

12-10-2005

Definition, Analysis, And An Approach For Discrete-Event Simulation Model Interoperability

Tai-Chi Wu

Follow this and additional works at: <https://scholarsjunction.msstate.edu/td>

Recommended Citation

Wu, Tai-Chi, "Definition, Analysis, And An Approach For Discrete-Event Simulation Model Interoperability" (2005). *Theses and Dissertations*. 1269.

<https://scholarsjunction.msstate.edu/td/1269>

This Dissertation - Open Access is brought to you for free and open access by the Theses and Dissertations at Scholars Junction. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholars Junction. For more information, please contact scholcomm@msstate.libanswers.com.

DEFINITION, ANALYSIS, AND AN APPROACH FOR DISCRETE-EVENT
SIMULATION MODEL INTEROPERABILITY

By

Tai-Chi Wu

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Industrial Engineering
in the Department of Industrial Engineering

Mississippi State, Mississippi

December 2005

DEFINITION, ANALYSIS, AND AN APPROACH FOR DISCRETE-EVENT
SIMULATION INTEROPERABILITY

By

Tai-Chi Wu

Approved:

Allen G. Greenwood
Professor of Industrial Engineering
(Director of Dissertation)

Larry G. Brown
Professor and Head Emeritus of
Industrial Engineering
(Committee Member)

Donna S. Reese
Professor of Computer Science &
Engineering
(Committee Member)

John M. Usher
Professor of Industrial Engineering
(Committee Member)

Stanley F. Bullington
Professor of Industrial Engineering
Graduate Coordinator in the
Department of Industrial Engineering
(Committee Member)

Kirk H. Schulz
Dean of the College of Engineering

Name: Tai-Chi Wu

Date of Degree: December 9, 2005

Institution: Mississippi State University

Major Field: Industrial Engineering

Major Professor: Allen G. Greenwood

Title of Study: DEFINITION, ANALYSIS, AND AN APPROACH FOR DISCRETE-
EVENT SIMULATION MODEL INTEROPERABILITY

Pages in Study: 217

Candidate for Degree of Doctor of Philosophy

Even though simulation technology provides great benefits to industry, it is largely underutilized. One of the biggest barriers to utilizing simulation is the lack of interoperability between simulation models. This is especially true when simulation models that need to interact with each other span an enterprise or supply chain. These models are likely to be distributed and developed in disparate simulation application software. In order to analyze the dynamic behavior of the systems they represent, the models must interoperate. However, currently this interoperability is nearly impossible. The interaction of models also refers to the understanding of them among stakeholders in the different stages of models' lifecycles. The lack of interoperability also makes it difficult to share the knowledge within disparate models.

This research first investigates this problem by identifying, defining, and analyzing the types of simulation model interactions. It then identifies and defines possible approaches to allow models to interact. Finally, a framework that adopts the strength of Structured Modeling (SM) and the Object-Oriented (OO) concept is proposed

for representing discrete event simulation models. The framework captures the most common simulation elements and will serve as an intermediate language between disparate simulation models. Because of the structured nature of the framework, the resulting model representation is concise and easily understandable.

Tools are developed to implement the framework. A Common User Interface (CUI) with software specified controllers is developed for using the proposed framework with various commercial simulation software packages. The CUI is also used to edit simulation models in a neutral environment. A graphical modeling tool is also developed to facilitate conceptual modeling. The resulting graphic can be translated into the common model representation automatically. This not only increases the understanding of models for all stakeholders, but also shifts model interactions to the “formulating” stage, which can prevent problems later in the model’s lifecycle. Illustration of the proposed framework and the tools will be given, as well as future work needs.

DEDICATION

I would like to dedicate this research to my parents and my wife Chiung-I.

ACKNOWLEDGMENTS

A profound gratitude is expressed to my major professor Dr. Allen G. Greenwood for his invaluable advice, invariable encouragement and guidance. His patience and kindness will never be forgotten.

A sincere gratitude is extended to Drs. Larry G. Brown, John M. Usher, Donna S. Reese, and Stanley F. Bullington for serving on my committee. Additional thanks are given to Dennis Mohr and Travis Hill for their help on my research and I wish the best of luck to them.

I wish to thank the Department of Industrial Engineering and Center for Advanced Vehicular Systems for their funding support.

Finally, every sincere and special appreciation is extended to my parents and my family, who gave effortlessly their love, support and prayers throughout my academic career.

CHAPTER	Page
III. REVIEWS OF EXISTING COMMON MODEL REPRESENTATIONS	37
3.1 Conditional Specification.....	38
3.2 Structural Modeling	39
3.3 Shop Data Model and Interface Specification	41
3.4 Other Approaches	43
3.4.1 Entity-Relationship Approach	43
3.4.2 Object-Oriented Approach.....	45
3.5 Comparison of Existing Model Representations	47
IV. PROPOSED SIMULATION INTERACTION APPROACH	49
4.1 Overview of the Proposed Approach.....	49
4.2 The Common Model Elements.....	56
4.2.1 General Information.....	57
4.2.2 Entity.....	59
4.2.3 Static Resource.....	60
4.2.4 Dynamic Resource	64
4.2.5 Linkage	66
4.2.6 Routing.....	67
4.2.7 Operation.....	69
4.2.8 Arrival	70
4.2.9 Non-Supported Elements	71
4.3 The Entity-Relationship Diagram.....	74
4.4 The Structural Modeling Schema	79
4.5 Common Graphic Elements.....	82
4.6 Summary.....	82
V. IMPLEMENTATION OF PROPOSED SIMULATION INTERACTION APPROACH	83
5.1 The XML DTD	83
5.2 The XML file	85
5.3 Graphical Modeling Tool.....	86
5.4 Common User Interface.....	93
5.4.1 The User Interface and XML Parser.....	93
5.4.2 The ProModel Controller.....	95
5.4.3 The QUEST Controller.....	97
VI. ILLUSTRATION OF THE PROPOSED APPROACH	99
6.1 Test Set 1.....	99
6.1.1 BullyBooks	100
6.1.2 Quarry problem.....	102
6.1.3 Shuttle bus problem	106

CHAPTER	Page
6.2	Test Set 2..... 107
6.2.1	Exercise 5.1..... 108
6.2.2	Exercise 5.2..... 110
6.2.3	Exercise 5.3..... 111
6.2.4	Exercise 5.4..... 111
6.2.5	Exercise 5.5..... 113
6.2.6	Exercise 5.6..... 114
6.2.7	Exercise 5.7..... 116
6.2.8	Exercise 5.8..... 118
6.2.9	Exercise 5.9..... 120
6.2.10	Exercise 5.10..... 122
6.2.11	Exercise 5.11..... 124
6.2.12	Exercise 5.12..... 124
6.2.13	Exercise 5.13..... 125
6.2.14	Exercise 5.14..... 125
6.3	Summary of the Chapter 125
VII.	LIMITATION OF THE PROPOSED APPROACH..... 127
7.1	Compatibility from Framework to ProModel (Opportunity 1)..... 129
7.2	Compatibility from Framework to QUEST (Opportunity 2)..... 132
7.3	Compatibility from ProModel to the Framework (Opportunity 3)..... 135
7.3.1	General Information..... 135
7.3.2	Entities 136
7.3.3	Locations..... 136
7.3.4	Resources 137
7.3.5	Arrival..... 138
7.3.6	Process 138
7.4	Compatibility from Quest to the Framework (Opportunity 4)..... 139
7.4.1	General Information..... 141
7.4.2	Parts..... 141
7.4.3	Machine and Buffer 142
7.4.4	Labor, AGV, and Controllers..... 144
7.4.5	Source 145
7.4.6	Connections..... 148
7.4.7	Process and Failure 148
7.5	Capturing Model Information Not Used in the Framework 151
VIII.	CONCLUSIONS AND RECOMMENDATIONS..... 157
REFERENCES 162

APPENDIX	Page
A STRUCTURAL MODELING SCHEMA.....	167
B XML DTD.....	170
C A SAMPLE XML FILE.....	173
D <i>ProModel</i> [®] LISTING.....	182
E SAMPLE LOG FILE.....	189
F SAMPLE UNINTERPRETED <i>ProModel</i> [®] XML FILE.....	197
G SAMPLE UNINTERPRETED <i>QUEST</i> [®] XML FILE.....	202

LIST OF TABLES

TABLE	Page
2.1 Mapping of Integration Requirements to Integration Approaches.....	28
4.1 General Information Elements Included in the Framework.....	58
4.2 Entity Elements Included in the Framework.....	59
4.3 Static Resource Elements Included in the Framework.....	61
4.4 Static Resource Elements Not Included in the Framework.....	63
4.5 Dynamic Resource Elements Included in the Framework.....	65
4.6 Dynamic Resource Elements Not Included in the Framework.....	66
4.7 Linkage Elements Included in the Framework.....	67
4.8 Routing Elements Included in the Framework.....	68
4.9 Operation Elements Included in the Framework.....	69
4.10 Arrival Elements Included in the Framework.....	71
4.11 Elements that are Not Supported by the Framework.....	73
4.12 Common Sequence Rules.....	75
4.13 Common Routing Rules.....	77
7.1 General Information Compatibility from Framework to ProModel.....	129
7.2 Entity Resource Compatibility from Framework to ProModel.....	130
7.3 Static Resource Compatibility from Framework to ProModel.....	130
7.4 Dynamic Resource Compatibility from Framework to ProModel.....	130

TABLE	Page
7.5 Arrival Compatibility from Framework to ProModel	131
7.6 Linkage Compatibility from Framework to ProModel	131
7.7 Routing Compatibility from Framework to ProModel.....	131
7.8 Operation Compatibility from Framework to ProModel.....	132
7.9 General Information Compatibility from Framework to QUEST	132
7.10 Entity Compatibility from Framework to QUEST	133
7.11 Static Resource Compatibility from Framework to QUEST.....	133
7.12 Dynamic Resource Compatibility from Framework to QUEST	133
7.13 Arrival Compatibility from Framework to QUEST	134
7.14 Linkage Compatibility from Framework to QUEST.....	134
7.15 Routing Compatibility from Framework to QUEST	134
7.16 Operation Compatibility from Framework to QUEST.....	135
7.17 General Information Compatibility from ProModel to Framework.....	136
7.18 Entity Compatibility from ProModel to Framework.....	136
7.19 Location Compatibility from ProModel to Framework	137
7.20 Resource Compatibility from ProModel to Framework.....	138
7.21 Arrival Compatibility from ProModel to Framework	138
7.22 Process Compatibility from ProModel to Framework.	139
7.23 General Information Compatibility from QUEST to Framework.....	141
7.24 Parts Compatibility from QUEST to Framework.....	142
7.25 Machine Compatibility from QUEST to Framework.....	143

TABLE	Page
7.26 Buffer Compatibility from QUEST to Framework	144
7.27 Labor Compatibility from QUEST to Framework	145
7.28 Source Compatibility from QUEST to Framework.....	147
7.29 Connection Compatibility from QUEST to Framework	148
7.30 Cycle Process Compatibility from QUEST to Framework	149
7.31 Failure Compatibility from QUEST to Framework	150
7.32 Repair Process Compatibility from QUEST to Framework.....	150

LIST OF FIGURES

FIGURE	Page
2.1 Sequential Model Integration	9
2.2 Many-to-One (Multiple Source) Integration	10
2.3 One-to-Many (Multiple Destination) Integration	11
2.4 Parallel Integration	12
2.5 Replacement Integration.....	12
2.6 Opportunities for Simulation Models to Interact During the Simulation Process	14
2.7 Conceptual and Programmed Model Interactions	16
2.8 Link Through Individual Observation.....	17
2.9 Link Through Distributions.....	19
2.10 Models Merged in Targeted Simulation Environment.....	21
2.11 Models Merged in Software Independent Environment	23
2.12 Peer-to-Peer Communication	25
2.13 Central Communication Bus	26
2.14 Model Interaction Opportunities at the Formulation Stage.....	29
4.1 General Interaction Strategy for Current Practice	50
4.2 General Interaction Strategy for the Proposed Approach.....	51
4.3 Components of Proposed Approach to Facilitate Simulation Model Interactions.....	54

FIGURE	Page
4.4 Relationships Between Common Data Elements	76
4.5 Performance Measurements of Common Data Elements	78
4.6 Example SM Schema	81
5.1 Example XML DTD	84
5.2 Example XML file	86
5.3 Visio Stencil of Model Elements	88
5.4 Graphical Modeling Tool Environment with Part1 Selected	91
5.5 Graphical Modeling Tool Environment with Part2 Selected	92
5.6 Overall Structure of the Common User Interface	93
5.7 Common User Interface	94
5.8 Management Functions in the Common User Interface	95
5.9 Transferring ProModel Files to the Proposed Framework	96
6.1 Graphical model of Bully Books	101
6.2 Graphical Model of Quarry Problem	104
6.3 Graphical Model of Exercise 5.1	109
6.4 Graphical Model of Exercise 5.2	110
6.5 Graphical Model of Exercise 5.4	113
6.6 Graphical Model of Exercise 5.5	114
6.7 Graphical Model of Exercise 5.6	116
6.8 Graphical Model of Exercise 5.7	117
6.9 Flow Chart of Exercise 5.7	118

FIGURE		Page
6.10	Graphical Model of Exercise 5.8.....	120
6.11	Exercise 5.9 With Lines for Different Customers	121
6.12	Exercise 5.9 with One Line for All Customers	122
6.13	Graphical Model of Exercise 5.10 With Orders Randomly Assigned	123
6.14	Graphical Model of Exercise 5.10 With Order Assigned to First Available	124
7.1	Relationships Between Simulation Packages and Implementations	127
7.2	Log File	152
7.3	Portion of the Uninterpreted XML File for ProModel Distribution Model ..	153
7.4	Portion of the Interpreted XML File for ProModel Distribution Model	154
7.5	Portion of the Uninterpreted XML File for a QUEST Model.....	156

CHAPTER I

INTRODUCTION

Decision-making is becoming increasingly complex, especially when the decisions involve entire production systems, enterprises, and supply chains. The complexity of these problems requires modeling to effectively address the large number of variables and interactions inherent in such systems. These decision problems are usually too difficult to be solved by traditional methods, such as linear programming. Discrete-event simulation is a primary means used to model and analyze complex systems. According to the Oak Ridge Centers for Manufacturing Technologies [1], “no other technology offers more than a fraction of the potential that M&S (modeling and simulation) does for improving products, perfecting processes, reducing design-to-manufacturing cycle time, and reducing product realization costs.” However, even though simulation provides great benefits to industry, it is largely underutilized [2]. It is especially underutilized when multiple models must interoperate in order to analyze large-scale organizational systems.

1.1 Problem of Simulation

One of the largest barriers to utilizing simulation to address problems in these complex systems is that disparate simulation models, those developed using different simulation software, do not *interact* or *interoperate* with each other; i.e., it is very

difficult for simulation models to act upon or influence each other. Entire models or portions of models oftentimes must be considered together so that the activities can be coordinated across models and the dynamic behavior of the overall system can be analyzed [3]. As McLean and Leong [2] point out, the most important factor that inhibits the use of simulation is cost. The extremely limited interoperability between disparate simulation models directly and indirectly increases the simulation implementation and model management costs.

Model reusability is also directly affected by the lack of interaction. For example, an existing model that is built using one simulation package cannot be executed in other environments. In order to execute it in other simulation packages, a new model must be built. However, if the model can be *transferred* to the new environment, it would save the duplication of modeling efforts. In addition to expanding the application domain, effective simulation model interactions also increase the useful life of existing models. Most models are built from scratch to solve an ad hoc problem and represent a portion of some larger system. Once the immediate problem is solved, these models are often no longer used. However, if they can be coupled with other models and enable the analysis of a broader system, their useful life would be greatly expanded. The modeling effort would provide a much greater return on the investment needed to build the models.

Poor, or no, model interactions create another problem – they create *islands* of simulation modeling and analysis. These islands result from inter- and intra-company barriers because of varying simulation expertise and software preferences. It also restricts the use of existing simulation models to analyze and improve the supply chain.

The lack of interactions also complicates the development of decision support systems (DSSs) and model management systems (MMSs). Since most DSSs require multiple simulation models to address a specific problem, the lack of interoperability of the simulation models greatly complicates the development of interfaces and communications protocol between disparate models. Selecting and managing models that are built and executed using different software or a different approach further complicates DSS development. This disparity in commercial software also inhibits the creation of a common representations of a simulation models. This lack of common representation creates problems of understanding among model builders and decision-makers (DMs) and complicates validation and verification.

1.2 Reasons for the Lack of Interoperability

There are several reasons why models do not interact with each other. A model is an abstraction of a system's behavior and there are many world views or ways to view, characterize, and represent a system as a model. For example, simulation time can be viewed as a continuous flow or a series of discrete-events. Even the discrete-event approach has several world views, such as *event-scheduling*, *activity-scanning*, and *process-interaction*. While the most common world view being used today is event scheduling (at least in the U.S.), there are numerous implementations; i.e., one for each simulation package on the market. Models that are built using different world views are not likely to interact with each other.

While the various implementations or simulation packages have many aspects in common (e.g., pseudorandom number generation, sampling from theoretical probability

distributions, and processing events over time), there are many significant differences. First, there is no common input/output format. For example, the output generated by a *ProModel*[®] model cannot be directly used as input to a *QUEST*[®] model (a simulation software package from the DELMIA Corporation). Second, there is no common interface or communication protocol between simulation software applications. For example, it is difficult to coordinate the execution of models in *ProModel*[®] and *QUEST*[®] because they cannot exchange information at run-time. Third, each simulation software vendor has their own approach to building models. Therefore, models of the same system are likely to be represented in such different formats that the behavior of the system is unintelligible unless someone has considerable expertise in the different packages. Fourth, there are few common elements and corresponding terminology between simulation packages. For example, *ProModel*[®] refers to the things that flow through a model as entities, General Purpose Simulation System (GPSS) refers to them as transactions, and *QUEST*[®] refers to them as parts. Finally, there is no common structure for simulation models, like there are for other types of models, e.g., mathematical programming system (MPS) format for defining linear programming models. Ideally, a system that is to be represented as a simulation model could be represented in a general format, then that format could be used by any applicable *solver*.

Also, typically a model of a system either lies in the modeler's mind or is embedded in some specific software; there is no generalized way to represent or formulate a simulation model. Given the same system, different model builders are likely to formulate and build the model in different ways.

1.3 Scenarios and Research Objectives

The poor interoperability between disparate simulation models creates a lot of problems. Consider the following scenarios: When an organization decides to build a simulation model to address a problem, it is not uncommon that decision-makers and the model builders are different groups of people. In this beginning, the model builders usually do not have enough information about the system to build a good model. Getting the right information is important to the success of simulation [4]. The decision-makers need to provide system information to the model builders, but they usually do not know what information is needed in order to build a model. Thus, the model builders may create a rough-cut model, ask the decision-makers to verify the model, then ask for more information, modify the model, come back to the decision-makers, and so on until the model is acceptable. This iterative model building process is usually very time consuming. If there were an intermediate language between model builders and decision-makers, this process would be considerably shortened.

Consider another scenario: Companies in a supply chain would like to integrate their simulation models to analyze the overall performance. However, they find it difficult to do it because their models are created using different simulation software packages. There is a need to identify possible methods to facilitate model interaction between disparate simulation models (meaning models that are built using different simulation software packages). But if the desired models can be translated into a common format, it will be easier to integrate disparate simulation models.

Regardless of all other possible model interaction approaches, from the above two scenarios, having an intermediate language (or a common model representation) between a human and a model, or between models, will largely improve the model interoperability. The purpose of this research is to develop a framework that enables a common model representation, and thus improves discrete-event simulation model interoperability. The study focuses on the model interaction problem and has two primary objectives:

- Define and analyze simulation model interactions, by:
 - Identifying, defining, and analyzing the types of simulation model interactions.
 - Identifying and defining possible approaches to allow models to interact.
- Develop an approach to improve model interoperability.

In Chapter 2, the various types of simulation model interactions are identified and defined. It also includes potential approaches for facilitating simulation model interactions, and the advantages and disadvantages of each. Chapter 3 reviews relative works on simulation model representation. Chapter 4 describes the proposed approach for facilitating model interaction. The implementation of the proposed approach is given in Chapter 5. Chapter 6 applies the proposed approach to a variety of simulation problems. Chapter 7 describes the limitations of the proposed approach and suggests methods to compensate for the limitations. Future research needs are discussed in Chapter 8. Finally, Chapter 9 draws conclusions from the research and summarizes the contributions.

CHAPTER II

DEFINITION AND ANALYSIS OF SIMULATION MODEL INTERACTIONS

Simulation models interact and interoperate when multiple models work together -- i.e., act upon or influence each other -- in order to analyze a system. To address the first research objective, and provide the basis for the second objective, this section identifies and defines the types of simulation model interactions and the general approaches for getting models to interact.

2.1 Types of Model Interactions

There are different types of interactions that are characterized along the following dimensions:

- degree to which the system description is revealed,
- where in the simulation process the interaction takes place, and
- the types of interaction relationships that exist between the models.

The degree to which a system description is revealed by a model can range from closed to open. A closed-type of interaction can be viewed as the exchange of information between “black boxes.” Only limited system knowledge is revealed. A very common example of this type of interaction is “blindly” using the output of one model as the input of the other. Another example of “closed” interaction is High Level Architecture (HLA). While HLA is described in a later section, basically each

federation (e.g., a simulation application) exchanges small specific sections of execution state information between others dynamically at run-time and thus only small portions of execution information are revealed; i.e., models essentially act as black boxes.

An “open” type of interaction is one where a full system description and operating logic are revealed and understandable by other models. This usually occurs when models are merged (e.g., into a common simulation package) and become integrated into a single implementation. It is very difficult to perform this form of interaction across simulation packages because of the lack of a common simulation model structure.

Another way to classify model interactions is pinpointing where in the simulation modeling process the interactions take place. Models either can interact at application or at formulation. Models are said to interact during application (not necessarily concurrently) when the state of one or more models depends on the state or output of another model or models; i.e., models interact with each other either to exchange information dynamically during run-time or the output from one or more models is used as input to another model or models. Models are said to interact during formulation when any stakeholder (not just modelers) needs to understand model logic and/or data or when common elements that will interact during execution need to be understood. For example, it is needed to ensure two models give the same entity an identical name.

The third way to characterize model interactions is by the type of relationships that need to exist between a set of models. The difficulty of getting the models to interact is heavily dependent upon the type of relationship that is required between models in

order to capture the system's behavior. There are five possible relationships between models: one-to-one, one-to-many, many-to-one, parallel, and replacement.

2.1.1 One-to-One Integration

One-to-one is the simplest type of integration. It is one where there is a simple sequential relationship between two models; i.e., the output of one model is the input to the other. As shown in Figure 2.1, this type of integration emphasizes the input/output relationships and ignores the content of simulation models. Blanning's [5] entity-relationship approach, where the relational data concept is applied to models, addresses this type of integration. Blanning [5] considers a model as a virtual file and all of the possible inputs and corresponding outputs are considered as records. Once the input/output types of two models fit, then it is possible for two models to be integrated.

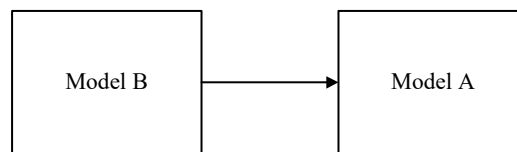


Figure 2.1 Sequential Model Integration

There are several advantages to this type of integration. First, models do not have to be run synchronously if there is no blocking effect; i.e., the output of one model is always accepted by the subsequent model. Blocking can occur when there is no space for transferring an entity and when multiple entities are combined in the receiving model. As long as the output of one model is stored properly, the subsequent model can execute any time. A second advantage is that two disparate simulation applications can be integrated

quite easily; the only requirement being a neutral data exchange format that is common or shared by the models. A third advantage is that the information and knowledge contained in the models are hidden from each other. This is an important characteristic in the case of supply-chain models since companies are usually reluctant to share the details of their operations.

2.1.2 *Many-to-One Integration*

As shown in Figure 2.2, a many-to-one integration is required when a model derives its inputs from multiple models; i.e., a model has multiple sources. This type of integration is similar to the sequential type of integration; however, blocking can have a more severe effect. If blocking is an issue, then all models need to run synchronously. Since the relationship between models is one of input/output, then the internal information of each model remains hidden.

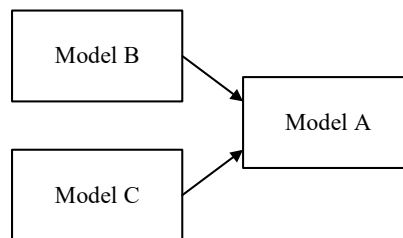


Figure 2.2 Many-to-One (Multiple Source) Integration

2.1.3 *One-to-Many Integration*

In a one-to-many integration, the output of one model drives multiple models; i.e., the output of a model has multiple destinations. This relationship is illustrated graphically in Figure 2.3. The main difference between it and a many-to-one relationship is the

routing logic involved. If the blocking is an issue, then a model needs to know the available capacity of subsequent models before it can send the outputs; therefore, the models need to run synchronously. Similar to many-to-one integration, model details remain hidden.

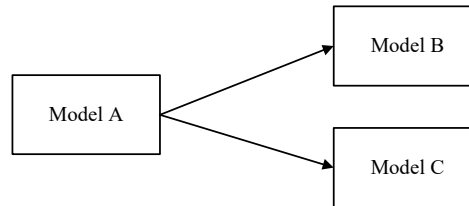


Figure 2.3 One-to-Many (Multiple Destination) Integration

2.1.4 *Parallel Integration*

As shown in Figure 2.4, parallel integration involves two tightly-coupled models that each receive input from, and provide input to, the other. The activities of each model need to be coordinated with the state of the other model. Analyzing a proposed expansion of a production facility is an example of parallel integration. This type of integration is obviously much more complex than input/output types defined above since the availability of dynamic resources, routing logic, linkage between static resources, and operation logic all need to be considered. In this case, models need to run synchronously and the information contained within the models is difficult to protect.

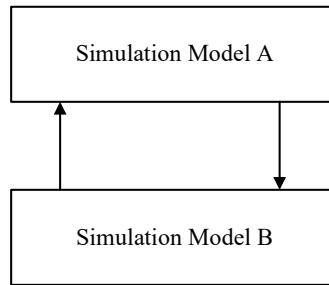


Figure 2.4 Parallel Integration

2.1.5 Replacement Integration

As the name implies, and as shown in Figure 2.5, one simulation model replaces a portion of another simulation model. This type of integration is typically used to build hierarchical models. For example, a high-level simulation model may represent an overall production facility with individual, more detailed models, used to represent work cells, shops, or departments. In this type of integration, models need to run synchronously and it is difficult to protect the internal information within each model.

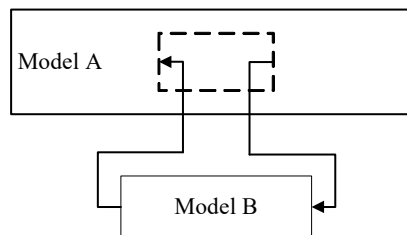


Figure 2.5 Replacement Integration

2.2 Approaches to Model Interactions

The simulation process is used to identify opportunities for model interaction. Figure 2.6 is a representation of a generic simulation modeling and analysis process.

While there is no standard process, Figure 2.6 represents a compilation of processes from a variety of sources (e.g., Banks & Carson [6], Law and Kelton [7], Shannon [8], Harrell, Ghosh, and Bowden [9]). As noted earlier, and as shown in the simulation process in Figure 2.6, there are two possible opportunities for simulation models to interact, i.e., at *application* and at *formulation*. Traditionally, model interactions occur at application when the system representation has been translated into a simulation language. The models that need to interact are called a *programmed* model. After models are built using different simulation languages, the model structure and terminologies are tied to a simulation vendor's specified model building approach. The content of the model is not likely to be understood by others that are using different simulation software. Only the observable information, such as output, can be used as means for interaction. However, even the output format and terminology vary greatly from package to package. Thus, the application stage primarily supports the closed type of interaction. Since most companies are not willing to share the details of their operation, this closed type of interaction is usually preferred when the interaction spans multiple enterprises.

In Figure 2.6, the steps that follow step “problem definition, statement of objectives, and develop project plan” are: “design experiments”, “model formulation”, and “data preparation.” This research focuses on model formulation. Commercial simulation software packages usually contain functions to assist in the “design experiments” step. For example, in *ProModel*[®], users can specify the number of replications and whether to exclude transient behavior. Those features are considered as experimental design and out of the scope of this research.

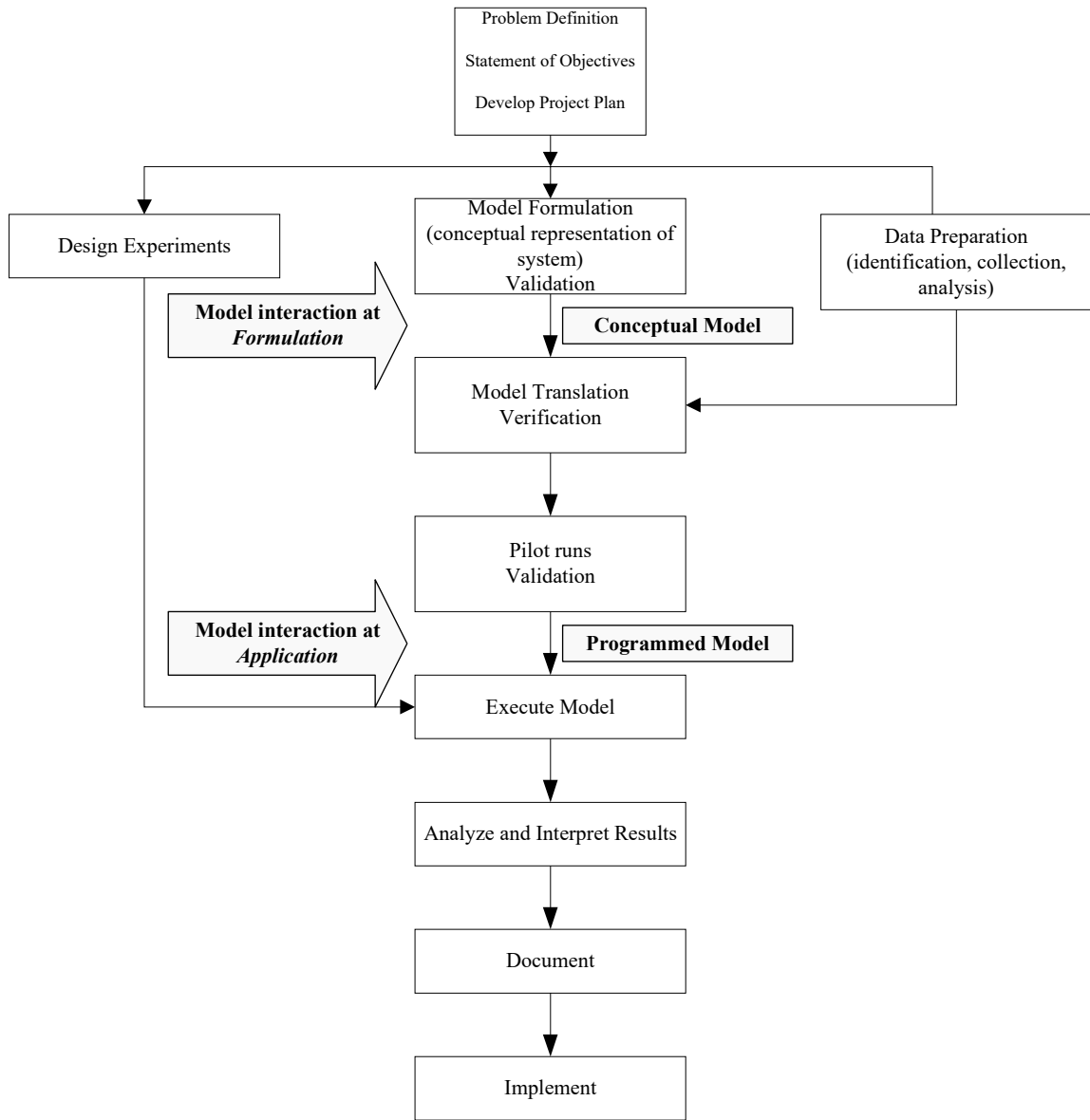


Figure 2.6 Opportunities for Simulation Models to Interact During the Simulation Process

As shown in Figure 2.6, interactions that occur at the formulation stage involve conceptual models. They are referred to as conceptual since the abstraction of a system's behavior typically lies in the mind of the modeler. The representation of the system at the formulation stage is used to verify that the modeler's observations and assumptions are correct, and to define the information that needs to be collected. Because all of the system information is revealed, the formulation stage primarily supports an open type of interaction. This stage is prior to the translation to a specific simulation package. Currently, there is no standard methodology for representing a system, formulating a model of the system, and depicting the relevant information that is needed to construct a functioning simulation model. The conceptual models can be text or graphical representations, e.g., activity diagram or flow chart. Addressing model interactions at the formulation stage, rather than at application, helps to avoid duplicate names, different measurement units, identical objects with different names, routing problems, reference problems, etc. Having well understood and clearly represented models at this stage greatly improves model validation and verification.

Figure 2.7 shows a more detailed view of the modeling process and the performer of each step in the process, i.e. human and machine performers. Typically, modelers manually translate conceptual simulation models into programmed models. However, in some cases, existing models need to be translated from one simulation environment to another, or integrated with other models. This is called model interaction at the application stage. Approaches and opportunities for interactions at the application and formulation stages are discussed in a subsequent section.

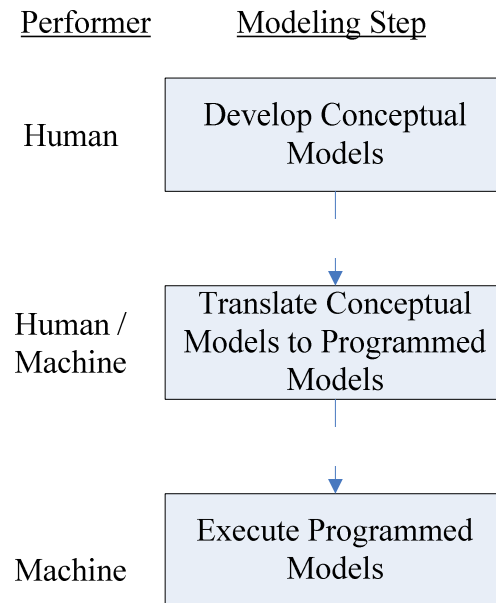


Figure 2.7 Conceptual and Programmed Model Interactions

2.2.1 Model Interactions at the Application Stage

There are several approaches for facilitating model interactions at the application stage. A review of them is given in this section.

2.2.1.1 Link Through Individual Observation

This approach stores the output from one model and then uses that information as input to subsequent models, as shown in Figure 2.8. To use this approach, a data exporter that records and saves the outputs of simulation is needed. Most simulation software applications have built-in functions to export the output data to an external file. The information that needs to be stored includes the type of output, quantity, and the time each observation leaves the system (or exit time between observations). Metadata is also

needed to indicate the description of output, the format of the file, the location of the file, etc. In the subsequent simulation software application, a data importer is needed to (1) establish the connection between the output data file(s) and the current simulation, (2) synchronize the exit time in the output data file(s) with the simulation time, and (3) map the output names and output types to appropriate arrival logics in the receiving simulation. This approach is only applicable to one-to-one, many-to-one, and one-to-many types of integration; it works well if blocking is not an issue. It is not suitable for integration that requires dynamic information from other models.

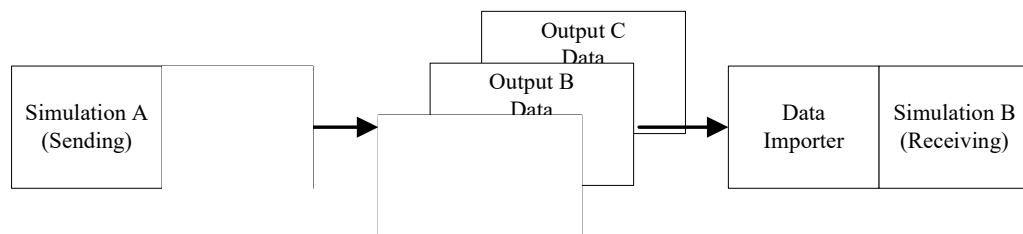


Figure 2.8 Link Through Individual Observation

Advantages:

- Models can be built in and run by different simulation software,
- Easy to implement,
- Information is protected within a model,
- Synchronous execution is not required.

Disadvantages:

- Only suitable for integrations that have no blocking effect,
- Requires a large amount of storage space,

- Transferring information between models is time-consuming and may cause errors,
- Access to external data may reduce the execution speed.

2.2.1.2 *Link Through Distribution*

Instead of linking two models with individual observations (output of one model is input to another model), the distribution of the output of each “sending” model is determined (primarily through distribution fitting), stored, and then used as input to a subsequent or “receiving” model, as shown in Figure 2.9. Similar to the link-through-individual-observation approach, the output data are exported through a data exporter. Then a data importer loads the output files into a statistical software application that fits the data to a probability distribution. These distributions are then used as part of the logic in the subsequent receiving simulations. For example, exit time data from the sending model may become the arrival time data for the receiving model. That is, the time between arrivals within the receiving model may be based on the time between exits from the sending model.

In this approach, synchronous executions are not required. The distribution may not adequately represent the actual behavior if the data set is not sufficiently large. The approach works well for one-to-one and many-to-one relationships. However, it is not suitable for integration that requires dynamic information from the other models.

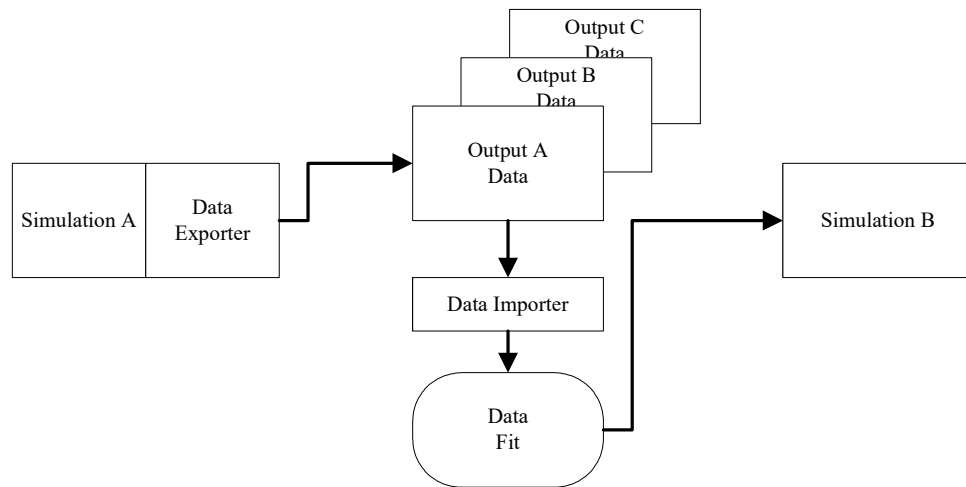


Figure 2.9 Link Through Distributions

Advantages:

- Models can be built in and run by different simulation software,
- Easy to implement,
- Information is protected within a model,
- Synchronous execution is not required,
- Fast execution speed.

Disadvantages:

- May not represent the actual behavior when sample size is small,
- Loss of dynamic interaction between models. Models become decoupled and semi-independent.

2.2.1.3 Common Structure/Common Application

In this approach, disparate models are translated to one common simulation structure, merged into a single simulation software application or a software independent environment, and executed in one implementation, as shown in Figure 2.10 and Figure

2.11. The first step of this approach is to develop a common structure, i.e., a common representation for all simulation models. The second step is to develop an interface in each simulation software application. This interface will enable the simulation application to export models into the common structure and import the models that are stored in the common structure. The final step is to merge the models. Because disparate models are frequently integrated during the application stage instead of the model building stage, the modeling methodology and naming rules are not the same. Some issues that arise from integrating models late in the models' lifecycle include duplicate names, different measurement units, identical objects with different names, routing problems, and reference problems. Because of these problems, manual integration of models is usually required.

Models can be merged either in a specified simulation software application or a software independent environment. In Figure 2.10, one model is saved into the common structure, and then translated into targeted simulation format. Then both models are merged in the targeted simulation environment. In Figure 2.11, models are translated into a common structure, then merged in a software independent environment. Then the merged models are translated into the targeted simulation format and executed in one implementation.

