2005

Speedes: A Case Study Of Space Operations

Amith Paruchuri
University of Central Florida

SPEEDES: A CASE STUDY OF SPACE OPERATIONS

by

Amith Paruchuri
B.S., Jawaharlal Nehru Technological University, India 2001

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Modeling and Simulation
in the College of Engineering & Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2005

ABSTRACT

This thesis describes the application of parallel simulation techniques to represent the structured functional parallelism present within the Space Shuttle Operations Flow using the Synchronous Parallel Environment for Emulation and Discrete-Event Simulation (SPEEDES), an object-oriented multi-computing architecture. SPEEDES is a unified parallel simulation environment, which allocates events over multiple processors to get simulation speed up. Its optimistic processing capability minimizes simulation lag time behind wall clock time, or multiples of real-time. SPEEDES accommodates an increase in process complexity with additional parallel computing nodes to allow sharing of processing loads.

This thesis focuses on the process of translating a model of Space Shuttle Operations from a procedural oriented and single processor approach to one represented in a process-driven, object-oriented, and distributed processor approach. The processes are depicted by several classes created to represent the operations at the space center. The reference model used is the existing Space Shuttle Model created in ARENA by NASA and UCF in the year 2001. A systematic approach was used for this translation. A reduced version of the ARENA model was created, and then used as the SPEEDES prototype using C++. The prototype was systematically augmented to reflect the entire Space Shuttle Operations Flow. It was then verified, validated, and implemented.

ACKNOWLEDGMENTS

Special thanks to those in the VTB program at UCF who made this effort possible for without their guidance and support, this effort would not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1	DES	Discrete Event Simulation
2	DFRC	Dryden Flight Research Center
3	ET	External Tank
4	KSC	Kennedy Space Center
5	NASA	National Aeronautics and Space Administration
6	OPF	Orbiter Processing Facility
7	PDES	Parallel Discrete Event Simulation
8	SDES	Sequential Discrete Event Simulation
9	SRB	Solid Rocket Boosters
10	UCF	University of Central Florida
11	UML	Unified Modeling Language
12	VAB	Vehicle Assembly Building
13	VTB	Virtual Test Bed

CHAPTER 1: INTRODUCTION

1.1 Preface

This thesis focuses on translating an Arena shuttle model into SPEEDES (Synchronous Parallel Environment for Emulation and Discrete Event Simulation) and running the SPEEDES model on multiple computers. The model created in this thesis also proves that SPEEDES can be used for large NASA simulations, to run with distributed technology. The model translated to SPEEDES (Object-Oriented environment) was verified and validated against the original implementation in Arena (procedural-oriented). Different testing sessions were run to demonstrate the distributed simulation. With the generated model, different scenarios can be run to test the feasibility of the changes.

1.2 Distributed Simulation

As mentioned, SPEEDES can run in distributed environments. This means that the processing load can be divided on multiple nodes. This feature is very useful to run large simulation models. Different processors called nodes can maintain their own set of events and communicate with each other during simulation. Using this capability a large simulation model can be run over the Internet, having processors at different physical locations.

To demonstrate this feature the SPEEDES shuttle model was run on 4 different computers simultaneously. Different objects were distributed automatically among available nodes and simulation results were obtained. These results are also documented in the Findings Section.

1.3 Virtual Test Bed Project

The objective of the NASA/UCF Virtual Test Bed (VTB) Project is to provide a collaborative computing environment to support simulation scenarios, reuse, and integration of multidisciplinary models to represent elements of operations, range, and spaceports. The VTB project aims to create a common platform for testing and validating future NASA Space Shuttle Simulation Models [13].

The central goal of the VTB is to provide a virtual environment of the launch and range at KSC. VTB will be integrating and adapting some of the existing simulation models and complementing some of the gaps present, to create a unique mission environment for the Intelligent Launch & Range Operations (ILRO) program. This realistic NASA mission environment will provide scientists within the Intelligent Systems (IS) project with a computing environment where they can implement schemes for high-performance human-automation systems. This integration will require the development of a computer architecture that allows the integration of the different models and simulation environments [1].

1.4 SPEEDES Engine

SPEEDES has been chosen as the simulation engine to be used as it is based on NASA-patented algorithms and has a good set of documentation. SPEEDES is built and supported by a software company (Metron, Solana Beach, California), and it runs in Linux environments, ensuring high security of the NASA simulation models. This simulation engine will allow spaceport managers and decision makers to access and manage data and processes. It is a software framework/toolbox for building parallel C++ simulation models. SPEEDES allocates events over multiple processors to get simulation speed-up, (a feature which enhances runtime, especially when exploiting the very large number of processors) and the high-speed internal communications found in high performance computing platforms. The allocation of different processes on different processors can be controlled by the user.

The source of the operation process flow model was the NASA Shuttle Simulation Model. It is a simulation model for the operational lifecycle of the Space Shuttle flight hardware elements through their respective ground facilities at KSC developed by NASA and the University of Central Florida [2]. This Space Shuttle model was built using windows based discrete-event simulation software Arena 3.0. Arena is built and supported by the company Rockwell.

This shuttle model simulated the hardware flow and processing at different facilities at a macro level. Different distributions available in Arena were used to model the flow as accurately as possible. This NASA Space shuttle model also has the feature of animating the flow to visualize the process.

As the hardware parts of the Shuttle, this model included the following parts:

- Orbiter
- Main Engines
- Left and Right Maneuvering System Pods
- Left and Right Orbiter Maneuvering System Pods
- Forward Reaction Control System
- The Solid Rocket Boosters
- External Tank

As the ground facilities, this model included the following systems:

- Orbiter Processing Facility
- Vehicle Assembly Building
- KSC Landing Facility
- KSC Landing Pads
- Engine Shop
- DFRC Landing Facility
- Palmdale Facility

A stepwise approach was used to transfer the process flows from ARENA to SPEEDES. In this approach the shuttle model was broken into different small modules. These modules were representing the shuttle flow up to some logical point, which was enhanced in each module. These modules were transferred creating different versions in SPEEDES.

It took 7 versions to transfer all the modules into one complete model in SPEEDES. The Unified Modeling Language (UML) was utilized to develop the hierarchy of objects and for the generation of the C++ code whenever required. UML facilitates better understanding of the process.

While transferring this model to SPEEDES, the following classes were created to represent the shuttle hardware and different facilities:

Table 1: List of Classes

S_Choosing_OPF_Logic	S_Crawler	S_DFRC
S_Engine_Shop	S_External_Tank	S_Flow_Type_Logic
S_Global	S_HMF_Logic	S_Hyster
S_KSC	S_Launch	S_MDM
S_MLP	S_OPF	S_OPF_2_Assembly
S_OPF_3_Assembly	S_OPF_Assembly	S_OPF_Final_Assembly
S_OPF_Logic	S_Orbitor	S_PADA
S_PADB	S_Palmdale	S_Post_Palmdale_OPF_Operations
S_Retrivel_Vessel	S_Route	S_Pre_Palmdale_Logic
S_RPSF	S_Shuttle	S_SRB
S_Tow	S_Train	S_SRB_Stacking_Logic
S_VAB	S_Vessel	

The generated SPEEDES shuttle model was tested and validated against the Arena NASA Space Shuttle Model. This validation included running the model for different scenarios using different simulation time and multiple replications. These testing results are documented in Findings section.

