2005

Developing An Object-oriented Approach For Operations Simulation In Speedes

Amit Wasadikar
University of Central Florida

DEVELOPING AN OBJECT-ORIENTED APPROACH FOR OPERATIONS SIMULATION IN SPEEDES

by

AMIT S. WASADIKAR
B.S. Dr. B.A.M University, July 2000

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Industrial Engineering and Management Systems
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2005

ABSTRACT

Using simulation techniques, performance of any proposed system can be tested for different scenarios with a generated model. However, it is difficult to rapidly create simulation models that will accurately represent the complexity of the system. In recent years, Object-Oriented Discrete-Event Simulation has emerged as the potential technology to implement rapid simulation schemes. A number of software based on programming languages like C++ and Java are available for carrying out Object Oriented Discrete-Event Simulation. These software packages establish a general framework for simulation in computer programs, but need to be further customized for desired end-use applications. In this thesis, a generic simulation library is created for the distributed Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES).

This library offers classes to model the functionality of servers, processes, resources, transporters, and decisions. The library is expected to produce efficient simulation models in less time and with a lesser amount of coding. The class hierarchy is modeled using the Unified Modeling Language (UML). To test the library, the existing SPEEDES Space Shuttle Model is enhanced and recreated. This enhanced model is successfully validated against the original Arena model.

ACKNOWLEDGMENTS

I offer my sincere and deepest thanks to my advisor Dr. Luis Rabelo for his continuous motivation and inspiration throughout this work. Dr. Luis Rabelo not only helped me as thesis advisor but also guided me to take important decisions in the academics. I take this opportunity to thank Dr. Jose A. Sepulveda for all the coaching I received from him during the coursework. I also thank Dr. Christopher D. Geiger for agreeing to be on my thesis committee.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DES	Discrete-Event Simulation
FDK	Federated Simulation Development Kit
FIFO	First In First Out
JAS	Java Agent Based Simulation
KSC	Kennedy Space Center
OMNet	Objective Modular Network
OOP	Object-Oriented Programming
OOS	Object-Oriented Simulation
PADS	Parallel and Distributed Simulation
RTI	Run Time Infrastructure
SPEEDES	Synchronous Parallel Environment for Emulation and Discrete-Event Simulation
UCF	University of Central Florida
UML	Unified Modeling Language
VTB	Virtual Test Bed
XML	Extensible Markup Language

CHAPTER 1: INTRODUCTION

1.1 Preface

In any simulation environment, a predefined set of frequently used classes is required. In this study, a generic library has been developed for the Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES). SPEEDES is a Linux based C++ Discrete-Event simulation environment. The classes in this library can be used in any SPEEDES simulation to generate quick and efficient models. This chapter will familiarize the reader with the concepts of Object-Oriented Simulation, Distributed Simulation, and will explain the background of this study.

1.2 Object-Oriented Simulation

An Object-Oriented Simulation consists of a set of objects that interact with each other over time [1]. The strength of Object-Oriented Simulation lies in producing independent, component based code that can be changed, enhanced, and reused easily. In Object-Oriented Simulation (OOS), the complex logic of the model can be embedded into implementation classes, and a real-world entity can be represented using simulation objects. Features of Object-Oriented Programming like Encapsulation, Data Hiding, Inheritance, and Reusability are significant in such simulation techniques [2]. Object-

Oriented Simulation is capable of creating generic base classes and inheriting them to create specialized classes. The classes contain the attributes and procedures used in the model where the attributes are used as state variables, and the procedures are mapped as simulation events by the mechanism provided in the simulation framework.

The common languages used for Object-Oriented Simulation are Java, C++, Simula, and Smalltalk, whereas environments such as SPEEDES, Simkit, Silk, and PSimJ provide the simulation framework.

1.2.1 Encapsulation and Inheritance

By definition, hiding details and providing a common interface to access the data, is called Encapsulation. Encapsulation ensures that one object cannot change the state of another object in unexpected way and can only pass the messages using the internal methods. Encapsulation also helps reduce the dependency of client code on internal manipulation code in the classes of the library. This way, sensitive simulation data can be hidden from public view.

The property of reusing the attributes and behavior from base class to child class is called Inheritance. This property is one of the strongest features of Object-Oriented Simulation techniques. Use of Inheritance in simulation allows a user to create models with the help of a pre-built library. Inheritance brings reusability to the code, so that the enhancements can be done without repeating the previous work.

1.3 Distributed Discrete-Event Simulation

In a distributed Discrete-Event simulation, execution load can be divided on multiple processors to help speed up the run. Distributed simulation is especially useful for running a large simulation model on computers at different physical locations via the Internet. It also helps to run the simulation with smaller capacity resources, such as low processor speed or less available memory [3]. In an Object-Oriented distributed simulation environment, simulation objects are distributed among the processors which can schedule local or global events on other objects. To coordinate the simulation among all processors, the distributed environment must provide a good communication infrastructure. Many communication algorithms have been developed to support distributed simulation. Due to each processor maintaining a separate event list and a separate simulation clock, fast processors may run ahead in time compared to the slower processors. If an event is scheduled by a slow processor in the past simulation time, the events on the fast processor need to be reverted back. There are very few environments which support distributed simulation, and SPEEDES is one of them. SPEEDES provides rollback features to reestablish the object state if an event in the past is received by the processor.

1.4 Problem Statement and Importance of Work

The need for this study is established from the Virtual Test Bed Project being executed in the department of Industrial Engineering and Management Systems at the University of

Central Florida. Virtual Test Bed (VTB) is a NASA-sponsored simulation project to establish a common platform to study and integrate heterogeneous simulation models. In the first phase of this project, an existing ARENA Space Shuttle Simulation model is converted into SPEEDES. The ARENA model is built to simulate the detailed hardware flow of the NASA Space Shuttle operations [4].

In translating the model from Arena to SPEEDES, there was a need for predefined classes for rapid development. In any Discrete-Event simulation some basic modules like Server, Resource, Queue, and Transporter are frequently used. Due to unavailability of these classes, the SPEEDES Space Shuttle model repeats the code at every instance. Because of more code, more memory, more debugging time and more maintenance of the model are required. Hence, to create truly Object-Oriented simulation models, it is necessary to develop a library of frequently used classes. This is the objective of this thesis work.

1.5 Synopsis

This study demonstrates the implementation of predefined classes to enhance the SPEEDES. The developed classes are of the plug-and-play type. In any simulation model, the user needs to include these classes and create the objects whenever required in the logic. This library will speed up the model building process in the SPEEDES. The code for any simulation model will be structured, maintainable and easy to understand, than what it would be without this library. The uniqueness of this library is the only

available generic library for DES in Linux environment.

To test the library the SPEEDES space shuttle model is recreated using these classes. At every opportunity, objects from these classes are used to replace the repeated code. Other enhancements in this model included using dynamic creation of simulation objects, and providing an interface to change the random number distributions outside the model.

The result of these enhancements is an approximate 40% reduction in code, thereby creating a more maintainable, reusable, easily understandable, and truly Object-Oriented space shuttle model. This study delivers the library of these generic classes in SPEEDES, with an implemented example of the SPEEDES space shuttle model. The targeted areas of application for this library are industrial simulation, war-gaming simulation, transportation simulation, etc.

This work serves as the reference for the developed simulation library. Chapter 2 reviews the concepts of Discrete-Event Simulation and discusses the past implementations of this kind of library in other simulation environments. The chapter also discusses distributed simulation in detail, previous implementations and distinct features of the distributed simulation environment SPEEDES. Chapter 3 explains the structure of this library in detail with UML diagrams. Chapter 4 discusses the implementation of this library to recreate the SPEEDES space shuttle model. This chapter also lists the benefits gained in this model because of the use of this library. Chapter 5 concludes this discussion by specifying future use of this library and potential areas of application.

CHAPTER 2: LITERATURE REVIEW OF DES LIBRARY AND SPEEDES

2.1 Preface

This chapter first introduces the basic concepts of Discrete-Event Simulation. Then, a survey of reusable libraries for fast creation of models using different DES environments is presented. The library survey is followed by the section of Distributed Simulation Concept, with examples of an application. Finally, in the last section, the distinct features of SPEEDES used in this library are discussed.

2.2 Discrete-Event Simulation

Discrete-Event simulation is the systematic approach of modeling the system as it advances through distinct time steps each of which changes the state of the system [5]. The simulation study starts with collecting data from the system which can be either past data or data from experimentation. This data is then modeled into mathematical distribution to produce a random number with similar behavior as of real system. The next step is to build the logic of the system. The logic can be built using modern programming languages or specialized Discrete-Event simulation software. At the end of modeling, process verification and validation of the model are performed. In the

verification model logic is checked and in the validation, the model is checked statistically [5].

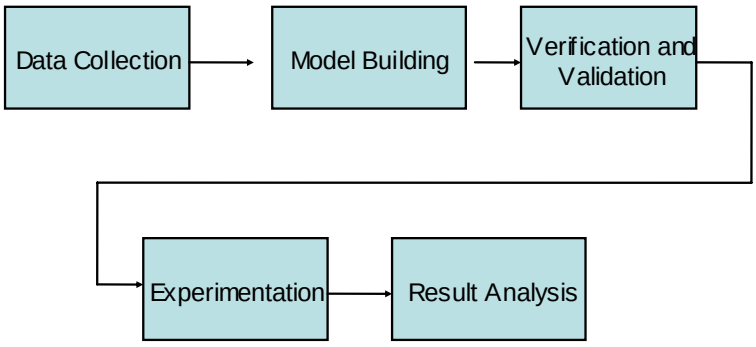


Figure 1: Steps in Discrete-Event Simulation

When it is evident that the model is representing the actual system under study, experiments can be performed on the model. Different scenarios to test the changes in the existing system can be run. The performance can be measured in several ways, such as by changing the number of resources, run time, or run period. The results from these experiments are analyzed to check the feasibility of the new system.

2.2.1 Entity and Event

An Entity is any physical, tangible element of the simulation system. It possesses a logical relationship with other entities and defines the behavior of the overall model. Generally, an entity passes between the events in the simulation.

An Event is a process which occurs on a particular timestamp. Normally, events are matched to the procedures in the programming language. The Event changes the state of the system by changing the attributes of the entity. One event can trigger another event to populate the event list. Events are executed at fixed times, i.e. the time at which they are scheduled.

2.2.2 Simulation Clock and Event List

Every Discrete-Event Simulation maintains an event list. This event list contains the events to be executed and the time at which they are to be executed. Events are sorted by the order in the event list. The primary order of events in the event list is the timestamp of the events. If two events have same the timestamp, then they are prioritized by some other criterion such as type of object. To execute events on their respective timestamps a simulation clock is maintained by the simulation engine. In Discrete-Event Simulation, the clock jumps from event to event creating increments in the time steps which allows optimization in the execution of the model. The simulation clock and the event list are maintained by the simulation engine, which establishes the required framework for simulation.

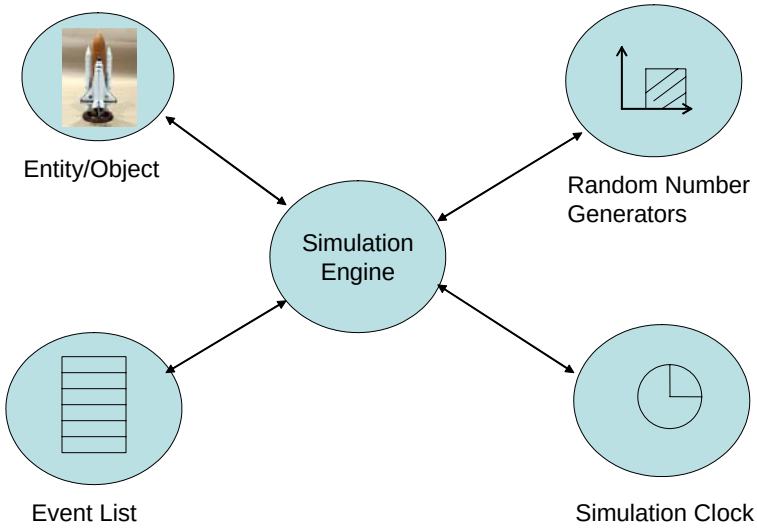


Figure 2: Components of Discrete-Event Simulation

2.2.3 Random Numbers

Random numbers are the base of any simulation model as they capture the stochastic nature in the simulation runs. Random numbers are generated by an algorithm which generally starts with a pseudo-number between 0 and 1. To change the range of random numbers between runs, a different seed is provided. The seed is any integer used to generate different series of random numbers in the simulation [26]. Simulation models typically employ random number distribution to specify service times, inter-arrival times, and routing times.

2.3 Generic Libraries for Discrete-Event Simulation

Discrete-Event Simulation benefits greatly from predefined libraries. Predefined libraries

provide generic functionality to create basic components, including servers, resources, and transporters, all of which ease the model creation. Using these libraries, model development time can be saved, and component based code can be created for easy maintenance. Most of the commercial simulation software employs these types of libraries with their applications. The simulation environments offered in C++ or Java gives a basic framework to create and execute DES, however, these packages do not fulfill the requirements for every domain. Users must create the predefined libraries to fulfill the specific requirement. Here, we review some examples of these libraries, provided in different simulation environments.

2.3.1 Arena Simulation Library

The Arena Simulation Software offers a rich set of simulation libraries. These libraries are divided into different panels: The Basic Process Panel provides the modules like Create, Process, and Decide, to produce a fast and simple simulation model. The Advanced Process panel provides complicated functionalities such as Hold-Signal, ReadWrite, and Search. The Advanced Transfer panel provides functionality for transferring entities using different means. In a similar fashion, the modules of Elements, Packaging, Blocks, and Script are used for specialized areas of application. When users create models using these libraries, Arena creates modular code in SIMAN [6]. Pre-defined object libraries make model creation in Arena very easy.

2.3.2 Flexsim Simulation Library

One of the examples of Object-Oriented Simulation libraries is Flexsim. Flexsim is the C++ simulation software offering a generic library to create custom simulations. The library includes modules such as Source, Queue, Processor, Conveyor, and Transporter [7]. Flexsim uses classes that represent process activities and queuing. It also offers the functionality to change the existing library to fit simulation needs by having the library create specific-use classes or by creating totally new libraries using the Microsoft Visual C++ compiler. C++ controls the behavior of the created classes. As Flexsim objects are open to the modeler, the customized classes can be exchanged between users and can be used as simple drag and drop in the application. Flexsim also provides the 3D visual representation of the objects to create state-of-the-art animation.

2.3.3 Simkit Simulation Library

Simkit is a Java-.based simulation environment for DES. It supports model building through the Event Graph approach. Event graphs are a way of representing the future event list, respective object and state transition in the simulation. Event graphs also demonstrate the associated Boolean condition and time delay in between the events. The Events graph is a simple method of event modeling and hence accelerates understanding and creation of DES models [8].