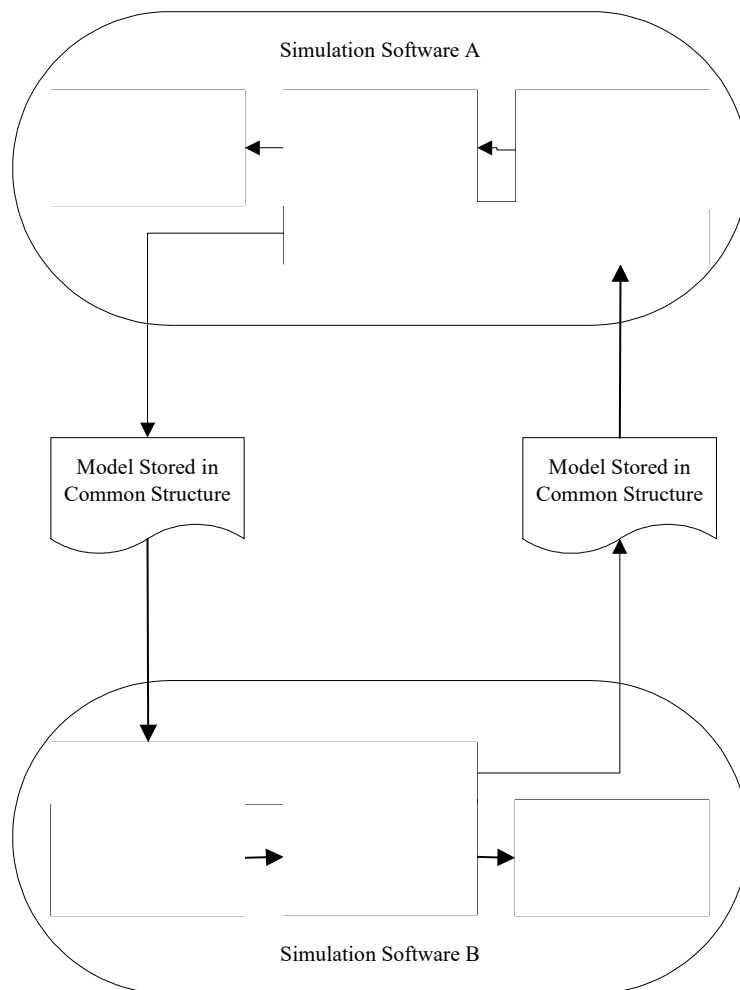


Figure 2.10 Models Merged in Targeted Simulation Environment

In order to implement this approach, a common structure is needed. Unfortunately, there is no common or standard format or means to represent discrete-event simulation models. If a common structure did exist, then a translator would be needed to convert models developed in one implementation to the standard and then the standard would be translated into another implementation. Several methods have been proposed to create a common structure, such as structural modeling [10], Simulation Data eXchange (SDX) [11][12], and condition specification (CS) [13] – none of them are widely accepted. Their lack of acceptance is primarily due to their not being capable of handling both the static and dynamic parts of discrete-event simulation models or they become extremely complex and difficult to implement. Both SM and CS (derived from the Conical Methodology) provide a methodology for facilitating model interactions in both the formulation and application stages of the simulation process. They are reviewed with “formulation” approaches because they provide a modeling methodology that covers both conceptual and programmed models. SDX, developed by Engineering Animation, Inc. (EAI), is a simulation standard that embeds simulation relative information with computer-aided design (CAD) objects. A library of CAD objects contains dimensional information as well as simulation-relevant information (e.g., processing time). The model building time can be shortened by pulling the predefined objects into either CAD or simulation software applications. The major shortfall of SDX is that it is not able to handle complex user logic.

Ideally, this integration approach is suitable for all five types of integration. Since this approach merges all of the information from each model implementation, the

resulting model should represent the behavior of the disparate models. However, the internal information of each model becomes revealed; also, the resulting model may become too large and hinder execution speed.

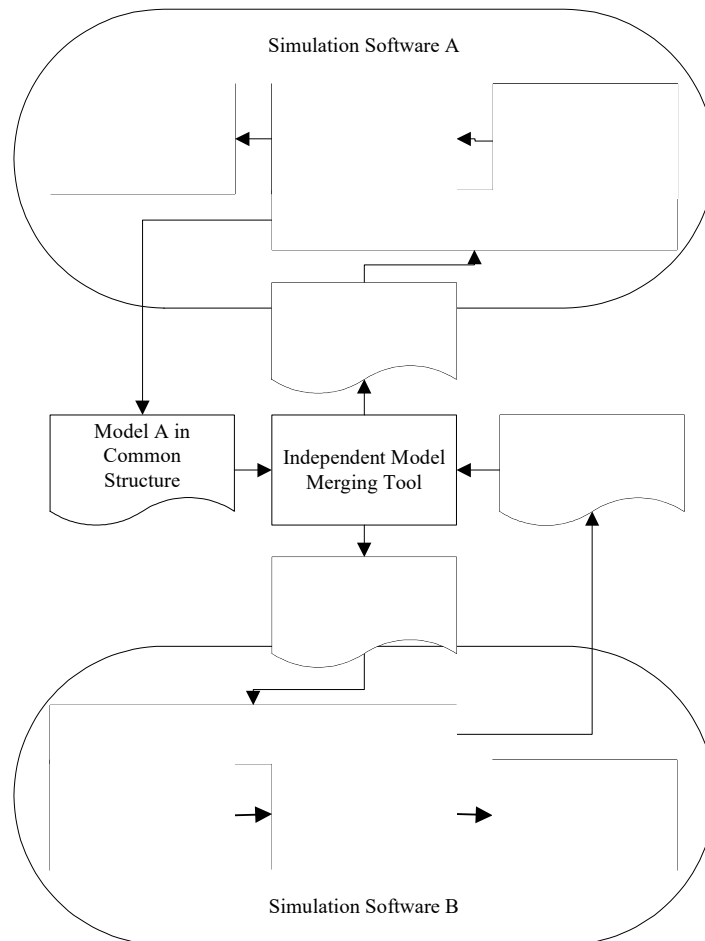


Figure 2.11 Models Merged in Software Independent Environment

Advantages:

- Very accurate since all information is captured in one implementation,
- Dynamic behavior is maintained,
- Applicable to all types of model integration,

- Applicable to all simulation application software.

Disadvantages:

- There is no common structure that can represent all simulation models,
- Information within simulation models is not protected,
- A translator needs to be developed for each simulation application software,
- Execution speed may degrade significantly since model size increases with integration.

2.2.1.4 Distributed Simulation

The fourth approach to simulation model interoperability during the application stage is distributed simulation. With the advancement of network technologies, it is possible to link disparate simulation models into a distributed simulation network. Object orientation concepts are widely used in this approach. Each simulation is viewed as an independent, yet interoperable, object. This approach primarily supports the closed form of model interaction in that only the essential information is revealed. An interface is needed for each simulation software application so that every simulation in the simulation network is able to communicate with each other. Heim [14] proposed a pure object-oriented distributed simulation network structure using peer-to-peer communication architecture that avoids large bandwidth. The general structure of peer-to-peer communication is shown in Figure 2.12. Because each simulation (object) in the distributed simulation network sends/receives information to/from others directly, the required communication bandwidth may be smaller but the overall information mechanism is harder to handle. Intelligent agents are used to help in integrating models.

The agent describes the information of a model, information needed, information generated, and the coordination requirements.

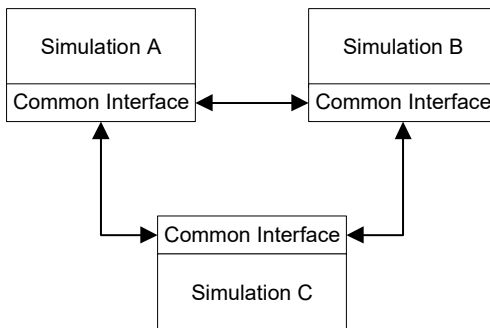


Figure 2.12 Peer-to-Peer Communication

The most notable work using the distributed simulation approach is the HLA that was developed by the U.S. Department of Defense (DoD) [15]. The HLA uses the HLA run-time infrastructure (RTI) to coordinate activities and information flows between different simulation models. An adaptor is needed in each simulation software application to communicate (send and receive specific information) with RTI [16]. In general, HLA uses a central communication bus that requires huge bandwidth. Figure 2.13 shows the general structure of HLA. A shortfall of HLA is that it fails to address the need for command and control systems, that is, a hierarchical structure of simulations, which is usually needed in a complex simulation network [17]. For example, a simulation model in the simulation network may be supported by several sub-models and databases. The HLA fails to support this kind of hierarchical structure. McLean and Riddick [15] also point out that a significant amount of coding is needed in order to integrate HLA with legacy simulation systems.

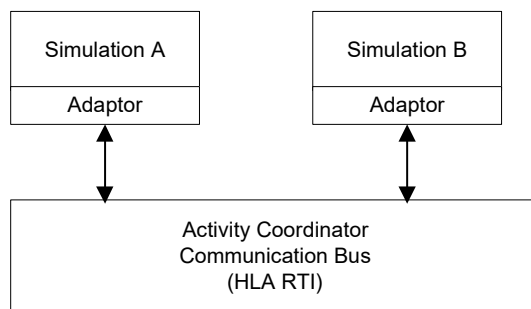


Figure 2.13 Central Communication Bus

The most significant difference between the distributed simulation approach and the common structure approach is that the latter merges the individual models into one large model as opposed to integrating simulation software applications together. All models in the distributed approach need to be run synchronously; therefore, there is a need for a good mechanism to coordinate the activities of models. Also, a communication and information filter mechanism is needed to transfer information between models either through a peer-to-peer bus or a central communication bus.

Advantages:

- Simulation behavior is accurate since model is unchanged and all data are used,
- Dynamic behavior is maintained,
- Different types of software can be integrated, e.g., database management system (DBMS) and simulation models can be integrated,
- Information is protected within a model,
- Overall execution speed may be improved through parallel computing,
- Utilize the strength of each simulation software application.

Disadvantages:

- Communication overhead may be very large,
- Execution speed may be greatly reduced due to large overhead,
- Considerable communication bandwidth may be required,
- Execution speed is tied to the slowest machine,
- An adaptor needs to be developed for every simulation software application,
- Model validation and verification may be difficult,
- Difficult to structure a hierarchical system of models.

2.2.1.5 Comparison of Approaches to Model Interaction at the Application Stage

Each model integration approach presented in this paper has its own pros and cons and no one approach dominates. However, in general, the common structure and distributed approaches support each type of model integration. The individual observations and distribution approaches are applicable only to the one-to-one, one-to-many, and many-to-one types of integration. Table 2.1 shows the level of applicability of the integration approaches to the different types of integration required. Since the maintenance of dynamic behavior between integrated models is usually very important, the cell contents in the table reflect to what extent dynamic behavior is maintained. Therefore, an “M” in the table indicates the dynamic behavior between the models is maintained, an “L” indicates the dynamic behavior is lost, and a “C” indicates the extent of the dynamic behavior is conditional on the extent of blocking. If the cell is blank, then the approach is not applicable to the type of integration.

Table 2.1 Mapping of Integration Requirements to Integration Approaches

Approaches to Integration at Application				
Requirements for Integration	Link Through Individual Observations	Link Through Distributions	Common Structure / Common Application	Distributed
One-to-One	C	L	M	M
Many-to-One	C	L	M	M
One-to-Many		L	M	M
Parallel			M	M
Replacement			M	M

2.2.2 Model Interactions at the Formulation Stage

The formulation stage primarily supports interactions between humans and models. At this stage, model contents are usually “open” to each other. Figure 2.14 illustrates three opportunities for human and simulation model interactions through: (1) visual means, (2) common data structure, and (3) commercial simulation software packages. The lettered white boxes in Figure 2.14 identify simulation model representations in three different formats: (a) conceptual framework of simulation model elements and relationships, (b) common graphical representation, and (c) common data representation. The dark boxes in Figure 2.14 represent software programs that enable the interactions between humans and the various model representations. The arrows mean that the representation at the source box can be translated into the representation at the destination box.

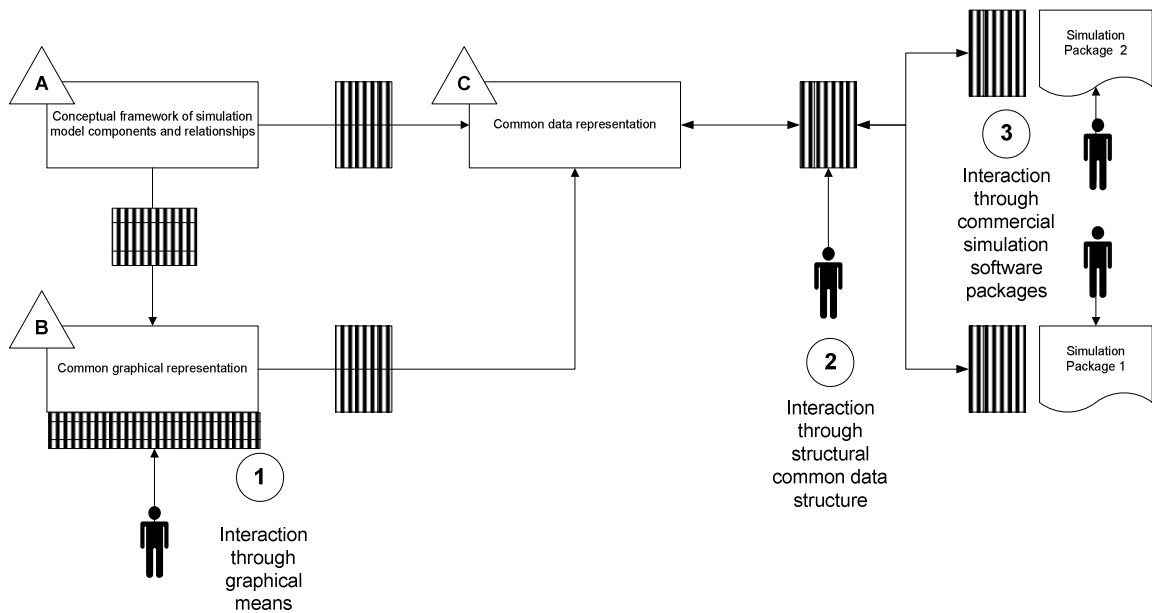


Figure 2.14 Model Interaction Opportunities at the Formulation Stage

The conceptual framework of simulation elements (represented by box A in Figure 2.14) is a set of rules for defining the basic elements of simulation models and their relationships, structure, data type, etc. The framework serves as the basis for developing various model representations, i.e., graphical representation and structural common data representation. Conical methodology (CM) [18], structural modeling [10], and the application of object-oriented concepts [19] [20] are examples of conceptual frameworks. A more detailed review of these approaches is provided in Chapter 3.

The most convenient and intuitive way to interact with a simulation model is through graphical or visual means. The common graphical representation (represented by box B in Figure 2.14) utilizes symbols to represent simulation elements and relationships. It provides a powerful tool for conceptual modeling. Early commercial simulation languages – such as *Q-GERT* [21], *SLAM II* [22], and *GPSS/H* [23]-- used combinations

of graphic symbols to construct the logic of simulation models. Each graphic representation is coupled with intermediate simulation statements that are subsequently executed on a specific “solver.” Unfortunately, each approach adopts a different world view and uses a different set of symbols. There is no standard way to represent simulation models graphically. A more detailed discussion of the graphical modeling approach is given in the following section.

With a common data representation (represented by box C in Figure 2.14), disparate simulation models can be translated into a common format. Having all of the models in the same format makes it much easier for the models to interact with each other. This standardized simulation data format can also serve as an intermediate language between conceptual models and programmed models, making it easier for non-modelers to understand the model contents in the common representation rather than a format associated with a specific simulation package. Examples of common data representations are condition specification [13] and structural modeling schema [10].

When human-model interaction occurs with specific simulation applications, at Interface 3 in Figure 2.14, different simulation vendors use completely different approaches and means for building models. Therefore, users must learn the approach, terminology, and syntax that are associated with each specific simulation software. A significant investment is required in order to become proficient in modeling using each package. Any resulting models are not able to interact with others, unless they were developed using the same simulation software.

2.2.2.1 *Graphical Modeling*

A trend of software design is toward the use of graphical representations. Visual programming is the concept of using graphical objects to present the system in mind. A good example of it is Microsoft™ Visual Basic. The main advantage of using intuitive graphical representations in programming is because it largely reduces the learning curve. Likewise, using graphical representations in the modeling process is called visual modeling. It is well known that a simulation model is an expensive tool. One important reason is the lack of effective, intuitive and general modeling approaches. Early simulation models were written in programming languages such as FORTRAN and C. These languages use unintuitive commands instead of graphics that are more intuitive and help users more easily connect portions of the model to real world objects.

Visual modeling is the idea of using graphical representations, such as icons and lines, to represent the model elements, constructs, and relationships. The most common graphical units include icons, lines, boxes, diamonds, ovals, and text strings. When arranging these graphical units under certain rules and providing the required data, the resulting drawing becomes an overall system representation and the data beneath it becomes a simulation model in a generic format. The resulting drawings are easier to understand than modeling languages. Thus, it can serve as a bridge between people who use models and people who build models. Another advantage of using graphical models is that they can be used to measure the complexity of simulation models [24].

It is common that decision-makers (DMs) and model builders are different groups of people. The DMs try to express what they want and the model builders try to

understand and translate this to models. This is usually an iterative process until the model builders come up with a model that meets DMs' needs. This process can take considerable time and is error prone since different groups of people usually speak different languages. Visual modeling can serve as a bridge between decision makers and model builders. The intuitive graphic representations avoid the need for DMs to memorize unintuitive commands. They can express their idea using high-level drawings. Then the model builders can use their draft models to develop them into more detailed and executable models.

Using graphic symbols to describe a system being modeled is not a new idea. PERT (Program Evaluation and Review Technique) is a graphical tool that describes the duration and dependency information of tasks that are required to accomplish a project. Lines in PERT represent tasks and nodes represent milestones. PERT is used to manage a project and does not have direct relationship with simulations.

Queuing – Graphical Evaluation and Review Technique (Q-GERT), developed by Pritsker [21], generalizes the PERT concepts with additional queuing and decision constructs. In Q-GERT, branches represent activities such as process or delay. Nodes represent model milestones, decision points, and queues. Nodes and branches in Q-GERT contain statements that portray key information and are executable by Q-GERT software. Each statement contains a three-character key word and parameters separated by commas. Entities that flow through the system are called transactions. Transactions differ only by their attribute values. Attributes can be assigned at any node. Like PERT, Q-GERT only focuses on how the tasks get done, the duration of tasks, the relationships

between tasks, and the flow of transactions. There is no direct relationship between the graphical symbols and real world objects. Q-GERT has fewer symbolic icons compared to other graphic modeling tools. The advantage is that it is easier to learn its symbology. The main disadvantage is that a node may contain too much information and the resulting statement becomes difficult to read. For example, a node may contain statistic collection information, label, initial number, capacity, user-defined functions, parameter set, etc. The resulting statement of the node may contain a long list of parameters.

Q-GERT analysis program is written in American National Standards Institute (ANSI) FORTRAN IV. Q-GERT has default FORTRAN functions to assist collecting statistical data. But users still need to explicitly specify how to collect the data. For example, to collect travel time for a transaction, an “M” must be put in the beginning node to record the time that a transaction enters the system. And an “I” must be put in the end to record the interval statistic. Q-GERT allows users to insert customized functions and sub-routines to accomplish complex logic. As a result, Q-GERT is a powerful graphical modeling system that permits direct computer analysis.

SLAM II evolved from Q-GERT and was also developed by Pritsker and Pegden [22]. It adds new features such as materials handling. Like Q-GERT, users can build graphical models with network symbols and translate them into input statements for computer analysis. SLAM II uses a hybrid world view and is able to handle discrete-event and continuous simulation modeling. Like Q-GERT, SLAM II implements activities as branches, nodes as milestones and decisions. SLAM II diagrams look very similar to Q-GERT. SLAM II fixes the problem that a node may contain too much

information by introducing more specific nodes. For example, in Q-GERT, attributes are assigned on basic nodes. SLAM II adds an assign node just to perform the attribute assignment task. In addition to using the combinations of nodes to represent complex logic, SLAM II also allows users to write their own code to accomplish complex logic. With the advance of hardware, it is easier to write custom codes. SLAM II is available in FORTRAN and C versions, which gives users more flexibility. Like Q-GERT, each graphic symbol in SLAM II directly ties to a statement. Each statement consists of a key word (not limited to three characters) and parameters separated by comma. SLAM II also introduces a statistics collection capability, called the COLCT node, that makes collecting performance measurements easier.

GPSS/H is also a graphical-based discrete-event simulation language. GPSS/H is used for “modeling system composed of units of traffic that compete with each other for the use of scarce resources” [23, p.16]. Units of traffic refer to unfinished products that flow through the system and are called transactions in GPSS/H. Each transaction possesses a unique ID and differs based on the attributes they carry. GPSS/H does not provide pictorial representations; however, a block diagram is used to express a GPSS/H model. The block diagram consists of different types of blocks and arrows. Each block represents an action, decision, or milestone. There are 60 types of blocks in GPSS/H while arrows have no meaning except to show the flow direction of transactions.

A model file in GPSS/H consists of block statements, control statements, and comment statements. Block statements directly correspond to the blocks in the block diagram. They describe the behavior of the system and are only executed at the time

transactions move into the blocks. Control statements are used to control the model execution, input/output of model, define features of model, etc. Comment statements provide additional information about the model and are optional, but strongly recommended. Statements in GPSS/H follow a specific format. Block and control statements can consist of three parts: label, operation, and operands. The label in the block diagram and the statements makes it easier to relate to real world objects. In GPSS/H, statistical data are collected automatically by the system. GPSS/H also allows users to write user functions and subroutines to accomplish complex tasks. GPSS/H is still one of the most general, flexible, and powerful simulation languages [25].

One problem with early graphic modeling languages is the graphic symbols do not have direct relationship with real world objects. For example, in Q-GERT, an activity may take 5 minutes to accomplish. This activity could be a machine drilling a hole on a panel, a doctor examining a patient, or an ATM machine finishing a transaction. Another problem is that the early languages try to teach the system “How” to do instead of “What” to do. This results in a complex representation of simulation models. For example, SLAM II has a series of blocks -- resource block, await node, free node, gate node, open node, preempt node, and alter nodes -- to manipulate the resources.

2.3 Summary of the Chapter

In this chapter, three ways to characterize model interaction types are discussed. The first way to classify model interaction is by the degree to which the system description is revealed, i.e., “open” versus “closed” type of interaction. The second way is by identifying where in the simulation process the interaction takes place, i.e., at

application or at formulation. The third way to classify model interaction is by the types of interaction relationships that exist between the models. Five possible relationships between models are discussed, they are: one-to-one, one-to-many, many-to-one, parallel, and replacement.

There are different approaches to achieving model interactions at application and at formulation stages. At the model application stage, the approaches include: link through individual observation, link through distribution, common structure/common application, and distributed simulation. The pros and cons of each are discussed as well as a mapping of interaction relationships to interaction approaches. At the model formulation stage, there are three opportunities for humans and simulation models to interact-through visual means, common data structure, and commercial simulation software. Reviews of existing visual means are provided in the chapter. A more detailed review on common data structure is given in Chapter 3.

CHAPTER III

REVIEWS OF EXISTING COMMON MODEL REPRESENTATIONS

Based on the analysis and definition of the types of simulation model interactions and approaches, the common structure and distributed simulation network approaches are applicable to all types of model interactions. As mentioned earlier, the models in a distributed simulation network act like “black boxes” and the model information is “closed;” thus, this approach does not facilitate model interaction in the formulation stage. The common structure approach supports all types of model interactions, in both formulation and application stages. Having a common model representation is essential to model interactions as well as model management [26]. To efficiently carry out the functionalities of model management systems, i.e., model development, model storage, and model manipulation, a common model representation format that is independent of software is needed [27][28][29][30]. There are several attempts to represent various types of models in a generic format. These include: structured modeling (SM), entity-relationship model, object-oriented approach, simulation data exchange (SDX), and condition specification (CS). Detailed discussions of these methods are given in the following sections.

3.1 Conditional Specification

Conical methodology (CM) is proposed by Richard E. Nance. “The CM is an object-oriented, hierarchical specification language that iteratively prescribes object attributes in a definitional phase that is top-down, followed by a specification phase that is bottom-up” [18, p.1]. Thus, CM defines the pieces, or components, of the system for building simulation models, especially for large complex models. The intent of CM is to provide disciplines and methods for the entire model lifecycle, e.g., from conceptual model to results. Consequently, CM divides model development into two phases, model definition and model specification. The top-down definition phase is to hierarchically decompose models into sub-models. At each level of hierarchy, attributes and elements are specified. The bottom-up specification phase defines the necessary information for a model. By completing the information of each hierarchical level from bottom to top, a complete representation of the model is thus finished.

The bottom-up specification phase focuses on creating a specification, which results in a CS as is proposed by Overstreet and Nance [13]. It is intended to “reduce modeling costs by interposing an intermediate form between a conceptual model and an executable representation of that model. As a model specification is constructed, the incomplete specification can be analyzed to detect some types of errors and provide some types of model documentation” [13, p. 190]. The CS consists of three components: interface specification, specification of model dynamics, and report specification. The interface specification defines the input and output of a model. The model dynamics specifications provide representation for the main body of the simulation. It consists of a

set of both object specifications and transition specifications. The object specifications define the elements and their attributes in a simulation model, such as facilities, resources, and positions. The transition specifications define the logic within the simulation model, such as initialization/termination logic, repair logic, arrival logic, and travel logic. The report specification defines the data that are to be collected and the logic how these data are to be collected.

The CS provides a complete method to define a simulation model. However, it is tedious to design a simulation model following CS since it requires the users to define every single piece of information about how to execute the model. Modern simulation software usually has built-in functions that handle a large part of the programming load for modeler. For example, the modern simulation software permits selecting a preferred routing rule from a list. In CS, however, modelers need to hard code every piece of logic.

3.2 Structural Modeling

Structured modeling is proposed by Geoffrion [10]. The purpose of SM is to identify the basic components of a model and store them in a structured format. It decomposes a model into manageable elements. The arrangement of elements and calling sequences decide the functionality of a model. The SM tries to cover major modeling areas, such as mathematical programming, data models, knowledge representation, and simulation models. Also, SM identifies the basic components of a model, the relationships between the components and then represents a model as an acyclic graph. There are three abstract degrees in SM: elemental structure, generic structure, and modular structure. The elemental structure defines the basic units of the models. These

units are primitive entity, compound entity, attribute, function, and test. Every element except primitive entity will “call” an associated element. This calling sequence is closed and acyclic. Thus, the element structure is a set of closed and acyclic elements. Generic structure groups the similar elements into a genus. Like the element structure, the generic structure is also a set of closed and acyclic genera. The modular structure is the highest level of SM and should have practical meaning to the users. It organizes all of the meaningful elements into a tree-like structure. For details of structured modeling, see Geoffrion [10] [31].

The SM was first designed for “static” models, i.e., the behavior of models will not change over time, such as linear programs. It is able to capture the numeric relationships between variables of the mathematical models. It also provides a standard format for representing models. However, SM is not able to capture the dynamic part of a simulation model in that SM cannot describe the behaviors of entities at a certain point of time. Researchers point out that SM is not designed for discrete-event simulation without considerable modification [31] [32]. The most significant difference between “static” and “dynamic” models is “time.” Since states within a simulation model change over time, it is necessary to define the behavior of model elements over time. If a time element can be properly integrated with static models, then it is possible to manage dynamic models [33].

Although SM was not originally designed for simulation models, considerable effort has been put forth to extend SM to discrete-event simulation. Lenard proposed an extended structured modeling (ESM) framework that adds three new types of elements

to SM, i.e., random attribute elements, action elements, and transaction elements [34]. A random attribute element is simply an attribute whose value is not known, but is based on a specified distribution. The transaction element is used to build up complex event lists. With certain preconditions, transactions invoke certain actions. An action element is used to specify a change to the current state of the simulation. Lenard's work suggests ESM is capable of capturing the dynamic behaviors of a simulation model. Since the transaction elements are modeled after the action clusters in CS, the framework also includes instructions on how to process the model and collect statistical data. As a consequence, the resulting schema is huge and complex. Yeo and Li tried to extend SM to discrete-event simulation by adding a new time element into the framework [33]. Their approach results in a complicated simulation time-advancement mechanism. Also, because each time point is recorded, it takes a tremendous amount of memory. Their approach also requires modelers to provide detailed instructions on how to perform computational tasks, thus, the resulting model representation becomes complex.

3.3 Shop Data Model and Interface Specification

Shop data model and interface specification is an extensible mark-up language (XML) based simulation specification developed by the National Institute of Standards and Technology (NIST) [35] [36] [37]. It contains all of the information needed to model a manufacturing system. It aims to provide a consistent data integration specification for discrete-event simulation. It is also referred to as the Machine Shop Data Model (MSDM) because, currently, it is only used to represent and exchange machine shop data [38]. The NIST will likely expand the specifications to business processes in the future. The work

is still ongoing and the resulting model will be promoted as a standard data interface for simulators.

The MSDM contains four major supporting data structures and fifteen major manufacturing data structures. The four major supporting data are:

- time sheets,
- probability distributions,
- references, and
- units of measurement.

The fifteen major manufacturing data structures are:

- organizations,
- calendars,
- resources (machines, stations, cranes, employees, tool-catalog, and fixture-catalog),
- skill-definitions,
- setup-definitions,
- operation-definitions,
- maintenance-definitions,
- layout, parts,
- bills-of-materials,
- inventory,
- procurements,
- process-plans (routing-sheets, operation-sheets, and machine-programs),

- work (orders, jobs, tasks, maintenance-orders, pick-orders, and tool-orders), and
- schedules.

Based on the data structure list, the MSDM completely covers all the information in a manufacturing system, from a manufacturing system's perspective. From a simulation perspective, the data elements in MSDM have a lot of similarities. For example, machines' and stations' data structures are represented the same way in *ProModel*[®]. Also, most discrete-event simulation packages use only a small subset of MSDM.

Currently, the shop data files are implemented through *QUEST*[®]. Using a translator, the shop data file is parsed into *QUEST*[®] batch control language (BCL) and Simulation Control Language (SCL) files and then executed in *QUEST*[®]'s simulation environment [35][39]. The advantage of this approach is that non-simulation experts can modify the shop data in MSDM and then generate *QUEST*[®] simulation models automatically. In the future, the NIST is going to apply this approach to various simulation software packages, as well as develop a graphical user interface (GUI) for collecting simulation data [35].

3.4 Other Approaches

3.4.1 Entity-Relationship Approach

The entity-relationship (ER) model has been used widely to represent entities and relationships between entities in database management [40]. Given the characteristics of the ER model, it is easy to represent the static part of simulation models, such as entity,

location, attributes, the inheritance of objects, and the relationships between them. However, the ER approach is unable to represent the dynamic part of a simulation model, e.g., performing different actions based upon different situations.

Blanning tries to apply the principles of the relational view of data to models [5]. He views a model as a virtual file, while all the possible inputs and its corresponding output are records. For example, consider a virtual file called “factory” (a factory model), where inside this file is a table(s) that contains records with fields named “raw materials” (for input) and “products” (for output). The “raw materials” and “products” fields represent the observable information of the “factory” model. It is usually easier to identify the purpose of a model by its observable information than its inner content. While it may not be easy to store most of the structure and logic of simulation models in traditional DBMSs, the input/output data can easily be stored in a DBMS. Thus, Blanning’s approach facilitates applying the DBMS practices to model management system (MMS) functions. For example, model selection can be done easier by sending the query “select product=“screw driver” .”

Blanning’s approach does not provide a standard format for models; instead, it tries to hide the physical part of models from users. This approach hides the tedious model details from the DMs and enables model selection by using query languages similar to structured query language (SQL) on input/output sets. This approach also helps model integration regardless of the physical difference of models. The limitation of the relational approach is that a virtual file is represented by its input and output set. It is possible that two completely different models have the same input/output set. This

approach can serve as an initial model selection approach, while additional information, such as metadata, is needed for accurate model selection.

3.4.2 Object-Oriented Approach

It is intuitive to view a manufacturing system as a set of objects, such as machines, pallets, and people. Object-Oriented Programming (OOP) has been in computer science for a long time and is also used in model representation [41]. In an object-oriented approach, models are built of reusable objects. An object is an entity that has its own private data and provides functionalities through a specified interface to others. The access to the data is via methods or functions that the object provides. That is, an object is a “black box” that receives inputs and sends outputs from certain ports to communicate with other objects. This mechanism facilitates building hierarchical models or command and control systems.

Because these objects (module models) are independent of their environment, they are easier to attach or detach from a base model in a order to build a new model or for conducting “what-if” experiments by plugging and unplugging the modeling components. Moreover, the principle of inheritance of objects makes it easier to develop new objects out of existing ones [42]. In short, the independent, yet interoperable, functional blocks (objects) largely increase the reusability of existing models [43]. Another advantage of object-oriented simulation is the encapsulation of objects that makes them more relevant to real world entities, thus facilitating the understanding of simulation models. It is also easy to represent a simulation model graphically because each object usually represents a real world entity [43].

Similar to distributed simulation discussed in Chapter 2, the object-oriented approach can be used to integrate existing objects into a new model based on a common structure. However, instead of integrating simulation software applications into a simulation network, objects are integrated into just one implementation. The point of using an object-oriented approach is to build a model quickly. Given that objects are from disparate environments, the problem becomes how to link the objects together. Zeigler suggests a coupling scheme that couples the input and output ports of the modules (objects) [44]. This input/output coupling is not restricted to physical input/output relationships, but also applies to state changes. For example, a machine's output port that indicates the state of the machine (busy/idle) may be coupled with its buffer's input port, so that the buffer knows when to send raw materials to the machine. A disadvantage of this approach is that the coupling may become very complex if there are a large number of objects in the system.

There are similarities between SM and OOP. The representation of a structured model is close to the object-oriented representation [20]. "Structured modeling formalizes the notion of a definitional system as a way of describing models. This is precisely what the object-oriented concept of a class and the class-composition graphs formalize" [45, p. 221]. Ma, Tian, and Zhou point out, using the object-oriented modeling concept, a discrete-event simulation model can be described by three different models, i.e., static model, functional model, and dynamic model [19]. A static model describes the properties of the system that are independent of time. A functional model (mathematical model) describes the numeric relationships between properties. A dynamic model

describes how the model changes over time, e.g., states, events, activities, and actions. SM is able to represent the first two kinds of model, but fails to represent the dynamic model. Ma et al. proposed a logic framework that takes advantage of object-oriented features to extend SM in order to capture dynamic part of models [19].

3.5 Comparison of Existing Model Representations

Structured Modeling fails to represent discrete-event simulation because it is not able to capture the dynamic part of a model. The ESM is able to capture both static and dynamic parts of simulation, but is too complex and difficult to implement. An object-oriented approach is able to handle both static and dynamic parts of a simulation and largely increases model reusability and interoperability. However, almost no commercial simulation software application supports the importing/exporting of objects from/to disparate simulation packages. Also, it is difficult to break down a simulation model into objects in some simulation software packages. The SDX provides a convenient way to build a model, but is not able to handle complex user logic [11][12]. Condition specification provides a complete method for representing simulation models, but it is complex and tedious to implement. While MSDM is a promising standard, it seems that fulfilling all of the required information in MSDM may be a real challenge. Also, the MSDM approach only supports one-way transformation, i.e., from MSDM to simulation. It does not help the reusability of existing simulation models.

Based on the literature reviewed in the above discussion, there is a need for a concise model representation that is:

- able to capture both static and dynamic parts of discrete-event simulation,

- easy to read and easy to write,
- easy to expand, and
- in a widely accepted format (open architecture).

Current simulation software applications usually provide some default options that are able to handle simple activities for users. Users no longer need to explicitly define how to perform some simple tasks. For example, to define the capacity of a queue, in the past, users needed to define events such as arrivals, departure, queue length handling, and statistic data collection methods. With modern simulation software application, users only need to specify the capacity of the queue, and then the application will handle the rest. This characteristic points out that it is possible to create a generic model format that only stores the critical information. Because the generic model format only contains important and intuitive information that supports all stakeholders, it will be concise, easy to read/write, and easy to learn.

It is believed that no common simulation model representation is able to satisfy all simulation software applications. Thus, expandability is an important factor in designing a generic format. Once the generic format is interacting with more simulation applications, new features can be added to it easily. A widely accepted storage media, such as XML, is also needed to develop a generic model format because it will interact with various simulation software applications.

CHAPTER IV

PROPOSED SIMULATION INTERACTION APPROACH

As discussed in Chapter 2, the common structure approach supports all types of model interactions in both formulation and application stages. However, no such approach exists. This section describes a proposed approach, based on the common structure concept, which is applicable at both the formulation and application stages.

4.1 Overview of the Proposed Approach

Figure 4.1 illustrates current practice, in terms of model interaction. Assume the three modelers in the left-hand portion of the figure all observe and model the same system. Because each model of the system either lies in each modeler's mind or is embedded in a specific simulation software package, the models will differ from each other. The conceptual models will be hard to understand by the other stakeholders because there is no common methodology or representation. Some modelers may use flow charts, but the symbols will likely differ; others may prepare detailed systems documentation. However, most will likely develop the model directly within a specific simulation package and employ no external visual representation. Similarly, the programmed models that are created by the modelers will be hard to understand by other modelers and other stakeholders, especially if they are not familiar with the specific

simulation software package. The resulting executable models will either be difficult or impossible to interpret by others.

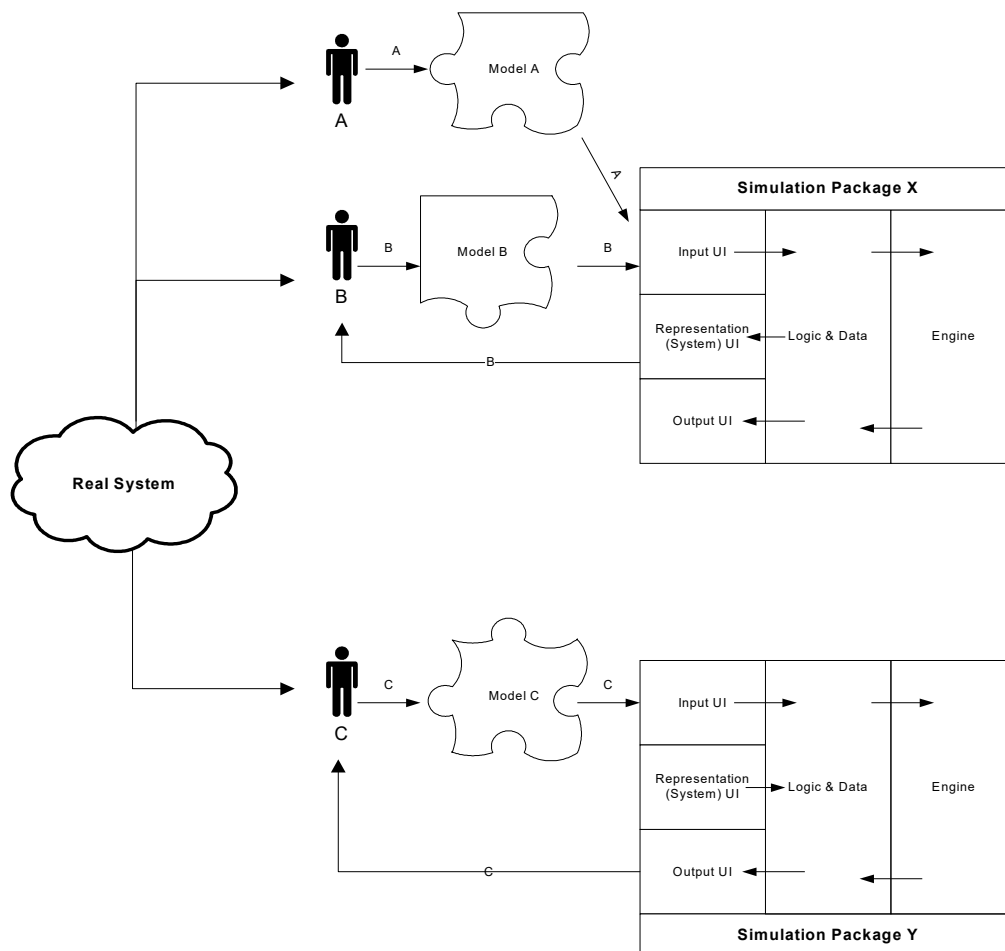


Figure 4.1 General Interaction Strategy for Current Practice

Commercial discrete-event simulation packages are represented in the right-hand portion of Figure 4.1. This conceptual representation shows that users interface with the software in three ways, by: 1) providing model logic in package-specific ways and providing system parameter and variable values (represented by Case A in the figure), 2) viewing the software’s representation of the model using its own constructs (Case B), and

3) receiving output resulting from execution of the model (Case C). As in also shown in Figure 4.1, each simulation package contains its own unique internal logic and methodologies for representing models. Similarly, each package also includes its own unique engine for executing the simulation models.

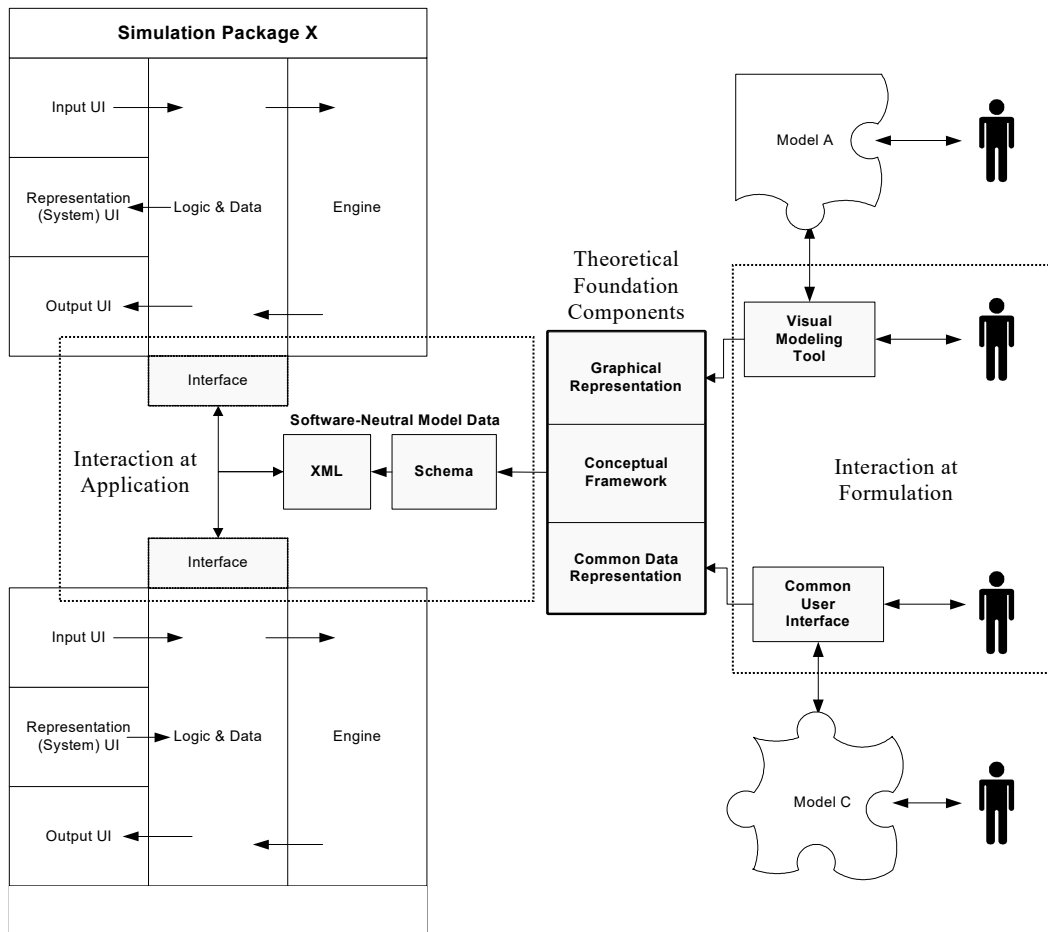


Figure 4.2 General Interaction Strategy for the Proposed Approach.