1.5 Summary

This thesis work is a classic example of using SPEEDES for NASA simulations. The Procedural code in Arena was successfully translated into object oriented C++ using SPEEDES. Complete translation was achieved through many steps described in further chapters. Major success was achieved by running this model on multiple computers demonstrating distributed simulation feature of SPEEDES. Finally, the deliverable of this thesis work is a simulation model in SPEEDES.

CHAPTER 2: LITERATURE REVIEW

2.1 What is Discrete Event Simulation

Discrete-event simulation is a tool for modeling and simulating complicated systems. A system can consist of subsystems which interact in a coordinated fashion to represent the physical system. Events and objects are the primary building blocks of a discrete-event simulation. An event provides functions that can modify the state of the system and schedule other events in future or at current simulation time. Object represents a system or a set of subsystems in a simulation.

A discrete event simulation differs from a time based simulation. In a time based simulation the whole system which is being simulated needs to be updated at regular intervals. But in a discrete-event simulation updates are done only when needed. This difference or capability of discrete-event simulation can be used to optimize the run-time performance.

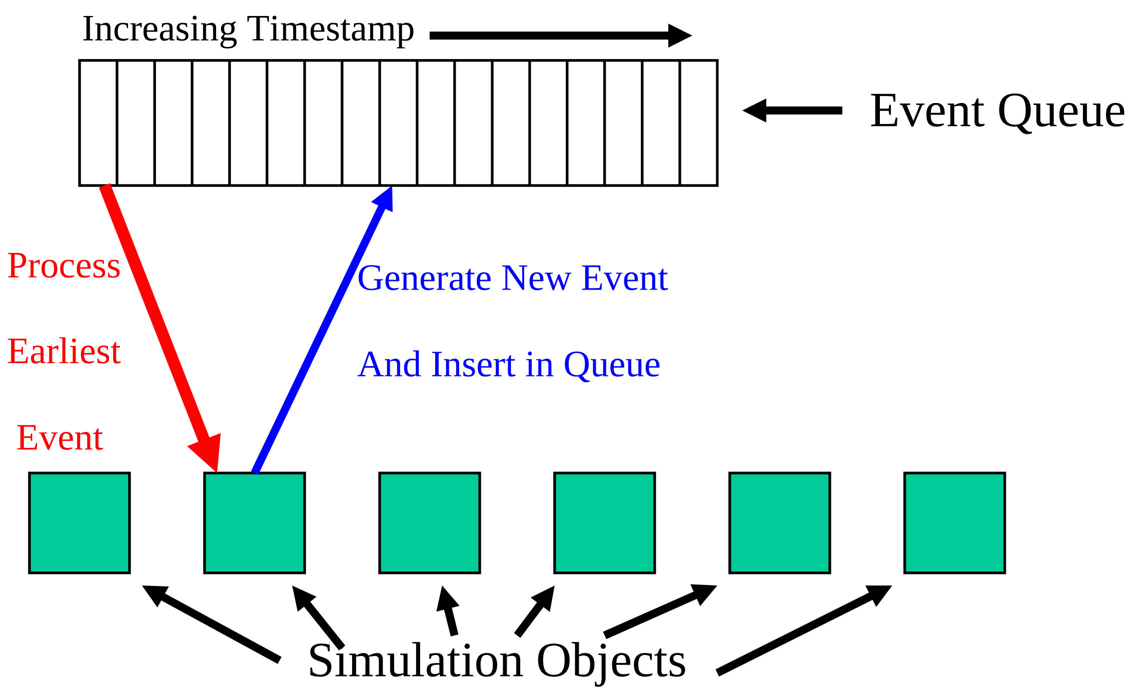


Figure 1: Discrete Event Simulation Event Ordering

Discrete-event simulation is one way of building up models to observe the time based (or dynamic) behavior of a system [3]. In discrete-event simulation, the event is the basic module of simulation. An event is any stored procedure changing the state of the system. One can also schedule another event to forward the execution. A central list of all the events in maintained in an event queue. The event queue contains all the events sorted by the priorities and time in most of the cases. These events execute one after another as the simulation clock forwards. Unlike a continuous simulation, the simulation clock jumps from one event time to another, which makes discrete event simulation faster.

Another basic module of DES is an Entity. An Entity is passed between events and each entity maintains its own state. Entities are physical elements found in the real world like a car, shuttle, machine etc. The most important component of a simulation is the Simulation Execution Engine. This engine controls the simulation clock, advances the event list and provides the basic functionality for a discrete-event simulation.

Another significant component in a discrete-event simulation is random number generation. Random numbers are used to bring dynamic behavior in the system and are used for server and transportation delays. Finally the simulation is run to analyze the system behavior which in turn will be achieved by result collection. These results are generally displayed in graphical manner or a structured format.

2.2 What is Distributed Discrete-Event Simulation

Distributing events of the simulation run on different nodes to execute in parallel is called as distributed discrete-event simulation. Each node (processor) maintains its own list of events and its own event clock. These clocks may run ahead or behind of each other. As a result, synchronization of events is the most important issue when considering distributed simulation.

2.2.1 Comparison between PDES and SDES

The events in a Sequential Discrete-Event Simulation (SDES) are executed sequentially on a single processor whereas in a Parallel discrete-event simulation (PDES), the events are processed in parallel on more than one processor. In SDES all the events are managed in a single event queue, the event with the earliest time stamp is removed for execution. In PDES each processor/node maintains its own event queue and various time synchronizing techniques take care of processing the events concurrently.

The advantages of PDES over SDES are computational speed-up, larger address spaces, more disciplined object-oriented approach and interoperability with other simulation systems. The greatest advantage of parallel simulation is that, the physical process can be matched with the logical process, i.e. when there are 2 servers performing the same job, they can independently execute on different processors. Parallel simulation is especially helpful when there is a large amount of dataset, or network because of large amount of events that are executed [15].

On the other hand, in a SDES, events can access or modify any of the system's state variables, but it cannot be done in PDES as the simulation is distributed on various processors. Another drawback of PDES is in that DES events are stored in a priority queue, based on time, which is a sequential process. So events executed by one process may affect or cancel other events. So it is necessary to maintain good communication between the processes in PDES. SPEEDES addresses this issue by providing a rollback feature.

PDES may not be a solution for every discrete-event simulation, sometimes SDES are much simpler to execute and can achieve more speed-up then PDES. PDES is mainly applicable to large military simulations, games or space simulations as examples.

2.2.2 Time Warp Algorithm and Breathing Time Buckets

To obtain minimum overheads in optimistic approach, various algorithms were generated [16]. Early methods of optimistic approach with anti-messages, with cascading rollbacks, are known as time warp method. But these methods showed signs of instability due to excessive overheads and bad workload overbalances. In the algorithm of Breathing Time Bucket, an event horizon in maintained and off node messages are released only when it is safe. While each node processes its pending events, the newly generated events are collected in an auxiliary queue. When the next event to be executed is in auxiliary queue, this queue is merged with main queue starting a new cycle. The benefit there is a no deadlock situation as processing events are defined by event horizon. Each node maintains its own local horizon and global horizon is the minimum of all local horizons. Once a node crosses the local horizon it broadcasts its local horizon to other nodes.

Breathing time Bucket algorithm may suffer from lot of synchronization and possibility of not enough events getting processed in a cycle [12].

2.2.3 Breathing Time Warp Algorithm

The time warp algorithm and the breathing time bucket algorithm suffer from exactly opposite drawbacks. The time warp algorithm woks with high risk by anti-message cascading system and on other hand the breathing time algorithm suffers much in terms of synchronization by holding back the messages. The breathing time warp combines these two algorithms and tries to overcome these drawbacks.