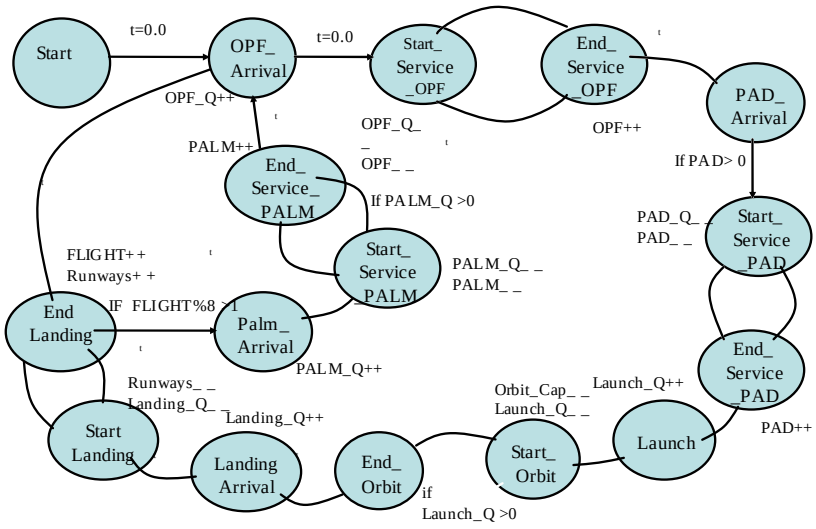


Figure 3: Example of Event Graphs in Simkit

To help build a quick simulation model in Simkit, a library of generic classes is developed at the University of Central Florida (UCF, 2003). These classes are built to bring in the functionality similar to that of the basic process panel in Arena. Different classes such as server, decision, resource, and queue, are created to support DES in Simkit. This library is based on Object-Oriented concepts, and is further inherited into specialized classes like OPF and KSC to develop NASA simulation models.

2.3.4 Silk Simulation Library

Silk is Java-based DES environment, supporting animation through the Java language. Silk explicitly uses the multi-threading capability of Java to implement process-based simulation. By executing the simulation in the Java environment, users can have access to

distinct features of Java, such as browser-based simulation, platform independent simulation, standard communication protocols, database connectivity, applets, and graphical user interface creation.

Silk offers a distinct feature called JavaBeans Component Modeling for animation purposes, which works similar to any generic simulation library. The advantage of this feature is that a simulation model can be built faster using predefined Silk components rather than starting from scratch. Silk can also write Individual, self-content simulation modeling components, which are automatically made functional and interoperable when incorporated into the JavaBeans environment. A user can assemble the components into the model by placing them in the workspace and editing their properties to produce the desired behavior [9].

2.3.5 Agent Based Simulation – E-commerce using Silk

This library is developed by at the Delft University of Technology, The Netherlands [10]. A number of predefined components containing mechanism and behavior to represent various roles relevant for the domain of electronic commerce are developed. Predefined components were defined in the java based simulation environment Silk. Agents are autonomous, goal driven entities that are able to communicate with other agents [10]. Their behavior is a consequence of their observation, their knowledge, and their interaction with the other agents [10].

Based on functional requirements, a generic agent model is derived consisting of

three layers (Control layer, Visualization layer and Process layer). Many domain-specific agents are built to support each layer. An essential feature of this approach is that several components are identified and constructed to represent behavior of independent organizations in e-commerce. These components can be further customized for specific use.

2.3.6 SimBeans – Library for component based DES using JavaBeans

SimBeans is a project conducted at Johannes Kepler University, Austria [11]. The JavaBean is a reusable software component that can be modified and composed interactively with other components. In this project, a powerful DES framework and a set of flexible java bin components are developed. The SimBeans system is developed in multiple layers. The lowest layer is Java programming language and the JavaBeans component model. The next layer is the Simulation Kernel providing infrastructure and implantation for simulation. The top layer consists of elementary and application specific simulation components. These simulation components are built as stand alone programs. The library of elementary simulation components for discrete process simulation has been designed to facilitate utmost reusability and extensibility [11].

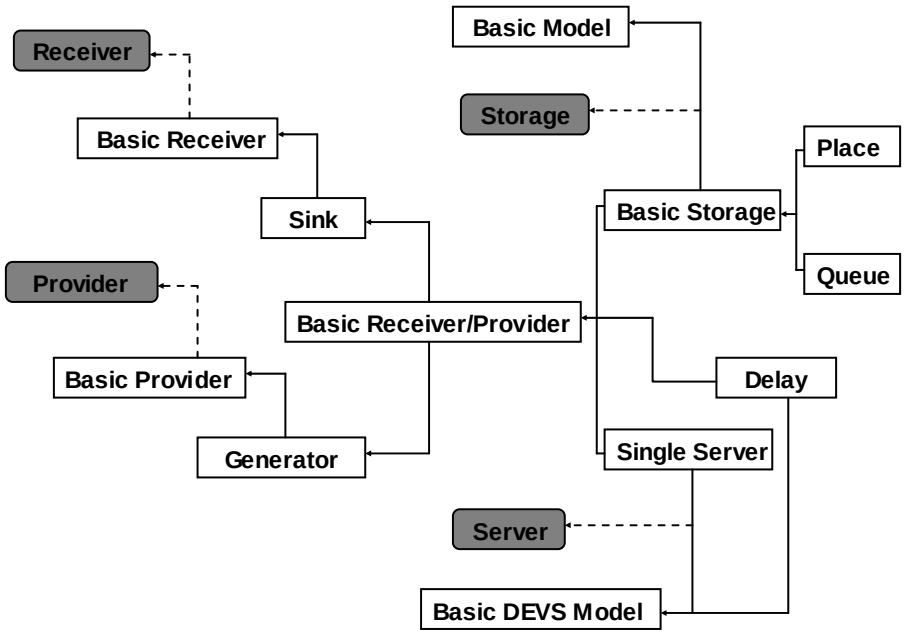


Figure 4: Class hierarchy for elementary model bean

The set of building blocks in the library include Generator, Sink, Processor, Queue, Place, and Delay. A separate set of components for visualization and animation is also provided. This component based library serves as an example of how predefined structures in Discrete-Event simulation can help to build the model rapidly.

2.3.7 JavaDemos – Library for DES using Java

JavaDemos is a Java library for DES, written by Olaf Matthes in his thesis at the University of Essen, Germany in 1999. This work is inspired by the Demos system written in Simula language. JavaDemos serves as a simulation environment as well as

providing classes for rapid creation of simulation models. These classes include Report, Random Number Generators, Entity, Queue, Resource and Bin. To use the Demos library, the user simply needs to include the Demos package in the simulation program written in Java. JavaDemos also provides graphical front end feature for visualization and interaction of the simulation run [12].

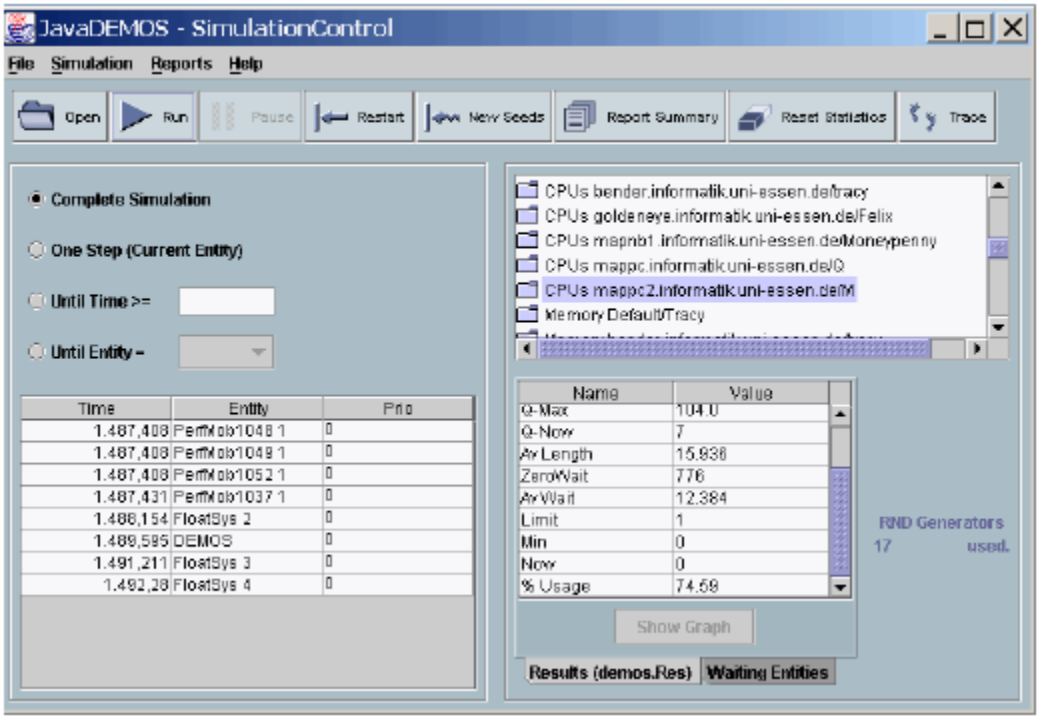


Figure 5: Screen capture of JavaDemos Model

As the JavaDemos example in figure shows, the current objects in the event list, statistical results of the usage of a resource object and the simulation trace can be seen while the simulation is running.

2.3.8 Mobile Agents – Reusable Building Blocks using JavaDemos

Mobile Agents is the simulation library for distributed information technology developed at the University of Jena, Germany. This library is developed using JavaDemos. Mobile Agents are autonomous, intelligent programs that move through a network, searching for and interacting with services on the user's behalf. These systems use specialized servers to interpret the agent's behavior and communicate with other servers.

This library is developed to analyze the performance of a Mobile Agent system during the development of an agent code. Modeling and simulating the Mobile Agent system using JavaDemos required plenty of simulation code. Therefore, to ensure faster development of the model reusable building blocks which fulfill the basic functionality of Mobile Agent system are developed. The simulation tool JavaDemos was extended to include new classes in order to enable performance analysis of a mobile agent system such as agent round trip time or the utilization of the agent server.

Basic goal of the library is to simulate the Mobile Agent system as a real agent system to test the performance. There are three top level classes; first is the Agent Server class containing objects for CPU, Resource, Agent Communication, Agent Wait Queue, and Message Queue. The second class is the Network Link class, containing attributes like Bandwidth, Medium Length, Active Stations, and Packet Size. Third class is the Router class, which stores the router table. This library serves as an example of how reusable classes can help to efficiently build simulation models in the specific domain [13].

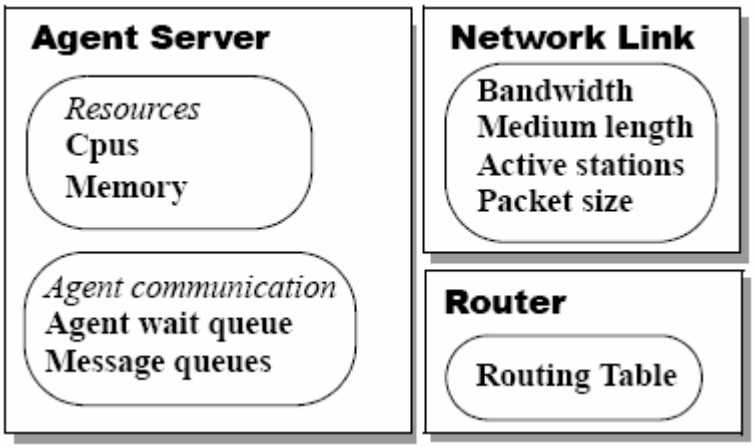


Figure 6: New classes for mobile agent simulation

2.3.9 JAS – Java Agent Based Simulation Library

JAS is developed at University of Torino, Italy [14]. JAS is a rich and open collection library, developed using Java and XML. It is designed for agent-based modeling to help create models and share them easily. JAS provides a collection of ready to use widgets and a set of thumb rules to build the simulation model using these widgets. Most of the libraries contained in JAS are reliable, well-tested third party libraries. Different packages in the JAS library are Discrete-Event Time Engine, Statistical Probes, Neural Networks/Genetic Algorithms, and Graph support for social network analysis. JAS is an example of interoperability and code reutilization enabled by open source licenses.

The special features of JAS include XML data input/output operations and a genetic algorithm library. JAS also supports Sim2Web architecture (for web publishing of simulations and remote user interaction) and automatic collection of statistical data

into a database [14].

2.3.10 FDK – Federated Simulation Development Kit

Federated Simulation Develop Kit (FDK) is the library of software modules developed at Georgia Institute of Technology, by Parallel and Distributed Simulation (PADS) research group. This library offers the modules for building run-time infrastructure (RTI). RTI is the software which provides a set of services used by federates to coordinate their operations and data exchange during a runtime execution [22]. RTI developers can choose from the set of provided FDK modules for developing RTI implementation. RTI developers can benefit from incorporating these ready-made modules, instead of developing on their own [15]. FDK is another example of how predefined simulation modules can help developer to build the model.

These different examples demonstrate how reusable domain specific classes are required in any simulation environment. Some of the simulation environments come with a reusable library and for some of the environments such a library needs to be built.

2.4 Distributed Discrete-Event Simulation

When a simulation grows large and has lots of events to be processed simultaneously, it is helpful to distribute those simulation event executions on multiple computer processors. This assists in sharing the load on multiple processors, reduces the simulation run time, and gets more address space [16]. Each processor is made responsible to

process certain kinds of events and each processor maintains its own event list and simulation clock. The main concern in distributed simulation is what happens when one processor schedules an event on another processor in the past. To implement parallel simulation, synchronization and communication between different nodes is very important. To achieve effective synchronization, two approaches, i.e., Conservative approach and Optimistic approach are generally used.

In the conservative approach, a check is performed so that no simulation object processes an event when it is possible to receive an event from other simulation objects at an earlier time stamp. By this approach, simulation objects will not be allowed to schedule an event on another simulation object on a different node for less than the look-ahead time gap. Hence, no event is scheduled out of order. This increases the interaction between simulation objects. If a node knows that the current simulation time of another node is greater than or equal to its current simulation time, only then the first node can continue processing events, otherwise it has to wait. To implement this approach, more communication between nodes is required.

In an optimistic approach, it is assumed that this kind of problem will not occur, and if it occurs, then the simulation is “rolled-back” to an earlier point. The simulation engine needs to have the capability of undoing the changes made to state variables and must also be able to keep track of already scheduled events. This kind of approach requires advanced simulation engine like SPEEDES. SPEEDES automatically retracts locally generated events through pointers and uses anti-messages to cancel events generated for simulation objects on another node [16]. This approach requires less

communication and it is also flexible, but roll-backing of the events is an overhead on the node.

2.4.1 Past Implementations of Distributed Simulation

Many researchers and scientists have performed different experiments to study speedup, computational load, and the number of processors required in distributed simulation. Distributed simulation performance is evaluated against sequential simulation. The following sections summarize some of those implantations and their conclusions.

2.4.2 Yaddes Simulation System

This is a study presented at the University of Waterloo, Canada [17]. It compares the distributed simulation with the sequential simulation of the same system. The performance is tested against four different performance measures: speedup, number of processors, lookahead time, and computation load. The Yaddes system is a tool for constructing a DES model. It compiles the instructions to produce C language code and links this code with a library to create an executable model. The Yaddes system supports sequential simulation as well as three distributed synchronization methods such as distributed simulation using multiple lists, conservative, and optimistic approaches.

Different experiments are conducted changing the initial load of the system. Out of 480 different simulation results, 432 are distributed simulations and 48 are sequential simulations. Each benchmark is run on one processor using traditional sequential

algorithms and on multiprocessors with two, four and eight processors using three different distributed synchronization methods. The result observed is that, a distributed simulation with multiple lists does not show speedup for the listed benchmarks, while the other two synchronization methods give good speedup.