As mentioned earlier, the proposed approach addresses interaction issues at both the application and formulation stages; this is depicted in Figure 4.2. The visual modeling tool and common user interface (shown in the right-hand portion of Figure 4.2) permits

model builders to share their observation of the real system and also allows non-modelers to interact with models. Thus the proposed approach can facilitate interaction at formulation stage. After the models are built, they can be transferred to software-neutral model data, as shown in the left-hand side of Figure 4.2. These models are saved in a common format that makes interaction easier. It is also possible to upload these models to various simulation packages. To demonstrate this capability, a model of the Bully Books problem described in Section 6.1.1 is developed, saved in the proposed framework format, and then uploaded to *QUEST*[®]. Due to the length of the model listings, they are not provided in this dissertation, but are available upon request.

The three main components of the proposed approach are:

- conceptual framework for representing discrete-event simulation models,
- graphical representation of simulation models, and
- common data representation of simulation models.

The conceptual framework is the basis for constructing a graphical representation and common data representation. As shown in Figure 4.2, in order to implement the three main components, the proposed approach requires: 1) a visual modeling tool, 2) a common user interface, 3) software-neutral model data, and 4) interfaces to commercial simulation software packages. The theoretical foundation components (conceptual framework, graphical representation, data representation, and software-natural model data) and the software implementations (visual modeling tool, common user interface, and interfaces to simulation packages) are building blocks to the proposed approach; their relationships can be further described, as shown in Figure 4.3.

Component A (common model elements and relationships) is the basis for the proposed approach. Based on Component A, a structural modeling (SM) schema (Component B) that serves as conceptual framework is developed. Based on the SM schema, a common data representation and software-neutral model data (Component C and D) are developed. Also based on the SM schema, a common graphical representation (Component F) is created. With a software-neutral model data, the interfaces to commercial simulation packages can be built (Component E). And the availability of a common graphical representation also enables the development of a graphical modeling tool (Component G).

The components shown in Figure 4.3 can be subdivided into theoretical foundation components (Component A, B, and F) and software implementations (Component C, D, E, and G). The theoretical foundation components are introduced in the following sections and the software implementations are discussed in Chapter 5.

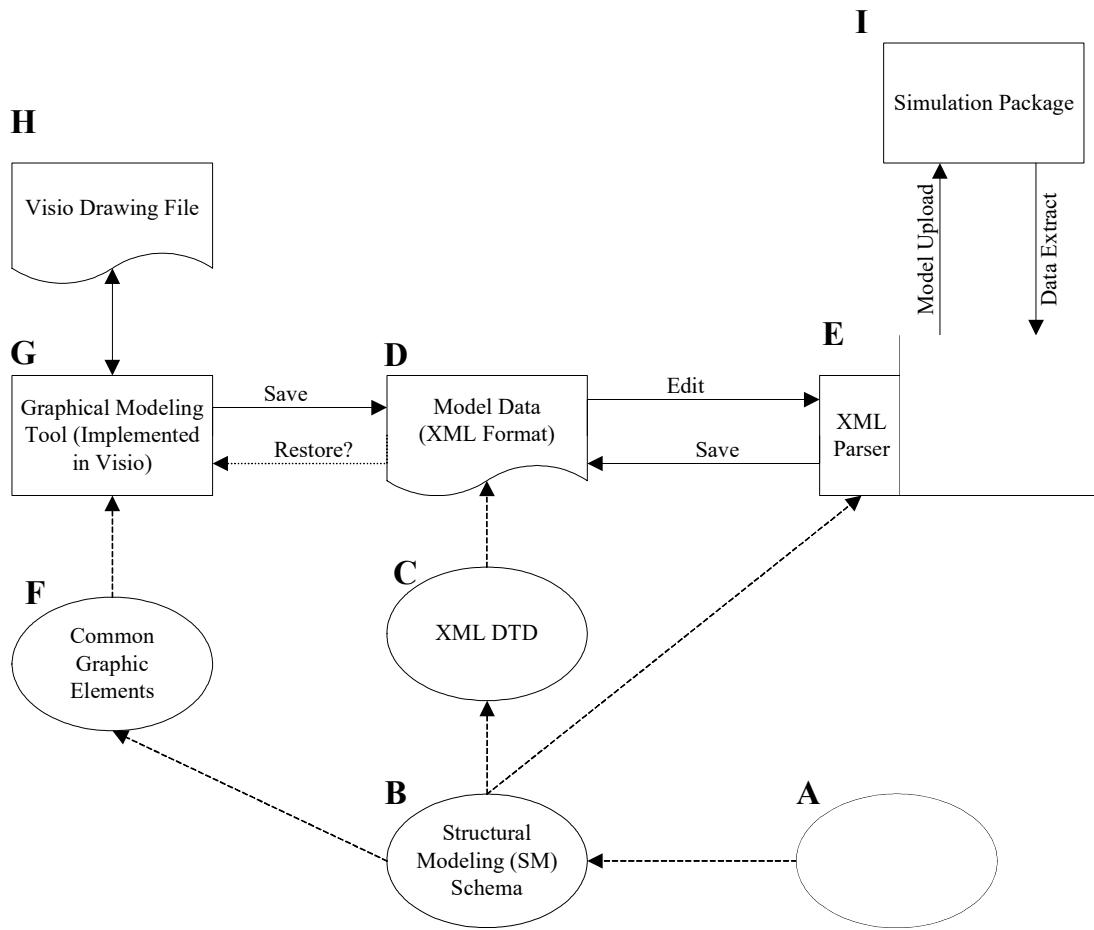


Figure 4.3 Components of Proposed Approach to Facilitate Simulation Model Interactions

Before developing a simulation package-independent common model structure, the most common simulation elements and their relationships must be defined. In Figure 4.3, component “A” represents an entity-relationship (ER) diagram that contains the most common elements of discrete-event simulation models, and the relationships between them. The common data elements are discussed in Section 4.2 while the relationships are discussed in Section 4.3. Based on “A”, a SM schema (Component B in Figure 4.3) is created. The detail of SM schema is discussed in Section 4.4. This model schema forms

the basis of the overall structure. In order to increase the interoperability and portability of simulation models, eXtensible Markup Language (XML) is used to store model information because of its structure and wide acceptance. In addition, XML documents are easy to maintain, read, write, and validate.

Component “C” is an XML Document Type Definition (DTD) that is based on the model schema and is used to verify the model information. A detail discussion of XML DTD is provided in Section 5.1. It is quite straightforward to transfer a model schema to XML DTD because both follow object-oriented principles. Component “D” represents the simulation model files stored in XML format, a example is given in Section 5.2. Component “E” is the common user interface (CUI) that allows modelers to edit and create models in a software-independent environment. This common user interface facilitates “application” interoperability. It reads and writes XML model files through *Microsoft*TM XML parser. It also contains a simulation software-specified control for uploading and extracting models to and from a specific simulation package (Component I in Figure 4.3). With the software specific control, models can be extracted from disparate simulation software and modified through the common user interface. Only the software controller needs to be changed in order to utilize different simulation software. *ProModel*[®] and *QUEST*[®] are used in this research because they provide ActiveX controls and batch control language (BCL) respectively that facilitate establishing connections to the CUI.

A finite set of common graphic elements (F) is derived from the model schema. With the common graphic icons, a graphical modeling tool (G) is developed that contains

a Visio template and macros. Modelers can create conceptual models by dragging and dropping the graphical icons that represent simulation elements onto a drawing. The drawing also serves as a communication tool between stakeholders that have varying levels of simulation expertise. The graphical modeling tool not only saves the drawing to Visio format drawing files (.H) but also contains macros for exporting the Visio files into standardized XML model files. Ideally, the XML model files can then be uploaded and executed using any simulation software package. Visio templates serve three purposes: they 1) facilitate “formulation” interactions, 2) limit simulation scope due to the finite set of model elements (no single discrete-event simulation representation can contain all the elements from all simulation packages), and 3) provide a means of documentation.

4.2 The Common Model Elements

This section defines the first step in building a common model structure -- determining the basic simulation elements and their relationships (Component A in Figure 4.3). Some research has focused on trying to define the basic simulation elements. Law identifies the most common model elements in a manufacturing system [46]. Bartolotta proposes an information exchange and interface protocols that contain required simulation information for solving manufacturing integration problems [47]. McLean at NIST developed a Machine Shop Data Model (MSDM) that defines the information requirements for a manufacturing system [38]. Among these proposed common data elements, MSDM is the most promising one, in that it may be promoted as a standard for all simulators by NIST.

The goal of the proposed framework is to create a bridge between all stakeholders. Thus, the framework should adopt both simulation and manufacturing experts' viewpoints. Therefore, the common model elements will represent a compromise between manufacturing (MSDN) and simulation (*ProModel*[®] and *QUEST*[®]). Also, it is believed no simulation representation can satisfy all simulation packages. The proposed framework should be as general as possible. The *ProModel*[®] information that is used to construct the framework is taken from [48] and [9]. The *QUEST*[®] information is taken from [49] and [50].

In the following sections of this chapter, the common simulation model elements are identified and defined. The same elements exist in *QUEST*[®], *ProModel*[®], and MSDM but may have different names, definitions, or characteristics. Establishing commonality, including critical properties of the elements, is essential for interoperability. The result of this step is a list of common simulation element names, properties, and definitions. It is recommended that the reader view Figure 4.4, the entity-relationship diagram of common data elements, first to get the overall view of the common data elements and the relationships between them.

4.2.1 General Information

General information stores the metadata of the model. In Table 4.1, the first row shows the application names, i.e., *ProModel*[®], *QUEST*[®], and MSDM. The last field in the first row, Framework, denotes the common model elements. The second row of Table 4.1 denotes the major elements of the application, and the properties of each element. *ProModel*[®] consists of several model elements; each element has attributes. *QUEST*[®] is

composed of element classes; each element class has properties. MSDM is a hierarchical structure, with the top-most level being the data structure; it may consist of multiple levels of complex data elements. The basic data elements are the lowest level in the hierarchy. Since later in this chapter, an ER diagram is used to represent the relationships between common model elements, ER diagram terminologies are used in the framework. The framework consists of eight elements; each element has its own set of properties.

Table 4.1 General Information Elements Included in the Framework

ProModel		<i>QUEST</i> [®]		MSDM			Framework	
Element	Attribute	Element class	Property	Data structure	Complex data element	(Basic) Data element	Element	Property
General Information	Title						General Information	Model Name
	Time units	Time units			Units of measurement	Time duration units		Time Unit
	Distance units					Length units		Distance Unit
								Date
								Builder
	Model notes	Model description						Notes

From Table 4.1, the common properties are: time unit, distance unit, and model description. Some properties are not shown under *ProModel*[®] or *QUEST*[®], but are implicitly implemented are model name, date, and builder. For example, the date property, when saving a *ProModel*[®] or *QUEST*[®] model, a date is automatically embedded. Note in *QUEST*[®], the time units and model description are placed at the same level as other element classes, thus they are placed under element class field.

In *ProModel*[®] there are initialization and termination logic sections that are not supported by the framework. These two properties are too programming-oriented and are usually used by experimental design experts; thus they are excluded from the framework.

4.2.2 Entity

Entities are objects that are routed and processed through the model. They are called parts in *QUEST*[®] and MSDM and transactions in SLAM II. In Table 4.2, the common properties are: name, attribute, and notes. Attributes denote a characteristic of the entity. Although it is not shown under MSDM, it is implicitly included in MSDM (e.g., bill-of-material). The speed property is a unique characteristic of *ProModel*[®]. *ProModel*[®] tries to simplify the modeling process by omitting some modeling details. In *QUEST*[®], to move an entity from one machine to another, a dynamic resource (labor) must be assigned, or there will be no move time between two machines. In *ProModel*[®], the entities can move by themselves (without the use of dynamic resources). The speed property is used to simplify the model, thus is adopted by the proposed framework.

Table 4.2 Entity Elements Included in the Framework

ProModel		QUEST		MSDM			Framework	
Element	Attribute	Element class	Property	Data structure	Complex data element	(Basic) Data element	Element	Property
Entity	Name	Part	Name	Parts		Name	Entity	Name
	Attribute		User attribute					Attribute
	Speed					Speed		
	Notes		Descriptions			Descriptions		Notes

Five *QUEST*[®] properties are not included in the framework: priority, routing labor requirement, routing sub-resource (SR) requirement, associated SR class, and required process. Priority is considered an advanced model feature, to keep the framework simple, it is excluded. The routing labor requirement, routing SR requirement, associated SR class, and required process define the resource requirement to move and process an entity. They are not required within entity definition because they are also defined in routing and process logics. Thus they are excluded from the entity element in the framework.

4.2.3 *Static Resource*

Static resources are places where entities are processed using certain manufacturing resource, e.g., machines. They are called locations in *ProModel*[®], machines and buffers in *QUEST*[®], and machines and stations in MSDM. In the proposed framework, they are referred to as static resources in contrast to dynamic resources (e.g., labor) because they do not move within the system. As shown in Table 4.3, the common properties are: name, type, capacity, units, downtime, repair, sequence rule, attribute, and notes. The type property denotes the class of static resource. Location in *ProModel*[®] may be a machine, buffer, conveyor, tank, etc. In *QUEST*[®], machines, buffers, and conveyors are viewed as different element classes. Machine and stations are considered resources in MSDM. To handle different types of static resources, the proposed framework adopts a type property. Currently, the type property only supports buffer and processing unit. The conveyor is not supported because the specifications of conveyor are too complex and different simulation software packages implement them differently. Besides, the

conveyor can be modeled as a buffer with delays to simplify the model. The fluid-process relative resources (e.g., tank) are not included in the framework because most manufacturing systems do not use fluid processes. The type property enables the framework to expand and support more types of static resources as needed.

Table 4.3 Static Resource Elements Included in the Framework

ProModel		QUEST		MSDM			Framework	
Element	Attribute	Element class	Property	Data structure	Complex data element	(Basic) Data element	Element	Property
Location	Name	Buffer, Machine	Name (Machine or Buffers)	Machine, stations		Name (Machine or Stations)	Static Resource	Name
	Capacity		Buffer		Part Capacity			Work piece capacity or employee capacity
	Units		No. of elements			Number		Units
	Downtime logic	Failure			Reliability statistics	MTBF		Downtime (TBF)
	Downtime logic	Repair process			Reliability statistics and Maintenance definition	MTTR, estimated duration, Maintained resource		Repair (TTR and repair resource)
	Rules	Buffer	Logics (queue logic)					Sequence Rule
	Attribute	Machine	Attribute					Attribute
	Notes	, Buffer	Description			Description		Notes

In *ProModel*[®] and *QUEST*[®], users can define the downtime and repair properties in great detail, e.g., scheduled downtime. To keep the framework simple and general, only three properties are supported: time between failures (TBF), time to repair (TTR), and repair resource. Also, *ProModel*[®] and *QUEST*[®] support multiple downtime for each static resource. Currently, the framework only supports one downtime and one

repair logic to simplify the framework, but the support of multiple downtimes and repairs can be easily added to the framework.

Elements that are not supported by the framework are shown in Table 4.4. In *ProModel*[®], rules include incoming entity selection rules, output queuing rules, and static resource selection rules. The framework only supports the output queuing rules because it is most commonly used. In *QUEST*[®], logics include: process logic, part input logic, route logic, initial logic, request input logic, queue logic, and request logic. The framework only supports route logic and queue logic. Other logics are left for future expansion of the framework. Shifts and costs are usually considered as advanced simulation features and are left for future expansion. Most manufacturing systems do not use fluid processes, thus this feature is also left for future expansions. Setup process, load process, and unload process are discussed in Section 4.2.7. In MSDM, the shift (schedule) information is defined in great detail; it is left for future expansions.

Table 4.4 Static Resource Elements Not Included in the Framework

ProModel		QUEST		MSDM		
Element	Attribute	Element class.	Property	Data Structure	Complex data element	(Basic) Data element
Location	Rules	QUEST	Logics			
	Shifts	Shifts	Shift break	Schedules	Resource section	Station-schedule
			Daily schedule			Machine-schedule
			Multi-day schedule			Crane-schedule
						Employee-schedule
						Work section
					Order-schedule	
					Job-schedule	
					Task-schedule	
					Maintenance-order-schedule	
					Pick-order-schedule	
			Tool-order-schedule			
Cost				Resource		Hourly-rate
Fluid process		Fluid process				
		Setup process		Setup definition		
		Load process				
		Unload process				

Several unique *QUEST*[®] properties are not included in the framework, they are: priority, part initial stock, process percentage, process group, thresholds, delay time, request routing, and dedicated labors. Priority is not included in the framework because it is an advanced feature. Part initial stock is an advanced modeling feature and belongs more to experimental design, thus is left for future expansions. The process percentage is not included in the framework because it is assumed all processes are executed in sequence. Process group is a unique feature of *QUEST*[®] that puts multiple processes together. This may facilitate the modeling process. This feature is not included in order to

keep the framework simple. Thresholds are also a unique feature to *QUEST*[®]; they define the safety inventory level for a buffer class. This feature is only available in pull systems and thus, is not included in the framework. Delay time is a unique attribute for the buffer class. It serves the same purpose as processes, thus is not included in the framework. Request routing is only available for pull systems and is not included by the framework. Dedicated labor is a unique attribute for *QUEST*[®]. The labor requirements are also defined in process and routing, thus this attribute is not necessary and is not included in the framework.

4.2.4 *Dynamic Resource*

Dynamic resources are objects that move around in the system to facilitate processing, moving entities, or maintaining resources. They are referred to as resources in *ProModel*[®], labors in *QUEST*[®], and employees or cranes in MSDM. In *ProModel*[®], a resource can represent an operator, a truck, or anything that moves around in the system. In *QUEST*[®], it explicitly divides dynamic resource into three resource classes: labor, automatic guided vehicle (AGV), and carrier. These three element classes are very similar except that AGV and carrier classes require a path system. Path systems are usually considered an advanced simulation feature, thus AGV, carrier, and path systems are not included in the framework. In MSDM, there are two types of dynamic resources: employees and cranes. These are very similar except that cranes have downtime/repair logics and employees have personal information, such as telephone numbers. The common properties shown in Table 4.5 are: name, units, downtime, repair, and notes. These common properties were discussed in the previous section.

Table 4.5 Dynamic Resource Elements Included in the Framework

ProModel		QUEST		MSDM			Framework	
Element	Attribute	Element class	Property	Data structure	Complex data element	(Basic) Data element	Element	Property
Resources	Name	Labor	Name	Crane		Name	Dynamic Resource	Name
	Units		No. of Elements	Employees		Number		Units
	Downtimes	Failure		Crane	Reliability statistics	MTBF		Downtime (TBF)
		Repair process			Reliability statistics and Maintenance definition	MTTR, estimated duration, Maintained resource		Repair (TTR and dynamic resource)
	Notes	Labor	Description			Description		Notes

Table 4.6 contains the elements that are not included in the framework. In *ProModel*[®], specifications define the path network and motions of resources. The search property contains the searching logic in the path network. The logics define entry and exit behaviors on the path network. The points property defines the positions of a resource's traveling path. All of the above features are related to path network systems and are considered advanced modeling features, thus are left for future expansion.

Table 4.6 Dynamic Resource Elements Not Included in the Framework

ProModel		QUEST		MSDM		
Element	Attribute	Element class	Property	Data Structure	Complex data element	(Basic) Data element
Resource	Specifications	Labor		Resource (employees)		
	Search					
	Logics					
	Points		Logics			
			Controller			
			Part capacity			
			Shifts			
			Speed			
			Priority			
			Load process			
			Unload process			
			Rotation speed			
						Skills definitions

In *QUEST*[®], a controller class contains the logics of the labor class. It allows users to define the behavior of the labor class in great detail. It is considered an advanced feature and is left for future expansion. Priority and shifts were discussed in the previous section. Unload process, load process, and part capacity are unique features of *QUEST*[®] that are left for future expansion. The labor speed and rotation speed are also unique features of *QUEST*[®]; they can be combined with the distances between static resources to calculate the move time automatically. The features are being considered for addition to the framework in the near future.

4.2.5 Linkage

Linkages define the input and output relationships between two static resources. They are referred as connections in *QUEST*[®] and paths in MSDM. As shown in Table 4.7, the common properties are: “begin loc.”, “end loc.”, distance, and notes. In

ProModel[®] and *QUEST*[®], the distance is implicitly implemented; the distance between static resources is automatically calculated. The distance property facilitates automatic calculation of move time if entity speed or dynamic resource speed are presented.

Table 4.7 Linkage Elements Included in the Framework

ProModel		QUEST		MSDM			Framework	
Element	Attribute	Element class	Property	Data structure	Complex data element	(Basic) Data element	Element	Property
Process	Location	Connection	Starting element	Layout	Paths	Path route	Linkage	Begin loc.
	Destination		Ending element			Path route		End loc.
								Distance
								Notes

4.2.6 Routing

In *ProModel*[®], the processing element consists of process and routing. In *QUEST*[®], routing logics are included in static resources. It is challenging to extract the routing information because both *ProModel*[®] and *QUEST*[®] have different modeling views. Table 4.8 contains the common properties for routing: linkages (which contain duration, dynamic resource, and user-defined logics), route entity, routing rules, quantity each, and notes. Routings are logics that define how entities (route entity) should move between static resources (linkages). In *ProModel*[®], linkages would refer to the location (in the process definition) and destination (in the routing definition). In *QUEST*[®], linkages refer to outputs. The sub-properties, i.e., durations, dynamic resource requirements, and user-defined logics define the duration and required resource to move

the entity along the linkage; they all have their corresponding fields in *ProModel*[®], *QUEST*[®], and MSDM. The routing rules in the framework only support commonly used rules (i.e., next free, by turns, random, least utilized, and percentage); other routing rules are left for future development. The attribute “Qty-each” defines the routing quantity. *QUEST*[®] only supports the routing of one entity per time.

The framework does not support pull systems. Thus, several *QUEST*[®] properties are not supported, they are: pull inputs, pull outputs, request routing, and request input logic.

Table 4.8 Routing Elements Included in the Framework

ProModel		QUEST		MSDM			Framework	
Element	Attribute	Element class	Property	Data structure	Complex data element	(Basic) Data element	Element	Property
Routing		Machine, Buffer	Outputs	Process plan	Routing sheet (plan definition)		Routing	Linkages
	Move logic		Labor move time		Routing sheet (plan definition)	Estimated duration		Linkage (Duration)
	Move logic		Labor requirements		Routing sheet (plan definition)	Resources required		Linkage (Dynamic resource)
	Move logic		Route logic (User function)		Routing sheet (plan definition)			Linkage (User logic)
	Output		Part routing (restrictions)		Routing sheet (plan definition)			Route Entity
	Rule		Logic (route logic)		Routing sheet (plan definition)			Rule
	Rule					Batch sizes		Qty Each
						Description		Notes

4.2.7 Operation

Operations define the tasks performed at static resources that turn entities into products or work-in-process (WIP). Operations are referred to as processes in *ProModel*[®], cycle processes in *QUEST*[®], and process-plan in MSDM. Table 4.9 contains the common properties, they are: Process (includes sub-properties duration, dynamic resource, and user-defined logics), Op. Location (static resource where operations are performed at), InEntity (entity that operations are performed on), OutEntity (products), and notes.

In *ProModel*[®], the process definition is a segment of code, thus users can define multiple processes. In *QUEST*[®], a static resource can have multiple process elements. Thus, the framework also allows for multiple processes.

Table 4.9 Operation Elements Included in the Framework

ProModel		QUEST		MSDM			Framework		
Element	Attribute	Element class	Property	Data structure	Complex data element	(Basic) Data element	Element	Property	
Process	Operation	Cycle process	Cycle time	Process-plan (Operation-sheet)	Plan-definition (plan-steps)	Plan-step (estimated duration)	Operation	Process (Duration)	
	Operation		Labor requirement		Plan-definition (plan-steps)	Resource required		Process (dynamic resource)	
	Operation				Plan-definition			Process (user logic)	
	Location		(Associate with Machine)		Operation-definition	Station		Op. Location	
	Entity		Part requirements		Plan-definition			InEntity	
Routing	Output		Products			Plan-definition			OutEntity
			Description					Description	Notes

As mentioned earlier, *QUEST*[®] also includes a fluid cycle process, setup process, repair process, load process, and unload process. As discussed earlier, the framework does not support fluid process. Repair process is partially supported in the repair property in static and dynamic resources. The setup process, load process, and unload process are essentially the same as the cycle process except they are resource-oriented. Since the framework supports multiple processes, the setup process, load process, and unload process are combined with processes to simplify the framework.

4.2.8 *Arrival*

Arrival defines entity creation logic in a simulation. Note MSDM does not have any corresponding elements, because it is based on a manufacturing view. As in the real world, orders always come before entities, i.e., orders create the arrivals of the entities. In MSDN, two elements are related to arrival logics: works (including jobs, tasks, orders, maintenance orders, pick-orders, and tool-orders) and bill-of-materials. In *QUEST*[®], the source class is the combination of arrival logic and a buffer class. The framework only takes the arrival logic part. Table 4.10 shows the common properties: *ArvEntity* (arrival entity), *ArvLocation* (arrival location), *QtyEach* (quantity each arrival), *Frequency*, *FirstTime*, *Occurrence*, and *User-defined logic*. These properties are self-explanatory.

Table 4.10 Arrival Elements Included in the Framework

ProModel		QUEST		MSDM			Framework	
Element	Attribute	Element class	Property	Data structure	Complex data element	(Basic) Data element	Element	Property
Arrival	Entity	Source	Name				Arrival	ArvEntity
	Location					ArvLocation		
	Qty each		Lotsize			QtyEach		
	Frequency		IAT*			Frequency		
	FirstTime		Start Offset			FirstTime		
	Occurrences		Max. Part Count			Occurrence		
	Logic					UserLogic		

* Inter-arrival time.

In *QUEST*[®], a source can create multiple parts (part fraction). To keep the framework simple, *ProModel*[®]'s approach is taken, i.e., an arrival element only supports one type of entity.

4.2.9 Non-Supported Elements

Table 4.11 contains the elements that are not supported by the framework and have not been discussed earlier. In *ProModel*[®], variables, arrays, sub-routines, and macros are significantly programming-oriented, and thus are excluded from the framework. In *QUEST*[®], a sink is an element class used to collect statistical data and generate orders for pull systems. Currently, the framework does not support pull systems and statistical data collections, thus the sink class is not included. *QUEST*[®]'s accessory class is for animation purposes only. The AGV controller, labor controller, and SR controller are unique features of *QUEST*[®]. They contain the logic for AGV, labor, and

sub-resource classes. They are considered auxiliary data and are left for future expansion. The AGV and carrier require the use of network system and thus are not included.

Decision points are auxiliary locations that allow user-defined logic. It is considered an advanced feature and is excluded from the framework. Layout is for display purposes only and thus is not included in the framework. *QUEST*[®]'s sub-resources correspond to the tool and fixture catalogs in MSDM. These are considered as advanced modeling features and are left for future expansions.

Because of different viewpoints, MSDM has several data elements that are not seen often in simulation software: inventory, procurement, process-plan, time sheets, references, organization directory, departments, and calendars. These data elements are primarily manufacturing-oriented and are not included in the framework.

Table 4.11 Elements that are Not Supported by the Framework

ProModel		QUEST		MSDM		
Element	Attribute	Element class	Property	Data Structure	Complex data element	(Basic) Data element
Variables						
Arrays						
Sub-routines						
Macros						
		Sink				
		Accessory				
		AGV controller				
		Labor controller				
		SR controller				
		AGV				
		Carrier				
		Decision points				
		Group				
		Layout				
		Sub-Resource		Resource	Tool catalog	
					Fixture catalog	
				Inventory	tool-inventory	
					fixture-inventory	
					part-inventory	
					materials-inventory	
				Procurements		
				Process-plans	routing-sheets	
					operation-sheets	
					machine-programs	
				Time sheets		
				References		
				Organization directory		
				Departments		
				Calendars		

4.3 The Entity-Relationship Diagram

The second step of developing the proposed framework is determining the relationships between common data elements. In previous sections, eight framework elements are defined: general information, entity, static resource, dynamic resource, arrival, operation, linkage, and routing. Among these, the general information contains the metadata of the model, and thus does not have a direct relationship to other elements. Under the definition of SM, entity, static resource, and dynamic resource are primitive entities because they do not call other elements. In other words, they are not based on the definition of other elements. By contrast, arrival, operation, linkage, and routing are compound entities because they reference other elements. Arrival is a compound entity of static resource (ArvLocation) and entity (ArvEntity). Operation is based on static resource (OpLocation) and entity (InEntity and OutEntity). Linkage defines the relationship between two static resources (BeginLoc and EndLoc). Routings reference a set of linkages and an entity (RouteEntity).

As shown in Figure 4.4, each primitive entity, identified above, is presented as an entity (represented as squares) in the ER-diagram, and compound entities are represented as a relationship (represented as diamonds). The ovals represent attributes of the entities. The heavier weighted ovals means the attribute can be defined multiple times. In a standard ER, when a relationship needs to reference to another relationship, aggregation needs to be used. The purpose of the dashed lined box in Figure 4.4 is to allow the routing element to reference the linkage element. There are two elements that are not considered in standard ER diagrams, i.e., dotted lines and arrowed dashed lines. In Figure

4.3, the dotted lines represent the selection among options. For example, static resource may be a processing unit or a buffer. The arrowed dashed lines represent references. For example, the dash arrowed line from repair resource of static resource to dynamic resource means that the repair requires a specific dynamic resource.

The sequence rules of static resources are very similar between *ProModel*[®] and *QUEST*[®]. Table 4.12 contains the sequence rules in *ProModel*[®] and *QUEST*[®], and identifies those that the two software have in common. The proposed framework only supports the common sequence rules.

Table 4.12 Common Sequence Rules

<i>ProModel</i> [®]	<i>QUEST</i> [®]	Framework
No queuing		
FIFO	FIFO	FIFO
LIFO	LIFO	LIFO
By Type		
Highest attribute value	Ascending order	By Att (Inc)
Lowest attribute value	Descending order	By Att (Dec)
	User function	

The routing rules vary considerably between *ProModel*[®] and *QUEST*[®]. Table 4.13 contains the routing rules in *ProModel*[®] and *QUEST*[®]. Again, the framework only supports the four common routing rules.

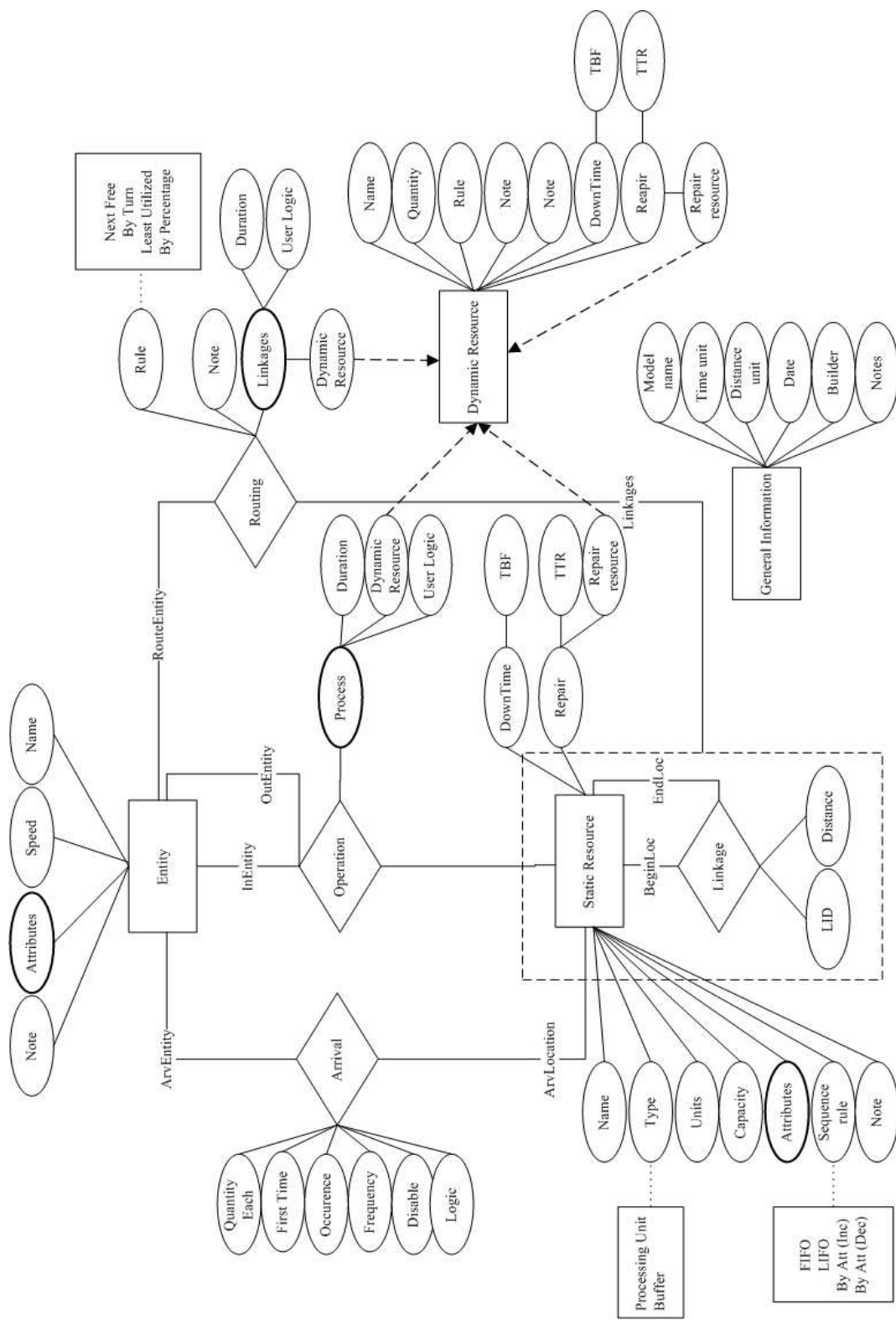


Figure 4.4 Relationships Between Common Data Elements

Table 4.13 Common Routing Rules

<i>ProModel</i> [®]	<i>QUEST</i> [®]	Framework
First available	Next free	Next free
Most available		
By turn	Cyclic order	By turn
Random		
If join request		
If load request		
If send		
Longest unoccupied	Least utilized	Least utilized
Until full		
If empty		
Probability	Proportions	By percentage
User condition	User function	
	Maximum room	
	Minimum queue	
	Minimum waiting	
	Priority	
	Fixed routing	
	First allowed output	

The ER diagram not only facilitates the representation of relationships between common data elements, but it also facilitates the collection of performance measurements. Figure 4.5 contains the same common data elements as in Figure 4.4, except that each element is attached to a list of performance measurements associated with the element. Although statistical data collection is beyond the scope of this research, Figure 4.5 would allow the users to select the performance measurements of interest and is thus useful in the experimental design portion of the simulation modeling process.

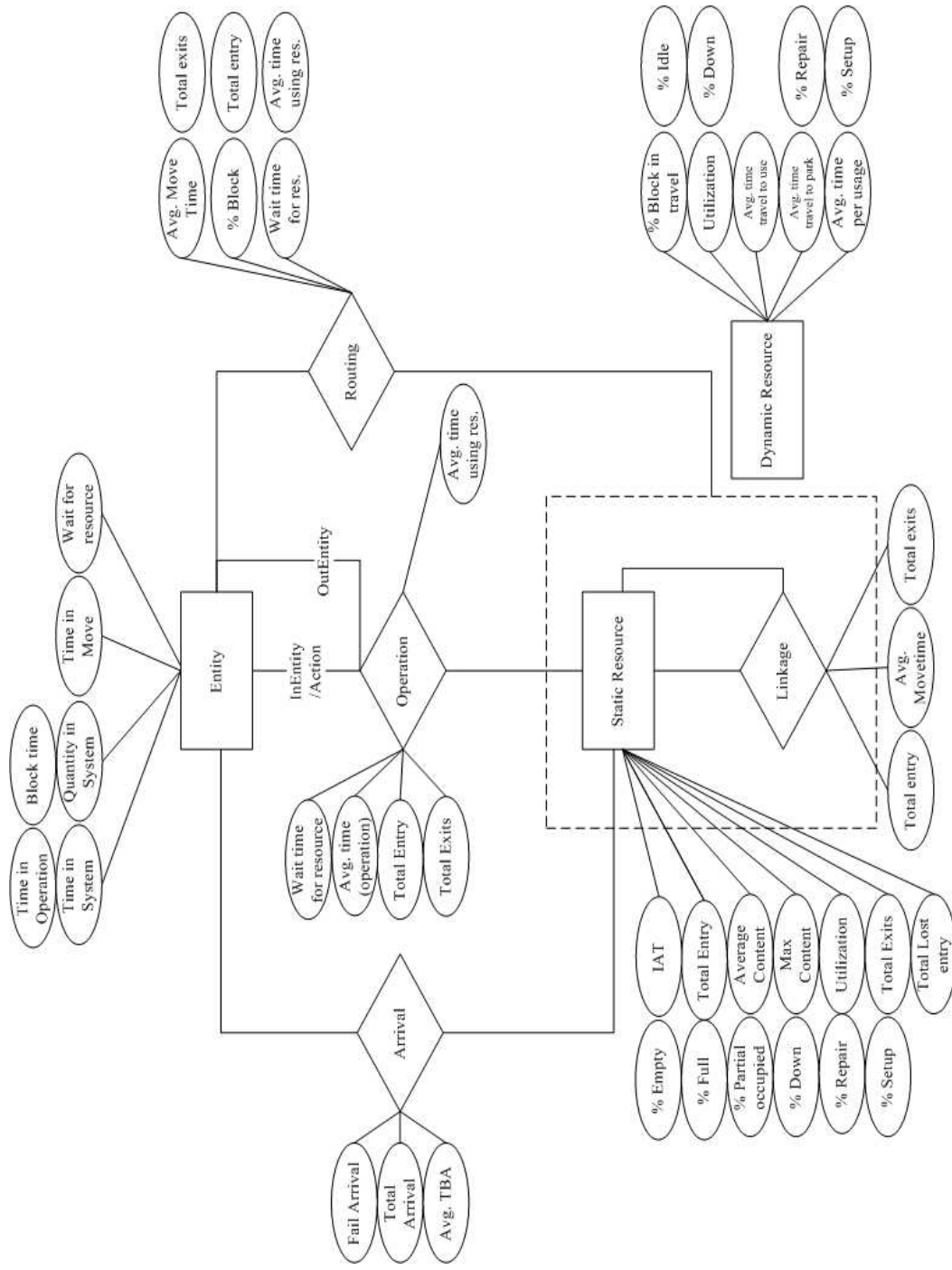


Figure 4.5 Performance Measurements of Common Data Elements

4.4 The Structural Modeling Schema

A SM schema (Component B in Figure 4.3) defines the relationships of model elements and indicates how the model elements should be stored. In this research the intent is to create a generic framework that can represent both the static and the dynamic parts of a simulation model. The static parts of a simulation are the information that will not change over simulated time. For example, the name of a machine is usually the same throughout the execution of a simulation. However, the dynamic parts of a simulation are the information that will change over execution time. For example, the state of a machine may become idle, busy, and/or down during a simulation execution. The SM is capable of handling the static part of the discrete-event simulation, but is not able to capture the dynamic part. As discussed in Section 3.2, several research efforts try to extend the SM to capture the dynamic part of simulation. The result is an extremely complex model representation that requires a tremendous amount of memory because they develop methods to: 1) describe how to perform certain operations or; 2) capture the state of the models at each time point in the simulation.

The proposed framework does not intend to capture the transient state of simulation models nor instruct the computer how to perform some tasks, as is done in extended structured modeling and condition specification. It is assumed that a solver (i.e., simulation software) when provided with high-level instructions, is available to read the generic model, simulate it, and generate results. The simulation software provides the transient state information automatically. Therefore, the model representation results

from the framework is concise and easy to understand because it only lists the tasks that need to be done instead of describing how to accomplish the tasks.