In this algorithm at the start of the cycle, events are sent out by a time warp algorithm taking risk and hence anti-message may be required if rollback takes place. A ‘N_{risk}’ factor is defined in ‘speedes.par’ to the specify number of events to be allowed with risk. After that the logic is switched to a breathing warp bucket algorithm, where new events are stored in an auxiliary queue and messages are held back. Now events which are rolled back do not need to send anti-messages as the messages were never released. At some time, the Global virtual time is updated to commit all the events and send the messages.

2.2.4 Past implementations of Distributed Simulation and Time Warp Algorithms

Frederick Wieland, Eric Blair, and Tony Zukas have presented a case study to implement parallel distributed simulation using SPEEDES. This paper concentrates on building a simulation model for National Airspace System (NAS) to study the average delay experience by aircraft from the beginning airport to the destination airport. NAS can be defined as a collection of airports, airfields, airspaces, the network of air routes connecting them and the system of navigation aids, radar sensors, and air traffic control that support flight through the Unites States.

This model is called as DPAT. The Simulation model was designed and developed explicitly to explore speed-up through parallel execution. It contains airport and sectors as data objects, solely to store the data and rest of the objects as event objects to execute the logic.

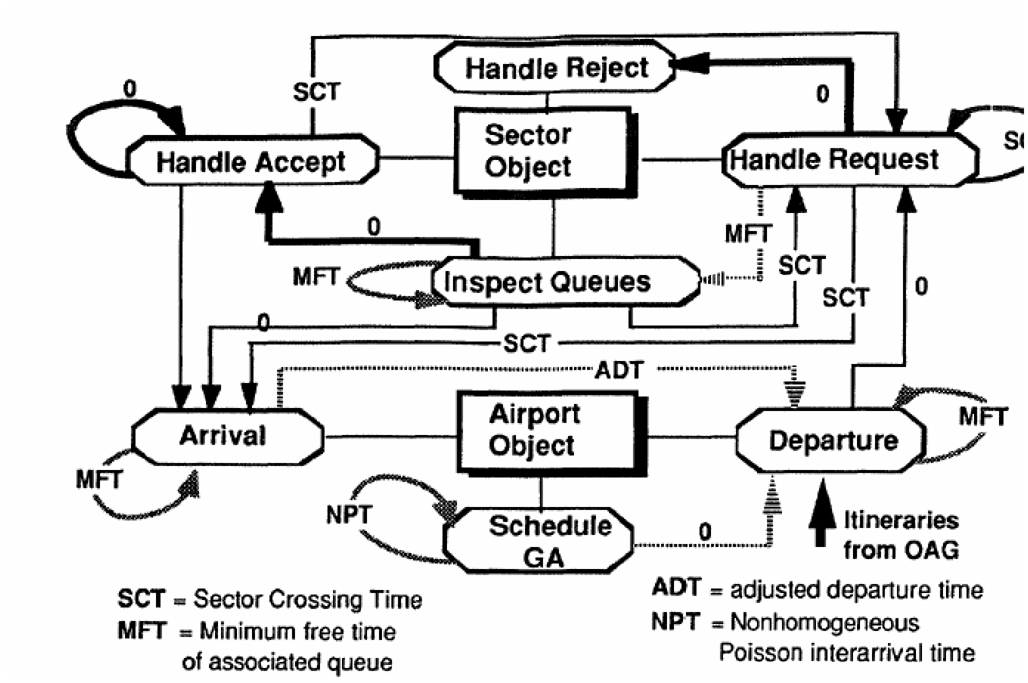


Figure 2: Software Architecture from DPAT, shown as SPEEDES diagram

This model was validated against NASPAC and other similar models of NAS. It was run on 3 SUN SPARC stations 2 of them consisting of 4 processors each and the other one having 1 processor. Despite the low granularity, use of networked workstations and relative lack of look-ahead, SPEEDES managed to extract speedup from the model. It was also observed that significant speedup was achieved by the Breathing Time Warp (BTW) protocol, which is a mixture of the fully optimistic Time Warp (TW) protocol and the risk-free Breathing Time Bucket (BTB) protocol [4].

2.3 What is SPEEDES

SPEEDES stands for Synchronous Parallel Environment for Emulation and Discrete-Event Simulation. It is an Object Oriented distributed discrete-event simulation framework developed in C++. SPEEDES executes discrete-event simulations in parallel by distributing the objects on the participating nodes/processors in the simulation. This feature is called as parallel simulation. SPEEDES can also implement High Level Architecture (HLA) federations of simulation.

2.3.1 Features of SPEEDES

SPEEDES provides a rich Object-oriented approach for developers to model and simulate various systems. As SPEEDES is a distributed discrete event simulation framework that allows distribution of various objects over multiple processors and co-ordinates the simulation activities among various objects that are distributed, it allows a simulation to perform optimistic parallel processing on high performance computers or network of different nodes [17].

SPEEDES provides interfaces for developing external modules. These modules provide functionalities that allow interoperability between various simulation systems and tools that will make sure the globally distributed simulation executes efficiently. External modules connected to a SPEEDES simulation control time advance of the simulation, receive information about simulation state and invoke events in the simulation. External modules can also be used to display the simulation status and provide inputs through hardware to control the simulation.

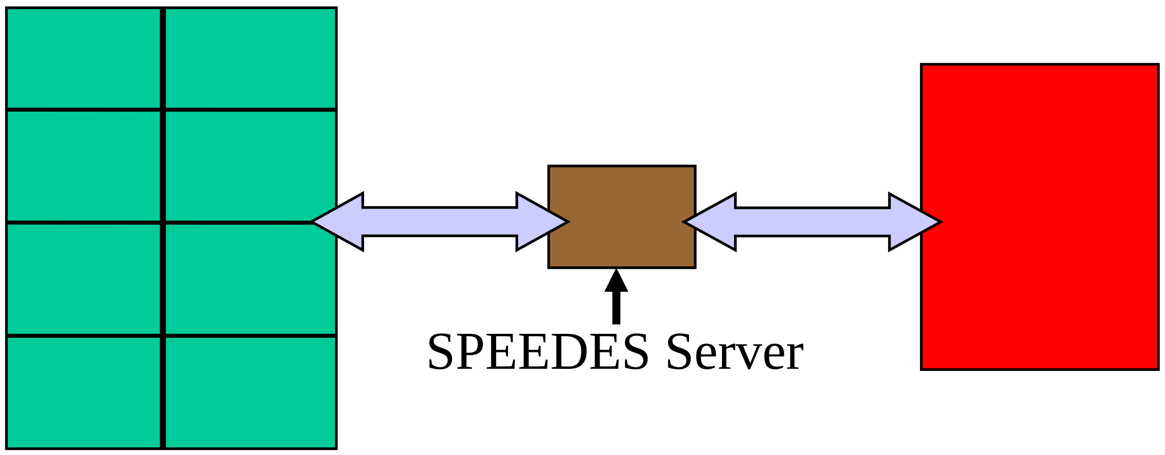


Figure 3: SPEEDES External Module

SPEEDES is HLA compliant. HLA stands for high level architecture, which was developed by the defense industry for supporting reuse and interoperability across large number of simulation models. HLA is the general purpose architecture for simulation reuse and interoperability which provides a gateway for communication between a federation of simulator's/federates [20]. Federation is a set of simulations, a common federation object .model and supporting infrastructure, which are used together to form a larger simulation model. Federate is the member of federation, used as one point attachment to the infrastructure [14].