The conservative approach shows the best speedup especially under heavy load situations whereas the optimization approach shows good speedup for all cases with a number of processors equal to or greater than four [17].

2.4.3 Remote OMNeT++ Distributed DES Environment

OMNeT++ (Objective Modular Network Tested in C++) is a remote simulation environment developed at the Budapest University of Technology and Economics, Hungary. In the referred paper OMNeT++ is presented to demonstrate its remote simulation capabilities i.e., simulation is not performed on the user's computer, but it is performed at a dedicated server. Users can access the simulation through a computer network. This resource concentration is mainly used for the reasons of hardware costs, data consistency, software upgrades, and maintenance.

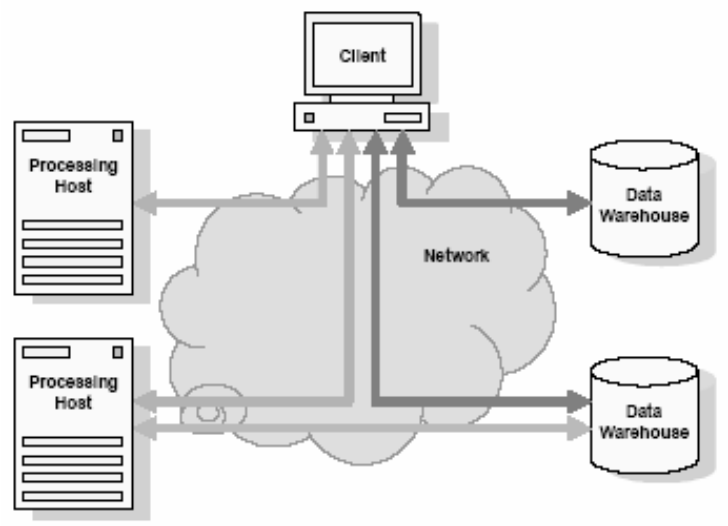


Figure 7: The components of Remote OMNeT++

The architecture for this simulation system is constructed with three components: Client, Processing Host and Data Warehouse. Processing host is mapped as the Manager, which is responsible for running the uploaded model. Data Warehouse can be a file, relational database or complete Object-Oriented database management system. Finally, the Client component is responsible for providing an easy to use interface to users through a GUI, while using minimum resources. The model in the system can be run with three basic configurations as follows: In Development Configuration, developers can create and test the models on same machine as simulation run time during development is small. In Laboratory Configuration, the model is uploaded on the server for testing or education purposes; hence no data warehousing is required. In Corporate Configuration, where heavy duty simulations are performed, a dedicated server and one or more data storage severs are required.

The features of OMNeT++ include a “Specialized Security system” with a variety of access rights, as required for distributed simulation. It also provides the feature of “off-line Monitoring”, using which allows the user to safely disconnect from the processing host and reconnect later to check the progress. OMNet++ supports Common Object Request Broker Architecture (CORBA) using the simulation manager which can be implemented in Java or C++ making it truly open architecture. The “Web based monitoring” can be enabled using Applets to monitor the simulation from any geographical location. OMNeT++ supports “Batch execution” to automatically distribute the simulation on different processing hosts. OMNeT++ also provides the feature of “Test case generation”, using which simulation can be run with changing parameter over a given range. Hence, OMNeT++ provides a solution for large simulations, high hardware costs and can be used as a platform independent environment [18].

2.5 SPEEDES

SPEEDES is the Linux based C++ Discrete-Event simulation environment used to simulate a wide range of models. SPEEDES has been used for military simulations, navy simulations, Space shuttle simulations, etc. The special feature of SPEEDES which makes it suitable for large simulations is simulation in parallel/distributed simulation. SPEEDES can distribute different objects on multiple processors. For implementing parallel processing, SPEEDES possesses the capability of rolling back the event. Sometimes when one processor is faster than another, the faster processor may need to

rollback if the slower processor schedules an event in the past. For rollback purposes, specialized data types in SPEEDES are provided, which can be used in the same way as the data types in C++ [16].

SPEEDES also posses the functionality of providing interface for external modules and federations. This interface can be used to develop an intricate communication and graphical user interfaces. This is also called high level architecture (HLA). High level architecture is widely used in military simulations, games, etc.

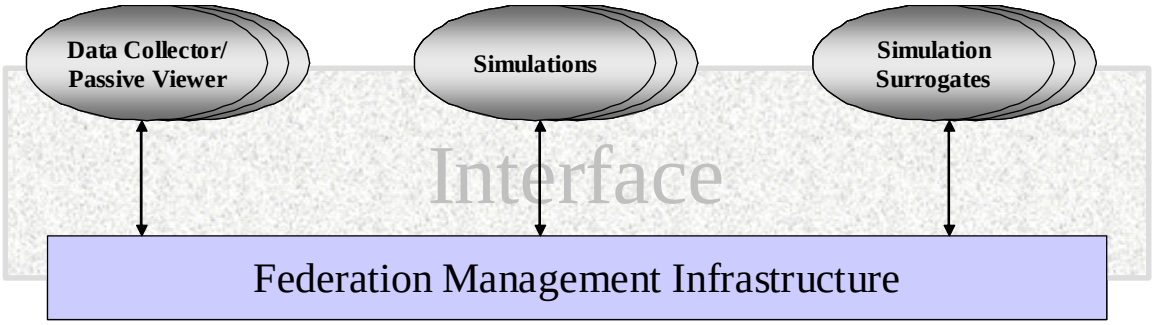


Figure 8: High Level Architecture

2.5.1 SPEEDES Simulation Object

SPEEDES provides a set of C++ classes and an Application Program Interface (API) as the framework. As the part of framework SPEEDES provides a base class “SpSimObj”. Any class which needs to schedule an event or needs to change state must inherit this class. This class provides the functionality of roll back.

To declare any object as a simulation object “DEFINE_SIMOBJ” needs to be

included in the class definition. Also, this object has to be plugged in the framework by calling the macro “PLUG_IN _SIMOBJ” in the main function. To initialize any object SPEEDES provides an Init method. This method is called at the start of the simulation. Similarly, to clean up the memory space before deleting the object SPEEDES provides the Terminate function.

The objects of each simulation class are managed by the simulation object manager. On each node, one simulation object manager is created per class. The simulation manager is used to manage initialization and termination of objects, creating dynamic objects, managing external module subscription, etc.

Each simulation object is identified by three IDs (i.e., Kind ID, Local ID, and Global ID). These IDs are useful in retrieving the simulation object. Kind ID is the ID assigned independently to each type of simulation object. For example, if we have four shuttles and seven transporters then shuttles will be assigned Kind ID (0-3) and transporters will be assigned Kind ID (0-6). Another type of ID is Local ID. It is the sequential number of the object on each CPU. It will count how many objects are on each node regardless of the type of the object. Finally, Global ID is the unique ID throughout all the nodes. In the above example, shuttle and transporter will have global ID’s (0-10).

Table 1: Example of IDs of Simulation object (S: Shuttle, T: Transporter)

	Node 1					Node 2					
Object	S1	S2	T1	T2	T3	S3	S4	T4	T5	T6	T7
Kind ID	0	1	0	1	2	2	3	3	4	5	6
Local ID	0	1	2	3	4	0	1	2	3	4	5
Global ID	0	1	2	3	4	5	6	7	8	9	10

Simulation objects can be assigned to different nodes using three decomposition algorithms. In Block algorithm, objects are assigned to each node in a chunk, in the Scatter algorithm, objects are distributed to each node like playing cards and in a File based decomposition the simulation object and desired node can be specified [16].

2.5.2 Point-to-Point Events

These are the most primitive types of event in SPEEDES. In this type of event one simulation object schedules an event on another simulation object at a point in simulated time. It is a one way type of communication. To define any point to point event first a method on the simulation object needs to be defined. Then that method needs to be mapped to the event by the macro “DEFINE_SIMOBJ_EVENT”, and then need to plug-in this method into the framework by the macro “PLUG_IN_EVENT”. Finally, a call is sent to the SCHEDULE_ macro to schedule this event [16].

There are three types of simulation macro available for point to point events: simulation object events, local events, and autonomous events. These events differ from each other by where the event method resides. In the simulation object, the event method the event resides on the class inherited from “SpSimObj” or one of its descendents. In local event, the method resides on the nested sub-object of the scheduling simulation object. In both these types the events method resides on the simulation object. But in an autonomous event, the method is separated from the simulation object.

2.5.3 Event Handlers

Event handlers are the substitute for point to point event. Additionally, event handlers can schedule one or more events and the event handler can be configured at execution time. That means the method attached to an event can be changed during the simulation, giving the flexibility of choosing implementation to the user. Another advantage of an event handler is that it can schedule an undirected event, which means they can trigger events on all simulation objects who have subscribed by broadcasting the message. There are three kinds of event handlers: Standard event handlers, Interaction event handlers, and Interface event handlers.

Generally standard handlers are used when no data needs to be sent to the event. This is the easiest form of handlers. When data needs to be sent in standard handlers, it has to be sent by variable buffer length. On the other hand, interaction handlers are used to send variable length of data to the event. It uses the class “SpParamSet” to define the

set of parameters. This set can be changed each time the handler is scheduled. The last type of handler is the interface handler. The Interface handler allows for multiple arguments in the handler method.

2.5.4 Process Model

This is a very specialized feature which sets apart SPEEDES from other simulation environments. This allows a method to execute in a time period rather than instantly at the same simulation time. Process is basically a point to point event, which uses some macro to have the capability of exiting at any point and reentering at some later point in time.

Event-based models execute the events instantaneously, where as process based models run over the period of time. Process based models can be mapped into treads to work in parallel. A Process based paradigm is also easy to implement.

A Process model uses macros for initializing the model, wait reentry point, semaphore reentry point and ask reentry point. Initializing macros mark up the start of the process model code and declare local state variables, i.e., variables which retain value in between the reentry. Wait macros used in wait reentry point can hold the process for a fixed time period. Semaphore reentry can hold the process until a resource semaphore or variable semaphore is available for further execution. Using these macros, the process model can be set on hold for a fixed time, waiting for a resource or for a variable. In this study, a process model is being used to implement the server class, where the server is on

hold until the required process time has passed.

2.5.5 Dynamic Objects

Traditionally, the user needed to specify the number of the objects required for a class at compile time. This method restricted the user from creating objects dynamically. Hence one has to count all the objects required for each class before the simulation starts. Dynamic object creation is an improved approach used to create objects in this study.

As a first step, it is necessary to declare dynamic initialization method. This method needs to be mapped into event using macro DEFINE_CREATE_EVENT. The byproduct of this macro call will be a schedule method identical to a standard schedule function. Then the dynamic objects can be created any time by calling macro SCHEDULE_, which returns the handle of newly created and initialized objects.

CHAPTER 3: LIBRARY STRUCTRURE WITH UML DIAGRAMS

3.1 Preface

This chapter describes the structure of the classes, technical aspects of the classes and the method of using these classes in any simulation model. The library contains the following classes: Server, Decision, Transporter, Entity, and the Data Interaction classes used for communication. Resource and Queue functionality is embedded in the Server class. Supporting UML diagrams are provided to give a better insight into the library.

This chapter initially describes the importance of UML diagrams in simulation and the significance of each diagram. Each class is explained in detail with UML diagrams and class code discussion follows in the next section. As mentioned before, the need of these classes is realized from the Virtual Test Bed Project. The basic aim of this library is to produce a modular code based on Object-Oriented principles.

3.2 UML Diagrams

Designing a complex simulation model is always associated with software engineering. Software engineering involves the utilization of the Unified Modeling Language (UML). Hence, UML forms a solid base for developing complex simulation models [19].

UML supports an iterative and incremental development process, promoting component

based architecture and the use of graphics and diagrams. All essentials of structures and dynamics of a simulation model can be easily described and the requirements of a simulation model can be easily formulated using UML. UML allows identification of reusable component in a simulation model. In simulation study interaction diagrams, state diagrams and class diagrams all are very useful. Each diagram will now be examined from a simulation aspect.

3.2.1 Interaction Diagrams

Interaction diagram depicts the interaction between objects/classes over time. There are two types of interaction diagrams: sequence and collaboration. Both diagrams are especially helpful in simulation study, because events can be mapped as messages (interaction) between the objects. Events can be plotted with respective objects in timely order [20], as shown in figure 9.

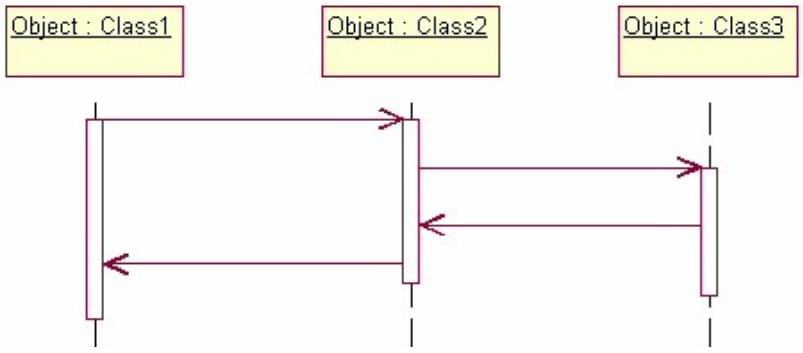


Figure 9: Example of Interaction Diagram

An interaction diagram describes the dynamic behavior of the simulation model. In this study, interaction diagrams were extensively used for explaining message passing between created library and external simulation classes. Interaction diagrams also help to understand the expected way of scheduling the events and using the objects from this library.

3.2.2 State diagrams

In Discrete-Event Simulation, simulation objects change their state over the period of time. It is very important to understand the different states a simulation object can possess. A state represents a stage in the behavior pattern of the simulation object. State diagrams depict the various states that an object may be in and the transitions between those states [27].

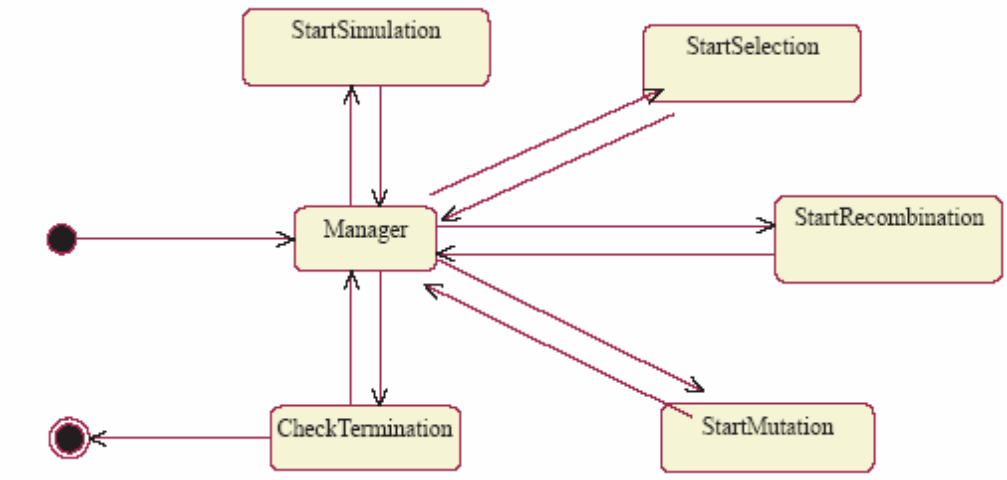


Figure 10: Example of State Diagram

As shown in the figure, a state has a distinct name and sequence of activities involved to achieve the state. States are represented by the values of the attributes of the simulation object and state transition is the result of the invocation of a method that causes an important change in the state. In this thesis, state diagrams are used to study different states the objects from this library can possess, and the events causing the state transition.