An SM schema is a set of rules that specifies how a model should be stored. To handle the dynamic part of a simulation, compound entities are used to represent logic. The logic property contains tasks that need to be done during the simulation. The logic property may be plain text that contains user-defined code, or consist of selections of predefined options. For example, the sequence rule may have first-in-first-out (FIFO) or last-in-first-out (LIFO) as a predefined option. Figure 4.6 contains a portion of the SM schema that represents the static resource part. A full SM schema is provided in Appendix A. As shown in Figure 4.6, the second line `/pe/` means static resource is an primitive entity. The third line `Name(Static_Resource;) /a/ : text` means name is an attribute (`/a/`) of static resource; the format is text (or string). The `I+` in Figure 4.6 means that data type is positive integer. The line `DownTime(Static_Resource;) /ce/` means downtime is an attribute and a compound entity (`/ce/`) of static resource. At the bottom of Figure 4.6, the compound entity downtime and sequence rules are defined. For the entire SM schema syntax, refer to Geoffrion [10].

<i>&Static_Resource Static Resource Data</i>	
Static_Resource _i /pe/	There are i static resource in the model.
Name(Static_Resource _i) /a/ : text	Name of static resource.
Type(Static_Resource _i) /a/ : text	Type of static resource.
Capacity(Static_Resource _i) /a/ : I+	Capacity of static resource.
Unit(Static_Resource _i) /a/ : I+	Units of location.
DownTime(Static_Resource _i) /ce/	Down time logic of the location.
Repair(Static_Resource _i) /ce/	Repair logic of the location.
SequenceRules(Static_Resource _i) /ce/	Dispatch rules.
Notes(Static_Resource _i) /a/ : text	Description of the location.
 <i>&DownTime Downtime data</i>	
DownTime _j (Static_Resource _i) /ce/	j downtime logics associated to static resource.
TBF(DownTime _j) /a/ : text	Time between failure.
 <i>&SequenceRules Sequence rules</i>	
SequenceRules _k (Static_Resource _i) /ce/	k sequence rules associated to static resource
SeqRule(SequenceRule _k) /a/ : text	sequence rule.

Figure 4.6 Example SM Schema

The reason for using SM schema to construct the framework is that the calling sequence is useful in simulation debugging and presentation. For example, if one tries to debug or view the flow of an entity in a model, the calling sequence tracks all of the operations performed on that entity. This is possible since operation is a compound entity, it reference an entitie and a static resource. Thus, the calling sequence tracks all of the static resources that were used to perform operations on entities. In the end, anything that is relative to the entity is found.

When transferring from framework to simulation software environments, a parser is needed to translate the text strings that contain user-defined code into specific modeling language syntax. Chapter 7 discusses a means to prevent the loss of data and avoid translation errors. Because the logic properties are saved as plain text, decision-makers can write the logic in their own words, and then model builders can interpret the

text and translate it into a proper format description using the syntax from the implantation language.

4.5 Common Graphic Elements

The common graphic elements (Component F in Figure 4.3) are the basis for graphic representation. They provide the pictorial display for common model elements and relationships. They are developed as a Visio stencil and are considered a part of the implementation of the proposed framework. Implementation is discussed in Chapter 5.

4.6 Summary

In this chapter, the common simulation elements are defined by combining viewpoints from manufacturing and simulation software using MSDM, *ProModel*[®], and *QUEST*[®]. The relationships among the elements are defined using an ER diagram. A structural modeling schema is used to construct the proposed framework. This chapter defines and describes all of the theoretical components of the proposed framework, except the common graphic elements. The graphical elements are defined in Chapter 5, along with other components of implementation.

CHAPTER V

IMPLEMENTATION OF PROPOSED SIMULATION INTERACTION APPROACH

In this section, the implementations of the proposed framework are discussed. The XML DTD (Component C in Figure 4.3) is discussed in Section 5.1. A sample XML file (Component D in Figure 4.3) is given in Section 5.2. There are two major parts to the software implementation: the Common User Interface (Component E in Figure 4.3) and the Graphical Modeling tool (Component G in Figure 4.3). These are discussed in Section 5.3 and 5.4, respectively.

5.1 The XML DTD

An XML DTD (Component C in Figure 4.3) is used to verify if an XML document follows a certain format. It defines the structure of an XML document with a list of legal elements. An XML DTD is developed according to the SM schema; in fact, the SM schema and XML DTD are very similar. Figure 5.1 is an example of a XML DTD file that defines the dynamic resource element. The full XML DTD file is provided in Appendix B.

```

<!ELEMENT Static_Resource (Name, Type, Capacity, Units, DownTime?, Repair?,
SequenceRule, Attribute*, Note?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Capacity (#PCDATA)>
<!ELEMENT Units (#PCDATA)>
<!ELEMENT SequenceRule (SeqRule, SeqAttribute?)>
    <!ELEMENT SeqRule(#PCDATA)>
    <!ELEMENT SeqAttribute(#PCDATE)>
<!ELEMENT DownTime (TBF)>
    <!ELEMENT TBF(#PCDATA)>
<!ELEMENT Repair (TTR, RepairResource?)>
    <!ELEMENT TTR(#PCDATA)>
    <!ELEMENT RepairResource(#PCDATA)>
    <!ELEMENT Note (#PCDATA)>
<!ELEMENT Attribute (ID, Type, Note?)>

```

Figure 5.1 Example XML DTD

The line “<!ELEMENT Static_Resource (Name, Type, Capacity, Units, DownTime?, Repair?, SequenceRule, Attribute*, Note?)>” means each static resource has only one name property, one type property, one capacity property, zero or one downtime property, zero or one repair property, a SequenceRule property, zero or more attributes, and zero or one notes property. The line “<!ELEMENT Capacity (#PCDATA)>” means the capacity property only accepts parsed character data, i.e., string type data.

If an XML document does not follow this rule, software, such as Microsoft Internet Explorer, issues an error and notifies the user that there is an error in the XML document. This is useful when transferring the simulation models from one environment to another. Because of the structural nature of the SM schema and XML DTD, it is straightforward to translate from one to the other.

5.2 The XML file

An XML file (Components D in Figure 4.3) is used to store the simulation model data in the proposed framework format and serves as a bridge between software implementations (Components G and E in Figure 4.3). An evolving research area is the use of XML to build an open-architecture model-exchange environment [51]. XML was originally designed to support large-scale publishing because XML has the following characteristics:

- simplicity – XML documents are easy to read and modify,
- extensibility – the format of an XML document can be easily extended to include more data, and
- interoperability – XML is widely accepted and works on various platforms and software.

The XML also plays an increasingly important role in various forms of data exchange [52]. Figure 5.2 is a portion of a sample XML file. The full file can be found in Appendix C.

```

<Location>
  <Name>Factory1</Name>
  <Type>ProcessingUnit</Type>
  <Capacity>5</Capacity>
  <Units>1</Units>
  <Downtime>
    <TBF>5 hr</TBF>
  </Downtime>
  <Repair>
    <TTR>10 min</TTR>
    <RepairResource></RepairResource>
  </Repair>
  <SequenceRule>FIFO</SequenceRule>
  <Attribute>
    <ID>SSAR</ID>
    <Type>Real</Type>
    <Note></Note>
  </Attribute>
  <Note>A Factory</Note>
</Location>

```

Figure 5.2 Example XML file

5.3 Graphical Modeling Tool

Graphical modeling is certainly not a new idea. However, previous graphical modeling practices usually do not have a direct relationship between the graphical symbols and real world objects [21] [22] [23]. For example, the SLAM II network model only shows the operation and routing logic. By just looking at the network, one may not know whether it represents a machine shop or a hospital. As a result, it is not intuitive to interpret the resulting representation of real world systems. The three primitive entities (entity, static resource, and dynamic resource) in the proposed framework are based on real world objects. The four compound entities (arrival, operation, linkage, and routing) are also associated with real world activities. Since the icons of the proposed graphical

representation are derived from the proposed framework, the resulting graphical representation should be more intuitive and easier to associate with a real world system.

The graphical representation provides a standardized methodology for conceptual modeling. Thus, stakeholders can better communicate and more easily share their understanding of the system. In addition, because various software vendors have different world views and terminologies, there is a need to restrict the scope of the simulation modeling process. With a finite set of standardized graphical icons, the scope of the simulation can be limited. The graphical representation also provides standardized simulation terminology, which will reduce conflicts at the application stage.

The graphical modeling tool (GMT) is developed using Microsoft™ Visio because it provides a well-developed drawing environment and supports Visual Basic for Application (VBA). Thus, VBA is used to develop customized functions. In GMT, some customized functions are developed to facilitate simulation data collection and input/output management. For example, in GMT, when an entity icon is dragged and dropped from the stencil to the drawing area, the icon is added to the drawing area (default action) and a form, created using VBA, pops up and requests the user to input simulation relative information.

Based on the proposed framework, a set of common graphical elements (Component F in Figure 4.3) is developed. A Visio stencil that contains seven corresponding masters to common model elements, as shown in Figure 5.3, is created. In the stencil, a rectangle represents a static resource, an arrowed line represents a linkage, a circle represents a routing, a rounded rectangle represents an entity, an ellipse represents

a dynamic resource, a database shape represents an arrival, and a box shape represents an operation. Note that general information is not included in the stencil; it is distributed on the drawing area, as shown in Figure 5.4.



Figure 5.3 Visio Stencil of Model Elements

The graphical modeling tool allows modelers to build simulation models by dragging and dropping icons from the stencil to the drawing area. Each master (icon in the stencil) contains its own attributes. These attributes were defined in Chapter 4 and are shown in Figure 4.4. When an icon is dropped onto the drawing area, a form pops up and asks the user to fill in the required simulation information. After a drawing is complete, users can save it as a Visio drawing or in the proposed framework format (i.e., XML file). This is accomplished using a VBA macro. Currently, the Visio drawings only save the information such as icons, lines, and positions; they do not store the simulation related data. The XML file stores simulation relative data as well as pictorial information. A VBA macro converts the XML files into Visio drawings.

As mentioned earlier, the purpose of the GMT is to create a high-level model. Thus, users do not have to fill in every single piece of simulation related information.

Only the required information is needed, such as the name of the machine, so the program can reference it to the icon. Because the GMT is based on the proposed framework, the sequence of dropping the icons into the drawing is restricted. At least one primitive element (entity, static resource, and dynamic resource) should be dropped before compound elements (arrival, linkages, routing, and operation) because compound elements reference primitive elements. For example, an arrival is a compound element that references an entity (what it generates) and a static_resource (where to create an entity). One cannot drop an arrival icon into the drawing if there are no entities and static resources already in the drawing.

A demonstration model is shown in the GMT environment in Figure 5.4. The left-hand side of Figure 5.4 contains the stencil. The right-hand side is the drawing area. Users can drag the icons from the stencil and drop them onto the drawing area to create a model. If an icon is selected on the drawing, the icons that reference the selected icon turn dark to show the reference relationship. As shown in Figure 5.4, when entity “Part1” is selected, arrival, operations, and routings that reference it turn dark. When entity “Part2” is selected, the set of icons that pertain to it turn dark, as shown in Figure 5.5. Currently, this feature only applies to primitive entities, i.e., entity, static resource, and dynamic resource.

In Figure 5.4, there are entities (i.e., part1 and part2) defined, as can be seen in the top left corner of the drawing. The text “Part1: Part1_Buff” in the database shape (arrival) contains the arrival logic of entity “Part1” at static resource “Part1_Buff”. “Link1” through “Link4” are the linkages; note that “Link4” leads to the exit of the

system. The circle shapes “R1” through “R5” defines the routing logic. The text in the box shape (operation) “Process1: Part1” defines the operation logic of entity “Part1” at static resource “Process1”. Note the operation “Process2: All” defines the operation logic for all entity types at static resource “Process2”. The “All” option is included in the GMT to simplify the modeling process.

To view the detail definition of each simulation element in the GMT, one can right click the icon and select the VBA function “Edit Properties”. In the future, the GMT should display more information on the drawing so that users do not have to click on each icon for simulation related data.

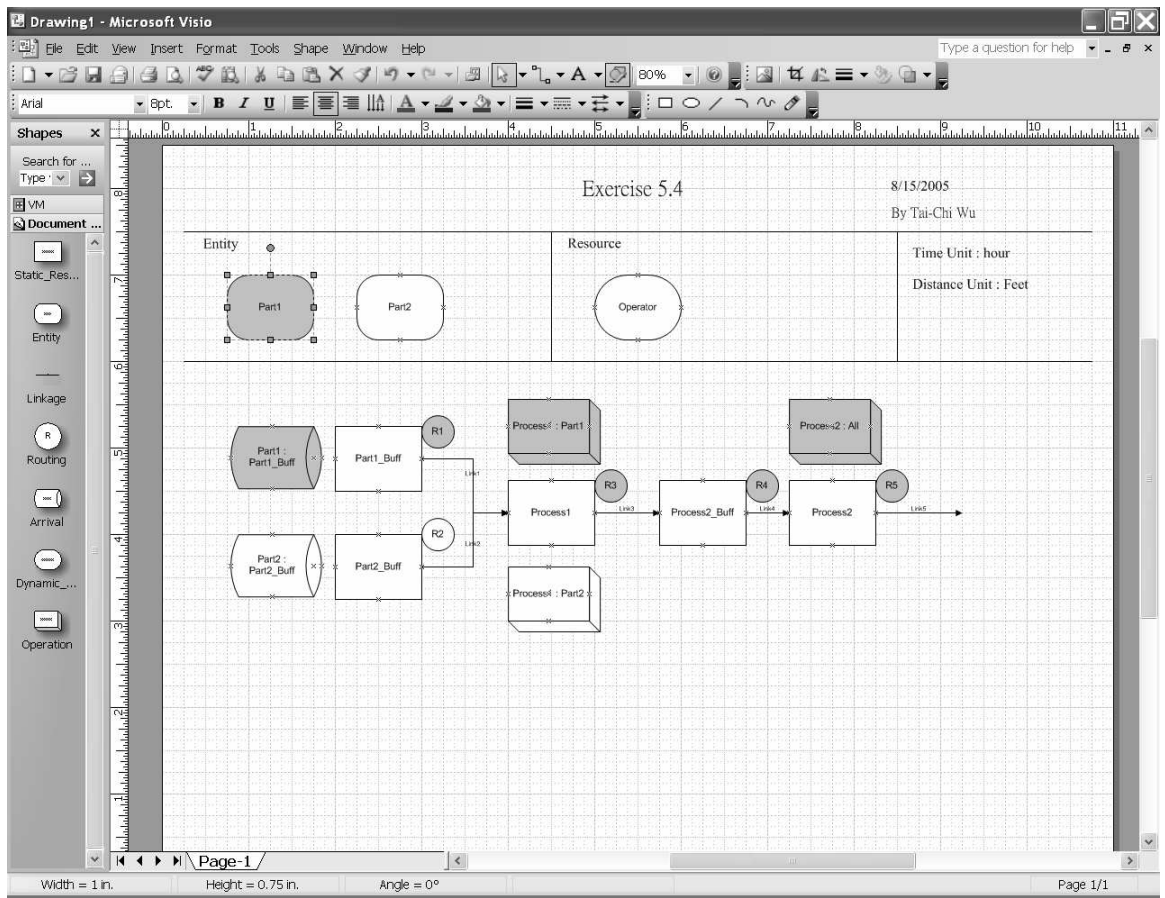


Figure 5.4 Graphical Modeling Tool Environment with Part1 Selected

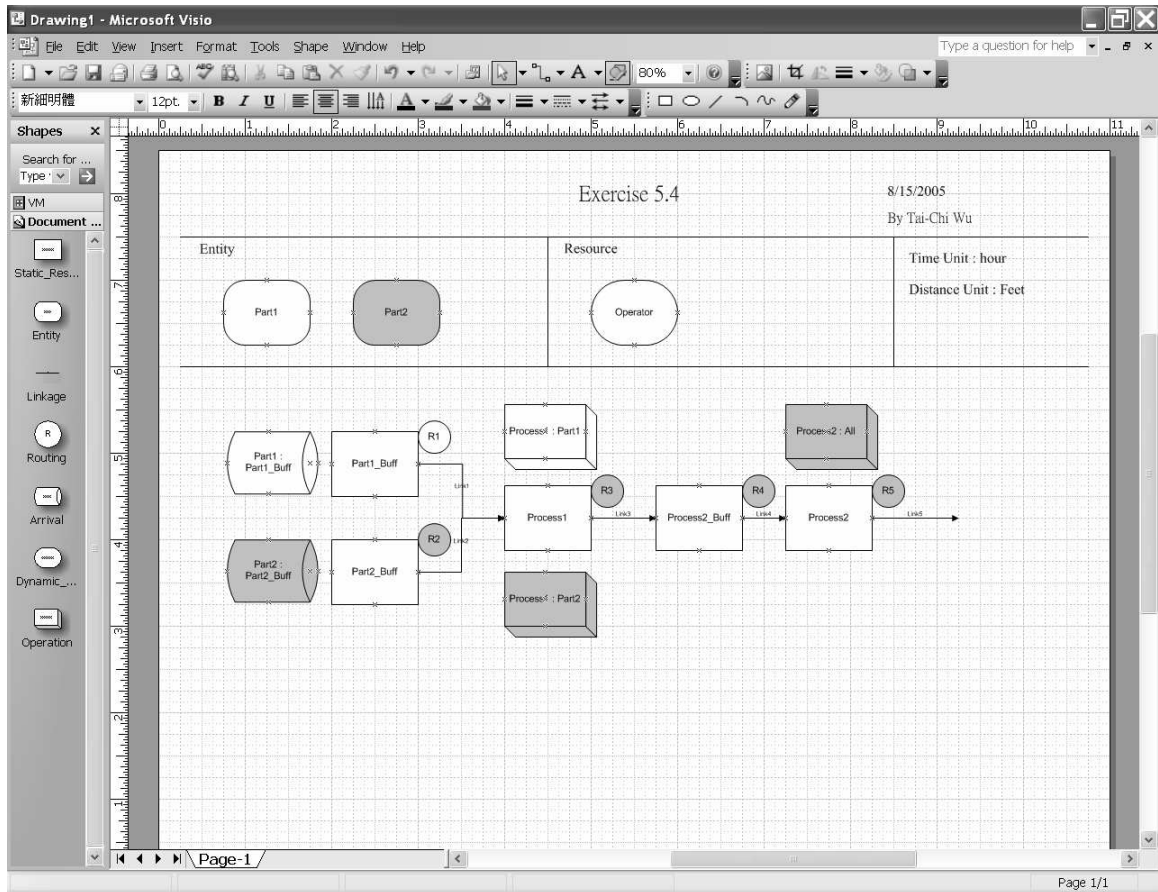


Figure 5.5 Graphical Modeling Tool Environment with Part2 Selected

Three macros were developed to control the environment: Initialize, ReadFile, and SaveModel. They are accessed from the Visio menu Tools-> Macros-> PublicDeclarations. The initialize macro cleans the drawing area and sets the memory to an initial state. The ReadFile macro reads an existing XML file and restores it as Visio drawing. The SaveModel macro saves the current drawing into an XML file.

The GMT is able to present the basic simulation elements and their relationships in a structured and intuitive way. The advantages of visual modeling include: facilitating model development, increasing the usage of simulation modeling, aiding in model

validation and verification, facilitating understanding and learning of existing models, and facilitating model documentation.

5.4 Common User Interface

Figure 5.6 shows the overall structure of the common user interface (CUI). The CUI is the bridge between XML files (in the proposed framework format) and simulation packages. The CUI contains 3 major parts: the user interface, the Microsoft XML parser, and the simulation package controllers. The user interface and the Microsoft XML parser are discussed in the following section. The *ProModel*[®] controller is discussed in Section 5.4.2. The *QUEST*[®] Controller is discussed in Section 5.4.3.

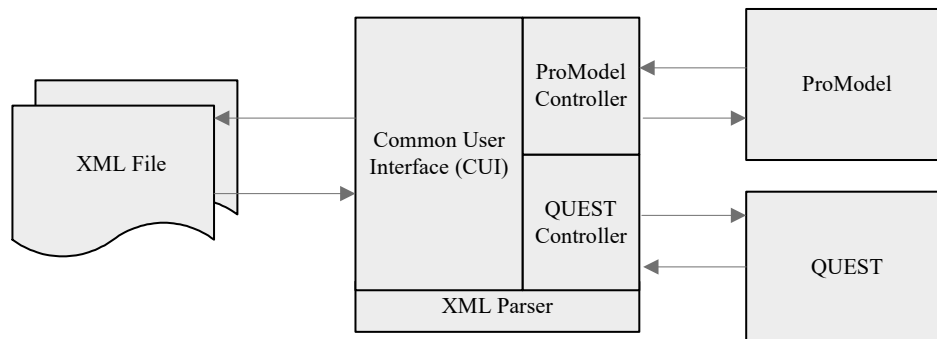


Figure 5.6 Overall Structure of the Common User Interface

5.4.1 The User Interface and XML Parser

The CUI serves two purposes:

- 1) provides a software independent model modification environment, and
- 2) facilitates the interaction between XML files and simulation packages.

Figure 5.7 is a screenshot of the CUI. The eight buttons allow users to modify all eight common simulation elements defined in the proposed framework. When a button is clicked, a form pops up that allows users to add, delete, or modify the simulation elements. The text box in the bottom shows the file being edited.

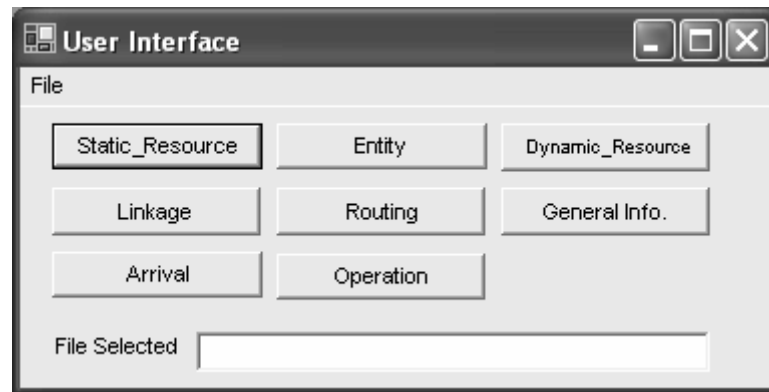


Figure 5.7 Common User Interface

There are six management functions in the CUI in the File menu. They are shown in Figure 5.8 and include: open, save, XML->ProModel, ProModel->XML, XML->QUEST, and QUEST-> XML. The open function reads an existing XML file into the CUI using the Microsoft XML parser (Document Object Model approach). It allows users to modify the content of the XML file. The save function can write the modified file to disk. The XML->ProModel and ProModel->XML functions are provided in the ProModel Controller and are discussed in Section 5.4.2. The XML->QUEST and QUEST->XML functions are contained in *QUEST*[®] Controller and are discussed in Section 5.4.3.

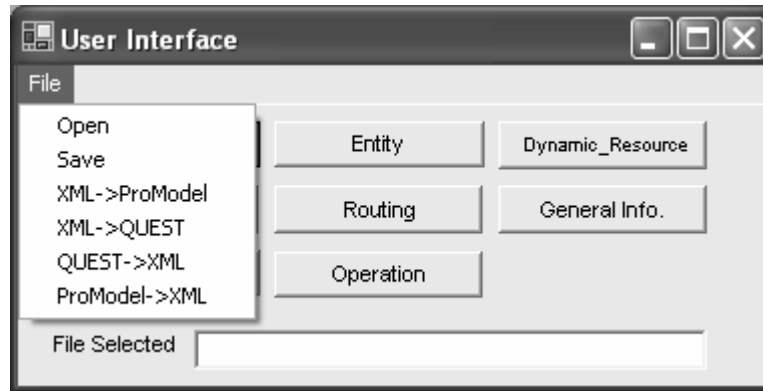


Figure 5.8 Management Functions in the Common User Interface

5.4.2 The ProModel Controller

The ProModel Controller is responsible for uploading XML files into the *ProModel*[®] application and extracting data from *ProModel*[®] into XML files. *ProModel*[®] provides an ActiveX control for creating intra-application communications [53]. Because everything in *ProModel*[®] is stored in tabular format, it is simple to access data within *ProModel*[®] through the ActiveX control. The biggest challenge with the ProModel Controller is that almost all operation, routing, and downtime related information is written using *ProModel*[®] specific code; therefore, a translator is needed to parse the code and map the data to the right places in the framework. Currently, the ProModel Controller can only recognize a few keywords: wait, get, free, move for, and move with. In Figure 5.9, the left-hand side is a section of a *ProModel*[®] model listing. Note that all simulation data is extracted from the *ProModel*[®] environment via ActiveX, instead of parsing them from the text file. Figure 5.9 is only for illustration purposes. When extracting a *ProModel*[®] model to the proposed framework, three files are generated: a log file, an interpreted XML file, and an uninterpreted XML file. The log file contains the translation

history, the interpreted XML file stores the simulation data supported by the framework, and the uninterpreted XML files stores the non-supported data. Details of these three files are discussed in Section 7.5.

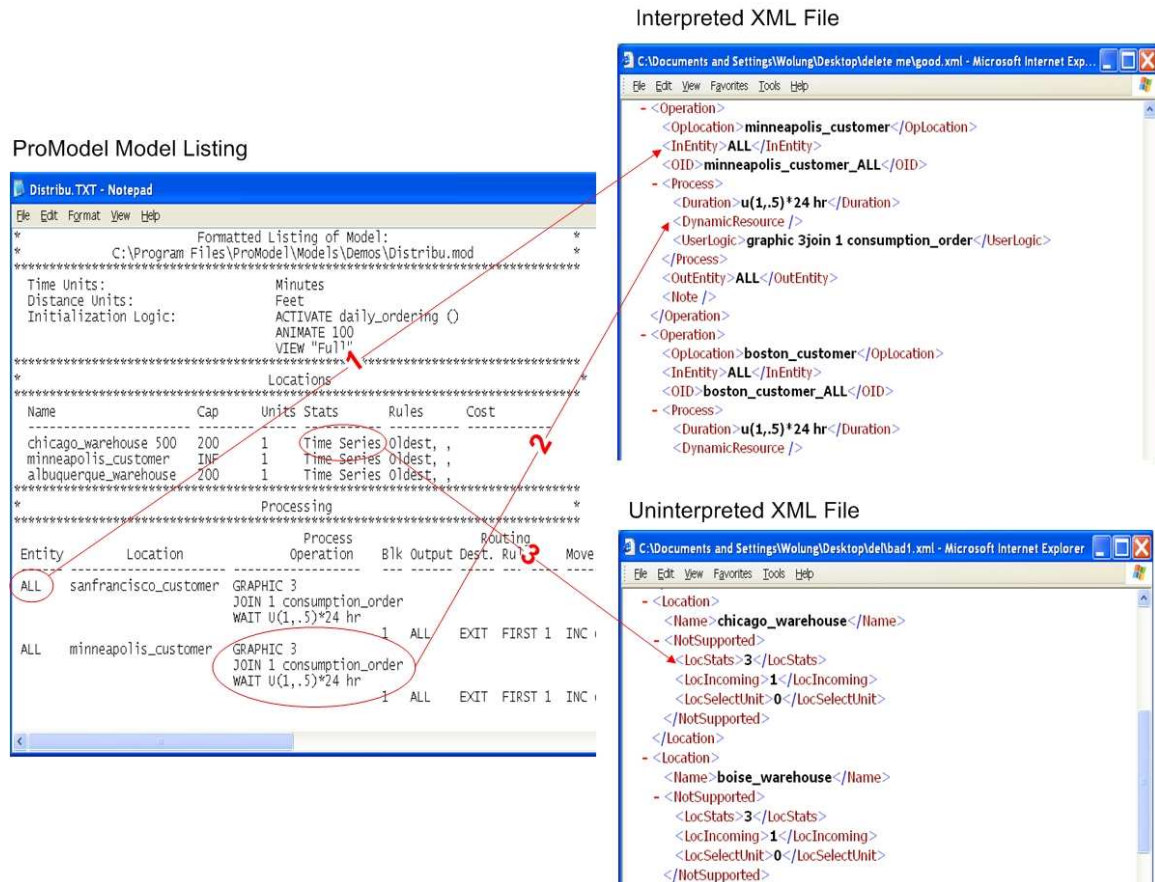


Figure 5.9 Transferring ProModel Files to the Proposed Framework

The supported data elements, shown as (1) in Figure 5.9, are placed in the interpreted XML file. The non-support data elements, shown as (3) in Figure 5.9, are placed in the uninterpreted XML file. As mentioned earlier, the ProModel Controller only recognizes a few keywords. Thus, (2) in Figure 5.9 illustrates that only the recognized logic (i.e., WAIT U(1,.5)*24 hr) is transferred, the unrecognized logic (i.e., “GRAPHIC

3” and “JOIN 1 consumption_order”) is dumped to the UserLogic node. The reason that unrecognized code is placed in the interpreted XML file instead of uninterpreted XML file is that the CUI does not support the editing of the uninterpreted XML file, and the information contained in UserLogic node is critical to model execution. The UserLogic node is for storing customized code written in any language. Currently, the CUI does not include a standard language for writing customized code.

5.4.3 *The QUEST Controller*

The *QUEST*[®] Controller is responsible for uploading XML files to *QUEST*[®] and extracting *QUEST*[®] models to XML files. When uploading an XML file to *QUEST*[®], a *QUEST*[®] wrapper class that contains a socket interface (developed by Travis Hill) is used to create an instance of *QUEST*[®]. The *QUEST*[®] Controller parses the model elements contained in the XML file and translates them into batch control language (BCL) statements [54]. Because the model data in *QUEST*[®] is distributed among multiple element classes instead of structurally stored in one place, such as tables in *ProModel*[®], it is difficult to generate correct BCL statements. Once generated, the statements are sent to *QUEST*[®] via the socket interface in the *QUEST*[®] wrapper one statement at a time. After each statement is sent, *QUEST*[®] responds with a status message. The *QUEST*[®] Controller creates a log file to record the transformation history and exceptions. A detailed discussion of exception handling is given in Section 7.5. Currently, the *QUEST*[®] Controller does not support the user-defined code translations because the code is not saved with the model file. An issue left for future development is that *QUEST*[®]-specific

code is written using simulation control language [55]. In order to fully translate the code, a lexical analyzer and a compiler are needed.

QUEST[®] provides limited functions and BCL statements for extracting information from the *QUEST*[®] environment. They are essentially useless if one does not know the content of the *QUEST*[®] model being extracted. Fortunately, the *QUEST*[®] models are saved in plain text format and it is possible to parse the model files into XML format. The results are three files: the log file, the interpreted XML file, and the uninterpreted XML file. A detailed discussion of these files is given in Section 7.5.

CHAPTER VI

ILLUSTRATION OF PROPOSED APPROACH

In this section, various simulation models are used to demonstrate the capabilities of the proposed framework and the software implementation. The models are built using the graphical modeling tool, then through the common user interface, uploaded to both *QUEST*[®] and *ProModel*[®]. Unless explicitly mentioned, the models can be properly uploaded to both *QUEST*[®] and *ProModel*[®] environments without any problems. Due to the length of the resulting models, they are not included in this document, but are available upon request. The graphical modeling tool generates two outputs: one Visio drawing and one XML file. The Visio drawings are shown with each problem in this chapter because they provide a good representation of the model. The XML files are available upon request.

6.1 Test Set 1

For Test Set 1, problems of different difficulty levels were selected from various sources in order to demonstrate the capabilities of the proposed framework. The purpose is to fully test the proposed framework, and identify the limitations of the framework. This will help to identify future research needs.

6.1.1 BullyBooks

This problem is taken from an exam (Text #2, Spring 2005 semester) of Mississippi State University's Simulation I class (IE4773/6773). This problem is relatively simple and requires the use of all seven model elements of the proposed framework, making it a good exercise for testing and demonstrating the basics of the framework.

Problem statement:

BullyBooks, Inc. (BBI) sells books through both a phone-in system and via the Internet. Order fulfillment for the phone orders is basically a three-step process: incoming orders arrive and are processed by order takers, they are then forwarded on to order fillers who collect the books in the order, and finally orders are packed in preparation for shipment. Internet orders do not require order takers; they go directly to order fillers and then are packed. The time between Internet orders is 4 minutes and the time between phone orders is 2.5 minutes, both exponentially distributed. The time to process phone orders by the 2 order takers is assumed to be uniformly distributed between 2 and 6 minutes. The estimates on the time to process each type of order by the 12 fillers are provided in the following table.

	Minimum	Maximum	Most Likely
Phone	12	21	15
Internet	8	25	15

The time to process an order by the order packers is normally distributed with a mean of 4 minutes and a standard deviation of 1 minute. The simulation will be run for 40 hours.

Bully Books

8/15/2005

By Tai-Chi Wu

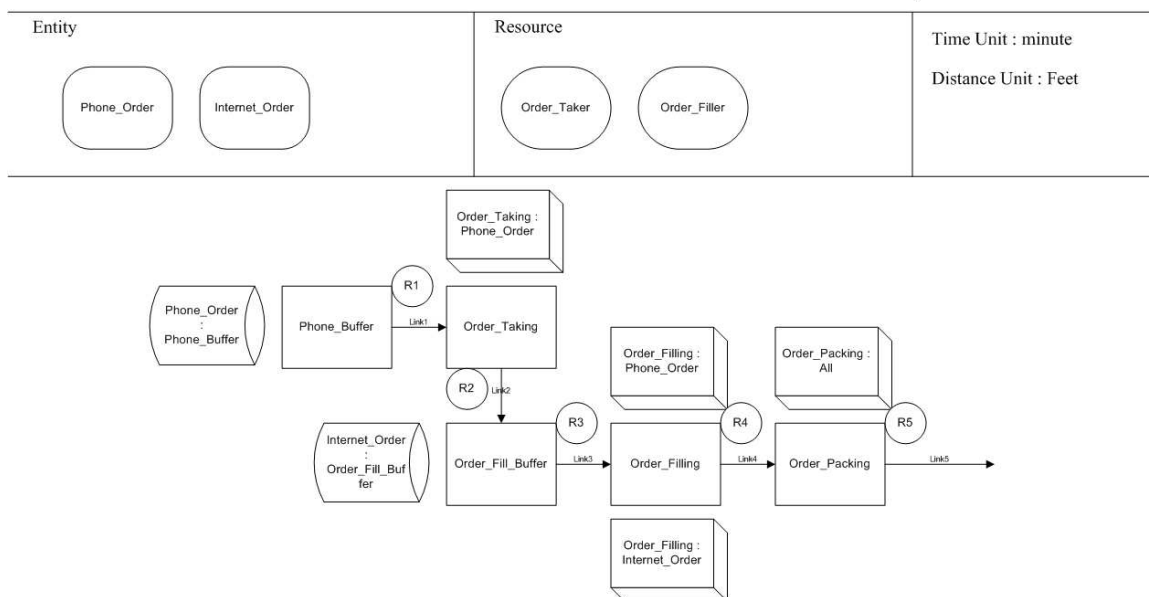


Figure 6.1 Graphical model of Bully Books

The key characteristics of this problem are that it requires the use of all seven model elements and involves no user-defined logic. Two arrival logics (shown as database shape in Figure 6.1) define the creation of two types of entities (phone orders and internet orders) that enter the system in specific places (static resource: Phone_Buffer and Order_Fill_Buffer). There are five static resources in the system (shown as rectangle in Figure 6.1) and two types of dynamic resources (order taker and order filler, shown as ellipse in Figure 6.1) that process the orders at various static resources. The linkages (shown as arrowed line in Figure 6.1) are very straightforward; each static resource except Phone_Order_Buffer has one incoming and one outgoing linkage. The routing (shown as circle in Figure 6.1) is also very straightforward, each routing node contains only one linkage, the parts are sent to the next static resource whenever it is possible.

Four operation logics (shown as box shape in Figure 6.1) are defined: 1) Order_Taking specifies the processing logic for phone orders, 2) two Order_Filling logics for one for each type of entity, and 3) Order_Packing for all types of orders. Since the processing time is the same for both types of orders, the “All” option is used to simplify the modeling process and the Visio drawing (two operation icons versus one operation icon).

As discussed in Section 4.1, this problem is also built in *ProModel*[®] and extracted to a XML file in the framework format. The XML file is then uploaded to *QUEST*[®] to demonstrate that the proposed framework is able to facilitate model interactions at application stage.

6.1.2 Quarry problem

This problem is taken from Law and Kelton [7, pp 187]. The exercise requires the use of entity attributes and a minimal amount of user-defined logic. This model uses perpetual entities; i.e., entities that continuously flow through the model. This is in contrast to using a large number of entities that are created, processed, and destroyed as the model executes. Also, the routing and sequence rules are more complex. This problem demonstrates additional capabilities of the proposed framework and implementations.

Problem statement:

In a quarry, trucks deliver ore from three shovels to a single crusher. Trucks are assigned to specific shovels, so that a truck will always return to its assigned shovel after dumping a load at the crusher. Two different truck sizes are in use, 20 and 50 tons. The size of the truck affects its loading time at the shovel, travel

time to the crusher, dumping time at the crusher, and return-trip time from the crusher back to its shovel, as follows (all times are in minutes):

	20-ton truck	50-ton truck
Load	Exponentially distributed with mean 5	Exponentially distributed with mean 10
Travel	Constant 2.5	Constant 3
Dump	Exponentially distributed with mean 2	Exponentially distributed with mean 4
Return	Constant 1.5	Constant 2

To each shovel is assigned two 20-ton trucks and one 50-ton truck. The shovel queues are all FIFO, and the crusher queue is ranked in decreasing order of truck size, the rule's being FIFO in case of ties. Assume that at time 0 all trucks are at their respective shovels, with the 50-ton trucks just beginning to be loaded. Run the simulation model for 8 hours and estimate the expected time-average number in queue for each shovel and for the crusher. Also estimate the expected utilizations of all four pieces of equipment. Use streams 1 and 2 for the loading times of the 20-ton and 50-ton trucks, respectively, and streams 3 and 4 for the dumping times of the 20-ton and 50-ton trucks, respectively.

Quarry

8/15/2005

By Tai-Chi Wu

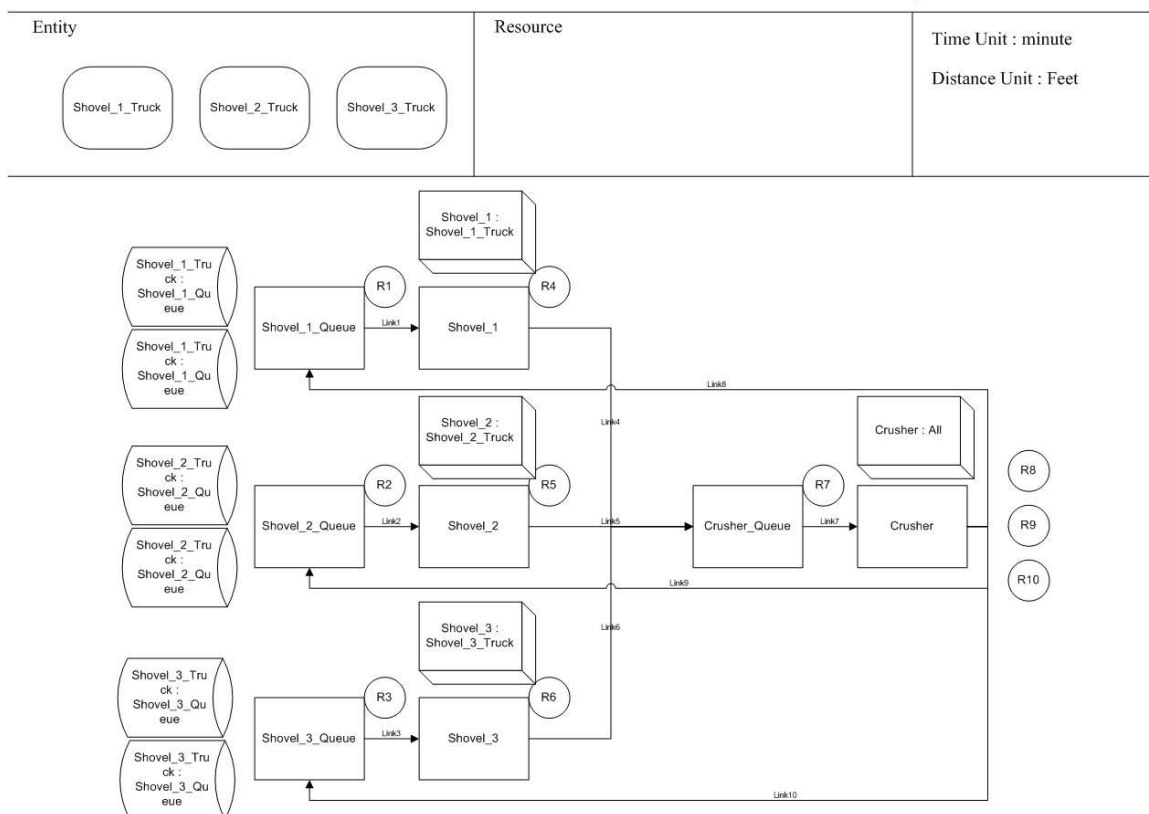


Figure 6.2 Graphical Model of Quarry Problem

In this problem, trucks return to their assigned shovels. Thus, there are three types of entities, one for each shovel: Shovel_1_Truck, Shovel_2_Truck, and Shovel_3_Truck. Each shovel is assigned two 20-ton trucks and one 50-ton truck. There are different ways to model this problem; one way is to have two types of entities (20-ton and 50-ton truck), another way is to use entity attributes. Here, entity attributes are used. A *weight* attribute is assigned to entities to distinguish the different size trucks. Upon the creation of the entities, i.e., arrival logic, the *weight* attribute is assigned a value of 20 or 50, respectively. Note there are six arrivals defined in Figure 6.2, each shovel has two

arrivals that create the trucks when that simulation starts and assigns their weight attribute. In *QUEST*[®], to assign a value to an entity attribute, one must write and compile user logic in a separate file. The *QUEST*[®] Controller within the Common User Interface (CUI) does not support this. To overcome this, using a manual transfer is required. The CUI provides a log file that contains a warning message if one tries to upload models that contain user-defined logic to *QUEST*[®]. Uploading to *ProModel*[®] is much easier because user logic is located in the same model file. Upon uploading user logic to *ProModel*[®], the CUI still create a log file that contains warning messages. The message is displayed even though the ProModel Controller places the user-defined logic in the correct field because it does not perform a syntax check. Therefore, *ProModel*[®] may not understand the user-defined logic.