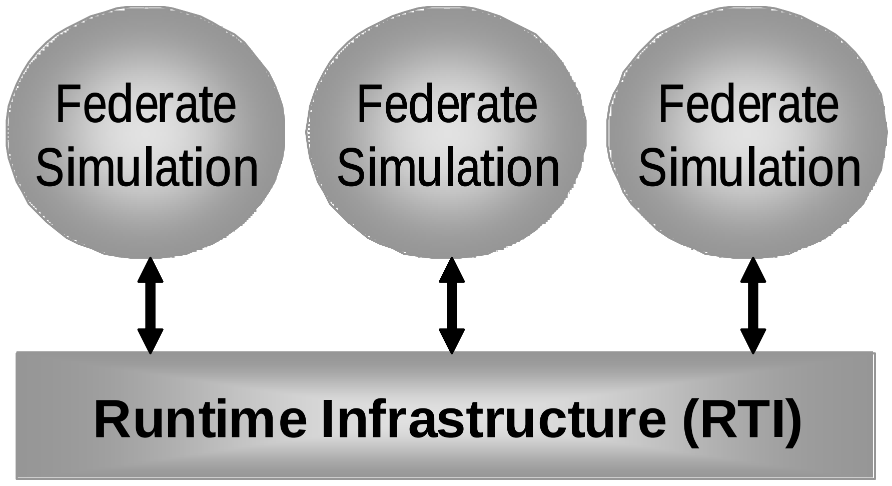


Figure 4: High Level Architecture

2.3.2 Simulation Object Decomposition

SPEEDES provides an advanced feature called Load balancing. This feature enables the user to balance the objects that require more processing on a faster processor, leading to increase in run time performance.

The distribution of objects in a SPEEDES simulation can be done in two ways; Automatic Object Placement and Manual Object Placement. Automatic Object Placement is done by two in-built algorithms called SCATTER and BLOCK. The SCATTER algorithm distributes objects like distributing cards in a card game. BLOCK distributes the objects evenly across the processors.

SPEEDES uses kind id for allocating the objects to different nodes. Kind ID is a unique ID starting at 0 for each kind of object. For the block algorithm, it calculates the total number of simulation objects, which can be placed on each node evenly. For example if we have 10 simulation objects with 3 nodes, then each node will receive 3 objects and last object will be allocated to the node next to the node where last simulation object was allocated.

Table 2: Block Decomposition

Kind ID	0	1	2	3	4	5	6	7	8	9
Node	0	0	0	0	1	1	1	1	1	1

The Scatter algorithm uses the card deal method for object distribution. The distribution starts at the node after the node where last object was allocated. So in the same above example each node will be allocated an object one after other.

Table 3: Scatter Decomposition

Kind ID	0	1	2	3	4	5	6	7	8	9
Node	0	1	2	0	1	2	0	1	2	0

There is a third decomposition method called File Driven Decomposition. In this decomposition, the user can specify in a file the allocation for each object on each node, giving more control to the user.

2.3.3 Rollback Feature of SPEEDES

SPEEDES can execute the DES model in parallel by distributing the objects on various nodes. Each node processes the events assigned to it, which may change the state of object, and/or schedule an event in past on some other node. To accommodate this change the already executed event on that node must be rolled back to bring the object in the past state. This can happen when one of the CPUs on which the simulation is distributed is relatively slower that other. In this case when the slower node schedules an event on a faster node, this event can be in the past time in since for faster node. This necessitates the faster node to roll back to that time state. The basic rule that can be followed is “If the value of variable in the simulation object changes as the result of processing an event, then that variable needs to be rollback able” [5].

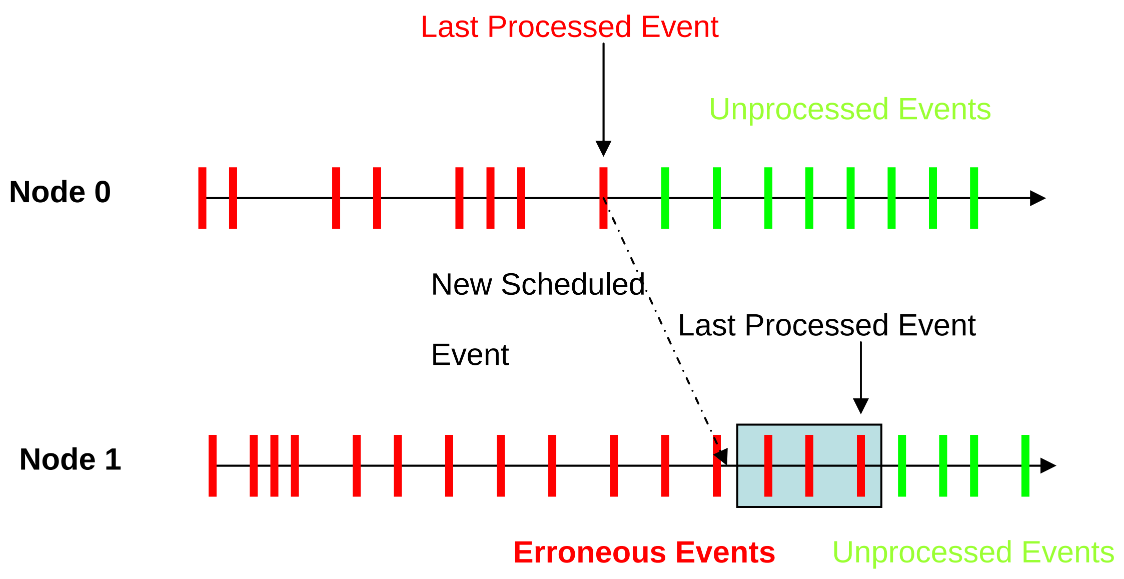


Figure 5: Synchronization by rollback in PDES (From SPEEDES Manual)

To support the rollback feature SPEEDES provides built in rollback able data types. These data types are similar to data types in C++. By adding word “RB_” to the primitive data type in C++, SPEEDES rollback able data types are built. For example “int” in C++ is “RB_int” in SPEEDES, “double” in C++ is “RB_double” in SPEEDES. During simulation execution if a simulation object is rolled back then these variables on simulation objects are returned to their old values. SPEEDES provides rollback able integers, doubles, strings, void pointers, Booleans, Streams, binary tree, hash tree, lists, priority trees, and dynamic pointer arrays.

CHAPTER 3: METHODOLOGY

3.1 Introduction

The main objective of this thesis is to convert a procedural Arena Space Shuttle model into an Object Oriented SPEEDES environment. The created model was required to be tested for distributed technology in SPEEDES. In the initial preparation stage the requirements of the new model were formulated from the Arena model. The model in Arena was challenging as it was built in the currently outdated Arena 3.0 version. This model had to be studied and understood thoroughly before start of development.