3.2.3 Class Diagrams

The purpose of a class diagram is to depict the classes used within a model. Classes have attributes, procedures and relationship with other classes. A class diagram can show the attribute visibility, association, multiplicity, constraints, inheritance, aggregation, and dependency in between other classes [21]. Static relationships between classes are shown in figure 11 using the class diagrams.

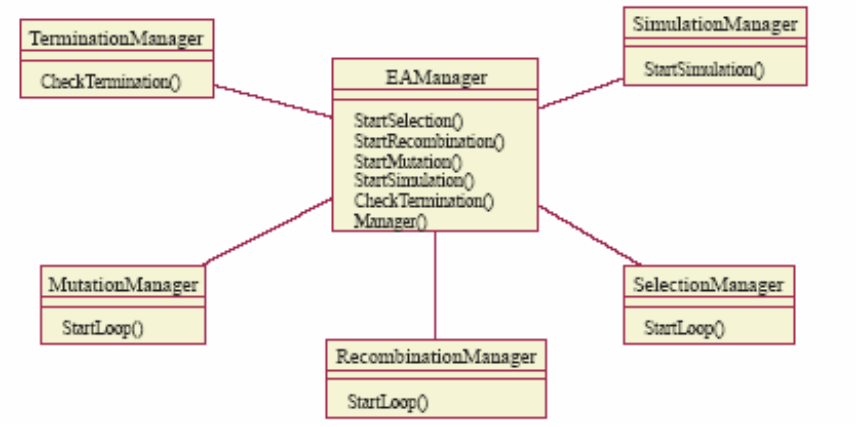


Figure 11: Example of Class Diagram

As shown in the diagram, a class diagram contains the methods in the simulation class, which are typically mapped as events. Attributes are shown with their data types. In this study class diagrams are used to describe internal structure of the classes in the created library and each type of the UML diagram shows a different view of the library. UML diagrams can be used as specification and documentation of the classes for this study. The following is the study of each class with details of its implementation.

3.3 Server Class

The Server is used to model the processing of entities using resources for a definite amount of time. The Server class in this library contains queue, and resource as built-in functionality. When an entity needs to be processed by the server, it is added to the server queue. After a required number of resources are available, the entity is removed from the queue and gets processed for a given amount of time. As soon as this processing finishes, the resource is released and the entity moves to the next module.

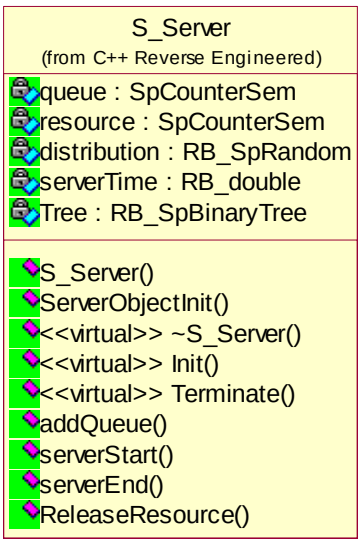


Figure 12: Server Class Diagram

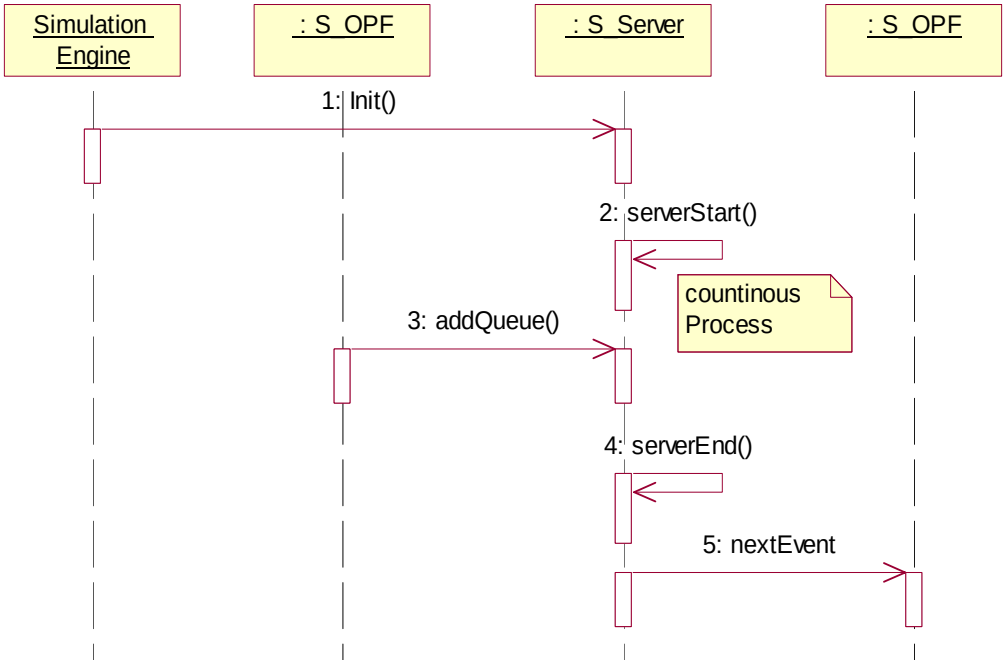


Figure 13: Server Sequence Diagram

3.3.1 Discussion of Server Class Code

The Server class uses the functionality of an Event based model as well as a Process based model. Functioning of the server starts from the 'Init' method as any other simulation class. The 'Init' method, triggered by the simulation engine at the start of simulation triggers the process model. The Process model runs continuously in 'serverStart' method and keeps checking the queue of the server. If any entity is present in the queue, and resource is also available for processing, this entity is taken out of the queue and scheduled for given processing time.

File Edit View Insert Project Build Tools Window Help

```
void S_Server::addQueue(SpObjHandle entityHandle,double processTime,int rel)
{
    Entity* E1 = RB_NEW_Entity();
    E1->Server_Handle = entityHandle;
    E1->Processing_Time= processTime;
    E1->Release_Resource = rel;
        Tree.Insert(E1,SpGetTime());
        queue++;
    //RB_cout<<"\nIn add queue";
}

void S_Server::serverStart()
{
    P_VAR;
        P_LV(Entity*,E2);
    P_LV(double,arrivaltime);
        P_BEGIN(2);
        {
    WAIT_FOR(1,queue,-1)
    WAIT_FOR(2,resource,-1)
    E2=(Entity*)Tree.RemoveFirstElement(arrivaltime);
        SCHEDULE_serverEnd(SpGetTime()+ E2->Processing_Time,SpGetObjHandle(),E2);
    queue--;
    resource--;
        }
        P_END;
}

void S_Server::serverEnd(Entity* E3){
    if(E3->Release_Resource == 1) resource++;
    RB_cout<<"\nIn server end"<<"Time="<<SpGetTime();
}

void S_Server::setResource(int res){
    resource = res;
```

Figure 14: Server Class code

To schedule processing of any entity by the Server, the entity is first added to the queue of the server by scheduling the event ‘addQueue’. The Process model running in the ‘serverStart’ event schedules a ‘serverEnd’ event with this entity after the given processing time has elapsed. The ‘serverEnd’ event scheduled at the end of processing time releases the resource and also schedules the next event on an entity outside the server. The information of the next event is found in the data interaction object, which is sent as an argument when scheduling the server event. The ‘serverObjectInit’ method is used for dynamic creation of a server, so that library files need not be modified to specify the number of the servers required in any simulation model.

As recommended by SPEEDES, we do not use constructors and destructors in any class. Rather we use ‘Init’ method to initialize the object and ‘Terminate’ to perform clean up operations after deletion of an object.

3.3.2 Queue

The ‘RB_SpBinary Tree’ data structure available in SPEEDES is used to implement the queue. The Queue is implemented on a First in First out (FIFO) principle by prioritizing all events on an entry time basis. The Init method sets up the tree object to implement as a queue. When any class sends an entity to the Server, it schedules an ‘addQueue’ event to add the entity in queue, and the server takes care of the further processing. In the ‘startServer’ process model, the execution of an event is halted until there is an entity in the queue. As soon as an entity is added to the queue, the queue count is incremented and

the process model sets off for the next halt, waiting for the required resource. When the resource is available, the process model takes the entity out of the queue, the queue count is decremented and the entity is scheduled for the ‘serverEnd’ event.

3.3.3 Resource

Resource is the basic functionality required for implementation of the server. Resource capacity can be set by an outer class using the ‘setResource’ method. In the process model ‘serverStart’, event execution halts until enough resources become available. As soon as the resources become available, the model proceeds by taking an entity out of the queue and sending it to the ‘’serverEnd’ event. In ‘serverEnd’, the resource is released and the count of a resource object is incremented. The flexible structure of this library allows resources to be released from outer class using the event ‘ReleaseResource’. Since, the queue count and resource are both declared as semaphore types, they can be used in process model to halt the execution.

3.3.4 Server Arguments class

The Server Arguments class is a data interaction class used to send the required data to the Server class in a specialized way. As server arguments are not basic data type, they need to be wrapped in an object before sending them to the server.

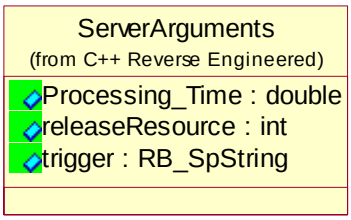


Figure 15: Server Argument Class Diagram

The Server Argument class shown in figure 15 contains the attribute 'Processing_Time', used to send the processing time in seconds generated by the external class. In this library, random number generation has been performed in the external class to reduce the complexity of the Server class. If in the future an advanced random number generation library is created for SPEEDES, then this generic library need not be modified to accommodate new random numbers. The external class will store the information of random number distribution, generate a random number and will send the number to the server through this wrapper class. The 'releaseResource' attribute stores the information of releasing the resource. The attribute 'trigger' is used to send the information about the next event to be scheduled following the server process, i.e., the 'serverEnd' event.

3.4 Decision Class

During a simulation, many times the next execution path is decided by probability. The Decision class shown in figure 16 can be used to model such probabilistic decision making processes in the SPEEDES simulation model. Currently, this class supports up to

four way decisions, but it is easy to enhance the class for more branches (see figure 17). The specialty of the decision class is; only one instance of the class is required for a complete simulation model, to make all kinds of probabilistic decisions.

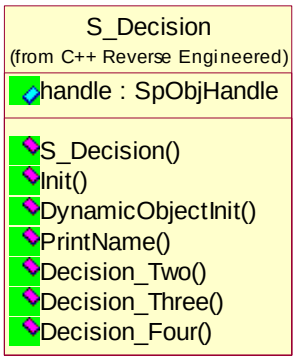


Figure 16: Decision Class Diagram

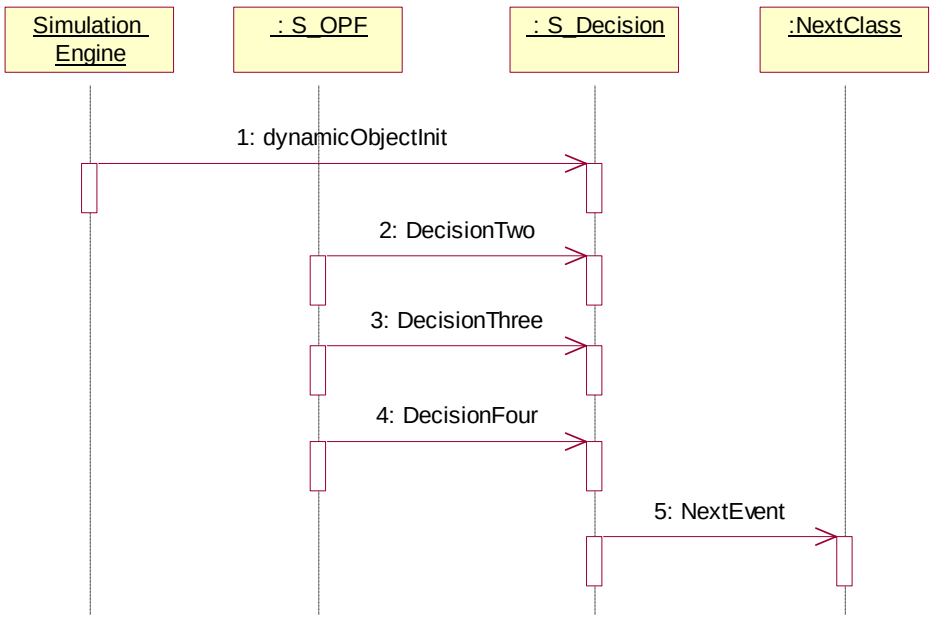


Figure 17: Decision Class Sequence Diagram

3.4.1 Decision Argument Class

The Decision Argument class shown in figure 18 is used as a wrapper class for sending arguments. Since, arguments of the decision class are not standard data types; they need to be wrapped in an object. The Decision Argument class contains a trigger for the first event of the branch and the probability of occurrence. One object of the Decision Argument class is required per decision branch in the Decision class. Hence, for a four-way decision, four objects of the Decision Argument class are sent to the Decision class.