Another reason for using the entity attribute approach in this problem is that the sequence rule of the crusher queue is ranked in decreasing order of truck weight, the weight attribute is a required variable. There are eight static resources in the system. The sequence rules are all FIFO (i.e., First In First Out) except for the crusher_queue. The routing logic is more complicated in this problem. There are travel times between static resources that are defined in the routing element. Also, after dumping at the crusher, the trucks return to their original shovel. Note there are three routing logics (R8, R9, and R10) at the end of the crusher; each routes the specified trucks to the right path back to the original shovels.

6.1.3 Shuttle bus problem

The shuttle bus problem is taken from an MSU Ph.D. comprehensive exam. This is an advanced test of the framework because it requires the use of global variables and a considerable amount of user-defined logic.

Problem statement:

A bus shuttles students from a remote parking facility to campus every 15 minutes. The capacity of the bus is 60 students. If students at the remote parking area are unable to board the bus because it is full, then they nearly always wait for the next bus because it takes more than 15 minutes to walk to campus.

Students have complained that the buses do not run frequently enough and they often have to wait for a second bus. The university would like to determine how frequently it needs to pick up students in order to keep up with demand. To do this, it would like to know how often the bus reaches capacity, and how many students are left waiting, if it picks up at 15, 12, and 10-minute intervals. Therefore, evaluate the impact of using 15, 12, and 10-minute intervals for the buses and make a recommendation for the interval that should be used. Also, provide a detailed report of your methodologies, assumptions, etc.

Note that students arrive sporadically. A brief study was performed the following time gaps between arrival of students to the bus stop (in seconds). The following times were recorded: 18, 45, 27, 9, 11, 12, 17, 30, 8, 17, 14, 3, 2, 15, 1, 26, 9, 18, 3, 9, 63, 10, 30, 9, 1, 58, 31, 11.

The original model uses numerous global variables and user-defined logic for: 1) calculating load/unload time, 2) counting the number of students that must wait for the second bus, 3) calculating bus utilization, and 4) scheduling the bus to arrive at specific intervals. But, the framework does not support global variables because they are too programming language-oriented. Including global variables into the framework will result in a very complex model representation, such as in the condition specification [13] approach. In order to model this problem without using global variables, some changes

need to be made. The load/unload time can be replaced by the average load/unload time. The bus interval can be replaced by average traveling time. However, to count the number of students that have to wait for the second bus, a global variable is needed. Also, a global variable is needed to record the number of students on bus. Thus, the proposed framework cannot solve this problem because the global variables are involved. This limitation will be discussed in the next chapter.

6.2 Test Set 2

In Test Set 2, exercises taken from various sources are used to test the proposed framework and the software implementation. This approach is biased because the exercises are not randomly selected; only the ones that fit into the framework's capabilities were chosen. In order to test the proposed framework and the software implementation more properly and completely, exercises are taken from Chapter 5 of a well-known *Arena*[®] textbook [56]. *Arena*[®] is a very popular discrete-event simulation software package. In Chapter 5 of [56], there is a set of 14 of exercises that utilize most of the features in *Arena*[®] and are designed to demonstrate the capability of *Arena*[®]. Thus, they can also be used to demonstrate the capabilities and the limitations of the proposed framework and its implementation. Also, because the software implementation only contains *ProModel*[®] and *QUEST*[®] controllers, if the proposed framework can solve the exercises in the *Arena*[®] textbook, then the generality of the proposed framework is also demonstrated. In each of the following sections, the framework is evaluated as to how well it represents each problem/exercise. For example, section 6.2.1 considers Exercise 5.1 in the textbook.

Note that some exercises require the simulation be terminated after a specified time or to collect specific statistical data. For example, Exercise 5.1 asks to run the simulation for 16 hours and to collect the time the traveler is in system, number of passengers completing check-in, etc. Because simulation termination and statistical data collection are a part of the experimental design aspect, they are beyond the scope of this research.

6.2.1 *Exercise 5.1*

Problem statement:

Travelers arrive at the main entrance door of an airline terminal according to an exponential interarrival-time distribution with mean 1.6 minutes. The travel time from the entrance to the check-in is distributed uniformly between 2 and 3 minutes. At the check-in counter, travelers wait in a single line until one of five agents is available to serve them. The check-in time follows a normal distribution with mean of 7 minutes and standard deviation of 2 minutes. Upon completion of their check-in, they are free to travel to their gates. Create a simulation model, with animation, of this system. Run the simulation for 16 hours to determine the average time in system, number of passengers completing check-in, and the average length of the check-in queue.

This is a very typical queuing system problem. The graphical model is shown as Figure 6.3.

Exercise 5.1

8/15/2005

By Tai-Chi Wu

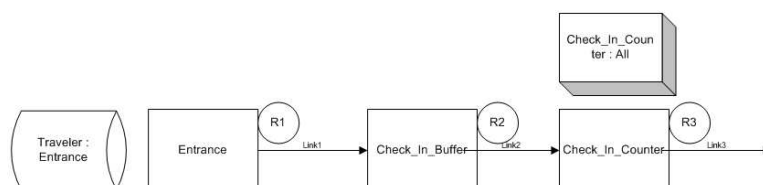
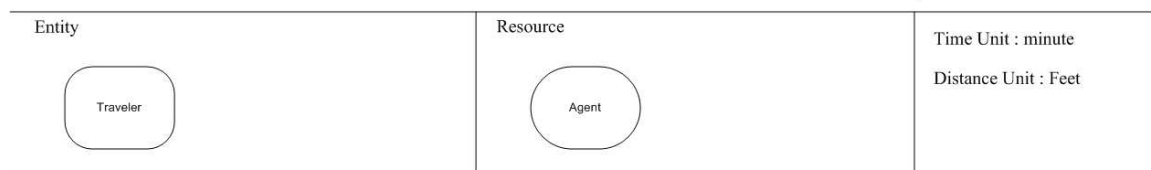


Figure 6.3 Graphical Model of Exercise 5.1

There are three notable challenges. First, there is a travel time from the entrance to the check-in. Second, the exercise uses two types of distributions: uniform and normal distributions. Third, there are five agents at the check-in counter. Note the number of agents is not shown in Figure 6.3. In future work, users should be allowed to choose what information is displayed in the drawing.

The first two challenges can be handled by the proposed framework in the routing and operation logics. There are couple ways to model the third challenge. One can model the check-in counter as a processing unit with capacity of five, or model it as five processing units each with a capacity of one. In Figure 6.3, the check-in counter is modeled as a processing unit with a capacity of five entities that require a dynamic resource (an agent) to complete the check-in process.

6.2.2 Exercise 5.2

Problem statement:

Develop a model of a simple serial two-process system. Items arrive at the system with a mean time between arrivals of 10 minutes. They are immediately sent to Process 1, which has an unlimited queue and a single resource with a mean service time of 9 minutes. Upon completion, they are sent to Process 2, which is identical to Process 1. Items depart the system upon completion of Process 2. Performance measures of interest are the average numbers in queue at each process and the system cycle time. Using replication length of 10,000 minutes, make the following four runs and compare the results:

- Run 1: exponential interarrival times and exponential service times
- Run 2: constant interarrival times and exponential service times
- Run 3: exponential interarrival times and constant service times
- Run 4: constant interarrival times and constant service times

This is also a very typical queuing system problem. The main purpose of this exercise is to let the users observe the effect of changing parameters on the behavior of the system. Changing system parameters can be done easily with the GMT. The model is shown in Figure 6.4.

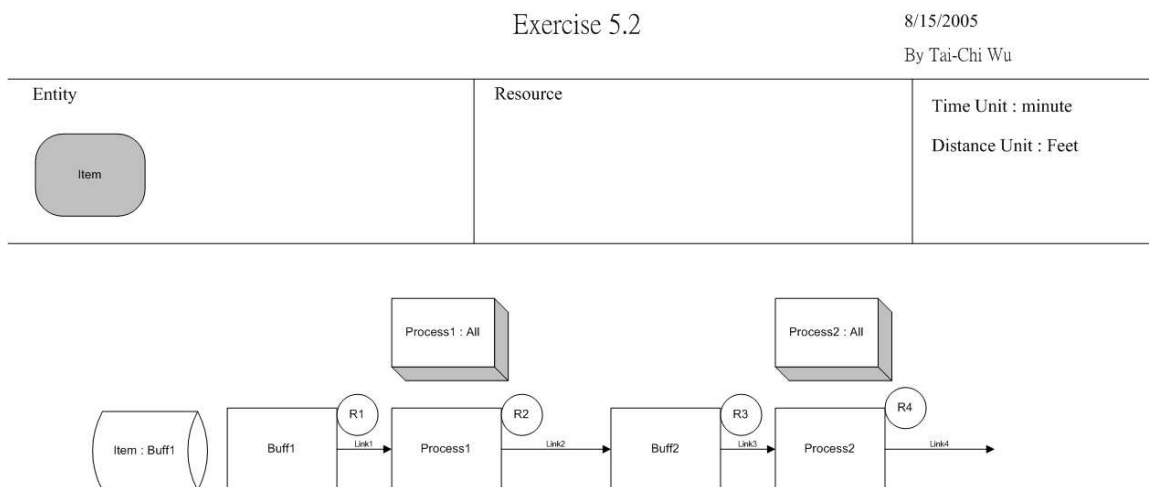


Figure 6.4 Graphical Model of Exercise 5.2

6.2.3 Exercise 5.3

Problem statement:

Modify the Exercise 5.1 check-in problem by adding agent breaks. The 16 hours are divided into two 8-hour shifts. Agent breaks are staggered, starting at 90 minutes into each shift. Each agent is given one 15-minute break. Agent lunch breaks (30 minutes) are also staggered, starting 3.5 hours into each shift. Compare the result of this model to the result without agent breaks.

This exercise is the extension of Exercise 5.1. The main challenge of this exercise is the addition of work shifts to the system. However, the proposed framework does not support shifts because, in most simulation software packages, it is considered an advanced modeling option. Work shifts can be modeled as downtime, but it requires the use of multiple downtime logic. Currently, the framework only supports a single downtime for each dynamic resource. Also, shifts affect resource availability but not the overall flow and operation of the system. A key aspect of the GMT during the model formulation stage is to capture the flow and operations logic and not necessarily resource availability. Different shift availabilities can be captured as notes to be implemented later in a specific simulation package. In future research, work shifts can be adopted as an attribute for certain model elements, such as static and dynamic resources. This is listed in Chapter 7 as future research needs.

6.2.4 Exercise 5.4

Problem statement:

Two different part types arrive at the same system for processing. Part Type 1 arrives according to a lognormal distribution with a log mean of 11.5 hours and log standard deviation of 2.0 hours. These arriving parts wait in a queue designated for Part Type 1's only until an operator is available to process them.

The processing time follows a triangular distribution with parameters 5, 6, and 8 hours. Part Type 2 arrives according to an exponential distribution with mean of 15 hours. These parts wait in a second queue until the same operator is available to process them. The processing time follows a triangular distribution with parameters 3, 7, and 8 hours. After being processed by the operator, all parts are sent for processing to a second operation that does not require an operator, triangular with parameters of 4, 6, and 8 hours. Completed parts exit the system. Assume that the times for all part transfers are negligible. Run the simulation for 5,000 hours to determine the average cycle time for all parts and the average number of items in the queues designated for the arriving parts.

This exercise has two major challenges. First, there are two types of entities. As shown in Figure 6.5, there are two operations defined in the first processing station and only one operation defined in the second processing station. Different types of entities may or may not require different operations. The GMT provides an “all” option in operation and routing element that allows assigning the same operation or routing logic to different types of entities, which simplifies the modeling process and also results in a cleaner drawing. The second challenge is that different entities share the same dynamic resource (an operator). The framework is able to capture this behavior.

Exercise 5.4

8/15/2005

By Tai-Chi Wu

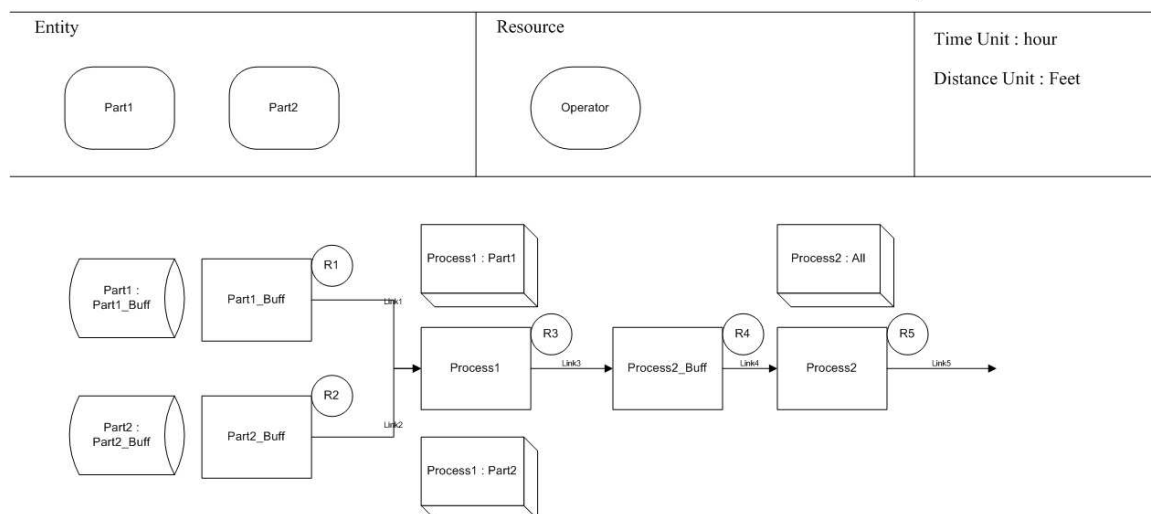


Figure 6.5 Graphical Model of Exercise 5.4

6.2.5 Exercise 5.5

Problem statement:

During the verification process of the airline check-in system from Exercise 5.3, it was discovered that there were really two types of passengers. The first passenger type arrives according to an exponential interarrival distribution with mean 2.4 minutes and has a service time following a normal distribution with mean of 6 minutes and standard deviation of 1.5 minutes. The second type of passenger arrives according to an exponential distribution with mean 4.4 minutes and has a service time following a normal distribution with mean of 11 minutes and standard deviation of 2 minutes. Modify the model from Exercise 5.3 to include this new information. Compare the results.

This exercise is an extension of Exercise 5.3, which contains shift information that the proposed framework cannot handle. Assuming there are no shifts in the system, the exercise can be modeled as shown in Figure 6.6.

Exercise 5.5

8/15/2005

By Tai-Chi Wu

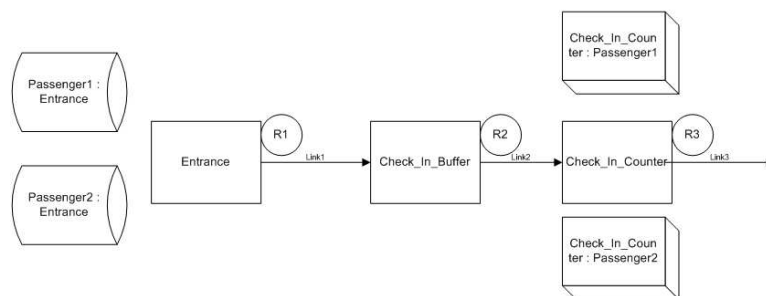
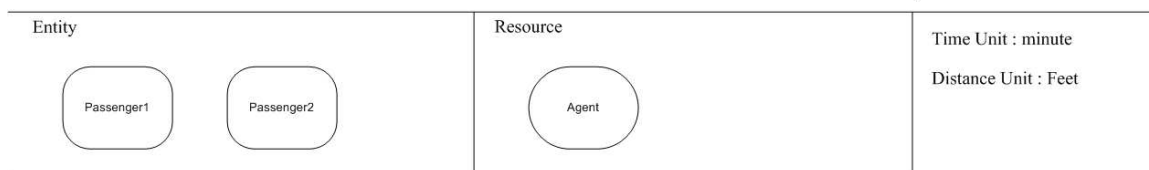


Figure 6.6 Graphical Model of Exercise 5.5

This exercise does not implement any new features. But it demonstrates that the GMT can modify existing models easily when new information becomes available.

6.2.6 Exercise 5.6

Problem statement:

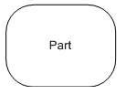
Parts arrive at a single workstation system according to an exponential interarrival distribution with mean 20 seconds. After being transferred to the workstation, the parts are processed. The processing time distribution is TRIA(16, 19, 22) seconds. There are several easily identifiable visual characteristics that determine if a part has a potential quality problem. These parts, about 10%, are transferred to a station where they undergo an extensive inspection. The remaining parts are considered good and are transferred out of the system. The inspection time distribution is NORM(120, 12) seconds. About 14% of these parts fail the inspection and are transferred to scrap. The parts that pass the inspection are classified as good and are transferred out of the system. Assume all transfer times are 2 minutes. Run the simulation for 10,000 seconds to determine the number of good parts that exit the system, the number of scrapped parts, and the number of parts that are inspected.

This exercise has two new major challenges. The first one is the use of percentage routing logic, which is handled by the proposed framework. The second challenge is the use of an entity attribute to denote the part quality. This exercise could be modeled without using attributes by putting two dummy dynamic resources at the end of the model to collect the total scrap and good part counts. To illustrate the capability of the proposed framework, user attributes are used to model the exercise. User attributes facilitate defining complex user-defined logic in certain places. For example, in the routing element, user-defined routing rules are very common. In the proposed framework, there is no standard modeling language defined, users may need to translate the user logic manually to the desired commercial simulation package environment using its proper syntax. In the CUI, when users try to upload the model that contains user logic to either *QUEST*[®] or *ProModel*[®], a log file containing warning message is created to indicate that user logic may require manual translation. The model is shown in Figure 6.7.

Exercise 5.6

8/15/2005

By Tai-Chi Wu

<p>Entity</p> 	<p>Resource</p>	<p>Time Unit : sec Distance Unit : Feet</p>
---	-----------------	---

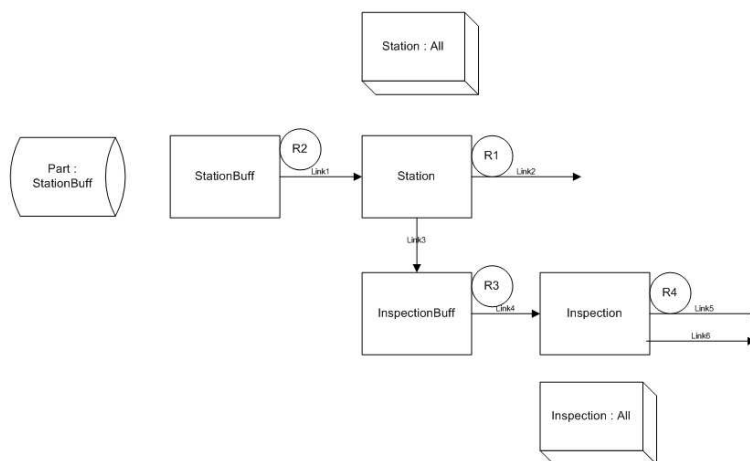


Figure 6.7 Graphical Model of Exercise 5.6

6.2.7 Exercise 5.7

Problem statement:

A proposed production system consists of five serial automatic workstations. The processing times at each workstation are constant: 11, 10, 11, 11, and 12 (all times given in this problem are in minutes). The part interarrival times are UNIF(13,15). There is an unlimited buffer in front of all workstations, and we will assume that the downstream transfer time is zero. The unique aspect of this system is that at workstations 2 through 5 there is a chance that the part will need to be reprocessed by the workstation that precedes it. For example, after completion at workstation 2, the part can be sent back to the queue in front of workstation 1. When this occurs, the transfer requires 3 minutes. The probability of revisiting a workstation is independent in that the same part could be sent back many times with no change in probability, the same for all four workstations, will be between 5% and 10%. Develop the simulation model and make six runs of 10,000 minutes each for probability of 5, 6, 7, 8, 9, and 10%. Using the results, construct a plot of the average cycle time (system time) against the probability of a revisit. Also include the maximum cycle time for each run on your plot.

This exercise primarily uses percentage routing. The model is shown in Figure 6.8.

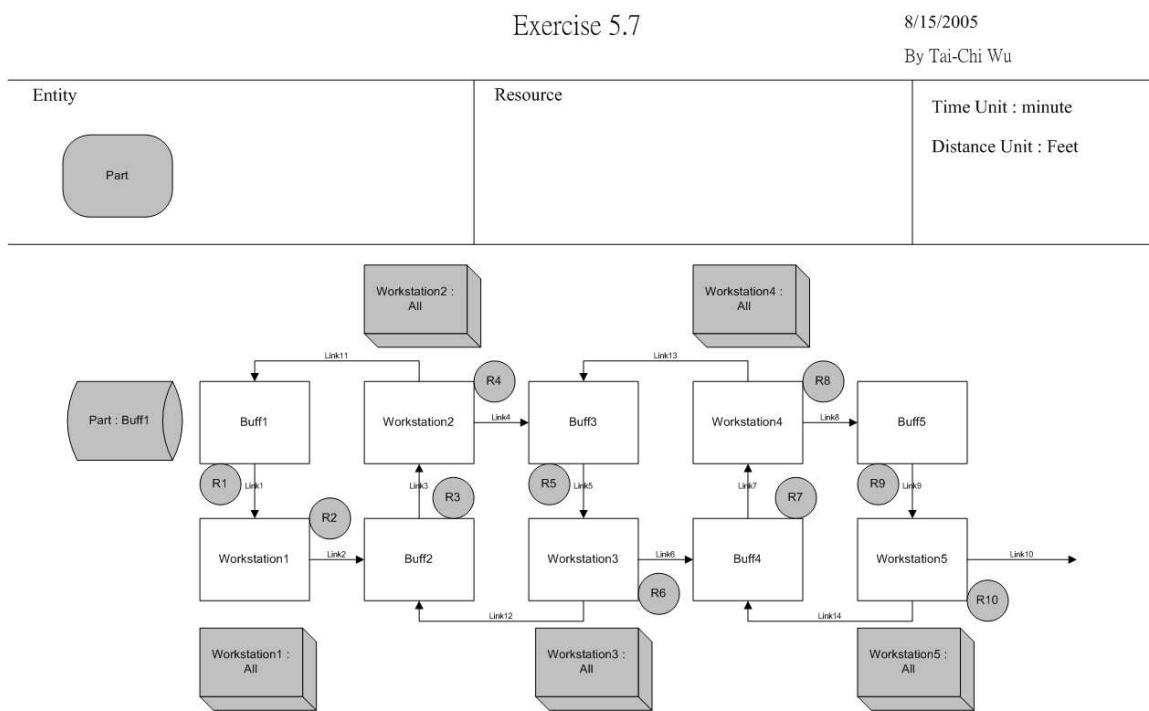


Figure 6.8 Graphical Model of Exercise 5.7

The model in Figure 6.8 looks very complex. It is hard to explain to others about the material flow of the system. Assume it is a physical layout of the real world system. With the graphical modeling tool, the model layout can be rearranged into a flowchart-like drawing. By easily dragging the model elements around, the physical layout drawing can be reformatted into Figure 6.9, a flowchart-like drawing, which is much easier to understand. This illustrates another advantage of the graphical modeling tool. Based on the perspective of users, the drawing can be modified easily to facilitate the understanding of different stakeholders.

Exercise 5.7

8/15/2005

By Tai-Chi Wu

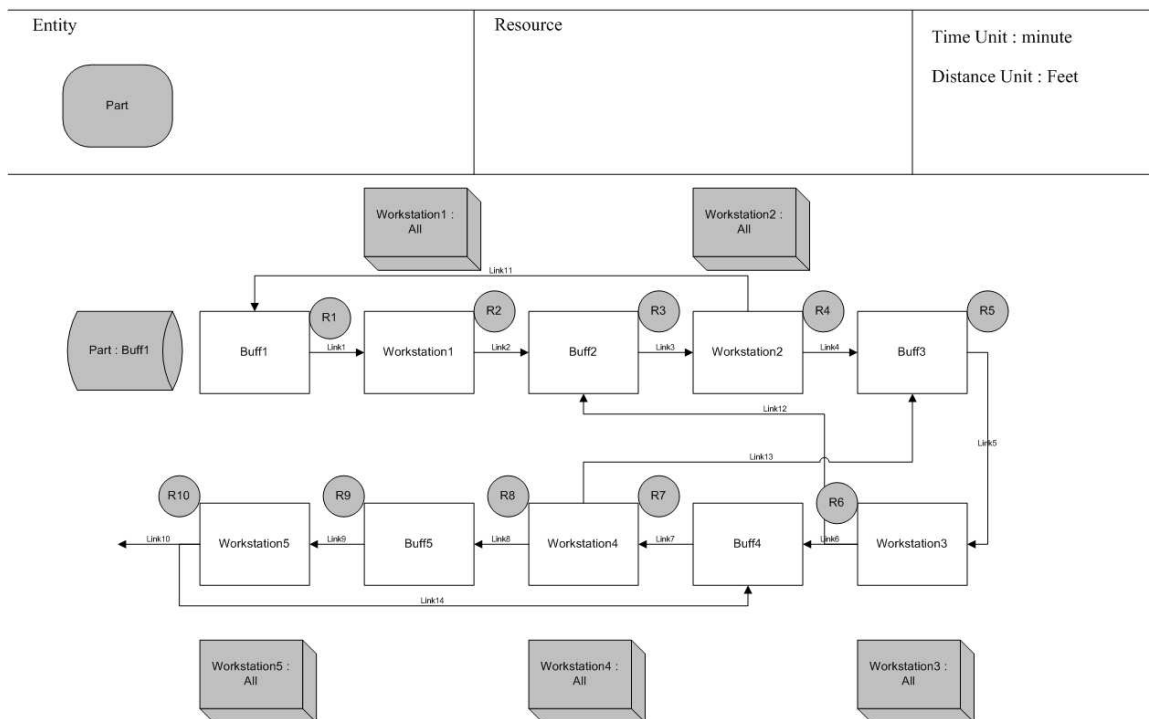


Figure 6.9 Flow Chart of Exercise 5.7

6.2.8 Exercise 5.8

Problem statement:

A production system consists of four serial automatic workstations. All transfer times are assumed to be zero and all processing times are constant. There are two types of failures: major and jams. The data for this system are given in the table below (all times are in minutes). Use exponential distributions for the up-times and uniform distributions for repair times (for instance, repairing jams at workstation 3 is UNIF(2.8, 4.2)). Model the major failures using the failure constructs (time with wait option) and the jams using the downtimes constructs. Run your simulation for 10,000 minutes to determine the percent of time each resource spends in the failure states and the ending status of each workstation queue.

Number	Mean	Major Failures		Jams	
	Process Time	MTBF	Repair	MTBF	Repair
1	8.5	475	20, 30	47.5	2, 3
2	8.3	570	24, 36	57	2.4, 3.6
3	8.6	665	28, 42	66.5	2.8, 4.2
4	8.6	475	20, 30	47.5	2, 3

This exercise uses two downtime and repair times for each station. The proposed framework only supports one downtime and one repair logic for each resource. It is not hard to modify the framework to support multiple downtimes and repair logic. However, the purpose of this research is to illustrate the usefulness of the proposed framework and its implementations. Multiple downtimes and repair logics will be listed as a future research need.

The purpose of this exercise is to get users more familiar with the downtime and repair logic capability in *Arena*[®]. Also, if there is no buffer between stations, the impact of downtime is more critical. Also, because the arrival time is not given, assume mean time between arrivals is constant eight minutes. Because the proposed framework only supports single downtime and repair logic, only the major failure is modeled. The model is shown as Figure 6.10.

Exercise 5.8

8/15/2005

By Tai-Chi Wu

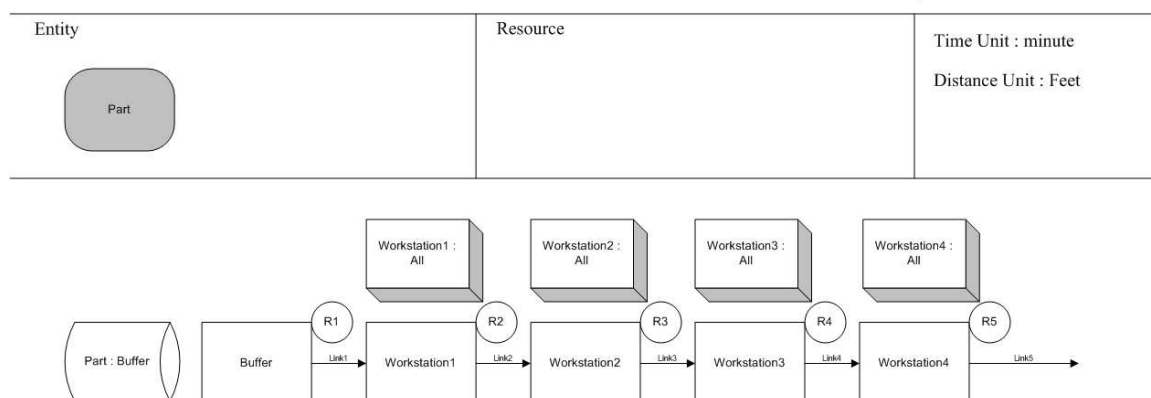


Figure 6.10 Graphical Model of Exercise 5.8

6.2.9 Exercise 5.9

Problem statement:

An office that dispenses automotive license plates has divided its customers into categories to level the office workload. Customers arrive and enter one of three lines based on their residence location. Model this arrival activity as three independent arrival streams using an exponential interarrival distribution with mean of 10 minutes for each stream. Each customer type is assigned a single clerk that processes the application forms and accepts payment. The service time in UNIF(8, 10) minutes for all three customer types. After completion of this step, all customers are sent to a second clerk who checks the forms and issues the plates. The service time for this activity is UNIF(2.66, 3.33) minutes for all customer types. Develop a model of this system and run the simulation for 5,000 minutes.

A consultant has recommended that the office eliminate the step of differentiating between customers and use a single line with three clerks who can process any customer type. Develop a model of this system, run it for 5,000 minutes, and compare the results with the first system.

The purpose of this exercise is to observe the difference between two customer service policies. Figure 6.11 shows the system with different lines for each type of

customer. Figure 6.12 shows the system with a single line for all customers. This exercise does not use any new characteristic of the proposed framework.

Exercise 5.9

8/15/2005

By Tai-Chi Wu

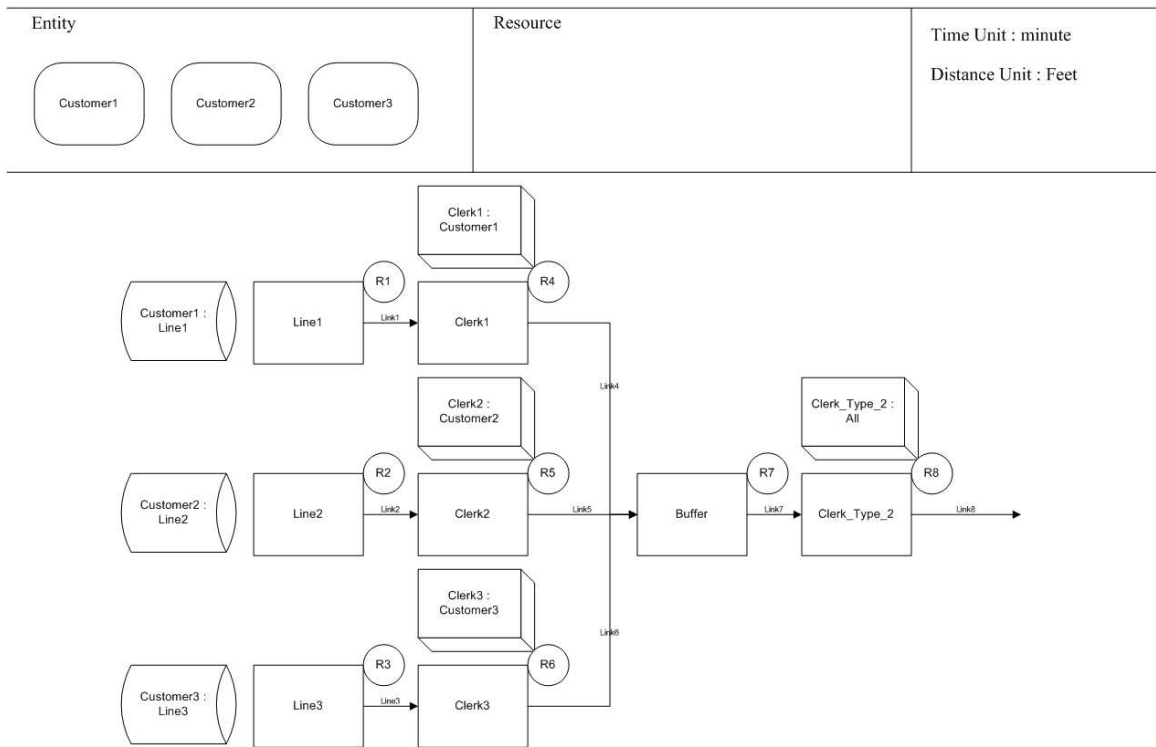


Figure 6.11 Exercise 5.9 With Lines for Different Customers

Exercise 5.9

8/15/2005

By Tai-Chi Wu

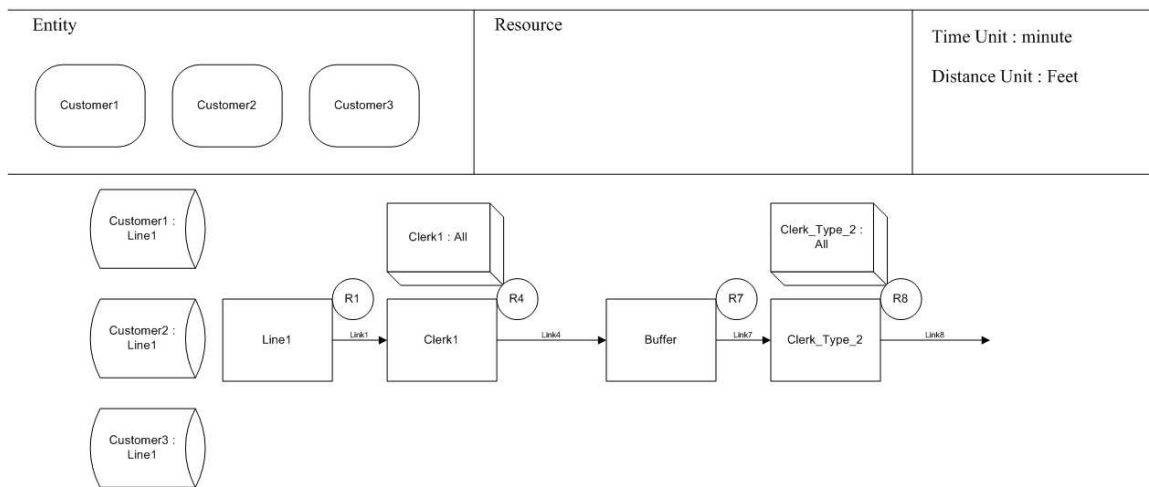


Figure 6.12 Exercise 5.9 with One Line for All Customers

6.2.10 Exercise 5.10

Problem statement:

Customers arrive at an order counter with exponential interarrivals with a mean of 10 minutes. A single clerk accepts and checks their orders and processes payments, UNIF(8,10) minutes. Upon completion of this activity, orders are randomly assigned to one of two available stock persons who retrieve the orders for the customers, UNIF(16, 20) minutes. These stock persons only retrieve orders for customers who have been assigned specifically to them. Upon receiving their orders, the customers depart the system. Develop a model of this system and run the simulation for 5,000 minutes.

A bright, young engineer has recommend that they eliminate the assignment of an order to a specific stock person and allow both stock persons to select their next activity from a single order queue. Develop a model of this system, run it for 5,000 minutes, and compare the results to the first system.

The purpose of this exercise is to observe the difference between two order processing polices. In Figure 6.13, there are separate buffers for each stock person. Customers are sent to the buffers randomly. In Figure 6.14, there is only one buffer prior

to the stock persons. All customers are sent to the buffer then processed by the first available stock person.

Exercise 5.10

8/15/2005

By Tai-Chi Wu

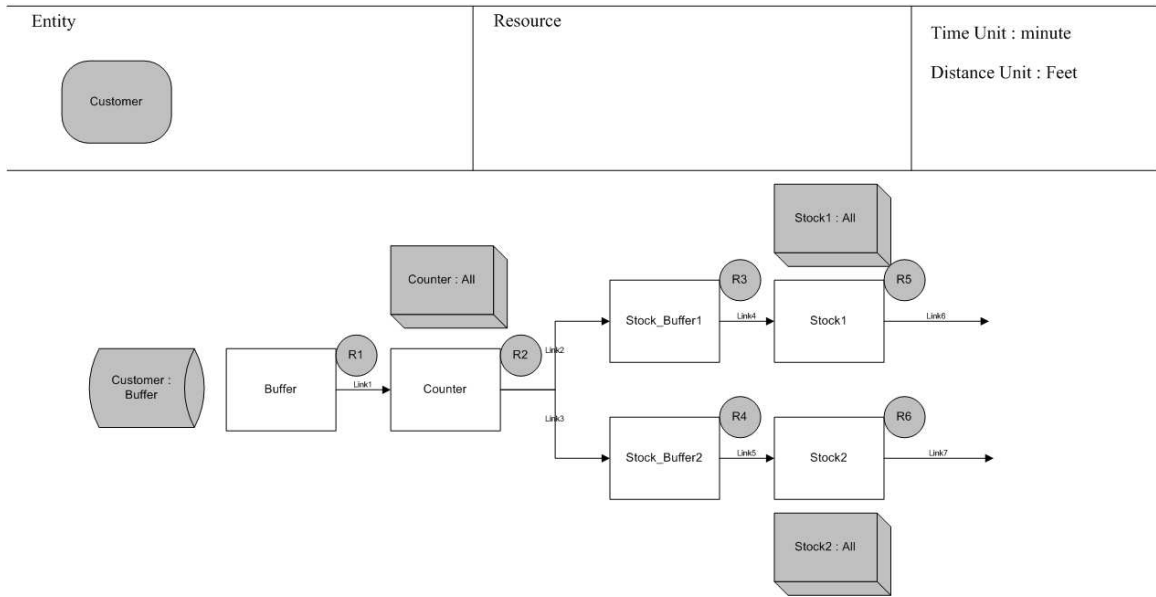


Figure 6.13 Graphical Model of Exercise 5.10 With Orders Randomly Assigned

Exercise 5.10

8/15/2005

By Tai-Chi Wu

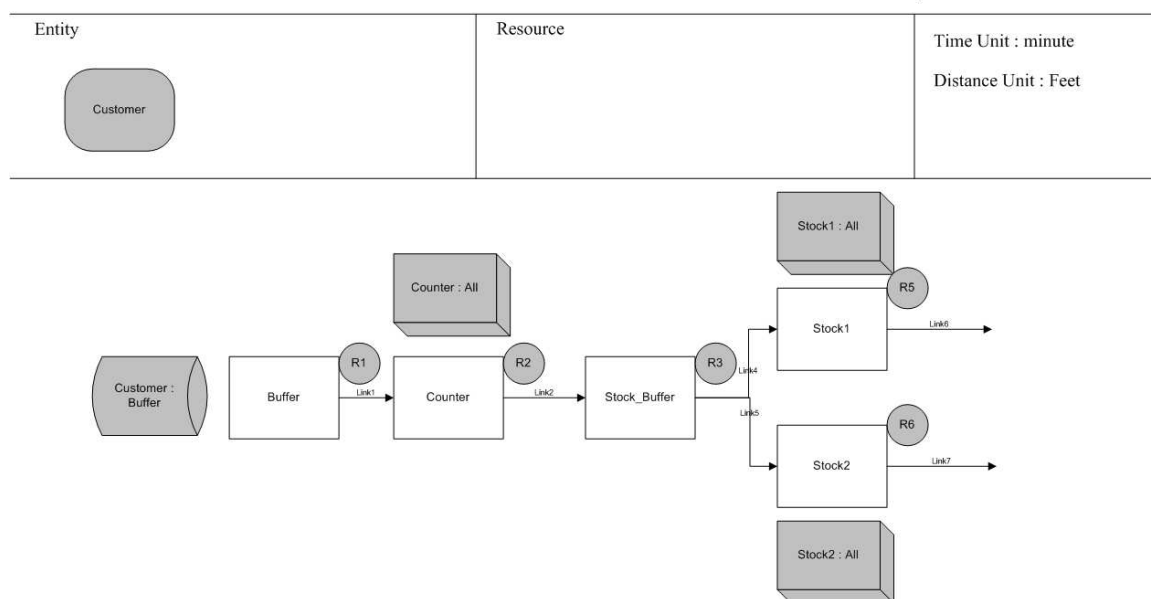


Figure 6.14 Graphical Model of Exercise 5.10 With Order Assigned to First Available

6.2.11 Exercise 5.11

Problem statement:

Using the model from Exercise 5.2, set the interarrival-time distribution to exponential and the process-time distribution for each process to normal with a mean of 9 minutes. Setting the standard deviation of the normal distribution to values of 1, 2, and 3. Make three different runs of 10,000 minutes each and compare the results.