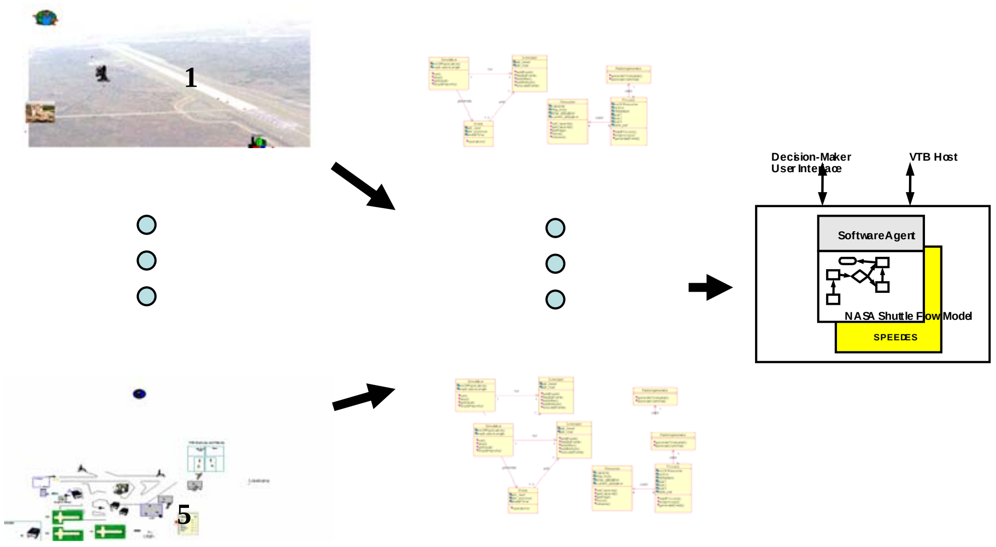


Figure 6: From Procedural to Object Oriented…

A stepwise approach was followed to convert the complete model into SPEEDES. The Arena model was broken down into smaller modules to better understand the behavior of the various entities and processing objects. Breaking the Arena model into smaller modules helped in classifying the objects required to form an infrastructure for developing the entire “NASA Space Shuttle Model” into SPEEDES. It took 7 versions/stages to convert the complete model from Arena to SPEEDES. In each stage some new modules were added and enhancements were done to the existing modules. At the end of each stage, we validated the SPEEDES model against the Arena model and documented the results.   So finally we had complete models in SPEEDES and Arena 5.0.

In this chapter different stages of development of the SPEEDES model are discussed in detail. Each stage talks about the new modules and enhancements done to the existing modules. Unified Modeling Language package Rational Rose had been used to draw UML diagrams during the development.

3.2 Using UML during development

UML stands for Unified Modeling language. The UML is a modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system intensive process [6]. It is the industry-standard modeling language for specifying and documenting both data and processes in software systems, created by OMG (Object Management Group).

UML provides different aspects of the systems under study. In this thesis UML is been used from formulating the requirements of desired model. Following are the important types of UML diagram from UML aspect:

3.2.1 Class Diagrams

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams also display relationships such as containment, inheritance, associations and others [8].

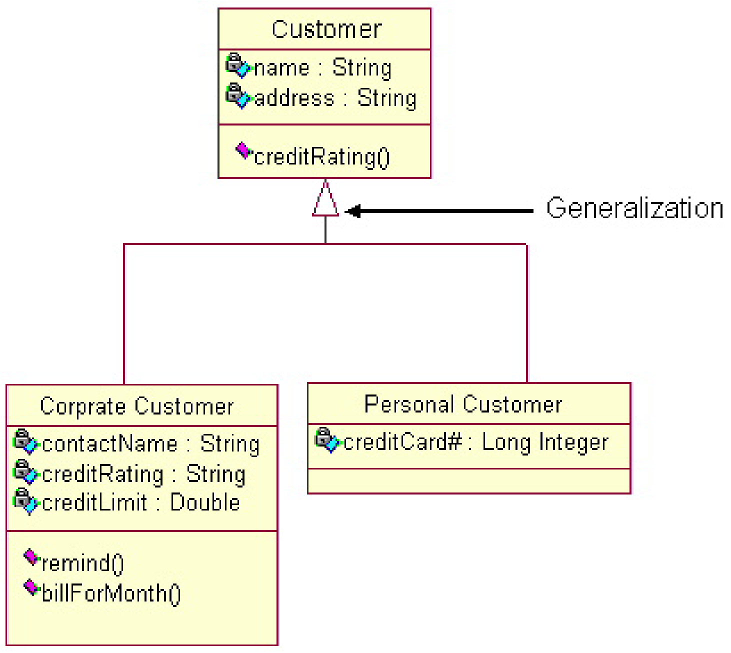


Figure 7: Example Class Diagram

Class diagrams were used to model the interaction between the classes. Class diagrams can explain which classes are inherited included in side the class under study.

3.2.2 Activity Diagram

An activity diagram is another way to describe use case behavior, focusing on how the behavior can be broken down in functions, internal to the system or system part. The activity diagram describes in what order different functions should be carried out and, if they are optional, under what circumstances the functions should be invoked [9].

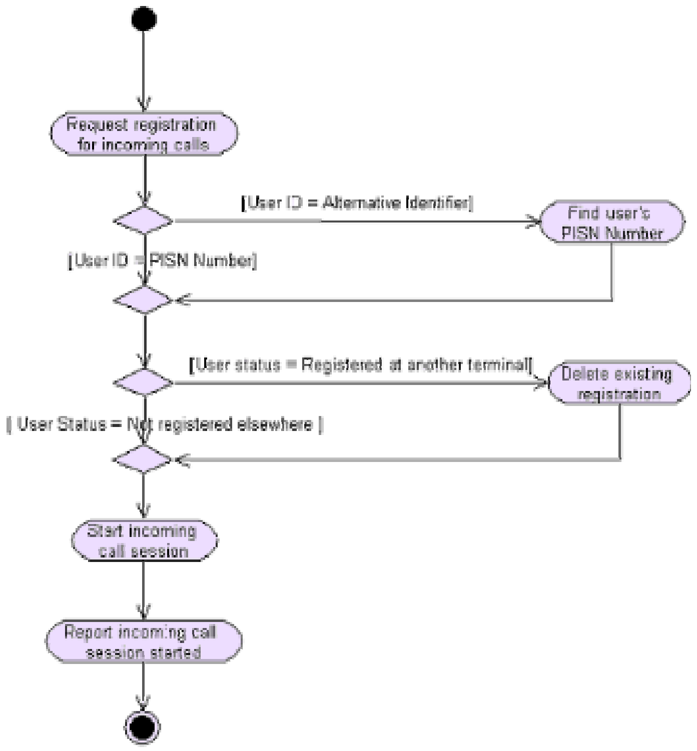

Request registration for incoming calls
[User ID = Alternative Identifier]
Find user's PISN Number
[User ID = PISN Number]
[User status = Registered at another terminal]
Delete existing registration
[User Status = Not registered elsewhere]
Start incoming call session
Report incoming call session started

Figure 8: Example Activity Diagram

Similar to the class diagrams, activity diagrams were especially very helpful in understanding the logic of events and activities involved with the event. Activity diagrams depict classes involved and scope of that stage. So it is a measure of how much transfer to SPEEDES has been completed.

3.2.3 Sequence Diagram

Sequence diagram is one of the possible diagrams to choose from in UML for simulation. Sequence diagram focus on describing the sequences of message interactions between communicating entities. In this thesis Sequence diagrams were used extensively to model sequence of event execution on respective objects.