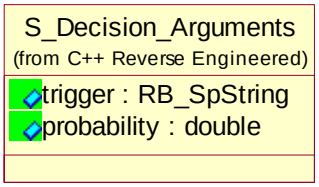
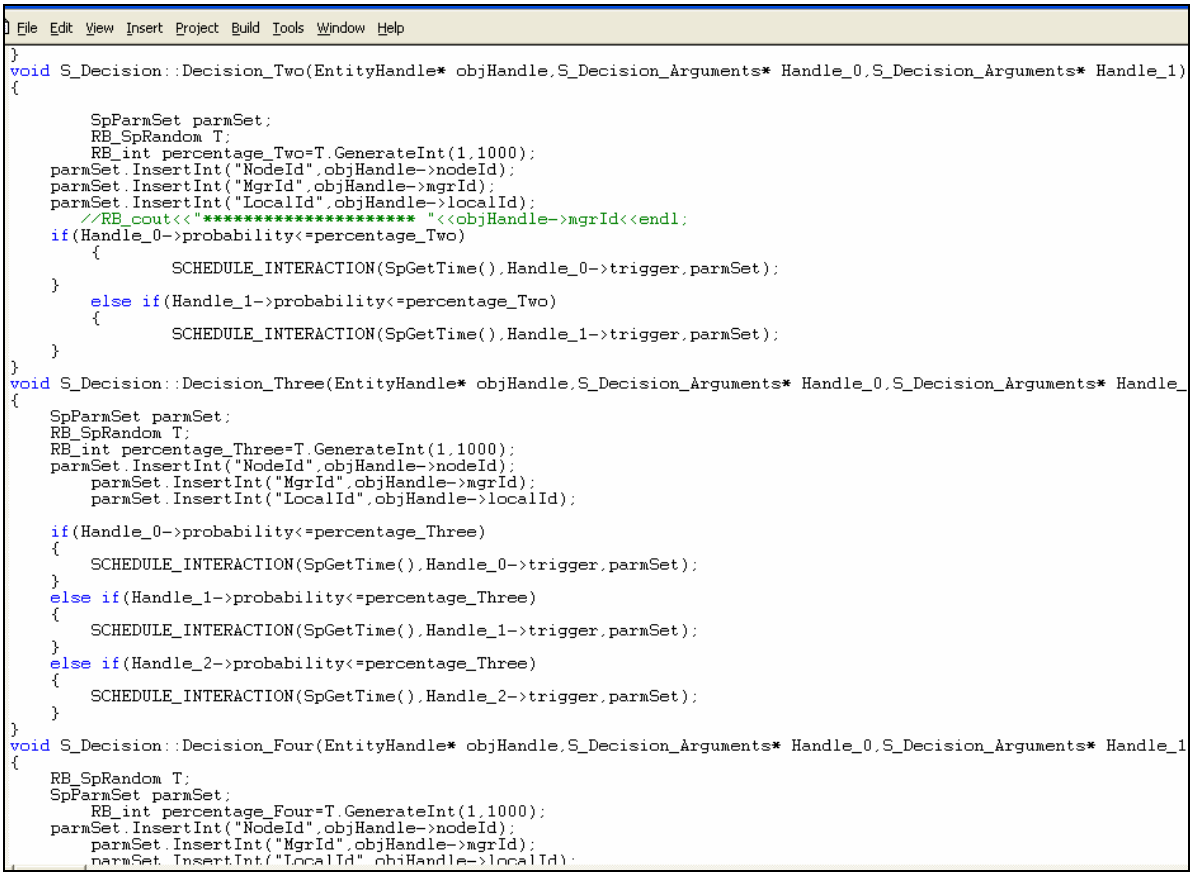


Figure 18: Decision Argument Class Diagram

3.4.2 Discussion of Decision Class Code

An object of the Decision class is created dynamically by scheduling the event 'DynamicObjectInit' on the Decision class. The Decision class has three methods to support multi-way decisions: Decision_Two, Decision_Three, and Decision_Four. These methods are mapped as events which can be scheduled by external class to implement the decision. These events require triggers, and the probability of each branch, wrapped in a Decision Argument class object. All probabilities need to be in the form of percentage

and must sum to 100.



```
File  Edit  View  Insert  Project  Build  Tools  Window  Help
}
void S_Decision::Decision_Two(EntityHandle* objHandle,S_Decision_Arguments* Handle_0,S_Decision_Arguments* Handle_1)
{

        SpParmSet parmSet;
        RB_SpRandom T;
        RB_int percentage_Two=T.GenerateInt(1,1000);
    parmSet.InsertInt("NodeId",objHandle->nodeId);
    parmSet.InsertInt("MgrId",objHandle->mgrId);
    parmSet.InsertInt("LocalId",objHandle->localId);
       //RB_cout<<"******************** "<<objHandle->mgrId<<endl;
    if(Handle_0->probability<=percentage_Two)
        {
                SCHEDULE_INTERACTION(SpGetTime(),Handle_0->trigger,parmSet);
    }
        else if(Handle_1->probability<=percentage_Two)
        {
                SCHEDULE_INTERACTION(SpGetTime(),Handle_1->trigger,parmSet);
    }
}
void S_Decision::Decision_Three(EntityHandle* objHandle,S_Decision_Arguments* Handle_0,S_Decision_Arguments* Handle_
{
    SpParmSet parmSet;
    RB_SpRandom T;
    RB_int percentage_Three=T.GenerateInt(1,1000);
    parmSet.InsertInt("NodeId",objHandle->nodeId);
        parmSet.InsertInt("MgrId",objHandle->mgrId);
        parmSet.InsertInt("LocalId",objHandle->localId);

    if(Handle_0->probability<=percentage_Three)
    {
        SCHEDULE_INTERACTION(SpGetTime(),Handle_0->trigger,parmSet);
    }
    else if(Handle_1->probability<=percentage_Three)
    {
        SCHEDULE_INTERACTION(SpGetTime(),Handle_1->trigger,parmSet);
    }
    else if(Handle_2->probability<=percentage_Three)
    {
        SCHEDULE_INTERACTION(SpGetTime(),Handle_2->trigger,parmSet);
    }
}
void S_Decision::Decision_Four(EntityHandle* objHandle,S_Decision_Arguments* Handle_0,S_Decision_Arguments* Handle_1
{
    RB_SpRandom T;
    SpParmSet parmSet;
        RB_int percentage_Four=T.GenerateInt(1,1000);
    parmSet.InsertInt("NodeId",objHandle->nodeId);
        parmSet.InsertInt("MgrId",objHandle->mgrId);
        parmSet.InsertInt("LocalId",objHandle->localId);
```

Figure 19: Decision Class code

Whenever any decision event on a decision object is triggered, a uniform integer between 1 and 1000 is generated by the function ‘GenerateInt’ from the ‘SpRandom’ object. Each probability is multiplied by 10 and the generated integer is compared to the ranges formed by the probabilities between 1 and 1000. Whenever a matching range is found, the trigger associated with that range is scheduled to follow that path. The trigger of the

next event is scheduled by the macro ‘SCHEDULE_INTERACTION’. As mentioned earlier, only one decision instance can serve the objective of probabilistic decision in any model.

3.5 Transporter Class

Transporter is one of the most frequently used modules in Discrete-Event Simulation. The Transporter class in this library, shown in figures 20, works similar to Transporter module in Arena. When an entity needs to be transported, it adds itself to the transporter queue. When the transporter is available, the entity seizes the transporter and gets transported. After reaching its destination, the entity frees the transporter and transporter either attends other calls or goes back to the home station.

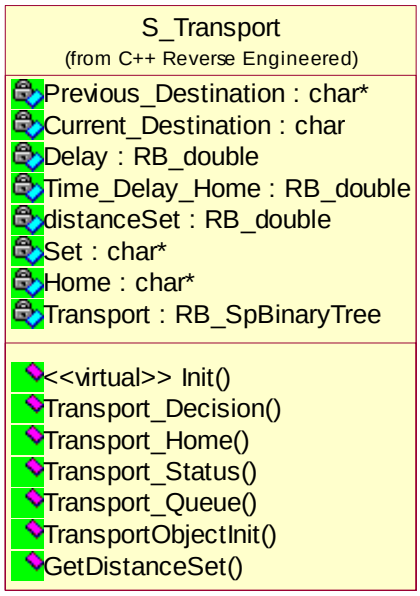


Figure 20: Transporter Class Diagram

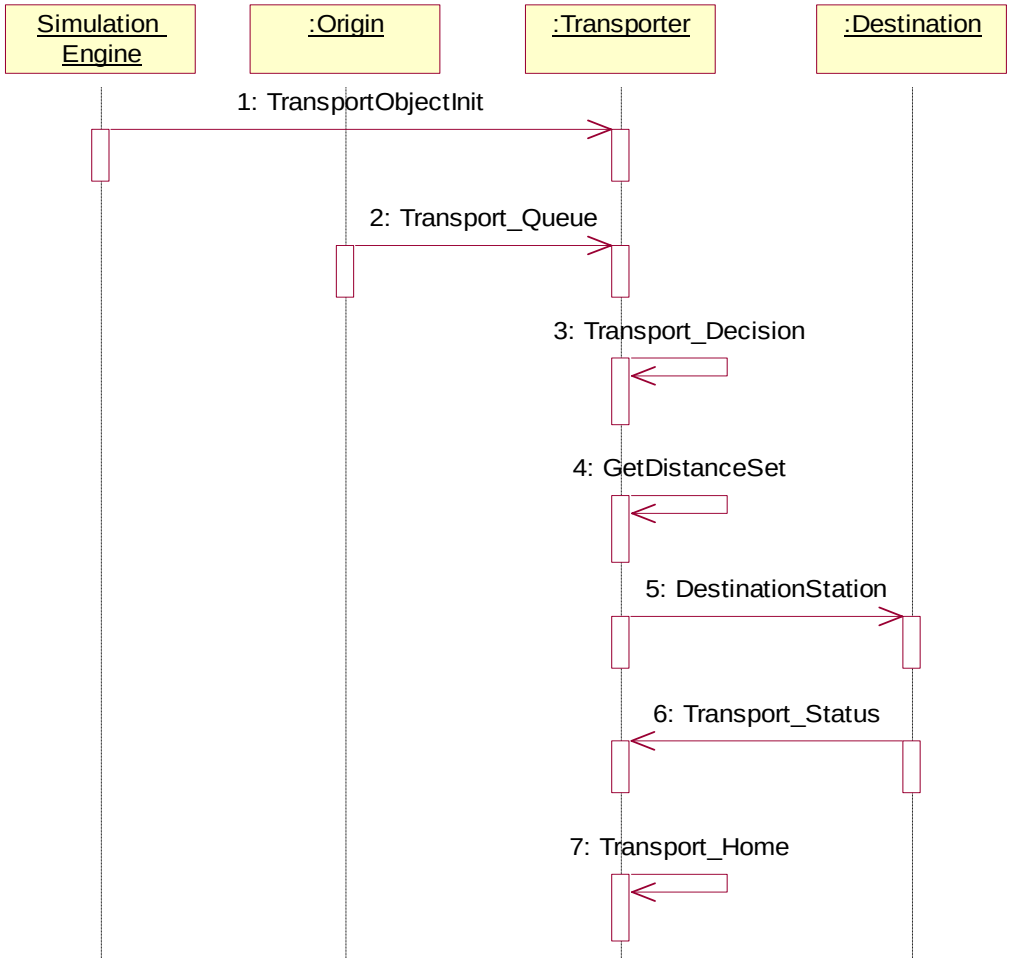


Figure 21: Transporter Class Sequence Diagram

3.5.1 Transporter Argument Class

The Transporter Argument class is used to supply the arguments to the transporter into wrapped format. Whenever an entity wants to be transported it supplies information of origin and destination to the transporter by creating the instance of the Transporter

Argument class and sending this instance as an argument to the transporter.

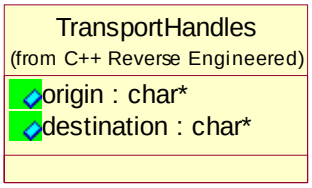
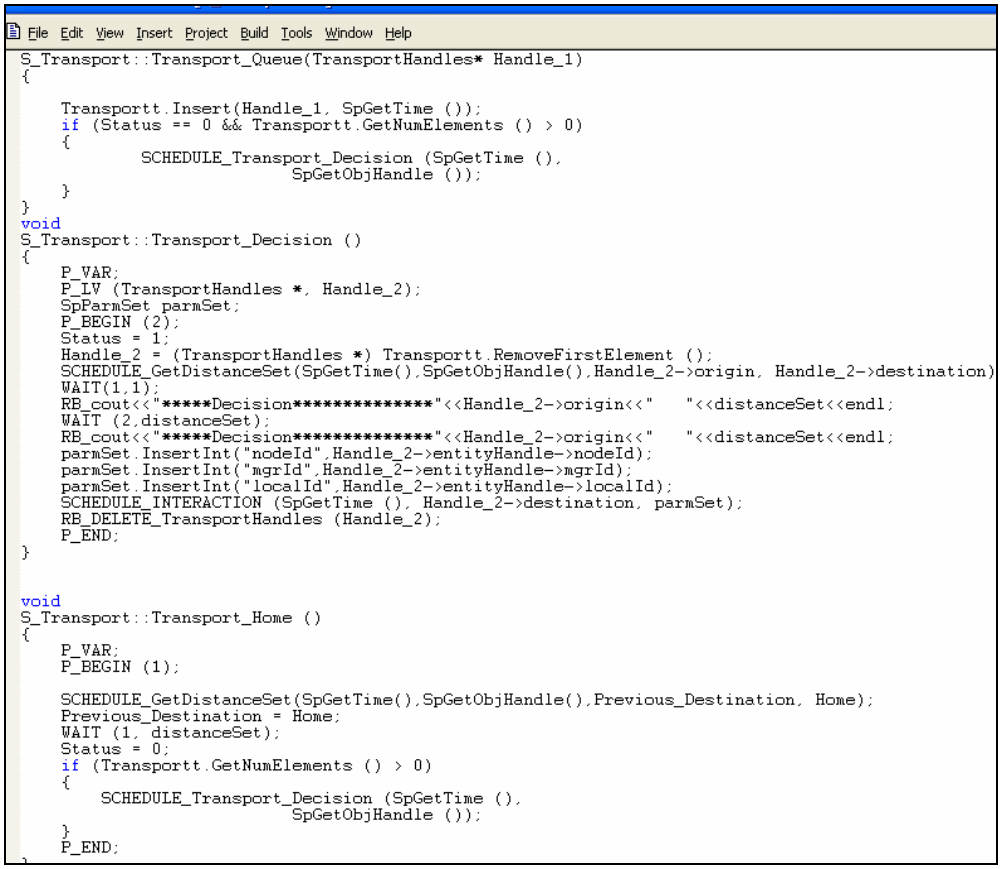


Figure 22: Transporter Argument Class Diagram

3.5.2 Discussion of Transporter Class Code

As shown in the UML diagram, the Transporter class is initialized dynamically at the start of the simulation by the event 'TransportObjectInit'. This event also assigns a distance set and a home station to the transporter. Whenever any entity wants to use the transporter it needs to schedule the 'Transport_Queue' event on that transporter. This event will add the entity in the transporter queue and check the availability of the transporter. If the transporter is available, a 'Transport_Decision' event will be scheduled on the transporter. 'Transport_Decision' is a process model, which continuously checks the queue of the transporter and removes the entity for transportation if the transporter is free. This function retrieves the origin and destination information from the entity by un-wrapping the Transporter Argument object. After getting origin and destination information, the 'GetDistanceSet' event is scheduled to get the transportation time in between stations from the Distance Set parser file. The Distance set parser file holds the information of transportation time in between the all possible pairs of stations for each

transporter. The transporter status is changed to busy and the entity is scheduled for the destination event after transportation time delay.



```
File  Edit  View  Insert  Project  Build  Tools  Window  Help
S_Transport::Transport_Queue(TransportHandles* Handle_1)
{

    Transportt.Insert(Handle_1, SpGetTime ());
    if (Status == 0 && Transportt.GetNumElements () > 0)
    {
            SCHEDULE_Transport_Decision (SpGetTime (),
                        SpGetObjHandle ());
    }
}
void
S_Transport::Transport_Decision ()
{
    P_VAR;
    P_LV (TransportHandles *, Handle_2);
    SpParmSet parmSet;
    P_BEGIN (2);
    Status = 1;
    Handle_2 = (TransportHandles *) Transportt.RemoveFirstElement ();
    SCHEDULE_GetDistanceSet(SpGetTime(),SpGetObjHandle(),Handle_2->origin, Handle_2->destination)
    WAIT(1,1);
    RB_cout<<"*****Decision**************"<<Handle_2->origin<<"    "<<distanceSet<<endl;
    WAIT (2,distanceSet);
    RB_cout<<"*****Decision**************"<<Handle_2->origin<<"    "<<distanceSet<<endl;
    parmSet.InsertInt("nodeId",Handle_2->entityHandle->nodeId);
    parmSet.InsertInt("mgrId",Handle_2->entityHandle->mgrId);
    parmSet.InsertInt("localId",Handle_2->entityHandle->localId);
    SCHEDULE_INTERACTION (SpGetTime (), Handle_2->destination, parmSet);
    RB_DELETE_TransportHandles (Handle_2);
    P_END;
}


void
S_Transport::Transport_Home ()
{
    P_VAR;
    P_BEGIN (1);

    SCHEDULE_GetDistanceSet(SpGetTime(),SpGetObjHandle(),Previous_Destination, Home);
    Previous_Destination = Home;
    WAIT (1, distanceSet);
    Status = 0;
    if (Transportt.GetNumElements () > 0)
    {
        SCHEDULE_Transport_Decision (SpGetTime (),
                        SpGetObjHandle ());
    }
    P_END;
```

Figure 23: Transporter Class Code

When the entity reaches the destination event, the destination class will release the transporter by scheduling a ‘Transport_Status’ event, which changes the status of transporter. This event also checks the transporter queue. If any entity is waiting in the queue, transporter is assigned to that entity, otherwise scheduled for home station.