The main purpose of this exercise is to observe the effect of modifying model parameters on system behavior. This has been demonstrated earlier.

6.2.12 Exercise 5.12

Problem statement:

Using the model from Exercise 5.11, assume the process time has a mean of 9 and a variance of 4. Calculate the parameters for the gamma, uniform, and normal distributions that will give these values. Make three runs (one for each distribution) and compare the results.

The calculation of statistical distribution parameters is not a part of this research.

6.2.13 Exercise 5.13

Problem statement:

Using the Input Analyzer, open a new window and generate a new data file containing 50 points for an Erlang distribution with parameters: ExpMean equal to 12, k equal to 3, and Offset equal to 5. Once you have the data file, perform a best fit. Repeat this process for 500, 5,000, and 25,000 data points, using the same Erlang parameters. Compare the results of the best fit for the four different sample sizes.

This exercise provides practice in probability distribution fitting and utilizes two support applications that come with *Arena*[®]. Statistical distribution fitting is not a part of this research.

6.2.14 Exercise 5.14

The problem statement of this exercise is skipped for brevity. This exercise uses shifts and schedules that cannot be handled by the proposed framework.

6.3 Summary of the Chapter

In this chapter, the limitations and the benefits of the proposed approach are identified. The framework does not support the shuttle bus problem in Test Set 1 because global variables are involved. In Test Set 2, nine exercises are fully applicable and three are out of scope of this research. Exercise 5.3 is not applicable because of the use of work

shifts. Exercise 5.8 is partially applicable because the framework only supports single downtime and repair logic for each static resource. The limitations of the framework are further discussed in the next chapter.

CHAPTER VII

LIMITATION OF THE PROPOSED APPROACH

In this chapter, the performance of the proposed framework and its implementations using *ProModel*[®] and *QUEST*[®] are discussed. In Figure 7.1, there are four opportunities for the proposed framework to interact with *QUEST*[®] and *ProModel*[®]:

- 1) from the XML file, through the *ProModel*[®] Controller within Common User Interface (CUI), to ProModel,
- 2) from the XML file, through the *QUEST*[®] Controller within CUI, to *QUEST*[®],
- 3) from *ProModel*[®], through the *ProModel*[®] Controller within CUI, to XML file, and
- 4) from *QUEST*[®], through the *QUEST*[®] Controller within CUI, to XML file.

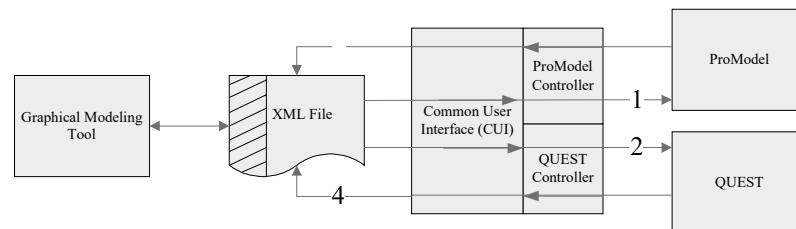


Figure 7.1 Relationships Between Simulation Packages and Implementations

In Figure 7.1, the shadowed part in the XML file icon means only the XML files generated by the graphical modeling tool (GMT) can be edited by the GMT because the position information that is required to place the icons in the Visio drawing may be missing. For example, in *ProModel*[®], processes are not shown on the model layout, thus they do not have position information. The extent to which the proposed framework can support these interaction opportunities are examined in detail in the following sections. Subsequently, in Section 7.5, the methodology for how exceptions of translations between the proposed framework and simulation software are discussed.

A series of tables are used to illustrate the performances of the proposed framework. There are eight major elements in the framework: general information, entity, static resource, dynamic resource, arrival, operation, linkage, and routing. Each table contains the properties that are a part of each major element. For example, the general information element contains model name, time units, distance unit, date, builder, and notes. These properties are represented as rows, each row contains four columns: property name, default value, support, and notes. A filled circle, ●, in the support field means this property is fully supported. A half filled circle, ◐, means it is partially supported. An empty circle, ○, means it is not supported. X means the property is not considered because of different modeling view or referencing purpose. The notes field explains why the property is partially supported or not supported.

Commercial simulation packages usually contain elaborate 2D/3D graphical icons or objects to represent simulation elements, such as machines. These display relevant data, such as CAD objects and colors, are excluded from this chapter because they are software

specific and are not critical to the execution of simulation models. Also, simulation software applications usually allow users to select a collection of performance measurements. This functionality belongs to the experimental design side and are therefore out of the scope of discussion of this chapter.

7.1 Compatibility from Framework to ProModel (Opportunity 1)

Tables 7.1 through 7.8 illustrate the translation capability between the various framework elements to *ProModel*[®]. Almost everything except that which contains user-defined logics can be translated from the framework to *ProModel*[®]. The user-defined logics have limited support; in fact, the CUI only recognizes a few keywords: wait, move for, move with, get, and free. The “-” in the default value field means there is no default value for the property or the default value is an empty string.

Table 7.1 General Information Compatibility from Framework to ProModel

Property name	Default value	Support	Note
Model name	-	●	
Time unit	“Minute”	●	
Distance unit	“Feet”	●	
Date	-	●	
Builder	-	○	ProModel does not have a builder field.
Notes	-	●	

Table 7.2 Entity Resource Compatibility from Framework to ProModel

Property name	Default value	Support	Note
Name	-	●	
Speed	-	●	
Notes	-	●	

Table 7.3 Static Resource Compatibility from Framework to ProModel

Property name	Default value	Support	Note
Name	-	●	
Type	“Processing unit”	●	
Capacity	1	●	
Units	1	●	
Downtime	-	●	
Repair	-	●	
Sequence rule	“FIFO”	●	
Notes	-	●	

Table 7.4 Dynamic Resource Compatibility from Framework to ProModel

Property name	Default value	Support	Note
Name	-	●	
Units	1	●	
Downtime	-	●	
Repair	-	●	
Notes	-	●	

Table 7.5 Arrival Compatibility from Framework to ProModel

Property name	Default value	Support	Note
AID	-	X	For reference purpose only.
ArvEntity	-	●	Arrival entity.
ArvLocation	-	●	Arrival location.
QtyEach	1	●	Arrival quantity.
Frequency	-	●	
FirstTime	-	●	
Occurrence	“Inf”	●	
UserLogic	-	◐	Limited support.

Table 7.6 Linkage Compatibility from Framework to ProModel

Property name	Default value	Support	Note
LID	-	X	For reference purpose only.
BeginLoc	-	●	Begin location.
EndLoc	-	●	End location.
Notes	-	○	ProModel does not have notes field for linkage in process.

Table 7.7 Routing Compatibility from Framework to ProModel

Property name	Default value	Support	Note
RID	-	X	For reference purpose only.
Linkage	-	◐	Duration and dynamic resource are supported. User logic is limited supported.
RouteEntity	“All”	●	
Rule	-	●	Routing rule is supported.
QtyEach	1	●	Route quantity.
Notes	-	○	ProModel does not have notes field for routing in process.

Table 7.8 Operation Compatibility from Framework to ProModel

Property name	Default value	Support	Note
OID	-	X	For reference purpose only.
Process	-	◐	Duration and dynamic resource are supported. User logic has limited support.
OpLocation	-	●	Operation location.
InEntity	“All”	●	Incoming entity.
OutEntity	“All”	●	Outgoing entity.
Notes	-	○	ProModel does not have notes field in process.

7.2 Compatibility from Framework to QUEST (Opportunity 2)

Tables 7.9 through 7.16 contain the translation performance from the framework to *QUEST*[®]. The format of these tables is similar to the ones in the previous section.

Table 7.9 General Information Compatibility from Framework to QUEST

Property name	Default value	Support	Note
Model name	-	○	There is no model name filed in QUEST.
Time unit	“Minute”	●	
Distance unit	“Feet”	●	
Date	-	○	There is no date field in QUEST.
Builder	-	○	There is no builder field in QUEST.
Notes	-	●	

Table 7.10 Entity Compatibility from Framework to QUEST

Property name	Default value	Support	Note
Name	-	●	
Speed	-	○	QUEST does not support part speed.
Notes	-	●	

Table 7.11 Static Resource Compatibility from Framework to QUEST

Property name	Default value	Support	Note
Name	-	●	
Type	“Processing unit”	●	Support both machine and buffer class in QUEST.
Capacity	1	●	*
Units	1	●	
Downtime	-	●	
Repair	-	●	
Sequence rule	“FIFO”	●	*
Notes	-	●	

* In QUEST, machine class only has capacity of one. Thus there is no sequence rule. Buffer class can have more than one capacity, and can define sequence rules.

Table 7.12 Dynamic Resource Compatibility from Framework to QUEST

Property name	Default value	Support	Note
Name	-	●	*
Units	1	●	
Downtime	-	○	**
Repair	-	○	**
Notes	-	●	

* QUEST supports three types of dynamic resources: labor, AGV, and carrier. Only the labor class is supported in the proposed framework.

** In QUEST, users need to write and compile Simulation Control Language code for downtime and repair logic in labor.

Table 7.13 Arrival Compatibility from Framework to QUEST

Property name	Default value	Support	Note
AID	-	X	For reference purpose only.
ArvEntity	-	●	Arrival entity.
ArvLocation	-	●	Arrival location.
QtyEach	1	●	Arrival quantity.
Frequency	-	●	
FirstTime	-	●	
Occurrence	“Inf”	●	
UserLogic	-	○	*

* In QUEST, users need to write and compile Simulation Control Language codes for arrival logics.

Table 7.14 Linkage Compatibility from Framework to QUEST

Property name	Default value	Support	Note
LID	-	X	For reference purpose only.
BeginLoc	-	●	Begin location.
EndLoc	-	●	End location.
Notes	-	●	

Table 7.15 Routing Compatibility from Framework to QUEST

Property name	Default value	Support	Note
RID	-	X	For reference purpose only
Linkage	-	◐	Duration and dynamic resource are supported. User logic is not supported.
RouteEntity	“All”	●	
Rule	-	●	Routing rule is supported
QtyEach	1	●	
Notes	-	○	*

* In QUEST, routing is embedded in machine and buffer classes, and thus does not have an independent field for routing descriptions.

Table 7.16 Operation Compatibility from Framework to QUEST

Property name	Default value	Support	Note
OID	-	X	For reference purpose only.
Process	-	◐	Sub-properties duration and dynamic resource are supported. Sub-item user logic is limited supported.
OpLocation	-	●	Operation location.
InEntity	“All”	●	Incoming entity.
OutEntity	“All”	●	Outgoing entity.
Notes	-	○	QUEST does not have note field in process.

7.3 Compatibility from ProModel to the Framework (Opportunity 3)

The major model elements in *ProModel*[®] are: general information, locations, entities, resources, path networks, processing, arrivals, shifts, attributes, global variables, arrays, macros, subroutines, path-networks, and cost. The proposed framework does not support shifts, path-networks, and costs because they are considered advanced features. The framework does not support global variables, macros, and subroutines because they are too programming or scripting oriented. All other major *ProModel*[®] elements are supported and are discussed in this section.

7.3.1 General Information

Table 7.17 shows the compatibility of the general information elements in terms of the interaction from *ProModel*[®] to the framework. The general information contains the metadata about the model. The framework supports everything except the initialization logic and termination logic. It is not difficult to include them in the

framework; however these two logic blocks are more relative to the experimental design aspect of simulation, and thus are excluded.

Table 7.17 General Information Compatibility from ProModel to Framework

Property name	Default value	Support	Note
Title	-	●	
Time Units	Minutes	●	
Distance Units	Feet	●	
Initialization logic	-	○	Not supported by the framework.
Termination logic	-	○	Not supported by the framework.

7.3.2 Entities

Table 7.18 shows the entity compatibility from *ProModel*[®] to framework. It is fully supported by the framework.

Table 7.18 Entity Compatibility from ProModel to Framework

Property name	Default value	Support	Note
Icon	-	●	
Name	-	●	
Speed	-	●	
Notes	-	●	

7.3.3 Locations

Locations correspond to static resources in the framework. The compatibility of locations is shown in Table 7.19. In *ProModel*[®], downtimes can be defined based on clock time, entries, usages, and setup downtime. Only one clock downtime per location is supported in the framework. Also, the logic field within the downtime property allows

users to write their own user-defined logic. The framework is limited to time to repair (TTR) logic and the associated resource required for repair. The rules property in *ProModel*[®] contains incoming entities selection rules and queuing for output rules. The framework does not support incoming entities selection rules.

Table 7.19 Location Compatibility from ProModel to Framework

ProModel attribute	Default value	Support	Note
Name	-	●	
Capacity	1	●	
Units	1	●	
Downtimes	-	◐	Limited support. *
Rules	-	◑	Select incoming rules are not supported.
Notes	-	●	

* Only supports single downtime and repair logics. First time, priority, and scheduled downtimes are not supported.

7.3.4 Resources

The resources correspond to the dynamic resources in the framework. Table 7.20 shows the compatibility of resources. Downtimes have limited support, as described in the previous section. There are two properties that are not supported by the framework: specifications and points. The specifications option contains information on such things as path networks, resource search rules, entity search rules, and motion data. The points option defines the resource traveling positions during model execution. It is mainly for animation purpose, thus is not included in the framework.

Table 7.20 Resource Compatibility from ProModel to Framework

ProModel attribute	Default value	Support	Note
Name	-	●	
Units	1	●	
Downtimes	-	◐	*
Specifications	-	○	Not supported.
Points	-	○	Not supported.
Notes	-	●	

* Only supports single downtime and repair logics. First time, priority, and scheduled downtimes are not supported.

7.3.5 Arrival

The compatibility of arrival properties is shown in Table 7.21. The framework supports everything except user-defined logic.

Table 7.21 Arrival Compatibility from ProModel to Framework

ProModel attribute	Default value	Support	Note
Entity	-	●	
Location	-	●	
Qty each	1	●	
First time	0	●	
Occurrences	“Infinite”	●	
Frequency	-	●	
Logic	-	◐	Limited support.

7.3.6 Process

The process element in *ProModel*[®] is a combination of three elements in the proposed framework: linkage, routing, and operation. It is challenging to subdivide process information and place them in the right place in the framework, but the ProModel

Controller is able to manage this. The operation and move logic properties in the process element allow users to write user-defined logic, thus they have limited support in the framework. The framework also only supports a subset of the *ProModel*[®] routing rules.

Table 7.22 Process Compatibility from ProModel to Framework.

ProModel attribute	Default value	Support	Note
Entity	-	●	
Location	-	●	
Operation	-	◐	User-defined logic has limited support.
Output	-	●	
Destination	-	●	
Routing rule	-	◐	Limited support. Framework supports first available, by turns, probability, and random routing.
Move logic	-	◐	User-defined logics are limited supported.

7.4 Compatibility from Quest to the Framework (Opportunity 4)

QUEST[®] is a very powerful and open simulation software package. It allows users to modify almost anything, thus it contains a considerable number of features that are not likely to be used by most model builders. *QUEST*[®] supports 2D and 3D simulation environments. As such, it contains features for display and animation purposes, such as CAD tools that are included within the package for creating 2D/3D objects and kinematic elements. Specifically, these features includes:

- Accessory – This element class adds 2D/3D objects into a model for display purposes only.
- Stack direction, stack points, and stack factor – These attributes affect the stacking mechanism of parts in buffer or source.

- Way points, labor points, and via path – These features are used to define the traveling path and standing points of the labor classes. It may affect the traveling time of labor classes if the traveling time is based on the labor speed. The framework uses fixed travel time. These features are for animation purposes only.

These features are excluded from this section, because they are not critical to the execution of simulation and the complexity involved makes it out of the scope of this research.

In this section, only major *QUEST*[®] elements critical to model execution are discussed, they are: general information, parts, machine, buffer, source, sink, conveyor, connections, labor controller, automated guided vehicle (AGV) controller, labor, AGV, carrier, path network system, shifts, cycle process, fluid process, repair process, setup process, load/unload process, and failure. As mentioned earlier, the framework does not support conveyors, path network systems, and shifts. General information element is discussed in Section 7.4.1. The part element is discussed in Section 7.4.2. The machine and buffer elements are discussed in Section 7.4.3. The source element is discussed in Section 7.4.5. The sink element is a unique to *QUEST*[®]. The main purpose of the sink class is to collect statistical data; therefore, it is excluded. Connections are discussed in Section 7.4.6. Dynamic resource relative elements (e.g., controllers, AGV, labor and carrier) are discussed in Section 7.4.4. Operation relative elements (cycle process, fluid process, repair process, setup process, load/unload process, and failure) are discussed in Section 7.4.6.

Also, *QUEST*[®] supports both push and pull systems. Currently, the framework only supports push systems. Two other unique features that are not supported by the framework are: *group* and *pop-up*. A *group* allows the grouping of multiple processes into one process union. A *pop-up* is a file containing a list of procedures that can be associated with elements in *QUEST*[®].

7.4.1 General Information

The general information element contains the metadata of the model. As mentioned earlier, the initial logic and termination logic are not supported. Also, the distance unit of measure is contained in a *QUEST*[®] model.

Table 7.23 General Information Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Developer	-	●	
Time/date	-	●	
Time unit	-	●	
Model description	-	●	

7.4.2 Parts

The parts class corresponds to the entity element in the framework. Properties that are not listed in Table 7.24 are display (e.g., 3D object), and history output file. Priority, routing labor requirements, routing sub-resource (SR) requirement, associated sub-resource class, and required processes are unique attributes of *QUEST*[®]. The framework does not support them.

Table 7.24 Parts Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Name	-	●	
Priority	1	○	Not supported by the framework.
Routing labor requirement	“No Labor”	○	Not supported by the framework
Routing SR requirement	“No SR”	○	Not supported by the framework.
Associated sub-resource class	“None”	○	Not supported by the framework.
User attribute	-	●	
Required processes	-	○	Not supported by the framework.
Description	-	●	

7.4.3 Machine and Buffer

In *QUEST*[®], machine, buffer, and conveyor correspond to static resources in the framework. As mentioned earlier, conveyors are not included in the framework. Unlike the framework, the machine and buffer classes are a combination of routing logic and static resource. Therefore, these classes must be split into two elements in the framework. The *QUEST*[®] Controller that implements the framework manages the splitting.

The compatibility of machine class is shown in Table 7.25. The properties that are not listed in the table are: “save in”, display, labor parking, SR parking, and random streams.

Table 7.25 Machine Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Name	-	●	
No. of elements	1	●	
Input, output type	-	◐	Only supports push system.
No. of processes	1	●	Supports multiple processes.
Priority	1	○	Not supported by the framework.
Part initial stock	0	○	Not supported by the framework.
Process percentage	-	○	Processes are executed in sequence in the framework.
Process group	-	○	Not supported by the framework.
Logics	-	◐	Use default logics on everything except route logic.
SR and labor controller	-	○	Not supported by the framework.
Shifts	-	○	Not supported by the framework.
Part routing	-	●	
Cycle Process	“Default process”	●	
Setup process	-	X	Included in multiple cycle processes.
Unload process	-	X	Included in multiple cycle processes.
Failure	-	●	
Request routing	-	○	Only supports push system.
User attribute	-	●	
Labor move time	-	●	
Dedicated labors	-	○	Not supported by the framework..
Labor depart requirement	-	●	
Description	-	●	

The compatibility of the buffer class is shown in Table 7.26. The properties that are not listed in the table are: “save in”, display, labor parking, SR parking, and random streams. Labor parking and SR parking are features for display purposes only.

Table 7.26 Buffer Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Name	-	●	
No. of elements	1	●	
Input and output type		○	Only supports push type.
Part capacity	“Infinite”	●	In QUEST, machine class only has capacity of 1. Buffer class can have capacity of any size.
Priority	1	○	Not supported by the framework.
Thresholds	-	○	Not supported by the framework.
Unload process	-	○	Not supported by the framework.
Failures	-	●	
Request routing	-	○	Pull processes are not supported.
Part initial stock	0	○	Not supported by the framework.
Load process	-	○	Not supported by the framework.
Logics	-	○	User default logics only.
SR and labor controller	-	○	Not supported by the framework.
Shifts	-	○	Not supported by the framework.
Part routing	-	●	
Delay time	0	○	Not supported by the framework.
User attribute	-	●	
Labor move time	-	●	
Labor depart requirement	-	●	
Description	-	●	

7.4.4 Labor, AGV, and controllers

Labor corresponds specifically to the dynamic resources in the framework. There are three types of material handling system (MHS) in *QUEST*[®]: labor, AGV, and carrier. The framework only supports the labor class. There are also two types of controllers in

QUEST[®]: the labor controller and the AGV controller. The controllers contain the logic associated with the MHS, such as the labor selection logic and labor path selection logic. In *QUEST*[®], to use the labor construct, at least one labor controller is needed. The compatibility of the framework with the labor class is shown in Table 7.27. The properties that are not listed in the table are: “save in”, animation mode, locate labor, locating space, move time mode, rotation speed, display, and random stream.

Table 7.27 Labor Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Name	-	●	
No. of elements	1	●	
Controller	-	○	Not supported by the framework.
Priority	1	○	Not supported by the framework.
Part capacity	-	○	Not supported by the framework.
Unload process	-	○	Not supported by the framework.
Failure	-	●	
Load process	-	○	Not supported by the framework.
Logics	-	○	Use only default logics.
Shifts	-	○	Not supported by the framework.
Move time	-	X	This is also defined in machine and buffer class.
Speed	-	○	Not supported by the framework.
Description	-	●	

7.4.5 Source

A source is a combination of arrival logic and a buffer. The framework views a source only in terms of its arrival logic. Thus, there are different modeling views between a *QUEST*[®] source and the framework. The compatibility of the source class within

framework is shown in Table 7.28. The properties that are not listed in the table are:

“save in”, display, SR parking, labor parking, random streams.

Table 7.28 Source Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Name	-	●	
No. of elements	1	●	
Max. part count	9999999	●	
Start offset	0	●	
Part creation mode	“Active”	○	Pull system is not supported.
Output type	“Push”	◐	Only supports push system.
Priority	1	○	Not supported by the framework.
Part initial stock	0	X	
Lotsize	1	●	
Unload process	-	X	
Failure	-	X	
IAT	-	●	
Part fractions	-	○	Only supports one type of entity arrival.
Logics	-	○	Only uses default logic.
SR controller	-	○	Not supported by the framework.
Shifts	-	○	Not supported by the framework.
Part routing	-	X	
User attribute	-	X	
Labor depart requirement	-	X	
Labor move time	-	X	
Description	-	●	

7.4.6 Connections

Linkage is fully supported by the framework, as shown in Table 7.29.

Table 7.29 Connection Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Begin location	-	●	
End location	-	●	

7.4.7 Process and Failure

There are six types of processes in *QUEST*[®]: cycle, fluid cycle, setup, repair, load, and unload processes. The framework only supports the cycle and repair processes. The cycle processes corresponds to the operation in the framework. A machine can contain multiple processes, which is supported by the framework.

The failure class in *QUEST*[®] relates to the resource downtime logic in the framework. The repair process also matches the resource repair logic in the framework. These two classes are viewed as processes in *QUEST*[®].

Table 7.30 shows the compatibility of the cycle process. The framework does not support priority, rejection rate, claim order, AGV requirement, sub-resource requirement, cycle process group, pop-ups, and user attributes. The framework also allows only one type of entity to be processed at a time. Likewise, only one type of labor can be claimed during the process.

Table 7.30 Cycle Process Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Name	-	●	
Priority	-	○	Not supported by the framework.
Rejection rate	0.0%	○	Not supported by the framework.
Claim order	-	○	Use default value only.
Part requirement	-	◐	Only supports processing one type of part at a time.
AGV requirement	-	○	Not supported by the framework.
Labor requirement	-	◐	Only supports one type of labor.
Sub-Resource req.	-	○	Not supported by the framework.
Cycle time	-	●	
Products	-	●	
Precedence processes	-	●	
Cycle process group	-	○	Not supported by the framework.
Attached popups	-	○	Not supported by the framework.
User attribute	-	○	Not supported by the framework.
Description	-	●	

Table 7.31 shows the compatibility of failure class with the framework. The framework only supports time between failures (TBF). Thus, only failure distribution is fully supported.

Table 7.31 Failure Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Name	-	○	Failure does not have a name in the framework.
Failure mode	Simulation time	◐	Only supports simulation time.
First failure by	System	◐	Only supports system-generated failures by default.
Schedule failure	After repair	◐	Only supports scheduled failure after repair by default.
Repair process	-	●	
Priority	1	○	Not supported by the framework.
Logics	-	○	User-defined logic are not supported.
Failure distribution	-	●	
Behavior	-	○	Use default settings.
User attributes	-	○	Not supported by the framework.

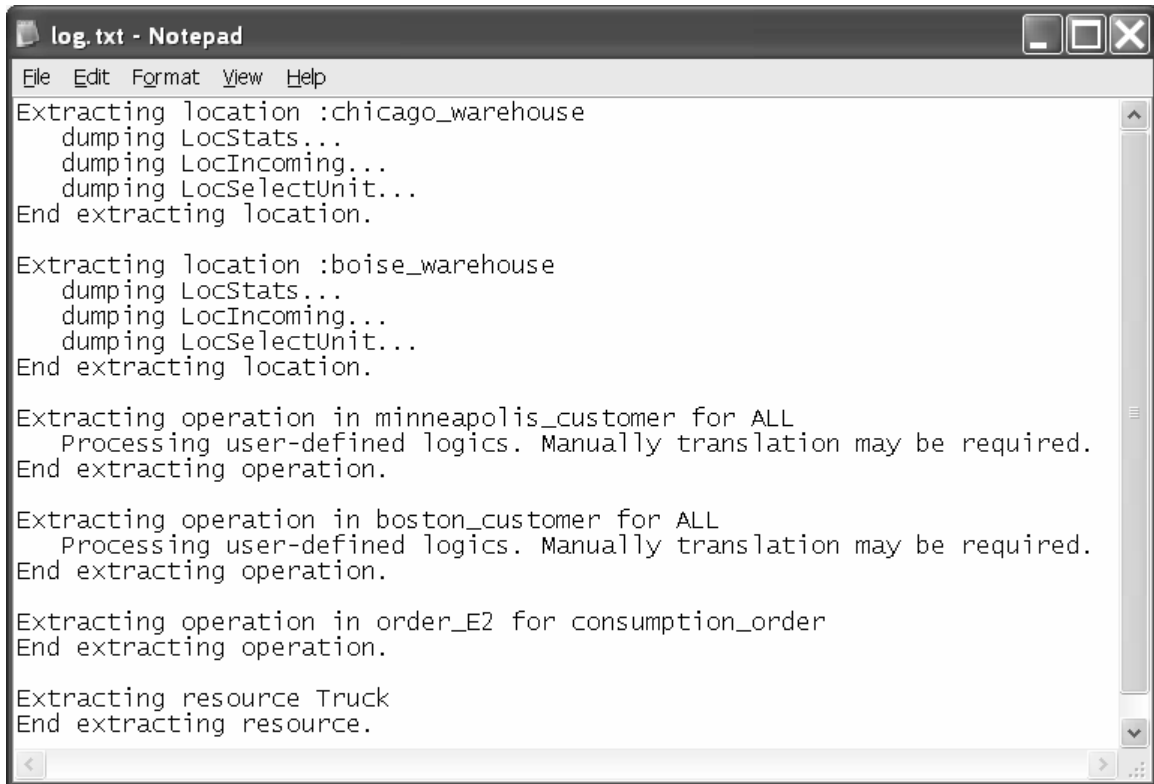
Table 7.32 shows the compatibility of the repair class. The framework only supports time to repair (TTR) and the required dynamic resource. Thus, only two properties are supported in the table.

Table 7.32 Repair Process Compatibility from QUEST to Framework

Property name	Default value	Support	Note
Name	-	○	Repair does not have a name in the framework.
Priority	1	○	Not supported by the framework.
Claim order	-	○	Use default values only.
Labor requirement	-	◐	Only supports one type of labor.
AGV requirement	-	○	Not supported by the framework.
Sub-resource requirement	-	○	Not supported by the framework.
Cycle time	-	●	
User attribute	-	○	Not supported by the framework.
Description	-	○	Not supported by the framework.

7.5 Capturing Model Information Not Used in the Framework

When transferring *ProModel*[®] or *QUEST*[®] models into the framework, there are substantial software-specific data that cannot be handled by the framework. To record the transferring process and provide users with important messages, a log file is incorporated. Figure 7.2 shows a portion of the log file that illustrates the transfer of one of ProModel's demonstration models (distribution.mod) to the framework. The *ProModel*[®] listing of the model contents is provided in Appendix D. A full example log file is included in Appendix E. To avoid the loss of data during the transferring process, in addition to the log file, two XML files are created: one that contains the data supported by the framework (referenced to as the "interpreted" XML file) and one that contains data that are not supported by the framework (referred to as the "uninterpreted" XML file).



```
log.txt - Notepad
File Edit Format View Help
Extracting location :chicago_warehouse
  dumping LocStats...
  dumping LocIncoming...
  dumping LocSelectUnit...
End extracting location.

Extracting location :boise_warehouse
  dumping LocStats...
  dumping LocIncoming...
  dumping LocSelectUnit...
End extracting location.

Extracting operation in minneapolis_customer for ALL
  Processing user-defined logics. Manually translation may be required.
End extracting operation.

Extracting operation in boston_customer for ALL
  Processing user-defined logics. Manually translation may be required.
End extracting operation.

Extracting operation in order_E2 for consumption_order
End extracting operation.

Extracting resource Truck
End extracting resource.
```

Figure 7.2 Log File

In Figure 7.2, the second line “dumping LocStats...” means extracting the non-supported data, location statistic, into the uninterpreted XML file. Figure 7.3 includes a portion of the uninterpreted XML file for the *ProModel*[®] distribution model. The entire uninterpreted XML file is provided in Appendix F.

```

- <Location>
  <Name>chicago_warehouse</Name>
  - <NotSupported>
    <LocStats>3</LocStats>
    <LocIncoming>1</LocIncoming>
    <LocSelectUnit>0</LocSelectUnit>
  </NotSupported>
</Location>
- <Location>
  <Name>boise_warehouse</Name>
  - <NotSupported>
    <LocStats>3</LocStats>
    <LocIncoming>1</LocIncoming>
    <LocSelectUnit>0</LocSelectUnit>
  </NotSupported>
</Location>
- <Entity>
  <Name>product_1</Name>
  - <NotSupported>
    <EntStats>3</EntStats>
  </NotSupported>
</Entity>

```

Figure 7.3 Portion of the Uninterpreted XML File for ProModel Distribution Model

In Figure 7.2, the message “Processing user-defined logics. Manually translation may be required” warns the user that the ProModel Controller that parses the model may not extract the user logic correctly. Users will likely need to manually interpret those segments of the code. Figure 7.4 is a portion of interpreted XML file for the example *ProModel*[®] distribution model. Note the ProModel Controller that parses the framework only picks up the duration part of the logic and leaves the unrecognized code in the UserLogic node. As mentioned earlier, the ProModel Controller only recognizes a limited number of key words.

```

- <Operation>
  <OpLocation>minneapolis_customer</OpLocation>
  <InEntity>ALL</InEntity>
  <OID>minneapolis_customer_ALL</OID>
- <Process>
  <Duration>u(1,.5)*24 hr</Duration>
  <DynamicResource />
  <UserLogic>graphic 3join 1 consumption_order</UserLogic>
</Process>
  <OutEntity>ALL</OutEntity>
  <Note />
</Operation>
- <Operation>
  <OpLocation>boston_customer</OpLocation>
  <InEntity>ALL</InEntity>
  <OID>boston_customer_ALL</OID>
- <Process>
  <Duration>u(1,.5)*24 hr</Duration>
  <DynamicResource />
  <UserLogic>graphic 3join 1 consumption_order</UserLogic>
</Process>
  <OutEntity>ALL</OutEntity>
  <Note />
</Operation>
- <Operation>
  <OpLocation>order_E2</OpLocation>
  <InEntity>consumption_order</InEntity>

```

Figure 7.4 Portion of the Interpreted XML File for ProModel Distribution Model

Likewise, when transferring from *QUEST*[®] to the framework. Three files will be created: one log file, one interpreted XML file, and one uninterpreted XML file. The log file and the interpreted XML file are very similar to that of *ProModel*[®]. Figure 7.5 is an example of an uninterpreted XML file for a *QUEST*[®] model. There are two parts to the XML file: the structured part and the unstructured part. The structured part stores the data that is recognized as a portion of the model elements but cannot be handled by the framework. For example, in Figure 7.5 the priority data are recognized, but the data are not handled by the framework. The unstructured part of the uninterpreted XML file stores the data that is not recognized. According to *QUEST*[®] Customer Service, the unstructured part is for *QUEST*[®] development use only [57]. A full uninterpreted *QUEST*[®] XML file can be found in Appendix G.

When transferring data from the framework to *ProModel*[®] or *QUEST*[®] environments, some data will not transfer properly. A log file will also be created to store the data transferring history; it is very similar to the one illustrated in Figure 7.2.



```
Unable.xml - Notepad
File Edit Format View Help
<UnableToHandle>
  <Entity>
    <Name>Item</Name>
    <NotSupported>
      Item NUM_DISPLAY 1
      PRIORITY 1
      PART_HISTORY 0
      DEVICE_CREATION_MODE 1
      DISPLAY_INDEX 0
      ...

      PCLASS_LBR_REQMT 'NO_LABOR'
      PCLASS_SR_REQMT 'NO_SR'
    </NotSupported>
  </Entity>
  <Operation>
    <OID>Process1All</OID>
    <NotSupported>
      PRIORITY 1
      REJECTION_RATE 0.000000
      ...

    </NotSupported>
  </Operation>
  ...

  <Unrecognized>
    MAGIC_NO
    1125972047
    MDL_VERSION 5
    SCL_FUNC SUB_set_get_part_dest 411
    ...

    LC1 1
    1 0 0 0
    0 1 0 0
    0 0 1 0
    0 0 0 1
    1

    10 327684 262145
    -4 1
    ...

  </Unrecognized>
</UnableToHandle>
```

Figure 7.5 Portion of the Uninterpreted XML File for a QUEST Model

CHAPTER VIII

CONCLUSIONS AND RECOMMENDATIONS

Discrete-event simulation plays a growing role in the design, analysis and management of modern enterprises. However, one of the major barriers to further application is the lack of interoperability of simulation models. There is a clear need for a means for better interaction between humans and simulation models, and between models themselves.

In order to address the simulation model interoperability problem, interactions are identified, defined, and analyzed along three dimensions: “open” versus “closed”, “formulation” versus “application”, and the type of interaction relationship. Interaction relationships between models are defined to be of the following types: one-to-one, many-to-one, one-to-many, parallel, and replacement integration. In addition to the types of interactions, this research defines possible interaction approaches that can occur in both formulation and application stages. The interaction approaches in the application stage includes: link through individual observation, link through distribution, common structure/common application, and distributed simulation. It is concluded that common structure/common application and distributed simulation approaches are able to handle all types of model interaction relationships. There are three opportunities for interactions between humans and simulation models: visual means, common model representation,

and commercial simulation software package. After reviewing the existing graphical modeling approaches and common model representations, it is concluded that only the common structure/common application approach is able to facilitate model interactions at both the formulation and application stages.

One of the main contributions of this research is a means for simulation models to interact that is based on the common structure/common application approach. The proposed approach includes a set of theoretical foundational components and components that enable the approach to be implemented in software. A very basic building block of the proposed approach is the definition of common simulation elements and their relationships, this step is done by combining elements from manufacturing and simulation (i.e., *ProModel*[®] and *QUEST*[®]). The resulting common simulation elements lead to a structure that represents the elements and relationships as well as a set of standard visual objects that facilitate use and understanding among more stakeholders.

The major components of the software implementations are the graphical modeling tool (GMT) and common user interface (CUI). The GMT facilitates the development and communication of the conceptual modeling phase. The CUI allows users to develop and modify simulation models in a simulation package neutral environment. It also contains controllers for interacting with commercial simulation software packages that enables the translation between the simulation packages and common data representations.

The capabilities of the proposed approach were tested on various simulation models. This not only illustrates the usefulness of the proposed framework and the

software implementations, but also identifies their limitations. It is shown that the proposed framework is compatible with simulation packages (i.e., *ProModel*[®] and *QUEST*[®]). The compatibility is illustrated through a series of tables that maps the capability of the framework to simulation packages and via versa. Also, methods are developed to capture model information that is not used in the framework.

The proposed approach serves as a bridge between stakeholders with varying levels of simulation expertise, and thus increases simulation interoperability at both formulation and application stages. The proposed approach that provides a solid foundation to simulation model interoperability is a prototype for future research and development. Improvement in the approach's capabilities can be made in three areas: framework, GMT, and CUI.

- The proposed framework can be extended to include:
 - more simulation elements, such as: work shifts, dynamic resource move/travel time, multiple downtimes and repair logic.
 - more routing and operation options, such as: join, combine, and send.
 - a simple programming language to describe the user-defined logic.
- The GMT can be enhanced by:
 - displaying more information in the Visio drawing view or allowing users to choose the information to be displayed.
 - providing a plain text summary of the model in addition to XML files; Extensible Stylesheet Language (i.e., XSL) can be used to provide the translation.

- integrating the GMT with the CUI and providing more controls and a more user-friendly interface.
 - improving the error proofing in GMT. For example, the GMT should generate an error message when two linkages have the same start and end locations.
 - improving the transferring capability from XML files to Visio drawings. The line representing linkages only contains the start and end positions. As a result, the line representing linkages in the Visio file may change positions after restoring XML files based on the location of other objects.
 - improving the display of information in Visio file when a simulation package model is input. When XML files are generated from a simulation package, the position information is likely not available and thus cannot be read by the GMT. It is possible to solve this problem by using relative positions. For example, entities are always placed at the top left corner of a drawing, even if the position information of entities is missing in the XML file, the entities can still be placed within a specific distance from the top left corner.
- The CUI can be improved by:
 - integrating CUI with a DSS to facilitate experimental design and statistical data collection. Performance measurement is a critical part to simulation.

- incorporating a simple standard syntax dictionary or lookup table. For example, in *ProModel*[®], the uniform distribution function is represented as U(mean, half range) while in *QUEST*[®], it is represented as U(min, max) in *QUEST*[®] and in *Arena*[®].
- improving error proofing, as in the GMT. For example, the duplicate names can occur in both the GMT and CUI.
- providing more functionality in the CUI. For example, including a report generation function that allows tracking of the flow of entities in a model is useful.

REFERENCES

- [1] Integrated Manufacturing Technology (IMTR) Roadmapping Modeling and Simulation Workshop Group and IMTR Roadmapping Project Team, 1998, "IMTR Roadmap for Modeling and Simulation," IMTR Project Office, Oak Ridge Centers for Manufacturing Technologies, Oak Ridge, TN; in McLean, Charles, R. "Manufacturing Simulation and Visualization" program status report, NIST. <http://www.mel.nist.gov/proj/msv.htm>. (Accessed January 3, 2004)
- [2] C. McLean and S. Leong, "The Expanding Role of Simulation in Future Manufacturing," *Proceedings of the 2001 Winter Simulation Conference*, Vol. 7, 2001, pp. 1478-1486.
- [3] J.P. Shim, M. Warkentin, J.F. Courtney, D.J. Power, R.Sharda, and C.Carlesson, "Past, Present, and Future of Decision Support Technology," *Decision Support Systems*, Vol. 33, 2002, pp. 111-126.
- [4] D.A. Sadowski and M.R. Grabau, "Tips for Successful Practice of Simulation," *Proceedings of the 1999 Winter Simulation Conference*, 1999, pp. 60-66.
- [5] R. Blanning, "Model Management Systems: An Overview," Owen Graduate School of Management, Vanderbilt University, Working Paper No. 89-23, November 1989.
- [6] J. Banks, J. Carson, B.L. Nelson, and D. Nicol, *Discrete-Event System Simulation*, Prentice Hall, 4th Edition, 2004.
- [7] A.M. Law, and W.D. Kelton, *Simulation Modeling and Analysis*, New York, McGraw-Hill, 2000.
- [8] R.E. Shannon, *Systems Simulation the Art and Science*, New Jersey, Prentice-Hall, 1975.
- [9] C.R. Harrell, B.K. Ghosh, and R.O. Bowden, *Simulation Using ProModel*, Boston, McGraw-Hill Company, July 2003.
- [10] A. Geoffrion, "An Introduction to Structured Modeling," *Management Science*, Vol. 33, No. 5, May 1987, pp. 547-588.
- [11] S. Moorthy, "Integrating The CAD Model with Dynamic Simulation: Simulation Data Exchange," *Proceeding of the 1999 Winter Simulation Conference*, 1999, pp. 276-280.