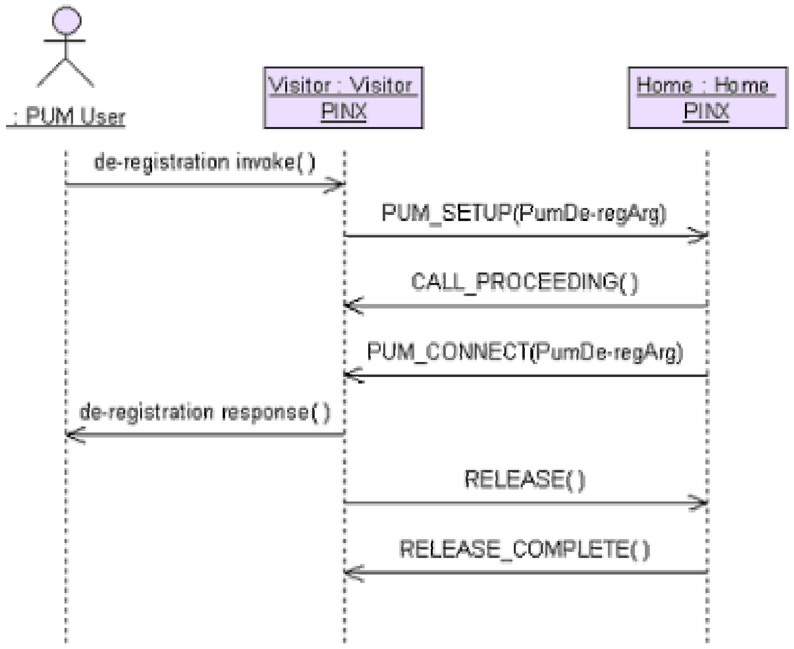


Figure 9: Example of Sequence Diagram

As SPEEDES model was to be run in distributed environment, component diagrams were also drawn to understand the architecture of the system to run the model on multiple computers. As mentioned before, this is a very detailed model. Hence each component cannot be shown in one diagram. Hence, in each stage newly added components are concentrated in the diagrams. In the next sections development in each stage is described and newly added modules are listed.

3.3 Stage I

The entire model developed in Stage I was named as "Version 1." It was proof of concept for SPEEDES. Previously SPEEDES had been used for military simulation and this was the first time SPEEDES was used for industrial purposes like shuttle hardware flow. This version consisted of only one hardware element - the orbiter, which would only go through the major processes that take place on a shuttle as it completes a flight. Each processing object contains many processes/servers. Executing version 1 on a single processor and multiple processors were major accomplishments in this model. The results were identical with that of the Arena model. This proved that SPEEDES models provided consistent results whether running on one or more than one processor.

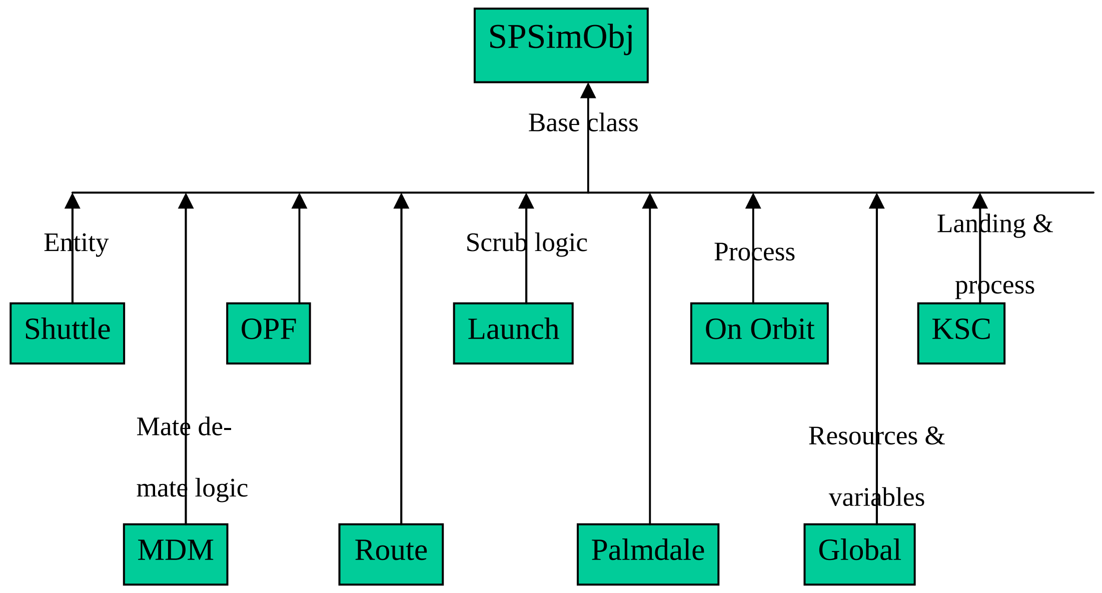


Figure 10: Hierarchy of classes

In this version there is one entity (shuttle) and eight processing objects, which were reviewed and implemented. These modules are:

Table 4: Classes Added in Stage 1

Orbiter Processing Facility (OPF)	Launch pad (launch)	On orbit (orbit)
Kennedy Space Center (KSC)	Mate de-mate (MDM)	Global
Route	Palmdale	Shuttle

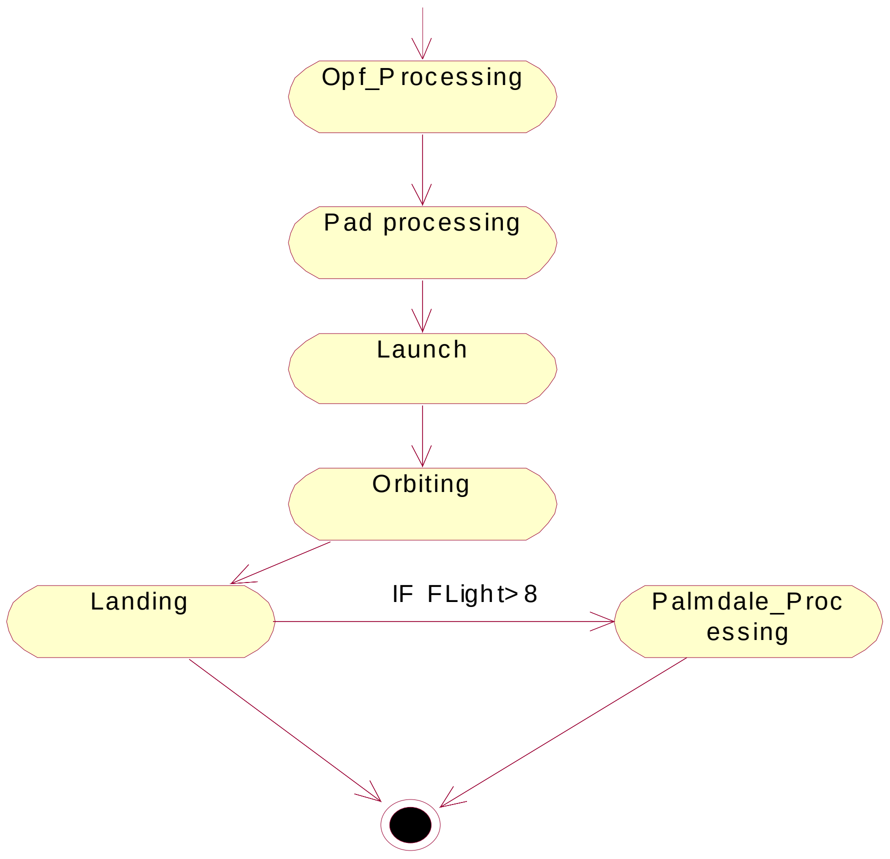


Figure 11: Activity diagram for Stage 1

Version 1 modeled the basic shuttle routing. 1 shuttle was created and routed to OPF, Launch Pad, On Orbit, and back to KSC. The flights made by the shuttle are counted and when the shuttle completes 8 flights, it is sent to Palmdale for service. This version demonstrated the capability of SPEEDES to implement basic components of industrial simulation like server, entity, routes, and stations. So this model served as the base needed to develop the platform for the future complexity and sub processes that needed to be modeled.