3.6 Entity Class

An Entity class is required to generalize the simulation entity definition in the library. This library works with the entity class; hence if any simulation entity, like a shuttle, is required to be sent to the object of this library, its information needs to be copied to an Entity class object.

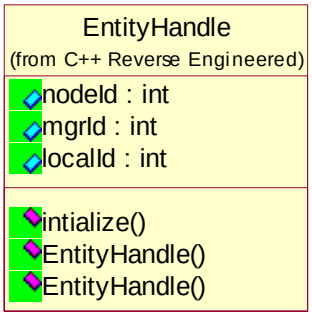


Figure 24: Entity Class

As shown in the figure, an entity class contains nodeID, managerID, and localID. In most of the classes in this library, events need to be subscribed and published, for calling by the other objects. SPEEDES does not allow the direct object handles as arguments for subscribing and publishing the events. Hence, information in the handle needs to be copied to entity object before publishing the trigger.

3.7 Synopsis

This chapter explains the technical details of the classes in the library. The task of creating these classes is challenging as Linux is a very restricted language and we were trying to create a library for general use. This is the first attempt to create such a generic library under Linux. This Library has most of the frequently used modules in any Discrete-Event Simulation. In the next section, we report the implementation results of this library in SPEEDES Space shuttle model.

CHAPTER 4: IMPLEMENTATION AND VALIDATION OF THE SHUTTLE MODEL

4.1 Preface

The classes in the generated library are tested and used to recreate the SPEEDES Space Shuttle Model. This chapter discusses the testing of the classes, implementation of the library and validation of the recreated SPEEDES Shuttle Model against the model in Arena.

4.2 Testing the Library

We have the Arena Basic Process Panel as an example of a predefined library. Requirements are formulated by keeping the functionality of Arena in mind. After completing the coding, each class is checked against these requirements. Though SPEEDES is not as flexible as the SIMAN language used for Arena, we try to match the requirements as closely as possible. During the testing some dummy scenarios are created to test the functionality. A dummy OPF (Orbiter Processing Facility) class to create the objects and a Main class to run the simulation model are used. The testing code is set up by including several output statements.

When coding of Server class is finished, 3-4 server objects are created in the

dummy OPF class and an entity is passed through each of the servers one after another. This Server class was tested for single resource capacity as well as multiple resource capacity. Late release of resources in server is also tested by releasing the resources in a class outside the server. To test the decision class two way, three way and four way decisions are implemented in dummy OPF class. Different events representing different paths are created in OPF class. The Decision class is tested to make sure that it is making the decisions and scheduling events according to the given probability. Testing the transporter is one of the challenging tasks as the transporter is more like a complex version of the Server class. It is tested to deliver the entity in the same class and in a different class. The library is tested to a satisfactory level and is expected to work according to the set requirements.

4.3 Implementation of the Shuttle Model

As the part of implementation, the SPEEDES Space Shuttle Model is recreated using the classes in this library. Each already existing class in SPEEDES shuttle model is closely examined for the purpose of finding the opportunities of using the objects from this library. To use the library, many changes are made to the Shuttle model. Following diagrams show the hierarchy of the classes formulated in this implementation. The library classes are extended from SPEEDES base class ‘SpSimObj’.

4.3.1 Class Hierarchy

As shown in figure 25, the existing shuttle model does not have any hierarchy. All the classes are made for a specific purpose and the model does not use any predefined components. For similar functionalities, the same code is reproduced which results in a large amount of coding and more development time.

In the recreated shuttle model, the classes from the generated library are in the middle level of the hierarchy, as shown in figure 26 and 27. For similar functionalities, the code from the library classes is reused. Due to use of the objects from these classes, the development time in the recreated Shuttle Model is reduced to a great extent.

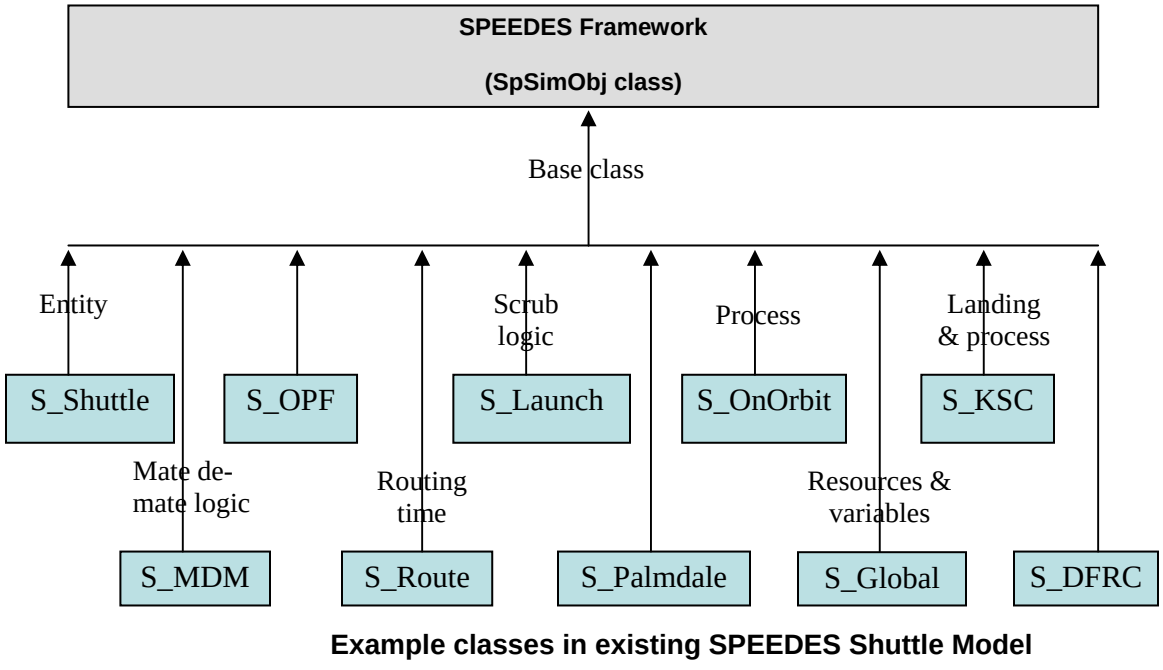


Figure 25: Class hierarchy in existing SPEEDES Shuttle Model

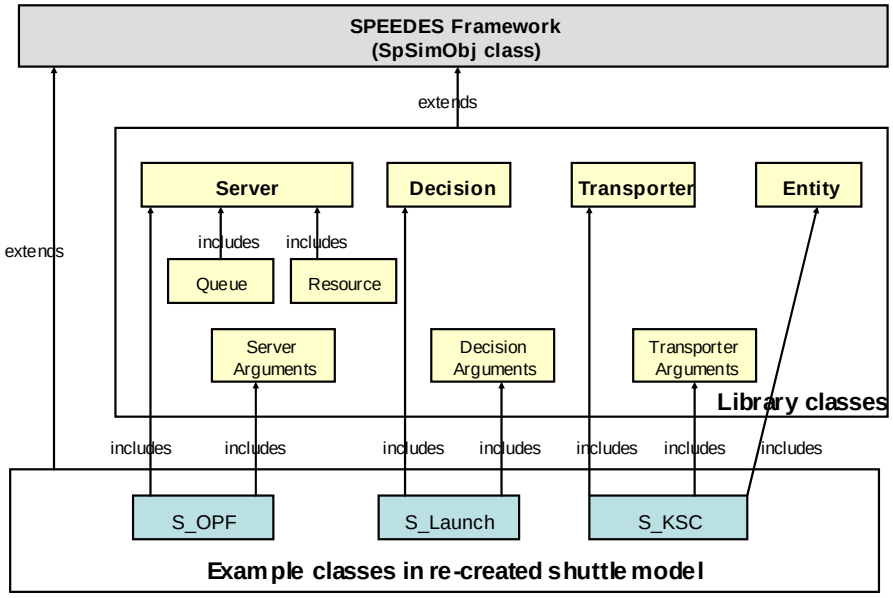


Figure 26: Class hierarchy in recreated SPEEDES Shuttle Model -1.

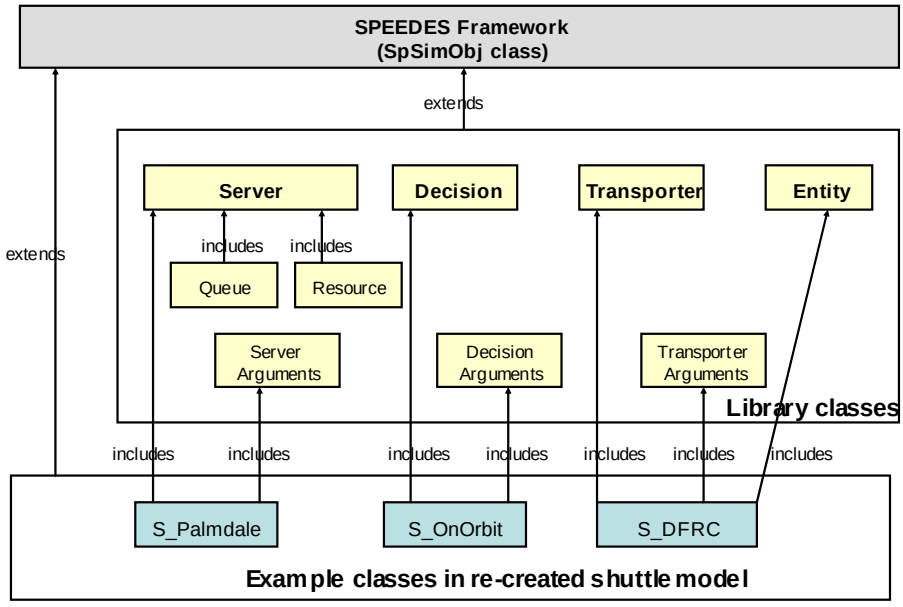


Figure 27: Class hierarchy in recreated SPEEDES Shuttle Model -2.

The objects from this library are included in the class definition of the recreated Shuttle Model. The reason behind including the objects of the library in class definition are of the Shuttle Model is any number of objects could be added in the same class. For example, the Palmdale class has two server processes. To model this logic, the Palmdale class includes two instances of the Server class.

4.3.2 Comparison of Existing and Replaced Code

In this section, we compare the length of the code for different modules in the existing SPEEDES Shuttle Model and the recreated SPEEDES Shuttle Model.

Existing Server Code	Replaced Server Code

void S_Orbiter::TAL_Queue(SpObjHandle shuttle)
{
OrbiterHandles* Handle_5 = RB_NEW_OrbiterHandles();
Handle_5->Orbiter_Handle = shuttle;
OR.Insert(Handle_5,SpGetTime());
Indi2++;
 	
}
void S_Orbiter::TAL()
{
 	
P_VAR;
P_LV(OrbiterHandles*,Handle_6);
P_BEGIN(2);
 	
while(1)
{
WAIT_FOR(1,Indi2,-1);
Handle_6=(OrbiterHandles*)OR.RemoveFirstElement();
//-----------------------TAl_Landing=1--------
SCHEDULE_Shuttle(SpGetTime(),Handle_6->Orbiter_Handle
//--------------------TAL_Descent=0 --------------
SCHEDULE_TAL_O(SpGetTime(),SpGetObjHandle("S_Global_M
Capture_TAL_Descent_R=0.3*60*60*24;
WAIT(2,Capture_TAL_Descent_R); //time for capturing
//--------------------TAL_Descent=1 -----------------
SCHEDULE_TAL_O(SpGetTime(),SpGetObjHandle("S_Global_M
SCHEDULE_Orbiter_Decision(SpGetTime(),SpGetObjHandle(
Indi2--;
}
P_END;
	ServerArguments* han = new ServerArguments();
	han->capacity=3;
	han->release=true;
	han->trigger=nextevent;
	han->processingTime=Distribution(10,100);
	eHandle->intialize(shuttle.GetNodeId(),shuttle.
	han->entityHandle=eHandle;
	SCHEDULE_Server_Queue(SpGetTime(),transportHand

Figure 28: Sample Server Code Comparison in SPEEDES Shuttle Model

As seen in the chart above, the server code was significantly reduced due to use of the Server class from the library (see figure 29 and 30). In the existing SPEEDES Shuttle Model, a separate code is required for the Server, Queue and Resource functionality, increasing the complexity of the model. Since, in the new Shuttle Model the Queue and Resource are embedded in the Server class definition, the implementation is much simpler.