- [12] D. Sly and S. Moorthy, "Simulation Data Exchange (SDX) Implementation and Use," *Proceedings of the 2001 Winter Simulation Conference*, 2001, pp.1473-1477.
- [13] M. Overstreet and R. Nance, "A specification language to assist in analysis of discrete event simulation models," *Communications of the ACM*, 1985, pp.190-201.
- [14] J. Heim, "Integrating Distributed Simulation Objects," *Proceedings of the 1997 Winter Simulation Conference*, 1997, pp. 532-538.
- [15] C. McLean and F. Riddick, "The IMS Mission Architecture for Distributed Manufacturing Simulation," *Proceedings of the 2000 Winter Simulation Conference*, 2000, pp. 1539-1548.
- [16] J.S. Dahmann, R.M. Fujimoto, and R.M. Weatherly, "The Department of Defense High Level Architecture," *Proceedings of the 1997 Winter Simulation Conference*, 1997, pp. 142-149.
- [17] W.J. Davis and G.L. Moeller, "The High Level Architecture: Is There A Better Way?" *Proceedings of the 1999 Winter Simulation Conference*, 1999, pp. 1595-1601.
- [18] R.E. Nance, "The Conical Methodology and the Evolution of Simulation Model Development," *Annals of Operations Research*, Vol. 53, 1994, pp. 1-45.
- [19] J. Ma, Q. Tian, and D. Zhou, "An Object-Oriented Approach to Structured Modeling," *1998 Information Resources Management Association International Conference*, May 1998, pp. 406-413.
- [20] M.L. Lenard, "A Prototype Implementation of a Model Management System for Discrete-Event Simulation Models," *Proceedings of the 1993 Winter Simulation Conference*, 1993, pp. 560-568.
- [21] A. Pritsker, *Modeling and Analysis Using Q-Gert Networks*, New York, John Wiley & Sons, Inc., 1977.
- [22] A. Pritsker, *Introduction to Simulation and SLAM II*, New York, John Wiley & Sons, Inc., 1995.
- [23] T.J. Schriber, *An Introduction to Simulation Using GPSS/H*, New York, John Wiley & Sons, 1991.

- [24] L. Schruben, and E. Yucesan, "Complexity of Simulation Models: a Graph Theoretic Approach," *Proceedings of the 1993 Winter Simulation Conference*, 1993, pp. 641-649.
- [25] R.C. Crain, "Simulation With GPSS/H," *Proceedings of the 1998 Winter Simulation Conference*, 1998, pp. 235-240.
- [26] A.E. Rizzoli, J.R. Davis, and D.J. Abel, "Model and data integration and re-use in environmental decision support systems," *Decision Support Systems*, Vol. 24, 1998, pp. 127-144.
- [27] A. Chang, C.W. Holsapple, and A.B. Whinston, "Model management issues and directions," *Decision Support Systems*, Vol. 9, 1993, pp. 19-37.
- [28] J.M. Pearson and J.P. Shim, "An empirical investigation into DSS structures and environments," *Decision Support Systems*, Vol. 13, 1995, pp. 141-158.
- [29] M.J. Ginzberg and E.A. Stohr, "Decision support systems: issues and perspectives," In Ginzberg, M.J., Reitman, W., and Stohr, E.A., eds. *Decision Support Systems*, Amsterdam: North-Holland, 1982, pp. 9-31.
- [30] M.L. Lenard, "An Object-oriented approach to model management," *Decision Support Systems*, Vol.9, 1993, pp. 67-73.
- [31] A.M. Geoffrion, "Structured Modeling: Survey and Future Research Directions," <http://www.anderson.ucla.edu/faculty/art.geoffrion/home/csts/index.htm>, June 1, 1999. (Accessed May 15, 2002)
- [32] E.J. Derrick, "Conceptual Frameworks for Discrete Event Simulation Models," M.S. Thesis, Dept. of Computer Science, Virginia Tech, Blacksburg, VA, Aug. 1988.
- [33] G.K. Yeo, and G. Li, "On Discrete Time Modeling," *IASTED International Conference on Modeling, Simulation, and Optimization*, Gold Coast, Australia, May 1996.
- [34] M.L. Lenard, "Extending the Structured Modeling Framework for Discrete-Event Simulation," *Proceeding of the 26th Hawaii International Conference on System Sciences*, Vol. 3, Jan. 1992, pp. 494-503.
- [35] R.F. Lu, G. Qiao, and C. McLean, "NIST XML Simulation Interface Specification at Boeing: a Case Study," *Proceedings of the 2003 Winter Simulation Conference*, 2003, pp. 1230-1237.

- [36] Y.T. Lee, C. McLean, and G. Shao, "A Neutral Information Model for Simulating Machine Shop Operations," *Proceedings of the 2003 Winter Simulation Conference*, 2003, pp. 1296-1304.
- [37] C. McLean, A. Jones, T. Lee, and F. Riddick, "An Architecture for a Generic Data-Driven Machine Shop Simulator," *Proceedings of the 2002 Winter Simulation Conference*, 2002, pp. 1108-1116.
- [38] McLean, C., Lee, T., Shao, G., Riddick, F., and Leong, S., "Shop data model and interface specification," Draft, Manufacturing Systems Integration Division, Manufacturing Engineering Laboratory, National Institute of Standards and Technology, Revision Feb. 24, 2003, from Simulation Standards Consortium Kickoff Meeting.
- [39] G. Qiao and F. Riddick, "Modeling Information for Manufacturing-Oriented Supply-Chain Simulations," *Proceedings of the 2004 Winter Simulation Conference*, 2004, pp. 1184-1188.
- [40] P.P. Chen, "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Trans. on Database Systems*, Vol. 1, No. 1, March 1976, pp. 1-36.
- [41] D.R. Dolk, "Data as Models: An Approach to Implementing Model Management," *Decision Support Systems*, Vol. 2, 1986, pp. 73-80.
- [42] O.B. Kwon and S.J. Park, "RMT, A modeling support system for model reuse," *Decision Support Systems*, Vol. 16, 1996, pp. 131-153.
- [43] D.P. Bischak and S.D. Roberts, "Object-Oriented Simulation," *Proceedings of the 1991 Winter Simulation Conference*, 1993, pp. 194-203.
- [44] B.P. Zeigler, "Hierarchical, Modular Discrete-Event Modeling in an Object-Oriented Environment," *Simulation*, Vol. 50, pp. 219-230.
- [45] W.A. Muhanna., "An object-oriented framework for model management and DSS development," *Decision Support Systems*, Vol. 9, 1993, pp. 217-229.
- [46] A.M. Law, "Simulation of Manufacturing Systems," *Proceedings of the 1999 Winter Simulation Conference*, 1999, pp. 56-59.
- [47] A. Bartolotta, C. McLean, and Y.T. Lee, "Production Systems Engineering: Requirements Analysis for Discrete-Event Simulation," NISTIR 6154, National Institute of Standards and Technology, Gaithersburg, MD, 1998.

- [48] R.N. Price, and C.R. Harrell, "Simulation Modeling and Optimization Using ProModel," *Proceedings of the 1999 Winter Simulation Conference*, 1999, pp. 208-214.
- [49] M.R. Barnes, "An Introduction to QUEST," *Proceedings of the 1997 Winter Simulation Conference*, 1997, pp.619-623.
- [50] DELMIA Corporation, *QUEST User Manual*, 2002.
- [51] H. Kim, "An XML-based modeling language for the open interchange of decision models," *Decision Support Systems*, Vol. 31, No. 4, October 2001, pp. 429-441.
- [52] L. Quin, "Extensible Markup Language (XML)," <http://www.w3.org/XML>, 2003 (Accessed July 19, 2005) .
- [53] PROMODEL Corporation, *ProModel ActiveX User Guide*, 2003.
- [54] DELMIA Corporation, *QUEST Batch Control Language Reference*, 2002.
- [55] DELMIA Corporation, *QUEST Simulation Control Language Reference Manual*, 2002.
- [56] D. Kelton, R. Sadowski, and D. Sadowski, *Simulation with Arena*, Boston, McGraw-Hill, 2003.
- [57] Personal email, September 12, 2005, from DELMIA Customer Service Department.

APPENDIX A
STRUCTURAL MODELING SCHEMA


```

&GeneralInfo
  GeneralInfo /pe/
  ModelName (GeneralInfo) /a/ : text
  TimeUnit (GeneralInfo) /a/ : text
  DistanceUnit (GeneralInfo) /a/ : text
  Date (GeneralInfo) /a/ : text
  Builder (GeneralInfo) /a/ : text
  Note (GeneralInfo) /a/ : text

&Entity
  Entityh /pe/
  Name (Entityh) /a/ : text
  Speed (Entityh) /a/ : real+
  Attributea (Entityh) /ce/
    ID (Attributea) /a/ : text
    Type (Attributea) /a/ : text
    Note (Attributea) /a/ : text
  Note (Entityh) /a/ : text

&Static_Resource
  Static_Resourcei /pe/
  Name (Static_Resourcei) /a/ : text
  Type (Static_Resourcei) /a/ : text
  Capacity (Static_Resourcei) /a/ : I+
  Units (Static_Resourcei) /a/ : I+
  Downtime (Static_Resourcei) /ce/
    TBF (Downtime, Static_Resourcei) /a/ : text
  Repair (Static_Resourcei) /ce/
    TTR (Repair, Static_Resourcei) /a/ : text
    RepairResource (Repair, Static_Resourcei) /a/ : text
  SequenceRule (Static_Resourcei) /ce/
    SeqRule (SequenceRule, Static_Resourcei) /a/ : text
    AeqAttribute (SequenceRule, Static_Resourcei) /a/ : text
  Attributea (Static_Resourcei) /ce/
    ID (Attributea) /a/ : text
    Type (Attributea) /a/ : text
    Note (Attributea) /a/ : text

&Dynamic_Resource
  Dynamic_Resourcej /pe/
  Name (Dynamic_Resourcej) /a/ : text
  Units (Dynamic_Resourcej) /a/ : I+
  Downtime (Dynamic_Resourcej) /ce/
    TBF (Downtime, Dynamic_Resourcej) /a/ : text
  Repair (Dynamic_Resourcej) /ce/
    TTR (Repair, Dynamic_Resourcej) /a/ : text
    RepairResource (Repair, Dynamic_Resourcej) /a/ : text
  Notes (Dynamic_Resourcej) /a/ : text

&Operation
  Operationk (Static_Resourcei, Entityh) /ce/
  OpLocation (Operationk) /a/ : text
  InEntity (Operationk) /a/ : text
  Processp (Operationk) /ce/
    Duration (Processp) /a/ : text
    DynamicResource (Processp) /a/ : text
    UserLogic (Processp) /a/ : text
  OutEntity (Operationk) /a/ : text
  Note (Operationk) /a/ : text

&Linkage
  Linkagel (Static_Resourcei, Static_Resourcej) /ce/
  LID (Linkagel) /a/ : text
  BeginLoc (Linkagel) /a/ : text
  EndLoc (Linkagel) /a/ : text
  Distance (Linkagel) /a/ : text

```

Note (Linkage_i) /a/ : text

&Routing

Routing_r (Entity_h, Linkage_i) /ce/
 RID (Routing_r) /a/ : text
 Link_n (Linkage_i) /ce/
 LID (Link_n) /a/ : text
 Duration (Link_n) /a/ : text
 DynamicResource (Link_n) : text
 UserLogic (Link_n) : text
 RouteEntity (Routing_r) /a/ : text
 Rule_u (Routing_r) /ce/
 RoutingRule (Rule_u) /a/ : text
 Rlink_d (Rule_u) /ce/
 LID (RLink_d) /a/ : text
 Percentage (RLink_d) /a/ : text
 QtyEach (Routing_r) /a/ : text
 Note (Routing_r) /a/ : text

&Arrival

Arrival_a (Entity_h, Static_Resource_i) /ce/
 ArvEntity (Arrival_a) /a/ : text
 ArvLocation (Arrival_a) /a/ : text
 QtyEach (Arrival_a) /a/ : text
 Frequency (Arrival_a) /a/ : text
 UserLogic (Arrival_a) /a/ : text
 FirstTime (Arrival_a) /a/ : text
 Occurrence (Arrival_a) /a/ : text
 Note (Arrival_a) /a/ : text

APPENDIX B

XML DTD

```

<!DOCTYPE Model [
    <!--          ****      Model Definition      ****          -->
    <!ELEMENT Model (GeneralInfo, Entity*, Static_Resource*, Dynamic_Resource, Operation*,
        Linkage*, Route*, Arrival*)>
    <!ELEMENT GeneralInfo (ModelName?, TimeUnit?, DistanceUnit?, Date?, Builder?, Note?)>
    <!ELEMENT ModelName (#PCDATA)>
    <!ELEMENT TimeUnit (#PCDATA)>
    <!ELEMENT DistanceUnit (#PCDATA)>
    <!ELEMENT Date (#PCDATA)>
    <!ELEMENT Builder (#PCDATA)>

    <!--          ****      General Definition      ****          -->
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Units (#PCDATA)>
    <!ELEMENT DownTime (TBF)>
    <!ELEMENT TBF(#PCDATE)>
    <!ELEMENT Repair (TTR, RepairResource?)>
    <!ELEMENT TTR(#PCDATA)>
    <!ELEMENT RepairResource(#PCDATA)>
    <!ELEMENT Icon (IconType, XPosition?, YPosition?, Width, Height)>
    <!ELEMENT IconType (#PCDATA)>
    <!ELEMENT XPosition (#PCDATA)>
    <!ELEMENT YPosition (#PCDATA)>
    <!ELEMENT Width (#PCDATA)>
    <!ELEMENT Height (#PCDATA)>
    <!ELEMENT Stats (#PCDATA)>
    <!ELEMENT Note (#PCDATA)>
    <!ELEMENT Attribute (ID, Type, Note?)>
    <!ELEMENT ID (#PCDATA)>
    <!ELEMENT Type (#PCDATA)>
    <!ELEMENT Speed (#PCDATA)>
    <!ELEMENT QtyEach (#PCDATA)>
    <!ELEMENT Duration(#PCDATA)>
    <!ELEMENT DynamicResource(#PCDATA)>
    <!ELEMENT UserLogic(#PCDATA)>
    <!ELEMENT QtyEach (#PCDATA)>

    <!--          ****      Static_Resource Definition      ****          -->
    <!ELEMENT Static_Resource (Name, Type, Capacity, Units, Icon?, DownTime?, Repair?,
        SequenceRule?, Attribute*, Note?)>
    <!ELEMENT Capacity (#PCDATA)>
    <!ELEMENT SequenceRule (SeqRule, SeqAttribute?)>
    <!ELEMENT SeqRule(#PCDATA)>
    <!ELEMENT SeqAttribute(#PCDATA)>

    <!--          ****      Entity Definition      ****          -->
    <!ELEMENT Entity (Name, Speed?, Icon?, Attribute*, Note?)>
    <!ELEMENT Speed(#PCDATA)>

    <!--          ****      Dynamic_Resource Definition      ****          -->
    <!ELEMENT Dynamic_Resource (Name, Units, DownTime?, Repair?, Icon?, Note?)>

    <!--          ****      Operation definition      ****          -->
    <!ELEMENT Operation (OID, OpLocation, InEntity, Process+, OutEntity, Note?)>
    <!ELEMENT OID(#PCDATA)>
    <!ELEMENT OpLocation (#PCDATA)>
    <!ELEMENT InEntity (#PCDATA)>
    <!ELEMENT Process (Duration, DynamicResource, UserLogic)>
    <!ELEMENT OutEntity (#PCDATA)>

    <!--          ****      Linkage definition      ****          -->
    <!ELEMENT Linkage (LID, BeginLoc, EndLoc, Distance?, Note?)>
    <!ELEMENT LID (#PCDATA)>
    <!ELEMENT BeginLoc (#PCDATA)>
    <!ELEMENT EndLoc (#PCDATA)>

```

```

<!ELEMENT Distance (#PCDATA)>

<!--          ****      Routing definition      ****          -->
<!ELEMENT Routing (RID, Linkage+, RouteEntity, Rule, Icon, QtyEach, Note?)>
<!ELEMENT RID (#PCDATA)>
<!ELEMENT Linkage (LID, Duration, DynamicResource, UserLogic)>
<!ELEMENT RouteEntity (#PCDATA)>
<!ELEMENT Rule (RoutingRule, Linkage*)>
<!ELEMENT RoutingRule (#PCDATA)>
<!ELEMENT Linkage (LID, Percentage)>
<!ELEMENT Percentage (#PCDATA)>

<!--          ****      Arrival Definition      ****          -->
<!ELEMENT Arrival (AID, ArvEntity, ArvLocation, QtyEach, Frequency, UserLogic?,
    FirstTime?, Occurence?, Note?)>
<!ELEMENT AID (#PCDATA)>
<!ELEMENT ArvEntity (#PCDATA)>
<!ELEMENT ArvLocation (#PCDATA)>
<!ELEMENT Frequency (#PCDATA)>
<!ELEMENT FirstTime (#PCDATA)>
<!ELEMENT Occurence (#PCDATA)>

```

⌋>

APPENDIX C
A SAMPLE XML FILE

```

- <Model>
  - <GeneralInfo>
    <ModelName>Exercise 5.4</ModelName>
    <TimeUnit> hour</TimeUnit>
    <DistanceUnit> Feet</DistanceUnit>
    <Date>8/15/2005</Date>
    <Builder>By Tai-Chi Wu</Builder>
    <Note />
  </GeneralInfo>
  - <Entity>
    <Name>Part1</Name>
    <Speed />
    <Note />
    - <Icon>
      <IconType>Entity</IconType>
      <XPosition>1.2500 in.</XPosition>
      <YPosition>6.6250 in.</YPosition>
      <Width>1.0000 in.</Width>
      <Height>0.7500 in.</Height>
    </Icon>
  </Entity>
  - <Entity>
    <Name>Part2</Name>
    <Speed />
    <Note />
    - <Icon>
      <IconType>Entity</IconType>
      <XPosition>2.7500 in.</XPosition>
      <YPosition>6.6250 in.</YPosition>
      <Width>1.0000 in.</Width>
      <Height>0.7500 in.</Height>
    </Icon>
  </Entity>
  - <Static_Resource>
    <Name>Part1_Buff</Name>
    <Type>Buff</Type>
    <Capacity>inf</Capacity>
    <Units>1</Units>
    - <DownTime>
      <MTBF />
    </DownTime>
    - <Repair>
      <MTTR />
      <RepairResource />
    </Repair>
    - <SequenceRule>
      <SeqRule>FIFO</SeqRule>
      <SeqAttribute />
    </SequenceRule>
    <Note />
    - <Icon>
      <IconType>Static_Resource</IconType>
      <XPosition>2.5000 in.</XPosition>
      <YPosition>4.8750 in.</YPosition>
      <Width>1.0000 in.</Width>
      <Height>0.7500 in.</Height>
    </Icon>
  </Static_Resource>

```

```

- <Static_Resource>
  <Name>Part2_Buff</Name>
  <Type>Buff</Type>
  <Capacity>inf</Capacity>
  <Units>1</Units>
  - <DownTime>
    <MTBF />
  </DownTime>
  - <Repair>
    <MTTR />
    <RepairResource />
  </Repair>
  - <SequenceRule>
    <SeqRule>FIFO</SeqRule>
    <SeqAttribute />
  </SequenceRule>
  <Note />
  - <Icon>
    <IconType>Static_Resource</IconType>
    <XPosition>2.5000 in.</XPosition>
    <YPosition>3.6250 in.</YPosition>
    <Width>1.0000 in.</Width>
    <Height>0.7500 in.</Height>
  </Icon>
</Static_Resource>
- <Static_Resource>
  <Name>Process1</Name>
  <Type>ProcessingUnit</Type>
  <Capacity>1</Capacity>
  <Units>1</Units>
  - <DownTime>
    <MTBF />
  </DownTime>
  - <Repair>
    <MTTR />
    <RepairResource />
  </Repair>
  - <SequenceRule>
    <SeqRule>FIFO</SeqRule>
    <SeqAttribute />
  </SequenceRule>
  <Note />
  - <Icon>
    <IconType>Static_Resource</IconType>
    <XPosition>4.5000 in.</XPosition>
    <YPosition>4.2500 in.</YPosition>
    <Width>1.0000 in.</Width>
    <Height>0.7500 in.</Height>
  </Icon>
</Static_Resource>
- <Static_Resource>
  <Name>Process2_Buff</Name>
  <Type>Buff</Type>
  <Capacity>inf</Capacity>
  <Units>1</Units>
  - <DownTime>
    <MTBF />
  </DownTime>

```



```

- <Repair>
  <MTTR />
  <RepairResource />
</Repair>
- <SequenceRule>
  <SeqRule>FIFO</SeqRule>
  <SeqAttribute />
</SequenceRule>
<Note />
- <Icon>
  <IconType>Static_Resource</IconType>
  <XPosition>6.2500 in.</XPosition>
  <YPosition>4.2500 in.</YPosition>
  <Width>1.0000 in.</Width>
  <Height>0.7500 in.</Height>
</Icon>
</Static_Resource>
- <Static_Resource>
  <Name>Process2</Name>
  <Type>ProcessingUnit</Type>
  <Capacity>1</Capacity>
  <Units>1</Units>
- <DownTime>
  <MTBF />
</DownTime>
- <Repair>
  <MTTR />
  <RepairResource />
</Repair>
- <SequenceRule>
  <SeqRule>FIFO</SeqRule>
  <SeqAttribute />
</SequenceRule>
<Note />
- <Icon>
  <IconType>Static_Resource</IconType>
  <XPosition>7.7500 in.</XPosition>
  <YPosition>4.2500 in.</YPosition>
  <Width>1.0000 in.</Width>
  <Height>0.7500 in.</Height>
</Icon>
</Static_Resource>
- <Linkage>
  <LID>Link1</LID>
  <BeginLoc>Part1_Buff</BeginLoc>
  <EndLoc>Process1</EndLoc>
  <Distance />
  <Note />
- <Icon>
  <IconType>Linkage</IconType>
  <XPosition>3.5000 in.</XPosition>
  <YPosition>4.5625 in.</YPosition>
  <Width>1.0000 in.</Width>
  <Height>-0.6250 in.</Height>
  <BeginX>3.0000 in.</BeginX>
  <BeginY>4.8750 in.</BeginY>
  <EndX>4.0000 in.</EndX>
  <EndY>4.2500 in.</EndY>

```

```

    </Icon>
  </Linkage>
- <Linkage>
  <LID>Link2</LID>
  <BeginLoc>Part2_Buff</BeginLoc>
  <EndLoc>Process1</EndLoc>
  <Distance />
  <Note />
  - <Icon>
    <IconType>Linkage</IconType>
    <XPosition>3.5000 in.</XPosition>
    <YPosition>3.9375 in.</YPosition>
    <Width>1.0000 in.</Width>
    <Height>0.6250 in.</Height>
    <BeginX>3.0000 in.</BeginX>
    <BeginY>3.6250 in.</BeginY>
    <EndX>4.0000 in.</EndX>
    <EndY>4.2500 in.</EndY>
  </Icon>
</Linkage>
- <Linkage>
  <LID>Link3</LID>
  <BeginLoc>Process1</BeginLoc>
  <EndLoc>Process2_Buff</EndLoc>
  <Distance />
  <Note />
  - <Icon>
    <IconType>Linkage</IconType>
    <XPosition>5.3750 in.</XPosition>
    <YPosition>4.2500 in.</YPosition>
    <Width>0.7500 in.</Width>
    <Height>0.2500 in.</Height>
    <BeginX>5.0000 in.</BeginX>
    <BeginY>4.2500 in.</BeginY>
    <EndX>5.7500 in.</EndX>
    <EndY>4.2500 in.</EndY>
  </Icon>
</Linkage>
- <Linkage>
  <LID>Link4</LID>
  <BeginLoc>Process2_Buff</BeginLoc>
  <EndLoc>Process2</EndLoc>
  <Distance />
  <Note />
  - <Icon>
    <IconType>Linkage</IconType>
    <XPosition>7.0000 in.</XPosition>
    <YPosition>4.2500 in.</YPosition>
    <Width>0.5000 in.</Width>
    <Height>0.2500 in.</Height>
    <BeginX>6.7500 in.</BeginX>
    <BeginY>4.2500 in.</BeginY>
    <EndX>7.2500 in.</EndX>
    <EndY>4.2500 in.</EndY>
  </Icon>
</Linkage>
- <Linkage>
  <LID>Link5</LID>

```

```

    <BeginLoc>Process2</BeginLoc>
    <EndLoc>Exit</EndLoc>
    <Distance />
    <Note />
  - <Icon>
    - <IconType>Linkage</IconType>
      <XPosition>8.7500 in.</XPosition>
      <YPosition>4.2500 in.</YPosition>
      <Width>1.0000 in.</Width>
      <Height>0.2500 in.</Height>
      <BeginX>8.2500 in.</BeginX>
      <BeginY>4.2500 in.</BeginY>
      <EndX>9.2500 in.</EndX>
      <EndY>4.2500 in.</EndY>
    </Icon>
  </Linkage>
- <Arrival>
  <AID>Part1 : Part1_Buff</AID>
  <ArvEntity>Part1</ArvEntity>
  <ArvLocation>Part1_Buff</ArvLocation>
  <QtyEach>1</QtyEach>
  <Frequency>lognormal(11.5,2.0)</Frequency>
  <FirstTime />
  <Occurence>inf</Occurence>
  - <Icon>
    <IconType>Arrival</IconType>
    <XPosition>1.2708 in.</XPosition>
    <YPosition>4.8906 in.</YPosition>
    <Width>0.9583 in.</Width>
    <Height>0.7188 in.</Height>
  </Icon>
  <UserLogic />
</Arrival>
- <Arrival>
  <AID>Part2 : Part2_Buff</AID>
  <ArvEntity>Part2</ArvEntity>
  <ArvLocation>Part2_Buff</ArvLocation>
  <QtyEach>1</QtyEach>
  <Frequency>exp(15)</Frequency>
  <FirstTime />
  <Occurence>inf</Occurence>
  - <Icon>
    <IconType>Arrival</IconType>
    <XPosition>1.2708 in.</XPosition>
    <YPosition>3.6406 in.</YPosition>
    <Width>0.9583 in.</Width>
    <Height>0.7188 in.</Height>
  </Icon>
  <UserLogic />
</Arrival>
- <Routing>
  - <Linkage>
    <LID>Link1</LID>
    <Duration />
    <DynamicResource />
    <UserLogic />
  </Linkage>
  <RID>R1</RID>

```

```

    <RouteEntity>Part1</RouteEntity>
  - <Rule>
    <RoutingRule>Next Free</RoutingRule>
  </Rule>
  <QtyEach>1</QtyEach>
  <Note />
  - <Icon>
    <IconType>Routing</IconType>
    <XPosition>3.1875 in.</XPosition>
    <YPosition>5.1875 in.</YPosition>
    <Width>0.3750 in.</Width>
    <Height>0.3750 in.</Height>
  </Icon>
</Routing>
- <Routing>
  - <Linkage>
    <LID>Link2</LID>
    <Duration />
    <DynamicResource />
    <UserLogic />
  </Linkage>
  <RID>R2</RID>
  <RouteEntity>Part2</RouteEntity>
  - <Rule>
    <RoutingRule>Next Free</RoutingRule>
  </Rule>
  <QtyEach>1</QtyEach>
  <Note />
  - <Icon>
    <IconType>Routing</IconType>
    <XPosition>3.1875 in.</XPosition>
    <YPosition>4.0000 in.</YPosition>
    <Width>0.3750 in.</Width>
    <Height>0.3750 in.</Height>
  </Icon>
</Routing>
- <Routing>
  - <Linkage>
    <LID>Link3</LID>
    <Duration />
    <DynamicResource />
    <UserLogic />
  </Linkage>
  <RID>R3</RID>
  <RouteEntity>All</RouteEntity>
  - <Rule>
    <RoutingRule>Next Free</RoutingRule>
  </Rule>
  <QtyEach>1</QtyEach>
  <Note />
  - <Icon>
    <IconType>Routing</IconType>
    <XPosition>5.1875 in.</XPosition>
    <YPosition>4.5625 in.</YPosition>
    <Width>0.3750 in.</Width>
    <Height>0.3750 in.</Height>
  </Icon>
</Routing>

```

```

- <Routing>
  - <Linkage>
    <LID>Link5</LID>
    <Duration />
    <DynamicResource />
    <UserLogic />
  </Linkage>
  <RID>R5</RID>
  <RouteEntity>All</RouteEntity>
  - <Rule>
    <RoutingRule>Next Free</RoutingRule>
  </Rule>
  <QtyEach>1</QtyEach>
  <Note />
  - <Icon>
    <IconType>Routing</IconType>
    <XPosition>8.4375 in.</XPosition>
    <YPosition>4.5625 in.</YPosition>
    <Width>0.3750 in.</Width>
    <Height>0.3750 in.</Height>
  </Icon>
</Routing>
- <Dynamic_Resource>
  <Name>Operator</Name>
  <Units>1</Units>
  - <DownTime>
    <MTBF />
  </DownTime>
  - <Repair>
    <MTTR />
    <RepairResource />
  </Repair>
  <Logic />
  <Note />
  - <Icon>
    <IconType>Dynamic_Resource</IconType>
    <XPosition>5.5000 in.</XPosition>
    <YPosition>6.6250 in.</YPosition>
    <Width>1.0000 in.</Width>
    <Height>0.7500 in.</Height>
  </Icon>
</Dynamic_Resource>
- <Operation>
  - <Process>
    <Duration>T(5,6,8)</Duration>
    <DynamicResource>Operator</DynamicResource>
    <UserLogic />
  </Process>
  <OID>Process1 : Part1</OID>
  <OpLocation>Process1</OpLocation>
  <InEntity>Part1</InEntity>
  <OutEntity>Part1</OutEntity>
  <Note />
  - <Icon>
    <IconType>Operation</IconType>
    <XPosition>4.4688 in.</XPosition>
    <YPosition>5.2500 in.</YPosition>
    <Width>0.9375 in.</Width>

```

```

        <Height>0.6250 in.</Height>
    </Icon>
</Operation>
- <Operation>
  - <Process>
    - <Duration>T(3,7,8)</Duration>
      <DynamicResource>Operator</DynamicResource>
      <UserLogic />
    </Process>
    <OID>Process1 : Part2</OID>
    <OpLocation>Process1</OpLocation>
    <InEntity>Part2</InEntity>
    <OutEntity>Part2</OutEntity>
    <Note />
  - <Icon>
    - <IconType>Operation</IconType>
      <XPosition>4.4688 in.</XPosition>
      <YPosition>3.3125 in.</YPosition>
      <Width>0.9375 in.</Width>
      <Height>0.6250 in.</Height>
    </Icon>
  </Operation>
- <Operation>
  - <Process>
    - <Duration>T(4,6,8)</Duration>
      <DynamicResource />
      <UserLogic />
    </Process>
    <OID>Process2 : All</OID>
    <OpLocation>Process2</OpLocation>
    <InEntity>All</InEntity>
    <OutEntity>All</OutEntity>
    <Note />
  - <Icon>
    - <IconType>Operation</IconType>
      <XPosition>7.7188 in.</XPosition>
      <YPosition>5.2500 in.</YPosition>
      <Width>0.9375 in.</Width>
      <Height>0.6250 in.</Height>
    </Icon>
  </Operation>
</Model>

```

APPENDIX D

ProModel[®] LISTING

```
*****
*
*           *
*   Formatted Listing of Model:           *
*   C:\Program Files\ProModel\Models\Demos\Distribu.mod   *
*           *
*****
```

```
Time Units:      Minutes
Distance Units:  Feet
Initialization Logic:  ACTIVATE daily_ordering ()
                   ANIMATE 100
                   VIEW "Full"
```

```
*****
*           Locations           *
*****
```

Name	Cap	Units	Stats	Rules	Cost
oklahomacity_production	500	1		Time Series Oldest,	
seattle_production	500	1		Time Series Oldest,	
sanfrancisco_customer	INF	1		Time Series Oldest,	
boston_customer	INF	1		Time Series Oldest,	
tampa_customer	INF	1		Time Series Oldest,	
minneapolis_customer	INF	1		Time Series Oldest,	
phoenix_customer	INF	1		Time Series Oldest,	
raleigh_warehouse	200	1		Time Series Oldest,	
stlouis_warehouse	200	1		Time Series Oldest,	
detroit_warehouse	200	1		Time Series Oldest,	
slc_warehouse	200	1		Time Series Oldest,	
dallas_warehouse	200	1		Time Series Oldest,	
neworleans_warehouse	200	1		Time Series Oldest,	
chicago_warehouse	200	1		Time Series Oldest,	
boise_warehouse	200	1		Time Series Oldest,	
albuquerque_warehouse	200	1		Time Series Oldest,	
order_E2	INFINITE	1		Time Series Oldest,	
order_E8	INFINITE	1		Time Series Oldest,	
order_E7	INFINITE	1		Time Series Oldest,	
order_E3	INFINITE	1		Time Series Oldest,	
order_E6	INFINITE	1		Time Series Oldest,	

```
*****
*           Entities           *
*****
```

Name	Speed (fpm)	Stats	Cost
product_1	2500	Time Series	
consumption_order	150	Time Series	

```
*****
*           Path Networks           *
*****
```

Name	Type	T/S	From	To	BI	Dist/Time	Speed	Factor
Net1	Passing	Speed & Distance	N1	N2	Bi	843317.06	1	
			N2	N3	Bi	1611816.97	1	
			N3	N4	Bi	2072924.43	1	
			N4	N5	Bi	2453348.13	1	
			N5	N1	Bi	4383024.73	1	
			N4	N6	Bi	2827068.80	1	
			N5	N6	Bi	4622741.79	1	


```

N6 N7 Bi 2058540.23 1
N7 N8 Bi 2734545.80 1
N8 N9 Bi 894472.82 1
N8 N10 Bi 2745685.46 1
N10 N11 Bi 1623467.49 1
N11 N12 Bi 1290302.66 1
N11 N13 Bi 2235946.07 1
N13 N14 Bi 3909327.87 1
N14 N15 Bi 3486195.55 1
N15 N16 Bi 4055509.58 1
N16 N17 Bi 3150838.72 1
N17 N10 Bi 3655533.90 1
N9 N17 Bi 3512620.76 1
N4 N12 Bi 7594014.10 1

```

```

*****
*                               Interfaces                               *
*****

```

```

Net  Node  Location
-----
Net1 N1      seattle_production
      N5      sanfrancisco_customer
      N3      boise_warehouse
      N4      slc_warehouse
      N6      phoenix_customer
      N7      albuquerque_warehouse
      N8      oklahomacity_production
      N9      dallas_warehouse
      N17     neworleans_warehouse
      N10     stlouis_warehouse
      N11     chicago_warehouse
      N12     minneapolis_customer
      N13     detroit_warehouse
      N14     boston_customer
      N15     raleigh_warehouse
      N16     tampa_customer

```

```

*****
*                               Resources                               *
*****

```

```

Res  Ent
Name Units Stats Search Search Path Motion Cost
-----
Truck 50 By Unit Closest Oldest Net1 Empty: U(3260,1200) fpm
Home: N1 Full: U(2640,800) fpm

```

```

*****
*                               Processing                               *
*****

```

```

Process Routing
Entity Location Operation Blk Output Destination Rule Move Logic
-----
ALL seattle_production WAIT 24*3 hr 1 ALL boise_warehouse SEND 1 MOVE WITH Truck
THEN FREE
ALL slc_warehouse SEND MOVE WITH Truck THEN FREE
ALL albuquerque_warehouse SEND MOVE WITH Truck THEN FREE
ALL chicago_warehouse SEND MOVE WITH Truck THEN FREE
ALL stlouis_warehouse SEND MOVE WITH Truck THEN FREE
ALL dallas_warehouse SEND MOVE WITH Truck THEN FREE

```

		ALL	detroit_warehouse	SEND	MOVE WITH Truck THEN FREE
		ALL	raleigh_warehouse	SEND	MOVE WITH Truck THEN FREE
		ALL	neworleans_warehouse	SEND	MOVE WITH Truck THEN FREE
ALL THEN FREE	oklahomacity_production	WAIT 24*3 hr	1 ALL	dallas_warehouse	SEND 1 MOVE WITH Truck
		ALL	albuquerque_warehouse	SEND	MOVE WITH Truck THEN FREE
		ALL	slc_warehouse	SEND	MOVE WITH Truck THEN FREE
		ALL	boise_warehouse	SEND	MOVE WITH Truck THEN FREE
		ALL	stlouis_warehouse	SEND	MOVE WITH Truck THEN FREE
		ALL	chicago_warehouse	SEND	MOVE WITH Truck THEN FREE
		ALL	detroit_warehouse	SEND	MOVE WITH Truck THEN FREE
		ALL	raleigh_warehouse	SEND	MOVE WITH Truck THEN FREE
ALL THEN FREE	raleigh_warehouse	GRAPHIC 2	1 ALL	sanfrancisco_customer	SEND 1 MOVE WITH Truck
		ALL	phoenix_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	minneapolis_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	tampa_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	boston_customer	SEND	MOVE WITH Truck THEN FREE
ALL THEN FREE	stlouis_warehouse	GRAPHIC 2	1 ALL	sanfrancisco_customer	SEND 1 MOVE WITH Truck
		ALL	phoenix_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	minneapolis_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	tampa_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	boston_customer	SEND	MOVE WITH Truck THEN FREE
ALL THEN FREE	detroit_warehouse	GRAPHIC 2	1 ALL	sanfrancisco_customer	SEND 1 MOVE WITH Truck
		ALL	phoenix_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	minneapolis_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	tampa_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	boston_customer	SEND	MOVE WITH Truck THEN FREE
ALL THEN FREE	slc_warehouse	GRAPHIC 2	1 ALL	sanfrancisco_customer	SEND 1 MOVE WITH Truck
		ALL	phoenix_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	minneapolis_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	tampa_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	boston_customer	SEND	MOVE WITH Truck THEN FREE
ALL THEN FREE	boise_warehouse	GRAPHIC 2	1 ALL	sanfrancisco_customer	SEND 1 MOVE WITH Truck
		ALL	phoenix_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	minneapolis_customer	SEND	MOVE WITH Truck THEN FREE
		ALL	tampa_customer	SEND	MOVE WITH Truck THEN FREE

```

ALL      boston_customer  SEND  MOVE WITH Truck THEN FREE
ALL      albuquerque_warehouse GRAPHIC 2      1 ALL      sanfrancisco_customer SEND 1  MOVE WITH Truck
THEN FREE

ALL      phoenix_customer  SEND  MOVE WITH Truck THEN FREE
ALL      minneapolis_customer SEND  MOVE WITH Truck THEN FREE
ALL      tampa_customer    SEND  MOVE WITH Truck THEN FREE
ALL      boston_customer    SEND  MOVE WITH Truck THEN FREE

ALL      sanfrancisco_customer GRAPHIC 3
      JOIN 1 consumption_order
      WAIT U(1,.)*24 hr
      1 ALL      EXIT      FIRST 1 INC done_at_E2

ALL      phoenix_customer    GRAPHIC 3
      JOIN 1 consumption_order
      WAIT U(1,.)*24 hr
      1 ALL      EXIT      FIRST 1 INC done_at_E8

ALL      tampa_customer      GRAPHIC 3
      JOIN 1 consumption_order
      WAIT U(1,.)*24 hr
      1 ALL      EXIT      FIRST 1 INC done_at_E6

ALL      minneapolis_customer GRAPHIC 3
      JOIN 1 consumption_order
      WAIT U(1,.)*24 hr
      1 ALL      EXIT      FIRST 1 INC done_at_E7

ALL      boston_customer     GRAPHIC 3
      JOIN 1 consumption_order
      WAIT U(1,.)*24 hr
      1 ALL      EXIT      FIRST 1 INC done_at_E3

consumption_order order_E2      1 consumption_order sanfrancisco_customer JOIN 1
consumption_order order_E8      1 consumption_order phoenix_customer    JOIN 1
consumption_order order_E6      1 consumption_order tampa_customer      JOIN 1
consumption_order order_E3      1 consumption_order boston_customer     JOIN 1
consumption_order order_E7      1 consumption_order minneapolis_customer JOIN 1

```

```

*****
* Arrivals *
*****

```

Entity	Location	Qty	Each	First	Time	Occurrences	Frequency	Logic
product_1	boise_warehouse	40	0	1	1			
product_1	slc_warehouse	40	0	1	1			
product_1	albuquerque_warehouse	40	0	1	1			
product_1	dallas_warehouse	40	0	1	1			
product_1	stlouis_warehouse	40	0	1	1			
product_1	chicago_warehouse	40	0	1	1			
product_1	detroit_warehouse	40	0	1	1			
product_1	raleigh_warehouse	40	0	1	1			
product_1	neworleans_warehouse	40	0	1	1			

```

*****
* Variables (global) *
*****

```

```

ID      Type      Initial value Stats

```

```

-----
done_at_E2 Integer 20 Time Series
done_at_E3 Integer 20 Time Series
done_at_E6 Integer 20 Time Series
done_at_E7 Integer 20 Time Series
done_at_E8 Integer 20 Time Series

```

```

*****
*                               *
*                               *
*****

```

* *

* *

```

ID                               Text
-----
raleigh_warehouse_level         28
stlouis_warehouse_level         198
detroit_warehouse_level         161
slc_warehouse_level             57
dallas_warehouse_level          47
neworleans_warehouse_level      115
chicago_warehouse_level        105
boise_warehouse_level           42
albuquerque_warehouse_level     154
oklahomacity_production_level   334
seattle_production_level        58

```

```

*****
*                               *
*                               *
*****

```

* *

* *

```

ID      Type      Parameter Type      Logic
-----

```

```

daily_ordering None          top:
                               WAIT 24 hr

                               ORDER N(8, 1) consumption_order TO order_E2
                               ORDER N(8, 1) consumption_order TO order_E3
                               ORDER N(8, 1) consumption_order TO order_E6
                               ORDER N(8, 1) consumption_order TO order_E7
                               ORDER N(8, 1) consumption_order TO order_E8

                               SEND done_at_E2 product_1 TO minneapolis_customer
                               SEND done_at_E3 product_1 TO boston_customer
                               SEND done_at_E6 product_1 TO phoenix_customer
                               SEND done_at_E7 product_1 TO sanfrancisco_customer
                               SEND done_at_E8 product_1 TO tampa_customer

                               IF raleigh_warehouse_level-CONTENTS(raleigh_warehouse)>0 THEN SEND
raleigh_warehouse_level-CONTENTS(raleigh_warehouse) product_1 TO raleigh_warehouse
                               IF stlouis_warehouse_level-CONTENTS(stlouis_warehouse)>0 THEN SEND
stlouis_warehouse_level-CONTENTS(stlouis_warehouse) product_1 TO stlouis_warehouse
                               IF detroit_warehouse_level-CONTENTS(detroit_warehouse)>0 THEN SEND
detroit_warehouse_level-CONTENTS(detroit_warehouse) product_1 TO detroit_warehouse
                               IF slc_warehouse_level-CONTENTS(slc_warehouse)>0 THEN SEND slc_warehouse_level-
CONTENTS(slc_warehouse) product_1 TO slc_warehouse
                               IF dallas_warehouse_level-CONTENTS(dallas_warehouse)>0 THEN SEND
dallas_warehouse_level-CONTENTS(dallas_warehouse) product_1 TO dallas_warehouse
                               IF neworleans_warehouse_level-CONTENTS(neworleans_warehouse)>0 THEN SEND
neworleans_warehouse_level-CONTENTS(neworleans_warehouse) product_1 TO neworleans_warehouse
                               IF chicago_warehouse_level-CONTENTS(chicago_warehouse)>0 THEN SEND
chicago_warehouse_level-CONTENTS(chicago_warehouse) product_1 TO chicago_warehouse
                               IF boise_warehouse_level-CONTENTS(boise_warehouse)>0 THEN SEND boise_warehouse_level-
CONTENTS(boise_warehouse) product_1 TO boise_warehouse

```

```
IF albuquerque_warehouse_level-CONTENTS(albuquerque_warehouse)>0 THEN SEND  
albuquerque_warehouse_level-CONTENTS(albuquerque_warehouse) product_1 TO albuquerque_warehouse
```

```
ORDER oklahomacity_production_level product_1 TO oklahomacity_production  
ORDER seattle_production_level product_1 TO seattle_production
```

```
done_at_E2 = 0  
done_at_E3 = 0  
done_at_E6 = 0  
done_at_E7 = 0  
done_at_E8 = 0
```

```
GOTO top
```

APPENDIX E
SAMPLE LOG FILE

Extracting GeneralInfo.

dumping GenInfoGLibFile...

dumping GenInfoInitLogic...