3.4 Stage II

It is focused on the sub-processes involved in each processing object that was modeled in Stage I and scaling the model with more hardware elements/entities and processing objects. The classes created in stage 1 were refined, and more attributes and methods were added to model the detailed functionality. In each stage SPEEDES was more and more explored to model the complex logic. More objects were then classified by distinguishing the entities and the processing objects.

In this version the model was created and run with 4 shuttles. Although the new entities were never modeled as a separate class, there were instances of the same class “shuttle”; for every entity/instance there was a separate set of state variables assigned. For example, the engine of orbiter was assigned a primary variable called “engine’. If a new entity was to be created, a new instance of the class shuttle was declared with different sets of state variables.

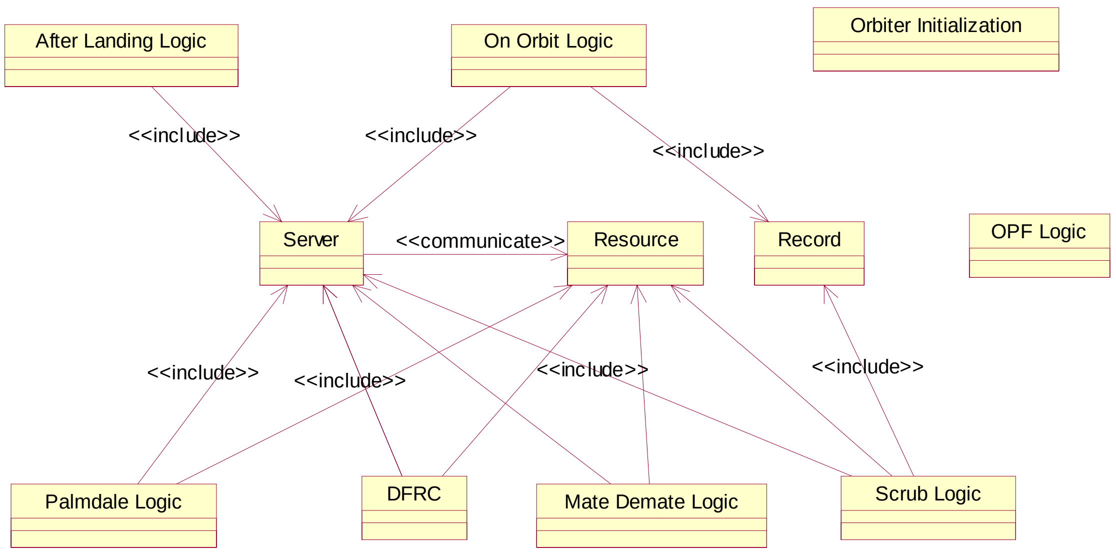


Figure 12: Stage 2 Class Diagram

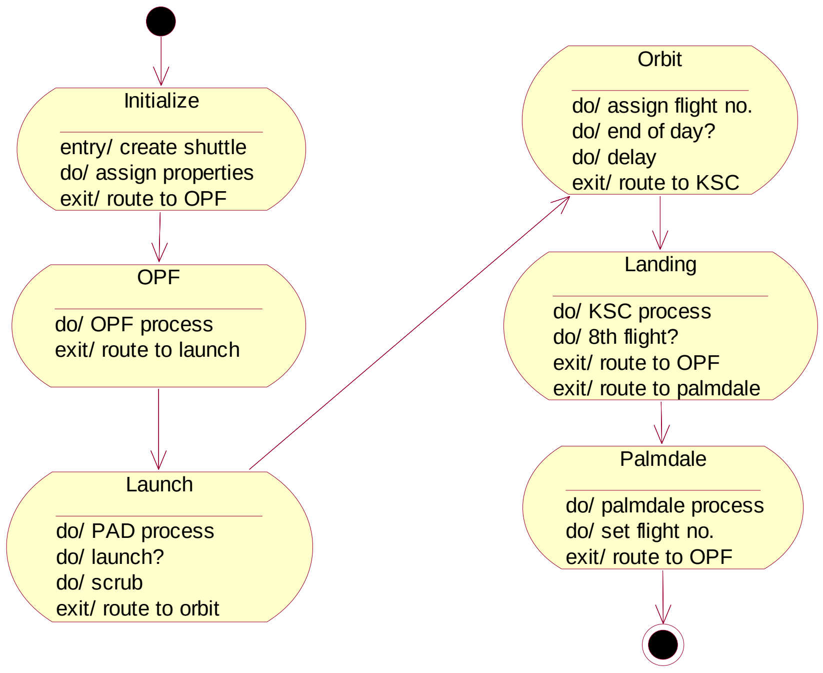


Figure 13: Activity Diagram for Stage 2

3.5 Stage III

SPEEDES can distribute or parallelize simulations to computers across networks such as LAN, WAN, or even the Internet. The processing speed of these nodes can be very different, which makes synchronization important. It could happen that a particular value needed by a process A is generated by another process B. So it is important that process a requests the value on or before the time process B generates it. This cannot be guaranteed since the various nodes are computers with different configurations and hardware. In order to accomplish this, the server uses two types of events: Rollback and Commit. Rollback is done when the data requested by process A has already been generated by B. So the server now asks process B to Rollback to the time where it had generated the value needed by A. Rollback ensures that process B generates the same values from the Rollback to the time before it did the Rollback. This makes sure that other processes which are dependent on B are not affected. Commit is done when all the processes dependent on B are in a state in which they do not require any value from process B, say after a point t in time. So the server will do a Commit at point t. Once a Commit is given, the system cannot Rollback to a point beyond t. Hence, to make a process Rollback-able, the events must also be Rollback-able, i.e. the occurrences of say event E should have the same distribution before and after the Rollback.

Arena has a large variety of random number generators built in, so do this model. These random numbers servers as the backbone of any simulation model to bring a stochastic nature. Hence to build the same kind of model in SPEEDES various kinds of random number generators were required, and these random numbers needed to be rollback able. SPEEDES has some basic random numbers like Uniform and Normal built in. Therefore, we added several random generators Rollback-able to SPEEDES. The algorithms for the Gamma, Lognormal,, Weibull, Triangular, Continuous, Discrete, and Johnson were written in C++ using SPEEDES library and were validated by fitting the output curve of these random number generators in Arena's input analyzer tool. This was a very important addition to enhance the SPEEDES.

Table 5: Logics Added in Stage 2

Scrub logic	On Orbit logic	Landing logic
Mate De-Mate logic	DFRC logic	Detailed Palmdale logic

In scrub logic the shuttle is checked against different probabilities for weather conditions, shuttle technical conditions etc. And if shuttle goes through all these probabilities then it is launched. In the On Orbit logic, the shuttle is checked against different probabilities for failure and if the launch is successful, the shuttle routes in orbit for the generated Time in Orbit. After landing logic decides the path of shuttle. If the shuttle has completed 8 flights, then it is sent to Mate de-mate logic and then to Palmdale for service, otherwise it is routed to OPF to prepare for the next flight. In mate de-mate logic, the shuttle is attached to a plane and transported to Palmdale. In Palmdale logic different shuttles and servers are modeled to bring Palmdale processing. Finally DFRC logic was added.

DFRC is alternate landing location and with some probability a shuttle can choose DFRC as landing site. When modeling in SPEEDES, all these modules were defined in separate classes i.e. S_MDM, S_DFRC etc.