Existing Decision Code	Replaced Decision Code

probability=T.GenerateInt(1,100000);
prob=((probability)/(1000));
 	
if(prob<=0.047)
{
SCHEDULE_ATO(SpGetTime(),SpGetObjHandle(),Han_2->Orb
}
if((prob>0.047)&&(prob<=0.189))
{
SCHEDULE_Mecco(SpGetTime(),SpGetObjHandle(),Han_2->O
}
if((prob>0.189)&&(prob<=0.331))
{
SCHEDULE_TAL_Queue(SpGetTime(),SpGetObjHandle(),Han_
}
if((prob>0.331)&&(prob<=0.897))
{
SCHEDULE_RTLS(SpGetTime(),SpGetObjHandle(),Han_2->Orb
}
if((prob>0.897)&&(prob<=100))
{
SCHEDULE_Process(SpGetTime(),SpGetObjHandle(),Han_2->
}
	_Decision_Arguments* hand= RB_NEW_S_Decision_Argument
	and->probability =90.0;
	and->trigger="LaunchSuccess";
	_Decision_Arguments* hand1= RB_NEW_S_Decision_Argumen
	and1->probability =10.0;
	and1->trigger="LaunchAbort";
	CHEDULE_Decision_Two(SpGetTime(),lookupObjHandle,hand

Figure 29: Sample Decision Code Comparison in SPEEDES Shuttle Model

Existing Transporter Code	Replaced Transporter Code

```
void PlugInTrain()
{
      PLUG_IN_SIMOBJ(S_Train);
      PLUG_IN_EVENT(Train_Decision);
      PLUG_IN_EVENT(Train_Home);
      PLUG_IN_EVENT(Train_Status);
      PLUG_IN_EVENT(Train_Queue);
}


void S_Train::Init()
{
      Traint.SetSplayTreeMode();
      Previous_Destination=0;
        Status=0;
      i=0;
      for(i=0;i<=3;i++)
      {
            Delay_Set[i][i]=0;
      }
      Delay_Set[Home][UTAH]=120960;
      Delay_Set[UTAH][Home]=120960;
      //RB_cout<<"      Delay Set    "<< Delay_Set[0][1]<<Previous_Destination<<endl;
      Delay_Set[Home][Hangar_AF]=Delay_Set[Hangar_AF][Home]=17280;// (2/24)*60*60*24;
      Delay_Set[Home][RSRM_In_KSC]=Delay_Set[RSRM_In_KSC][Home]=17280;
      Delay_Set[UTAH][Hangar_AF]=604800;
      Delay_Set[Hangar_AF][UTAH]=120960;
      Delay_Set[UTAH][RSRM_In_KSC]=604800;
      Delay_Set[RSRM_In_KSC][UTAH]=120960;
      Delay_Set[Hangar_AF][RSRM_In_KSC]=Delay_Set[RSRM_In_KSC][Hangar_AF]=17280;

}

void S_Train::Train_Queue(SpObjHandle shuttle,int Orig,int Destiny)
{
      TrainHandles* Handle_1=RB_NEW_TrainHandles();
      Handle_1->Train_Handle=shuttle;
      Handle_1->Origin=Orig;
      Handle_1->Destination=Destiny;


      ------------------------------------;
      if(Status==0 && Traint.GetNumElements()>0)
      {
            SCHEDULE_Train_Decision(SpGetTime(),SpGetObjHandle());
      }
}

void S_Train::Train_Decision()
{
      P_VAR;
      P_LV(TrainHandles*,Handle_2);
      P_BEGIN(1);

if(Status==0 && Traint.GetNumElements()>0)
{
      Status=1;
      Handle_2=(TrainHandles*)Traint.RemoveFirstElement();
      Delay=Delay_Set[Previous_Destination][Handle_2->Origin]+Delay_Set[Handle_2->Origin][Handle_2->Destination];

      //RB_cout<<"Previous_Destination"<<Previous_Destination<<"   Origin  "<<Handle_2->Origin<<"    Destination  "<<Handl

      WAIT(1,Delay);
      if( Handle_2->Destination==UTAH)
      {
            SCHEDULE_Queue_UTAH(SpGetTime(),SpGetObjHandle("S_SRB_MGR",0),Handle_2->Train_Handle);
      }
      else if(Handle_2->Destination==Hangar_AF)
      {
            SCHEDULE_Hangar_AF(SpGetTime(),SpGetObjHandle("S_SRB_MGR",0),Handle_2->Train_Handle);
      }
      else if(Handle_2->Destination==RSRM_In_KSC)
      {
            SCHEDULE_RSRM_IN_KSC(SpGetTime(),SpGetObjHandle("S_SRB_MGR",0),Handle_2->Train_Handle);
      }

}
else


else
{
      Status=1;
      Time_Delay_Home=Delay_Set[Previous_Destination][Home];
      Previous_Destination=Home;
      SCHEDULE_Train_Home(Time_Delay_Home+SpGetTime(),SpGetObjHandle());
}
P_END;

}

void S_Train::Train_Home()
{
      Status=0;
      if(Traint.GetNumElements()>0)
      {
            SCHEDULE_Train_Decision(SpGetTime(),SpGetObjHandle());
      }
}
void S_Train::Train_Status(int Current_Destiny)
{
      Status=0;
      Previous_Destination=Current_Destiny;
      SCHEDULE_Train_Decision(SpGetTime(),SpGetObjHandle());
}
```

```
//Intialize:

SpEnableDynSimObjLookup();
SCHEDULE_TransportObjectInit(SpGetTime(),dynamic,0,"Train","Home");

Getting Dynamic Object Handle :

simObjMgrId = SpGetSimObjMgrId("S_Transport_MGR");
GET_DYN_OBJ_HANDLE(1,SpGetTime(),simObjMgrId,ane,transportHandle);

//Scheduling:
TransporterHandle han = new TransporterHandle();
han->origin="Hangar_AF";
han->destination="UTAH";
eHandle->intialize(shuttle.GetNodeId(),shuttle.GetSimObjMgrId(),shut
han->entityHandle=eHandle;
SCHEDULE_Transport_Queue(SpGetTime(),transportHandle,han);

//Releasing:
SCHEDULE_Transport_Status(SpGetTime(),transportHandle,"UTAH");
```

Figure 30: Sample Transporter Code Comparison in SPEEDES Shuttle Model

As seen in the figure 29 and 30, a large amount of transporter code is replaced with the generated library. In the existing SPEEDES Shuttle Model, each transporter had separate .CPP and .H files. But in the recreated Shuttle Model, a generic transporter class replaced all these files. Finally, in this implementation, we incorporate about 15 Server objects, 15 Queue objects, 15 Resource objects, 5 Transporter objects and 4 Decisions objects from the generated library.

4.3.3 Flowcharts of the Implementation

Objects from this library are used in recreating the SPEEDES Shuttle Model. This discussion is followed by the flowcharts of the Server, Decision, and Transporter functionality in the recreated model using the generated library. Many opportunities are identified for use of the Server class objects. The Server functionality is much more straightforward and easy to replace. Since, the Resource and Queue objects are embedded in the server class; the amount of code required is significantly reduced. Similarly, most of the probabilistic decisions are replaced by the object from the Decision class. The key advantage of using the Decision class is that only one global decision object is required for the complete simulation model, as any decision process does not cause a delay.

Most of the transporter functionality is also replaced with the objects from the transporter class. Since, the transport functionality is not very well-defined in the existing SPEEDES Shuttle Model; code had is changed to use the transporter class. The figures 31 through 33 show all the events and state changes in the implementation of the

SPEEDES shuttle model.

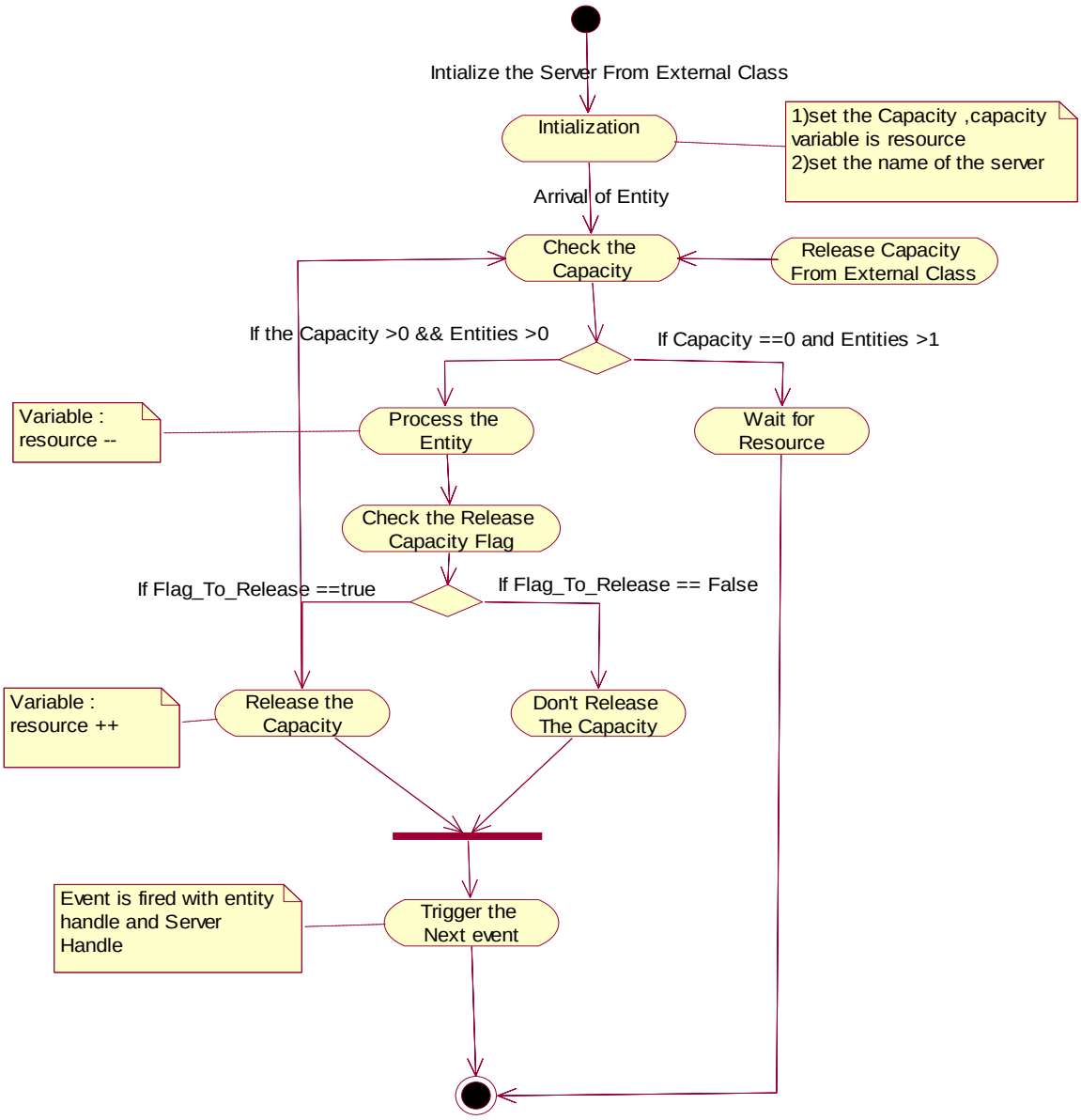


Figure 31: Server Implementation/Activity Diagram

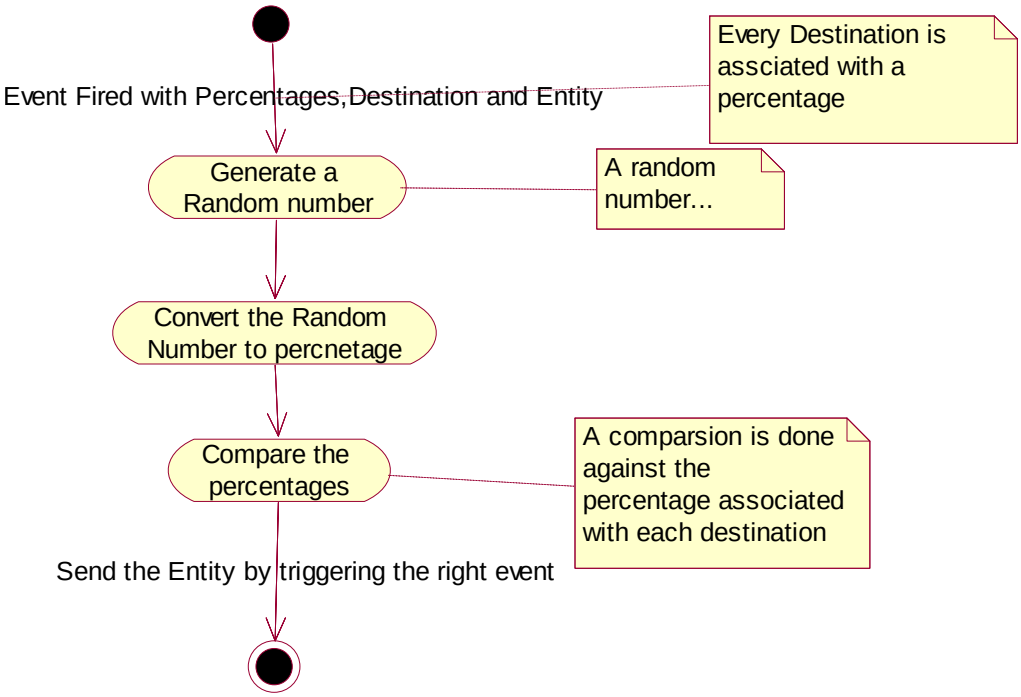


Figure 32: Decision class Implementation/Activity Diagram

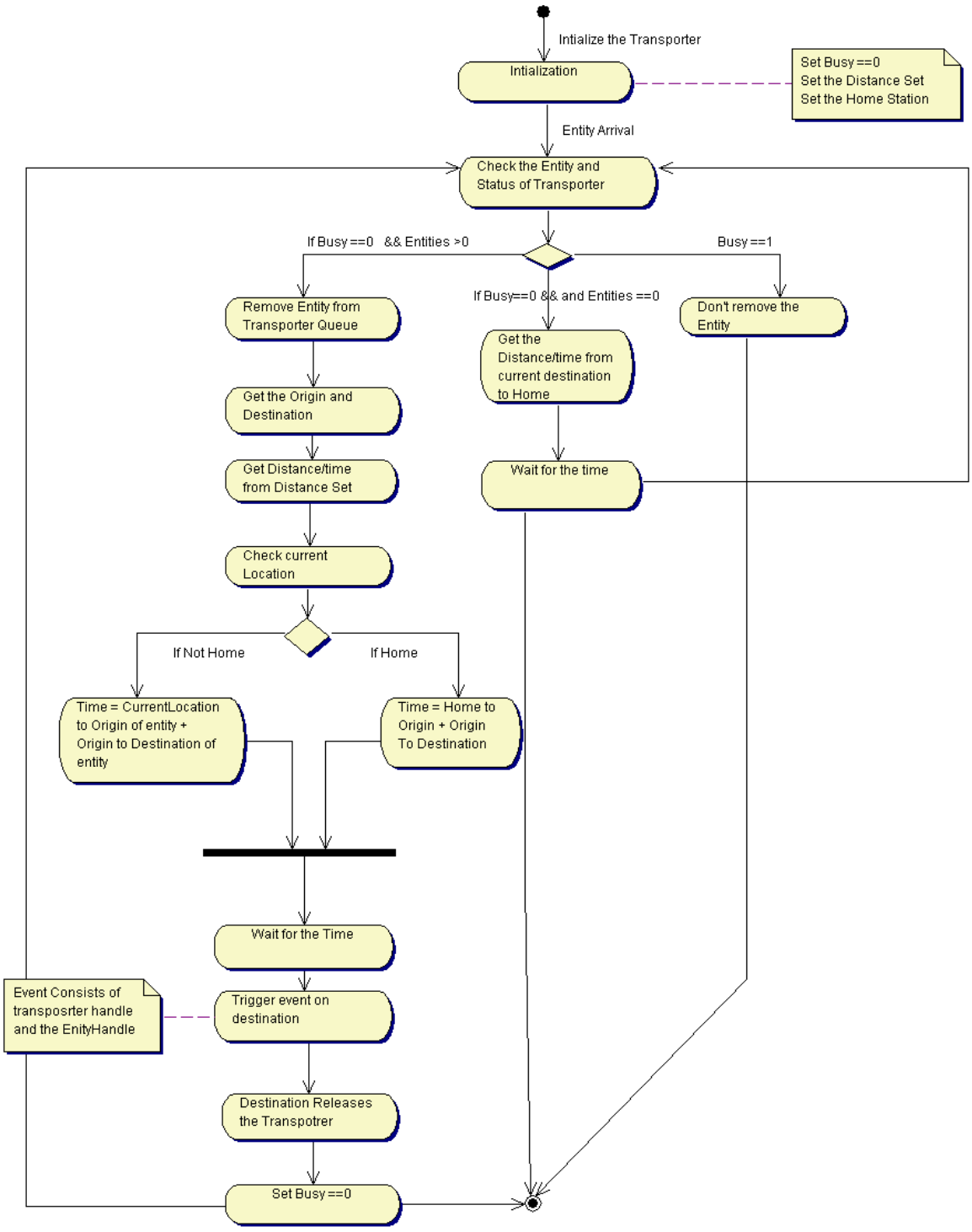


Figure 33: Transporter class Implementation/Activity diagram

4.3.4 Dynamic Creation of Objects

In the old shuttle model the numbers of entities required for each class are defined at compile time. This result in excessive memory usage and extra burden to precisely estimate the number of the objects required for the Shuttle Model. During development of the shuttle model, many duplicate entities are created to implement the complex logic. Hence, the developers have to keep track of all duplicates and make sure that existing duplicates are used before declaring a new one.