End extracting general information.

Extracting location :oklahomacity_production

dumping LocStats...

dumping LocIncoming...

dumping LocSelectUnit...

dumping downtime data...

processing repair logic, manually translation may be needed...

End extracting location.

Extracting location :seattle_production

dumping LocStats...

dumping LocIncoming...

dumping LocSelectUnit...

End extracting location.

Extracting location :sanfrancisco_customer

dumping LocStats...

dumping LocIncoming...

dumping LocSelectUnit...

End extracting location.

Extracting location :phoenix_customer

dumping LocStats...

dumping LocIncoming...

dumping LocSelectUnit...

End extracting location.

Extracting location :raleigh_warehouse

dumping LocStats...

dumping LocIncoming...
dumping LocSelectUnit...
End extracting location.

Extracting location :dallas_warehouse
dumping LocStats...
dumping LocIncoming...
dumping LocSelectUnit...
End extracting location.

Extracting location :albuquerque_warehouse
dumping LocStats...
dumping LocIncoming...
dumping LocSelectUnit...
End extracting location.

Extracting location :order_E2
dumping LocStats...
dumping LocIncoming...
dumping LocSelectUnit...
End extracting location.

Extracting location :order_E8
dumping LocStats...
dumping LocIncoming...
dumping LocSelectUnit...
End extracting location.

Extracting entity : product_1
dumping EntStats...
End extracting entity.

Extracting entity : consumption_order


```
dumping EntStats...
End extracting entity.

Dumping pathnetwork : Net1
End extracting pathnetwork.

Extracting arrival for entity: product_1 at boise_warehouse
  dumping ArrivalCycle...
  dumping ArrivalDisable...
End extracting arrival.

Extracting arrival for entity: product_1 at slc_warehouse
  dumping ArrivalCycle...
  dumping ArrivalDisable...
End extracting arrival.

Extracting arrival for entity: product_1 at albuquerque_warehouse
  dumping ArrivalCycle...
  dumping ArrivalDisable...
End extracting arrival.

Extracting arrival for entity: product_1 at dallas_warehouse
  dumping ArrivalCycle...
  dumping ArrivalDisable...
End extracting arrival.

Extracting arrival for entity: product_1 at raleigh_warehouse
  dumping ArrivalCycle...
  dumping ArrivalDisable...
End extracting arrival.

Extracting arrival for entity: product_1 at neworleans_warehouse
  dumping ArrivalCycle...
```

dumping ArrivalDisable...

End extracting arrival.

Dumping variable : done_at_E2

Extracting operation at stlouis_warehouse for entity ALL

Processing user-defined logics. Manually translation may be required.

End extracting operation.

Extracting operation at detroit_warehouse for entity ALL

Processing user-defined logics. Manually translation may be required.

End extracting operation.

Extracting operation at slc_warehouse for entity ALL

Processing user-defined logics. Manually translation may be required.

End extracting operation.

Extracting operation at dallas_warehouse for entity ALL

Processing user-defined logics. Manually translation may be required.

End extracting operation.

Extracting operation at sanfrancisco_customer for entity ALL

Processing user-defined logics. Manually translation may be required.

End extracting operation.

Extracting operation at phoenix_customer for entity ALL

Processing user-defined logics. Manually translation may be required.

End extracting operation.

Extracting operation at tampa_customer for entity ALL

Processing user-defined logics. Manually translation may be required.

End extracting operation.

Extracting operation at minneapolis_customer for entity ALL

Processing user-defined logics. Manually translation may be required.

End extracting operation.

Extracting operation at boston_customer for entity ALL

Processing user-defined logics. Manually translation may be required.

End extracting operation.

Extracting operation at order_E2 for entity consumption_order

End extracting operation.

Extracting operation at order_E7 for entity consumption_order

End extracting operation.

Extracting routing for entity: ALL at seattle_production

dumping RtgPriority...

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

End extracting routing.

Extracting routing for entity: ALL at oklahomacity_production

dumping RtgPriority...

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
End extracting routing.

Extracting routing for entity: ALL at raleigh_warehouse

dumping RtgPriority...
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
End extracting routing.

Extracting routing for entity: ALL at stlouis_warehouse

dumping RtgPriority...
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
End extracting routing.

Extracting routing for entity: ALL at detroit_warehouse

dumping RtgPriority...
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
Processing user-defined logics. Manually translation may be required.
End extracting routing.

Extracting routing for entity: ALL at albuquerque_warehouse

dumping RtgPriority...

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

Processing user-defined logics. Manually translation may be required.

End extracting routing.

Extracting routing for entity: ALL at tampa_customer

dumping RtgPriority...

Processing user-defined logics. Manually translation may be required.

End extracting routing.

Extracting routing for entity: consumption_order at order_E7

dumping RtgPriority...

End extracting routing.

Extracting resource : Truck

dumping ResStats...

dumping specifications...

End extracting resource.

Dumping Subroutine : daily_ordering

End extracting subroutine.

APPENDIX F

SAMPLE UNINTERPRETED *PROMODEL*[®] XML FILE

```

_ <UnableToHandle>
  _ <GeneralInfo>
    _ <NotSupported>
      <GenInfoGLibFile>distrib.glb</GenInfoGLibFile>
      <GenInfoInitLogic>ACTIVATE daily_ordering () ANIMATE 100 VIEW
        "Full"</GenInfoInitLogic>
      <GenInfoTermLogic />
    </NotSupported>
  </GeneralInfo>
  _ <Location>
    <Name>oklahomacity_production</Name>
    _ <NotSupported>
      <LocStats>3</LocStats>
      <LocIncoming>1</LocIncoming>
      <LocSelectUnit>0</LocSelectUnit>
      _ <DownTime>
        <DTFirstTime />
        <DTPriority>99</DTPriority>
        <DTScheduled>0</DTScheduled>
        <DTDisable>0</DTDisable>
      </DownTime>
    </NotSupported>
  </Location>
  _ <Location>
    <Name>seattle_production</Name>
    _ <NotSupported>
      <LocStats>3</LocStats>
      <LocIncoming>1</LocIncoming>
      <LocSelectUnit>0</LocSelectUnit>
    </NotSupported>
  </Location>
  _ <Location>
    <Name>phoenix_customer</Name>
    _ <NotSupported>
      <LocStats>3</LocStats>
      <LocIncoming>1</LocIncoming>
      <LocSelectUnit>0</LocSelectUnit>
    </NotSupported>
  </Location>
  _ <Location>
    <Name>raleigh_warehouse</Name>
    _ <NotSupported>
      <LocStats>3</LocStats>
      <LocIncoming>1</LocIncoming>
      <LocSelectUnit>0</LocSelectUnit>
    </NotSupported>
  </Location>
  _ <Location>
    <Name>stlouis_warehouse</Name>
    _ <NotSupported>
      <LocStats>3</LocStats>
      <LocIncoming>1</LocIncoming>
      <LocSelectUnit>0</LocSelectUnit>
    </NotSupported>
  </Location>
  _ <Location>
    <Name>detroit_warehouse</Name>
    _ <NotSupported>

```

```

        <LocStats>3</LocStats>
        <LocIncoming>1</LocIncoming>
        <LocSelectUnit>0</LocSelectUnit>
    </NotSupported>
</Location>
- <Location>
    <Name>boise_warehouse</Name>
    - <NotSupported>
        <LocStats>3</LocStats>
        <LocIncoming>1</LocIncoming>
        <LocSelectUnit>0</LocSelectUnit>
    </NotSupported>
</Location>
- <Location>
    <Name>albuquerque_warehouse</Name>
    - <NotSupported>
        <LocStats>3</LocStats>
        <LocIncoming>1</LocIncoming>
        <LocSelectUnit>0</LocSelectUnit>
    </NotSupported>
</Location>
- <Location>
    <Name>order_E2</Name>
    - <NotSupported>
        <LocStats>3</LocStats>
        <LocIncoming>1</LocIncoming>
        <LocSelectUnit>0</LocSelectUnit>
    </NotSupported>
</Location>
- <Location>
    <Name>order_E6</Name>
    - <NotSupported>
        <LocStats>3</LocStats>
        <LocIncoming>1</LocIncoming>
        <LocSelectUnit>0</LocSelectUnit>
    </NotSupported>
</Location>
- <Entity>
    <Name>product_1</Name>
    - <NotSupported>
        <EntStats>3</EntStats>
    </NotSupported>
</Entity>
- <Entity>
    <Name>consumption_order</Name>
    - <NotSupported>
        <EntStats>3</EntStats>
    </NotSupported>
</Entity>
- <PathNet>
    <Name>Net1</Name>
    <PathColor>16711680</PathColor>
    <PathVisible>0</PathVisible>
    <PathType>1</PathType>
    <PathBasis>1</PathBasis>
</PathNet>
- <Arrival>
    - <NotSupported>

```



```

        <ArrivalCycle />
        <ArrivalDisable>0</ArrivalDisable>
    </NotSupported>
    <ArrivalEntName>product_1</ArrivalEntName>
    <ArrivalLocName>boise_warehouse</ArrivalLocName>
</Arrival>
- <Arrival>
  - <NotSupported>
    <ArrivalCycle />
    <ArrivalDisable>0</ArrivalDisable>
  </NotSupported>
  <ArrivalEntName>product_1</ArrivalEntName>
  <ArrivalLocName>slc_warehouse</ArrivalLocName>
</Arrival>
- <Arrival>
  - <NotSupported>
    <ArrivalCycle />
    <ArrivalDisable>0</ArrivalDisable>
  </NotSupported>
  <ArrivalEntName>product_1</ArrivalEntName>
  <ArrivalLocName>albuquerque_warehouse</ArrivalLocName>
</Arrival>
- <Arrival>
  - <NotSupported>
    <ArrivalCycle />
    <ArrivalDisable>0</ArrivalDisable>
  </NotSupported>
  <ArrivalEntName>product_1</ArrivalEntName>
  <ArrivalLocName>raleigh_warehouse</ArrivalLocName>
</Arrival>
- <Arrival>
  - <NotSupported>
    <ArrivalCycle />
    <ArrivalDisable>0</ArrivalDisable>
  </NotSupported>
  <ArrivalEntName>product_1</ArrivalEntName>
  <ArrivalLocName>neworleans_warehouse</ArrivalLocName>
</Arrival>
- <Routing>
  - <NotSupported>
    <RtgPriority />
  </NotSupported>
  <RouteEntity>ALL</RouteEntity>
  <Location>raleigh_warehouse</Location>
</Routing>
- <Routing>
  - <NotSupported>
    <RtgPriority />
  </NotSupported>
  <RouteEntity>ALL</RouteEntity>
  <Location>dallas_warehouse</Location>
</Routing>
- <Routing>
  - <NotSupported>
    <RtgPriority />
  </NotSupported>
  <RouteEntity>ALL</RouteEntity>
  <Location>neworleans_warehouse</Location>

```

```
</Routing>
- <Routing>
  - <NotSupported>
    - <RtgPriority />
    </NotSupported>
    <RouteEntity>ALL</RouteEntity>
    <Location>chicago_warehouse</Location>
  </Routing>
```

APPENDIX G

SAMPLE UNINTERPRETED *QUEST*[®] XML FILE

```

<UnableToHandle>
<Entity>
<Name>Item</Name>
<NotSupported>
NUM_DISPLAY 1
PRIORITY 1
PART_HISTORY 0
DEVICE_CREATION_MODE 1
DISPLAY_INDEX 0
PART_COLOR 'White'
PART_RENDER_MODE 1
PART_DISPLAY_BBOXES 0
PART_DISPLAY_BACKFACE 0
PART_DISPLAY_EDGES 0
PART_GEOMETRY 'default'
PCLASS_LBR_REQMT 'NO LABOR'
PCLASS_SR_REQMT 'NO_SR'
</NotSupported>
</Entity>
<Operation>
<OID>Process1All</OID>
<NotSupported>
PRIORITY 1
REJECTION_RATE 0.000000
PREEMPT_LEVEL ASSIGNMENT_FUNC 'Distributions' 'Constant'
ARG 304, 1.000000
END_ASGNMT
WGT_CLAIM CLAIM_AS_AVAIL, FIRST
LBR_CLAIM CLAIM_AS_AVAIL, SECOND
SR_CLAIM CLAIM_AS_AVAIL, FOURTH
AGV_CLAIM CLAIM_AS_AVAIL, THIRD
FLUID_PROCESS 0
SCLHOOK ASSIGNMENT_FUNC 'Start Process' 'None'
END_ASGNMT
SCLHOOK ASSIGNMENT_FUNC 'Match Exact Part' 'None'
END_ASGNMT
SCLHOOK ASSIGNMENT_FUNC 'End Requirement' 'None'
END_ASGNMT
SCLHOOK ASSIGNMENT_FUNC 'Start Cycle' 'None'
END_ASGNMT
SCLHOOK ASSIGNMENT_FUNC 'Start Production' 'None'
END_ASGNMT
SCLHOOK ASSIGNMENT_FUNC 'End Process' 'None'
END_ASGNMT
CREATED_BY 1
</NotSupported>
</Process/>
</Operation>
<Operation>
<OID>Process2All</OID>
<NotSupported>
PRIORITY 1
REJECTION_RATE 0.000000
PREEMPT_LEVEL ASSIGNMENT_FUNC 'Distributions' 'Constant'
ARG 304, 1.000000
END_ASGNMT
WGT_CLAIM CLAIM_AS_AVAIL, FIRST
LBR_CLAIM CLAIM_AS_AVAIL, SECOND
SR_CLAIM CLAIM_AS_AVAIL, FOURTH
AGV_CLAIM CLAIM_AS_AVAIL, THIRD
FLUID_PROCESS 0
SCLHOOK ASSIGNMENT_FUNC 'Start Process' 'None'
END_ASGNMT
SCLHOOK ASSIGNMENT_FUNC 'Match Exact Part' 'None'
END_ASGNMT
SCLHOOK ASSIGNMENT_FUNC 'End Requirement' 'None'

```

```

END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Start Cycle' 'None'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Start Production' 'None'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'End Process' 'None'
END_ASGNMT
CREATED_BY 1
</NotSupported>
</Process>
</Operation>
</Labor_Controller>
</Name>LC1</Name>
</NotSupported>
LENGTH 1123.421265
WIDTH 885.909668
HEIGHT 1785.747803
PRIORITY 1
GEO_FILE 'default'
NUM_ICONS 1
ICON_FILE 1 'default'
CLASS_COLOR 'Yellow'
PROCESS_LOGIC_ASSIGNMENT_FUNC 'Labor Controller Process Logic' 'Default Labor Ctr Logic'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Selection' 'Closest Labor'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Path Selection' 'Minimum Distance Path'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Part Route' 'Part Route Default'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Departure' 'Minimum Part Departure'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Destination' 'Last Part with Destination'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Ctr Event Selection' 'First Pending Event for Labor'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Parking' 'Park At Current Location'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Failure' 'Default Failure'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Reroute' 'Default Reroute'
END_ASGNMT
SCLHOOK_ASSIGNMENT_FUNC 'Labor Selection By Preemption' 'Labor Moving to Load'
END_ASGNMT
SR_PT_START_INDEX 1
CREATED_BY 1
NUM_ELEMENTS_IN_CLASS 'LC1' 1
</NotSupported>
</Labor_Controller>
</Buffer>
</Name>Buff1</Name>
</NotSupported>
LENGTH 939.799988
WIDTH 1016.000000
HEIGHT 114.300003
PRIORITY 1
KINEMATIC NO
GEO_FILE 'default'
NUM_ICONS 1
ICON_FILE 1 'default'
CLASS_COLOR 'Yellow'
PROCESS_LOGIC_ASSIGNMENT_FUNC 'Buffer Process Logic' 'Default Buffer Logic'
END_ASGNMT
PULL_PROCESS_LOGIC_ASSIGNMENT_FUNC 'Pull Buffer Process Logic' 'Default Pull Buffer Process'
END_ASGNMT
ROUTE_LOGIC_ASSIGNMENT_FUNC 'Route Logic' 'Next Free'

```

```

END_ASGNMT
PULL_ROUTE_LOGIC_ASSIGNMENT_FUNC 'Pull Route Logic' 'Default Pull Route'
END_ASGNMT
REQUEST_LOGIC_ASSIGNMENT_FUNC 'Buffer Request Logic' 'Default Buffer Request'
END_ASGNMT
STACK_DIRECTION STACK_Z_AXIS
STACK_FACTOR 1.000000
INPUTS 1
PULL_INPUTS 0
PULL_OUTPUTS 0
INPUT_TYPE PUSH
OUTPUT_TYPE PUSH
SCLHOOK_ASSIGNMENT_FUNC 'Request Propagation' 'Where Part Available'
END_ASGNMT
NEED_LABOR_CONTROLLER YES
RES_LABOR_CONTROLLER 'LC1' 1
NEED_SR_CONTROLLER NO
NUM_STACK_POINTS 1
STACK_POINT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000,
0.000000, 1.000000, 0.000000, 1.651733, -1.210052, 114.300003, 1.000000
SR_PT_START_INDEX 1
NUM_LABOR_POINTS 1
LBR_PT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000,
1.000000, 0.000000, 1.651764, -1.210037, 0.000000, 1.000000
CREATED_BY 1
</NotSupported>
</Buffer>
<Machine>
<Name>Process1</Name>
<NotSupported>
LENGTH 1040.152100
WIDTH 1075.000000
HEIGHT 1725.000000
PRIORITY 1
KINEMATIC NO
GEO_FILE 'default'
NUM_ICONS 1
ICON_FILE 1 'default'
CLASS_COLOR 'Yellow'
PROC_OCCURENCE 'Process1All' 100.000000 , 0
PROCESS_LOGIC_ASSIGNMENT_FUNC 'Machine Process Logic' 'First Possible Process'
END_ASGNMT
PULL_PROCESS_LOGIC_ASSIGNMENT_FUNC 'Pull Machine Process Logic' 'Default Pull Machine Process'
END_ASGNMT
ROUTE_LOGIC_ASSIGNMENT_FUNC 'Route Logic' 'Next Free'
END_ASGNMT
PULL_ROUTE_LOGIC_ASSIGNMENT_FUNC 'Pull Route Logic' 'Default Pull Route'
END_ASGNMT
REQUEST_LOGIC_ASSIGNMENT_FUNC 'Machine Request Logic' 'Default Machine Request'
END_ASGNMT
STACK_DIRECTION STACK_Z_AXIS
STACK_FACTOR 1.000000
INPUTS 1
PULL_INPUTS 0
PULL_OUTPUTS 0
INPUT_TYPE PUSH
OUTPUT_TYPE PUSH
SCLHOOK_ASSIGNMENT_FUNC 'Request Propagation' 'Where Part Available'
END_ASGNMT
NEED_LABOR_CONTROLLER YES
RES_LABOR_CONTROLLER 'LC1' 1
NEED_SR_CONTROLLER NO
NUM_STACK_POINTS 1
STACK_POINT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000,
0.000000, 1.000000, 0.000000, 8.059203, -239.391739, 850.316956, 1.000000
SR_PT_START_INDEX 1

```

```

NUM_LABOR_POINTS 1
LBR_PT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000,
1.000000, 0.000000, 128.135239, -3.391754, 0.000000, 1.000000
CREATED_BY 1
</NotSupported>
</Machine>
<Buffer>
<Name>Buff2</Name>
<NotSupported>
LENGTH 939.799988
WIDTH 1016.000000
HEIGHT 114.300003
PRIORITY 1
KINEMATIC NO
GEO_FILE 'default'
NUM_ICONS 1
ICON_FILE 1 'default'
CLASS_COLOR 'Yellow'
PROCESS_LOGIC ASSIGNMENT_FUNC 'Buffer Process Logic' 'Default Buffer Logic'
END_ASGNMT
PULL_PROCESS_LOGIC ASSIGNMENT_FUNC 'Pull Buffer Process Logic' 'Default Pull Buffer Process'
END_ASGNMT
ROUTE_LOGIC ASSIGNMENT_FUNC 'Route Logic' 'Next Free'
END_ASGNMT
PULL_ROUTE_LOGIC ASSIGNMENT_FUNC 'Pull Route Logic' 'Default Pull Route'
END_ASGNMT
REQUEST_LOGIC ASSIGNMENT_FUNC 'Buffer Request Logic' 'Default Buffer Request'
END_ASGNMT
STACK_DIRECTION STACK_Z_AXIS
STACK_FACTOR 1.000000
INPUTS 1
PULL_INPUTS 0
PULL_OUTPUTS 0
INPUT_TYPE PUSH
OUTPUT_TYPE PUSH
SCHHOOK ASSIGNMENT_FUNC 'Request Propagation' 'Where Part Available'
END_ASGNMT
NEED_LABOR_CONTROLLER YES
RES_LABOR_CONTROLLER 'LC1' 1
NEED_SR_CONTROLLER NO
NUM_STACK_POINTS 1
STACK_POINT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000,
0.000000, 1.000000, 0.000000, 1.651733, -1.210052, 114.300003, 1.000000
SR_PT_START_INDEX 1
NUM_LABOR_POINTS 1
LBR_PT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000,
1.000000, 0.000000, 1.651764, -1.210037, 0.000000, 1.000000
CREATED_BY 1
</NotSupported>
</Buffer>
</Machine>
<Name>Process2</Name>
<NotSupported>
LENGTH 1040.152100
WIDTH 1075.000000
HEIGHT 1725.000000
PRIORITY 1
KINEMATIC NO
GEO_FILE 'default'
NUM_ICONS 1
ICON_FILE 1 'default'
CLASS_COLOR 'Yellow'
PROC_OCCURENCE 'Process2All' 100.000000 , 0
PROCESS_LOGIC ASSIGNMENT_FUNC 'Machine Process Logic' 'First Possible Process'
END_ASGNMT
PULL_PROCESS_LOGIC ASSIGNMENT_FUNC 'Pull Machine Process Logic' 'Default Pull Machine Process'

```

```

END_ASGNMT
ROUTE_LOGIC_ASSIGNMENT_FUNC 'Route Logic' 'Next Free'
END_ASGNMT
PULL_ROUTE_LOGIC_ASSIGNMENT_FUNC 'Pull Route Logic' 'Default Pull Route'
END_ASGNMT
REQUEST_LOGIC_ASSIGNMENT_FUNC 'Machine Request Logic' 'Default Machine Request'
END_ASGNMT
STACK_DIRECTION STACK_Z_AXIS
STACK_FACTOR 1.000000
INPUTS 1
PULL_INPUTS 0
PULL_OUTPUTS 0
INPUT_TYPE PUSH
OUTPUT_TYPE PUSH
SCLHOOK_ASSIGNMENT_FUNC 'Request Propagation' 'Where Part Available'
END_ASGNMT
NEED_LABOR_CONTROLLER YES
RES_LABOR_CONTROLLER 'LC1' 1
NEED_SR_CONTROLLER NO
NUM_STACK_POINTS 1
STACK_POINT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000,
0.000000, 1.000000, 0.000000, 8.059203, -239.391739, 850.316956, 1.000000
SR_PT_START_INDEX 1
NUM_LABOR_POINTS 1
LBR_PT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000,
1.000000, 0.000000, 128.135239, -3.391754, 0.000000, 1.000000
CREATED_BY 1
</NotSupported>
</Machine>
<Source>
<AID>ItemBuff1</AID>
<NotSupported>
LENGTH 1228.710205
WIDTH 1045.617432
HEIGHT 1676.400024
PRIORITY 1
KINEMATIC NO
GEO_FILE 'default'
NUM_ICONS 1
ICON_FILE 1 'default'
CLASS_COLOR 'Yellow'
PROCESS_LOGIC_ASSIGNMENT_FUNC 'Source Process Logic' 'Default Source Logic'
END_ASGNMT
PULL_PROCESS_LOGIC_ASSIGNMENT_FUNC 'Passive Source Process Logic' 'Default Passive Source Process'
END_ASGNMT
ROUTE_LOGIC_ASSIGNMENT_FUNC 'Route Logic' 'Next Free'
END_ASGNMT
PULL_ROUTE_LOGIC_ASSIGNMENT_FUNC 'Pull Route Logic' 'Default Pull Route'
END_ASGNMT
REQUEST_LOGIC_ASSIGNMENT_FUNC 'Source Request Logic' 'Default Source Request'
END_ASGNMT
STACK_DIRECTION STACK_Z_AXIS
STACK_FACTOR 1.000000
OUTPUTS 1
PULL_INPUTS 0
INPUT_TYPE PUSH
OUTPUT_TYPE PUSH
NUM_PART_FRACTIONS 1
PART_FRACTION 'Item' 100.000000
NEED_LABOR_CONTROLLER YES
RES_LABOR_CONTROLLER 'LC1' 1
NEED_SR_CONTROLLER NO
NUM_STACK_POINTS 1
STACK_POINT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000,
0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000
SR_PT_START_INDEX 1

```



```

NUM_LABOR_POINTS 1
LBR_PT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000
CREATED_BY 1
</NotSupported>
</Source>
<Sink>
<Name>Sink1</Name>
<NotSupported>
LENGTH 1168.400024
WIDTH 897.904297
HEIGHT 1676.400024
PRIORITY 1
KINEMATIC NO
GEO_FILE 'default'
NUM_ICONS 1
ICON_FILE 1 'default'
CLASS_COLOR 'Yellow'
PROCESS_LOGIC ASSIGNMENT_FUNC 'Sink Process Logic' 'Default Sink Logic'
END_ASGNMT
PULL_PROCESS_LOGIC ASSIGNMENT_FUNC 'Pull Sink Process Logic' 'Default Pull Sink Process'
END_ASGNMT
REQUEST_LOGIC ASSIGNMENT_FUNC 'Sink Request Logic' 'Default Sink Request'
END_ASGNMT
STACK_DIRECTION STACK_Z_AXIS
STACK_FACTOR 1.000000
INPUTS 1
PULL_OUTPUTS 0
INPUT_TYPE PUSH
IRT ASSIGNMENT_FUNC 'Distributions' 'Constant'
ARG 304, 1.000000
END_ASGNMT
REQ_LOTSIZE ASSIGNMENT_FUNC 'Distributions' 'Constant'
ARG 304, 1.000000
END_ASGNMT
MAX_REQ_COUNT 10000000
REQ_START_OFFSET 0.000000
SCLHOOK ASSIGNMENT_FUNC 'Request Propagation' 'Where Part Available'
END_ASGNMT
NEED_LABOR_CONTROLLER YES
RES_LABOR_CONTROLLER 'LC1' 1
NEED_SR_CONTROLLER NO
NUM_STACK_POINTS 1
STACK_POINT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000,
0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000
SR_PT_START_INDEX 1
NUM_LABOR_POINTS 1
LBR_PT_XFORM 1, 1, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
1.000000, 0.000000, 8.334778, 9.375381, 0.000000, 1.000000
CREATED_BY 1
</NotSupported>
</Sink>
<Unrecognized>
MAGIC_NO
1125972047
MDL_VERSION 5
SCL_FUNC SUB_set_get_part_dest 411

ROUTING_MODE FROM_CLASS
REQUESTING_MODE FROM_CLASS
CREATED_BY 1
END_DEFINE

RAND_STREAM_REQUEST_PERCENTAGE 1
RAND_STREAM_ROUTE_PERCENTAGE 1
ROUTING_MODE FROM_CLASS

```

```

REQUESTING_MODE FROM_CLASS
CREATED_BY 1
END_DEFINE

```

```

RAND_STREAM REJECTION_RATE 1
RAND_STREAM REQUEST_PERCENTAGE 1
RAND_STREAM ROUTE_PERCENTAGE 1
RAND_STREAM PROC_OCCURENCE 1
ROUTING_MODE FROM_CLASS
REQUESTING_MODE FROM_CLASS
CREATED_BY 1
END_DEFINE

```

```

RAND_STREAM REQUEST_PERCENTAGE 1
RAND_STREAM ROUTE_PERCENTAGE 1
ROUTING_MODE FROM_CLASS
REQUESTING_MODE FROM_CLASS
CREATED_BY 1
END_DEFINE

```

```

RAND_STREAM REJECTION_RATE 1
RAND_STREAM REQUEST_PERCENTAGE 1
RAND_STREAM ROUTE_PERCENTAGE 1
RAND_STREAM PROC_OCCURENCE 1
ROUTING_MODE FROM_CLASS
REQUESTING_MODE FROM_CLASS
CREATED_BY 1
END_DEFINE

```

```

RAND_STREAM ROUTE_PERCENTAGE 1
RAND_STREAM PART_FRACTION 1
ROUTING_MODE FROM_CLASS
REQUESTING_MODE FROM_CLASS
CREATED_BY 1
END_DEFINE

```

```

RAND_STREAM REQUEST_PERCENTAGE 1
RAND_STREAM REQUEST_FRACTION 1
ROUTING_MODE FROM_CLASS
REQUESTING_MODE FROM_CLASS
CREATED_BY 1
END_DEFINE

```

```

FINAL_ASSMBLY_MODE NO

```

```

SIM_TIMES 1000.000000, 1, 1, 2, 0, 1.000000, 60.000000, 0.000000, 0, 0.000000

```

```

MODEL_UNITS 0, 0, 0, 0, 0, 0, 0
SIM_MODES 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
SIM_MODES2 1, 2, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0
STATE_PRIORITY 'Idle' 21
STATE_PRIORITY 'Idle - Parked' 20
STATE_PRIORITY 'Busy - Processing' 5
STATE_PRIORITY 'Busy - Loading' 6
STATE_PRIORITY 'Busy - Unloading' 7
STATE_PRIORITY 'Busy - Setup' 8
STATE_PRIORITY 'Busy - Repair' 9
STATE_PRIORITY 'Busy - Transferring' 10
STATE_PRIORITY 'Busy - Loaded Travel' 11
STATE_PRIORITY 'Busy - Empty Travel' 12
STATE_PRIORITY 'Blocked - Travel Block' 13
STATE_PRIORITY 'Blocked - Unload Block' 14
STATE_PRIORITY 'Blocked - Requirement Block' 15
STATE_PRIORITY 'Blocked - Depart Requirement Block' 16

```

STATE_PRIORITY 'Blocked - Claim Block' 17
STATE_PRIORITY 'Blocked - Requirement Preempted' 18
STATE_PRIORITY 'Blocked - Wait Block' 19
STATE_PRIORITY 'Unavailable - Shift Out' 2
STATE_PRIORITY 'Unavailable - Shift Break' 3
STATE_PRIORITY 'Unavailable - Failed' 4
STATE_PRIORITY 'Unavailable - Not Considered' 1

NUM_RAND_SEEDS 100
RAND_SEED 1, 1088421888
RAND_SEED 2, 2176843776
RAND_SEED 3, 3265265664
RAND_SEED 4, 58720256
RAND_SEED 5, 1147142144
RAND_SEED 6, 2235564032
RAND_SEED 7, 3323985920
RAND_SEED 8, 117440512
RAND_SEED 9, 1205862400
RAND_SEED 10, 2294284288
RAND_SEED 11, 3382706176
RAND_SEED 12, 176160768
RAND_SEED 13, 1264582656
RAND_SEED 14, 2353004544
RAND_SEED 15, 3441426432
RAND_SEED 16, 234881024
RAND_SEED 17, 1323302912
RAND_SEED 18, 2411724800
RAND_SEED 19, 3500146688
RAND_SEED 20, 293601280
RAND_SEED 21, 1382023168
RAND_SEED 22, 2470445056
RAND_SEED 23, 3558866944
RAND_SEED 24, 352321536
RAND_SEED 25, 1440743424
RAND_SEED 26, 2529165312
RAND_SEED 27, 3617587200
RAND_SEED 28, 411041792
RAND_SEED 29, 1499463680
RAND_SEED 30, 2587885568
RAND_SEED 31, 3676307456
RAND_SEED 32, 469762048
RAND_SEED 33, 1558183936
RAND_SEED 34, 2646605824
RAND_SEED 35, 3735027712
RAND_SEED 36, 528482304
RAND_SEED 37, 1616904192
RAND_SEED 38, 2705326080
RAND_SEED 39, 3793747968
RAND_SEED 40, 587202560
RAND_SEED 41, 1675624448
RAND_SEED 42, 2764046336
RAND_SEED 43, 3852468224
RAND_SEED 44, 645922816
RAND_SEED 45, 1734344704
RAND_SEED 46, 2822766592
RAND_SEED 47, 3911188480
RAND_SEED 48, 704643072
RAND_SEED 49, 1793064960
RAND_SEED 50, 2881486848
RAND_SEED 51, 3969908736
RAND_SEED 52, 763363328
RAND_SEED 53, 1851785216
RAND_SEED 54, 2940207104
RAND_SEED 55, 4028628992
RAND_SEED 56, 822083584
RAND_SEED 57, 1910505472

RAND_SEED 58, 2998927360
 RAND_SEED 59, 4087349248
 RAND_SEED 60, 880803840
 RAND_SEED 61, 1969225728
 RAND_SEED 62, 3057647616
 RAND_SEED 63, 4146069504
 RAND_SEED 64, 939524096
 RAND_SEED 65, 2027945984
 RAND_SEED 66, 3116367872
 RAND_SEED 67, 4204789760
 RAND_SEED 68, 998244352
 RAND_SEED 69, 2086666240
 RAND_SEED 70, 3175088128
 RAND_SEED 71, 4263510016
 RAND_SEED 72, 1056964608
 RAND_SEED 73, 2145386496
 RAND_SEED 74, 3233808384
 RAND_SEED 75, 27262976
 RAND_SEED 76, 1115684864
 RAND_SEED 77, 2204106752
 RAND_SEED 78, 3292528640
 RAND_SEED 79, 85983232
 RAND_SEED 80, 1174405120
 RAND_SEED 81, 2262827008
 RAND_SEED 82, 3351248896
 RAND_SEED 83, 144703488
 RAND_SEED 84, 1233125376
 RAND_SEED 85, 2321547264
 RAND_SEED 86, 3409969152
 RAND_SEED 87, 203423744
 RAND_SEED 88, 1291845632
 RAND_SEED 89, 2380267520
 RAND_SEED 90, 3468689408
 RAND_SEED 91, 262144000
 RAND_SEED 92, 1350565888
 RAND_SEED 93, 2438987776
 RAND_SEED 94, 3527409664
 RAND_SEED 95, 320864256
 RAND_SEED 96, 1409286144
 RAND_SEED 97, 2497708032
 RAND_SEED 98, 3586129920
 RAND_SEED 99, 379584512
 RAND_SEED 100, 1468006400

mdl_end_of_info
 10
 0 0 1 10 892.874
 0 0
 1
 3
 1
 0
 0
 0
 0
 7
 LC1 1
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
 1

```

10 327684 262145
-4 1
1 1 1
-1 -1 -1
-1 -1 -1
1 0 0
0 1 0
0 0 1
-8000 8000 0 1
1
3
0
0
0
0
0
Buff1 1
1 0 0
0 1 0
0 0 1
0 0 1
1

```

```

10 327684 262145
-4 1
1 1 1
-1 -1 -1
-1 -1 -1
1 0 0
0 1 0
0 0 1
-3250 125 0 1
1
3
0
0
0
2
Buff1_ls_way_pts
2
0
4
3
0
0
0
0
Buff1_ls_lbr_pts
5
0
4
3
0
0
0
0
Process1 1
1 0 0
0 1 0
0 0 1
0 0 1
1

```

```

10 327684 262145
-4 1
1 1 1
-1 -1 -1
-1 -1 -1

```

```

1 0 0 0
0 1 0 0
0 0 1 0
-1750 125 0 1
1
3
0
0
0
2
Process1_1s_way_pts
2
0
4
3
0
0
0
Process1_1s_lbr_pts
5
0
4
3
0
0
0
Buff2 1
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1

10 327684 262145
-4 1
1 1 1
-1 -1 -1
-1 -1 -1
1 0 0 0
0 1 0 0
0 0 1 0
250 125 0 1
1
3
0
0
0
2
Buff2_1s_way_pts
2
0
4
3
0
0
0
Buff2_1s_lbr_pts
5
0
4
3
0
0
0
Process2 1
1 0 0 0

```

```
0 1 0 0
0 0 1 0
0 0 0 1
1

10 327684 262145
-4 1
1 1 1
-1 -1 -1
-1 -1 -1
1 0 0 0
0 1 0 0
0 0 1 0
1750 125 0 1
1
3
0
0
0
2
Process2_1s_way_pts
2
0
4
3
0
0
0
Process2_1s_lbr_pts
5
0
4
3
0
0
0
ItemBuff1 1
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1

10 327684 262145
-4 1
1 1 1
-1 -1 -1
-1 -1 -1
1 0 0 0
0 1 0 0
0 0 1 0
-4520.7998046875 140.6000061035156 0 1
1
3
0
0
0
2
ItemBuff1_1s_way_pts
2
0
4
3
0
0
0
```

ItemBuff1_1s_lbr_pts

5

0

4

3

0

0

0

Sink1 1

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

1

10 327684 262145

-4 1

1 1 1

-1 -1 -1

-1 -1 -1

1 0 0 0

0 1 0 0

0 0 1 0

8000 0 0 1

1

3

0

0

0

2

Sink1_1s_way_pts

2

0

4

3

0

0

0

Sink1_1s_lbr_pts

5

0

4

3

0

0

0

0

17

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

2350.44 4530.1 75501.7 0.804202 7565.27 1237.68 42

-1

-1

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

0 0.5 0.866025 0.523599 0

0 0.547576 0.921903

0.476122 0.333573 0.282585


```

0 -1 8198 0 1 1 1.5 60 0
9
1 2 0.250000 0.250000 0.250000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.250000 0.000001 90.000000 1
0 0 0 0 -1
1 0 0.550000 0.550000 0.550000
-0.043978 -0.135349 0.989821 0.000000 0.000000 1.000000
1.427997 -0.314159 0.550000 0.000001 90.000000 1
0 0 0 0 -1
1 0 0.400000 0.400000 0.400000
0.332374 -0.690182 0.642788 0.000000 0.000000 1.000000
0.698132 0.448799 0.400000 0.000001 90.000000 1
0 0 0 0 -1
1 0 0.400000 0.400000 0.400000
0.700629 0.404509 0.587785 0.000000 0.000000 1.000000
0.628319 2.094395 0.400000 0.000001 90.000000 1
0 0 0 0 -1
1 0 0.400000 0.400000 0.400000
-0.769421 0.250000 0.587785 0.000000 0.000000 1.000000
0.628319 4.398230 0.400000 0.000001 90.000000 1
0 0 0 0 -1
0 1 0.400000 0.400000 0.400000
0.000000 0.000000 1.000000 0.000000 0.000000 1.000000
1.570796 0.000000 0.400000 0.000001 90.000000 1
0 0 0 0 -1
0 1 0.400000 0.400000 0.400000
0.000000 0.000000 1.000000 0.000000 0.000000 1.000000
1.570796 0.000000 0.400000 0.000001 90.000000 1
0 0 0 0 -1
0 1 0.400000 0.400000 0.400000
0.000000 0.000000 1.000000 0.000000 0.000000 1.000000
1.570796 0.000000 0.400000 0.000001 90.000000 1
0 0 0 0 -1
0 -8 1.000000 1.000000 1.000000 1.000000
1 3 0
0
0
0
0
0
0
5
***** START ANNOTATION NAMES AND ARROWS *****
2
2
0
***** END ANNOTATION NAMES AND ARROWS *****
0
0
3
4
0 0 0 0 0
20 8000 3 1e+008 0.523599 32
20 3000 20 3000 1 20000
200 10000 50 25000 3 500
3
1
0 3 50000
1
0 10 1.5 1.5 100
1
0 200 50 20

```

0
0
</Unrecognized>
</UnableToHandle>