While implementing the shuttle model using this library, the advanced feature of Dynamic Object Creation is used. Whenever any entity, server, decision, or transporter object is required, it is created dynamically, resulting in the easy implementation of the model. Dynamic Object Creation results in improved performance such as increased execution speed, low memory usage, and high scalability of the model [16]. The Dynamic Object Creation feature can be used in all future SPEEDES simulation models.

4.3.5 Publishing and Subscribing the Events

This is one of the major changes in the SPEEDES Shuttle Model. The existing model used simple Point-to-Point method for event scheduling. In Point-to-Point method, the scheduler needs to have complete information of the event to be scheduled, which results in a hard-coded, tightly coupled static model. For a simple change such as event name, a lot of the code has to be revisited to check the integrity of the program.

Since, we use a generic library in the recreated shuttle model the events have to be

made flexible. The objects from the created library would not have any information about next event to be scheduled, as that event would be outside the library. Hence, instead of Point-to-Point events, event handlers are used. Event Handlers works on the principle of late binding of the events. To explain this feature let us take the example of the Server class from the library. In the recreated shuttle model, the external class can send the entity to the server by scheduling the regular Point-to-Point event, without any change. While scheduling server event, a trigger (string) of the event following the server is also sent. The next event (event following the server) needs to subscribe itself as the listener to the sent trigger. After the server finishing the processing of the entity, it publishes that trigger using special macros, resulting in scheduling of the subscribed event outside the server. This mechanism is successfully implemented all over the recreated shuttle model to include the objects of newly created classes from the library.

4.3.6 Use of External Distance Set

In implementation of the SPEEDES shuttle model, all the distance sets were defined in XML style, in the SPEEDES parameter file. The advantage of this scheme was distance/time between stations could be changed without recompiling the shuttle model. This is helpful in creating different scenario by changing the distance set values outside the classes (see figure 34). Each transporter is linked to a fixed distance set, resulting in an Arena-like implementation of the model.

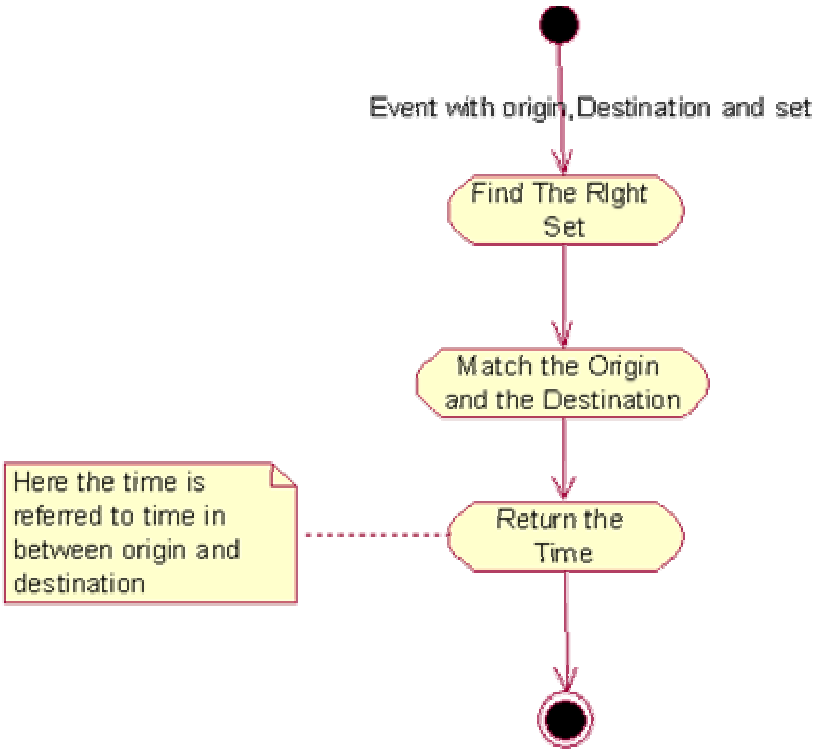


Figure 34: Implementation of Distance sets for use in Transporter

4.4 Validation of Shuttle Model

The recreated shuttle model is validated against original Arena model. The criterion used for validation is the Number of Flights completed in the given years. Both models are run for 10 years and 50 replications. A Confidence interval around the mean of Number of Flights in the Arena model is calculated. The Mean of Number of Flights from recreated SPEEDES shuttle model is fitted into that interval.

As the following Table 2 shows, the mean of recreated SPEEDES Shuttle Model fits within the lower and upper confidence limits of the Arena Shuttle model. Hence, we

can validate that the SPEEDES shuttle model closely resembles the Arena model. The means of each data set also match closely; though do not match exactly due to different random numbers streams used.

Table 2: Comparing the results from Arena and SPEEDES

	Run Time	No. of replications	Mean	Std. Dev.	95 % Conf. Interval	
					Low	High
Original Arena Model	10 years	50	69.2	1.45	66.34	72.05
Recreated SPEEDES Model	10 years	50	70.2	2.05		

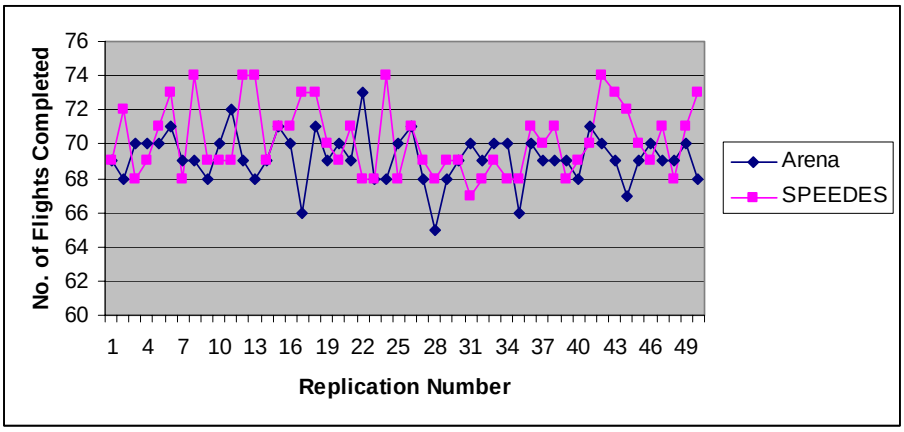


Figure 35: Chart of Comparison

The above chart of comparison shows that value of validation parameter from recreated SPEEDES Shuttle Model closely follows the Arena Shuttle Model, supporting our conclusion.

4.5 Synopsis

In this chapter we discuss the testing of the generated library with implementation and validation of the shuttle model. New features of SPEEDES are researched and used in this implementation for better performance. The implemented shuttle model serves as an example of using this library for simulations using SPEEDES. Users of this library can refer to the implemented shuttle model as a reference example.

CHAPTER 5: CONCLUSIONS AND FUTURE RESEARCH

This chapter provides conclusions of the results obtained from the use of the created Discrete-Event Simulation library. Further, this chapter also discusses the future scope of enhancements in the library.

5.1 Conclusion

The reusable Discrete-Event Simulation library for SPEEDES is developed following an Object-Oriented approach. The details of the class structure are illustrated with UML diagrams. The classes in the library are tested and validated against the formulated requirements. The classes are used to recreate the SPEEDES NASA Space Shuttle Model and this model is validated against the original Arena NASA Shuttle Model. The results of the validation are shown in the previous chapter. The advantages gained with use of the library are explained as follows:

- Modular, easy to maintain, and truly Object-Oriented code;
- Rapid creation of the model due to use of predefined components;
- Less amount of work required to create the same model;
- Allows ease and flexibility in modifying the model; and
- Permits runtime changes in input parameters;

5.2 Contributions

This simulation library is based upon its requirements from the Virtual Test Bed (VTB) project. There are contributions to the VTB in creating the structure of the library. The library contributes to the rapid development of simulation models using SPEEDES. The users of SPEEDES will greatly benefit with use of this library as it will also help standardize the implementation of operations simulation using SPEEDES.

5.3 Future Work

Employing the reusable simulation library is a significant improvement in the model creation process in SPEEDES. This library is created for the operations simulation domain. Hence, it has frequently used modules for operations simulation such as server, and transporter. However, other domains like transportation simulation and electric simulation have their own sets of frequently used modules. Enhancements in this library can be done by adding the classes used for other domains. This will extend the area of application of SPEEDES.

Another considerable improvement can be, providing a graphical interface to this library. Due to the lack of a graphical interface, the model creation process in SPEEDES is somewhat complicated. Therefore, in future, if this library is provided with an Arena like drag and drop interface, where the user can select the right class for implementation, the areas of application of SPEEDES will be broadened.

Automated generation of code using extensible modeling language is also one of

the potential areas of research in SPEEDES. With this new approach, users will specify the parameters of the simulation model and create an XML schema. The code generator of the extensible modeling language will use the set of XML schemas as input and will produce the C++ implementations for each attribute defined in the XML schema [23].

XML's meta-language aspect and extensive tool support makes it a potential technology to build such modularly extensible modeling languages. Using XML document transformation technology, it has become easier to develop custom code generators. The code generator will work like a parser which will parse the XML schema and generate the code for simulation models. XML has a fixed grammar, which has enabled the development of excellent parsers for code generation [24]. The benefits of this approach in the SPEEDES will be model driven programming, automatic update propagation and a higher degree of consistency enforced by a generative approach.

LIST OF REFERENCES

1. Joines J.A, Roberts S.D. (1998) “Fundamentals of Object-Oriented Simulation”, Proceedings of the 1998 Winter Simulation Conference D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan. http://www.informs-cs.org/wsc98papers/018.PDF
2. Stroustrup B. (1991) “What is Object-Oriented Programming?” AT&T Bell Laboratories Murray Hill, NJ 07974 http://www.research.att.com/~bs/whatis.pdf
3. Marzolla M. “Distributed Simulation of Large Computer Systems” INFN Padova and Dept. of Computer Science, Univ. of Venice, Italy http://www.ihep.ac.cn/~chep01/paper/8-010.pdf
4. Paruchuri A., Wasadikar A., Marin F., Sepulveda J.A., Rabelo L. (2004) “Parallel Discrete-Event Simulation of Space Shuttle Operations, Design, Development, and Performance using SPEEDES.” Winter Simulation Conference 2004
5. Ball P. (1996) “Introduction to Discrete-Event Simulation” 2nd DYCOMANS workshop on Management and Control: Tools in Action" in the Algarve, Portugal. May 15-17 1996, pp. 367-376. http://www.dmem.strath.ac.uk/~pball/simulation/simulate.html
6. Kelton W., Sadowski P., Sadowski A., Sadowski P. (2002) “Simulation with Arena”, McGraw-Hill New York, NY.

7. Sodhi S. (2004) “Software Review – Flexsim”, OR/MS Today
http://www.lionhrtpub.com/orms/orms-8-04/swr.html
8. Buss A. (2001), “Technical Notes Basic Event Graph Modeling”,
Operations Research Department, Naval Postgraduate School Monterey, CA 93943-5000 U.S.A.
http://diana.gl.nps.navy.mil/~ahbuss/papers/BasicEventGraphModeling.pdf
9. Richard A. K. (2000), “Silk Java and Object-Oriented Simulation”, Proceedings of the 2000 Winter Simulation Conference
http://www.informs-cs.org/wsc00papers/037.PDF
10. Janssen M, Verbraeck A., and Henk G. S. “Agent-Based Simulation for Evaluating intermediate roles in electronic commerce” School of technology, Policy and Management, The Delft University of Technology, The Netherlands
http://www.betade.tudelft.nl/publications/JanssenVerbraeckSol_AGENT2000.pdf
11. Praehofer H., Sametinger J., Stritzinger A., “ Discrete-Event simulation Using the JavaBeans Component Model” Johannes Kepler University, Austria
12. Flüs C., Mohamed H., Müller-Clostermann B. (2001), “JavaDEMOS: Java-based Discrete-Event Simulation”
http://www.informatik.unibw-muenchen.de/mmb/mmb41/JavaDemosKurz.pdf
13. Flüs C., “Performance Engineering of Mobile Agents Using JavaDEMOS”, University of Essen.
http://www.cs.uni-essen.de/SysMod/publikationen/PE2002_CFlues.pdf
14. Sonnessa M. (2004), “JAS library – User guide”

http://jaslibrary.sourceforge.net/files/UserGuide.pdf

15. Federated Simulation Development Kit homepage, retrieved on 12/03/04 http://www.cc.gatech.edu/computing/pads/fdk/
16. "SPEEDES User Guide" (2003) Metron Inc Prepared for The joint national integration center 730 Irwin avenue Schriever AFB, CO 80912-7300 http://www.speedes.com/docs/SpUG.pdf
17. Preiss B.R. (1990) Performance of Discrete-Event Simulation on a Multiprocessor using Optimistic and Conservative Synchronization Department of Electrical and Computer Engineering University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1, Copyright (c) 1990 by The Pennsylvania State University http://www.brpreiss.com/papers/published/1990/icpp/paper.pdf
18. Erdei M., Sója K., Wagner A., "The remote OMNet++ distributed Discrete-Event Simulation environment" Budapest University of Technology and Economics
19. Heemink A.W., Dekker L., De Swaan Arons H., Smit I., Van Stijn Th.L. (2001), "UML based business systems Modeling and simulation" Proceedings of eurosim 2001 Shaping future with simulation. http://www.betade.tudelft.nl/publications/Shishkov_EUROSIM2001.pdf
20. Li X., Liu Z., and Jifeng H. (2004) "A Formal Semantics of UML Sequence Diagrams" proceedings of ASWEC2004, 13-16 April, 2004, Melbourne, Australia http://www.iist.unu.edu/newrh/III/1/docs/techreports/report292.pdf
21. Eichelberger H., "UML Class Diagrams - State of the Art in Layout Techniques" W¨urzburg University , Germany

http://www.cs.uvic.ca/~mstorey/vissoft2003/submissions/eichelberger_pospaper.pdf

22. Defense Modeling and Simulation Office website, retrieved on 12/03/04 https://www.dmso.mil/public/transition/hla/rti/
23. Dashofy E. M. (2001) "Issues in Generating Data Bindings for an XML Schema-Based Language" Proceedings of the Workshop on XML Technologies and Software Engineering (XSE2001), Toronto, ONT, Canada. http://www.ics.uci.edu/~edashofy/papers/xse2001.pdf
24. Sarkar S., Cleaveland C. "Code Generation Using XML Based Document Transformation, published on "TheServerSide Your J2EE Comminity" http://www.theserverside.com/articles/content/XMLCodeGen/xmltransform.pdf
25. Rizzoli A. E., IDSIA (2004), A Collection of Modeling and Simulation Resources on the Internet Galleria 2 CH - 6928 Manno Switzerland http://www.idsia.ch/~andrea/simtools.html
26. Law A.M., Kelton, D.W. (2000) Simulation Modeling and Analysis, McGraw-Hill, Inc.
27. Object Oriented Analysis and Design Team website, Kennesaw State University http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/state.htm