3.6 Stage IV

The addition of more classes to this environment has been planned and will include the different visualization environments. Visualization is a very important feature of modern simulation modeling environments. The investigation of different visualization paradigms continues. There are many visualization tools available. However, for space operations among the most sophisticated tools are the Real-Time Graphics Engine (RAGE) from White Sands Missile Range, EDGE from Boeing Autometric, and customized environments using JAVA 3D and the Virtual Reality Modeling Language (VRML) and other extensions using the extensible Markup Language (XML), such as X3D, Web3D, and Xj3D [19]. Figure 6 shows the development of multiple windows (one for each Shuttle) using JAVA 3D and VRML Objects and manipulated from different computers using SPEEDES.

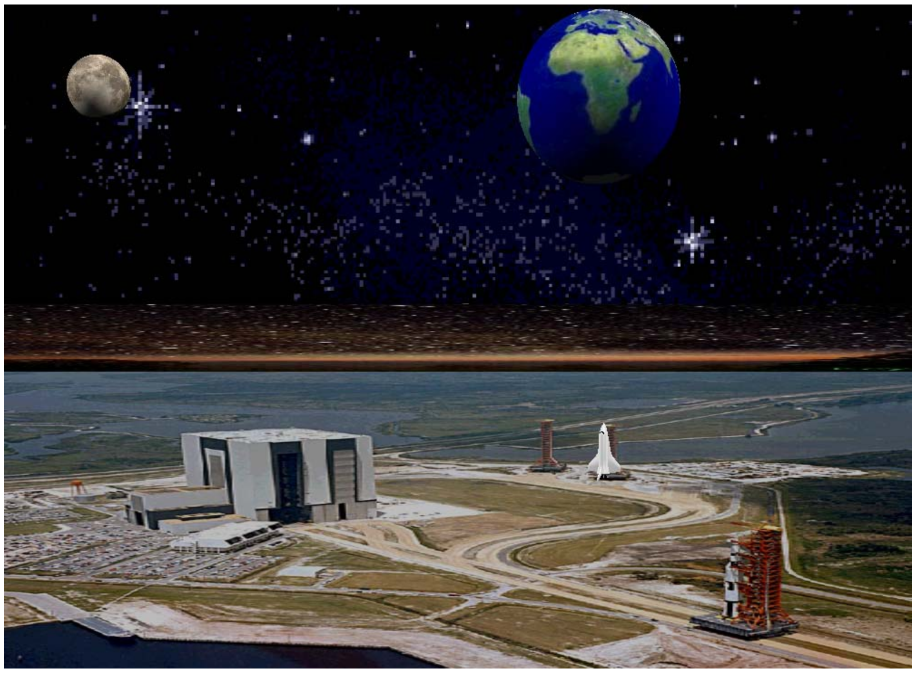


Figure 14: JAVA 3D Environment and VRML Object

In this version the detailed OPF was modeled into modules OPF Single Queue, OPF Logic and OPF Assembly logic. The Shuttle is sent to Flow type logic from the OPF single queue, to determine the next destination i.e. Pre-Palmdale flow, Post Palmdale Flow normal OPF logic. This is decided on the number of flights the shuttle completed after the last servicing. When the shuttle is routed to OPF logic, it is dismantled into different parts for further processing. These parts are Engine, FRCS, Left OMS, and Right OMS. These parts were modeled as separate objects. The Engine shop and HMF were modeled as classes for processing of these parts. In the engine shop, after server process resource was released at a later stage. This complicated logic was implemented. After processing of all part is complete, the shuttle is assembled back in OPF assembly.

In this class the shuttle object waits for matching with other parts. When a shuttle is to be sent to Palmdale it goes through pre-Palmdale flow and after return it is routed through post Palmdale flow. These logics were added in this version.

Table 6: Logics Added in Stage 4

Initialization	OPF Single Queue	OPF Logic
OPF Assembly Logic	Engine Shop	HMF Logic
Pre Palmdale Logic	Post Palmdale Logic	VAB Logic
PAD A Logic	Pad B Logic	SRB Separation
ET Separation	Retrieval Vessel Logic	Flow Type logic

Shuttle model uses 2 launch pads Pad A and Pad B which were added in this version. SRB and ET are separated from shuttle after launch one after another. Retrieval vessels wait for SRB in and recover the SRB after launch. These logics were modeled by creating separate classes. In stage 4 about 60% of Arena model was transferred into SPEEDES.

3.7 Stage V

Continuing with enhancements in version 4, new modules were added in version 5. These new modules are VAB logic, MLP Park Site logic, Utah logic, Hanger AF logic, ARF logic, ET logic, MLP logic, RPSF logic, and SRB stacking logic. After the VAB process , the shuttle waits in the high bay for the external tank, solid rocket boosters, and other parts.

SRB recovered from the past flight are sent to Hanger AF (separate class) and from Hanger AF it is transported to Utah. SRBs are serviced in Utah and brought back for RPSF servicing in KSC. In RPSF the SRBs are further serviced and sent to the high bay for assembly with motors. The External tank is created every 50 days and routed to VBA port got joining the shuttle at high bay. After launch, the MLP (mobile launch pad) is sent to the MLP site for processing and prepared for the next flight. The MLP is delayed for various logics to occur and finally sent to the high bay for mating with the shuttle.

Hence all these different entities i.e. SRB, ET, MLP are modeled as different objects. These objects were created at the time of shuttle creation in a separated module called ‘Other Element initialization’. Each of these logics is a combination of different servers, decisions, and probabilities. While building the model in this stage, the transporter functionality used in Arena was not developed in SPEEDES. Hence to complete the logic, transporters were replaced by direct routing and later transporters were added as a separate version. In this stage about 90% of final Arena model was transferred into SPEEDES.

3.8 Stage VI

As mentioned earlier, using transporters, the distance functionality was not yet achieved in SPEEDES. This version totally concentrates on modeling transporters in SPEEDES. Transporters are used to transport an entity from one station to another. When the entity needs to be transported from one station to another, it calls the transporter and waits on the station. The Transporter when receives the call; starts from the beginning station and reaches the entity station, and transports the entity to destination station of the entity. The Transporter can travel with different speed depending upon the load it is transporting. The delays occurring in this process are due to the transporter moving from its beginning station to entity station, entity loading delay, transportation delay and finally unloading delay. If a transporter is transporting another entity when called, then the calling entity has to wait until the transporter becomes available. Each transporter works in between some fixed stations and the distance between those stations is fixed.

There are total 6 transporters used in the Arena model i.e. ‘Tow’, ‘Trnsp’, ‘Crwlr’, ‘Vessel’, ‘EngineHyster’, and Train. In SPEEDES each of these transporter were modeled as a separate class, with specialized functionality. Each transporter maintains its current station and velocity as attributes. Each transporter is linked with a set of distances. In SPEEDES these distances were maintained as two dimensional arrays and each station was assigned a constant number for indexing in an array. When an entity requires transportation, it adds itself to the queue of transporter, and after the transporter becomes available, it is transported to the desired station. The delay is calculated by extracting the distance between stations from the distance sets array and dividing by the velocity of transporter. With addition of transporter the SPEEDES version was modeling Arena version as close as 90%.

3.9 Stage VII

Arena posses a rich variety of random numbers distributions, and so does the Arena shuttle model. To model the Arena shuttle model in SPEEDES, all the distributions used in the Arena shuttle model needed to be developed in SPEEDES. So, new distributions were developed using functionality provided in SPEEDES as a separate task with model development. The new distributions were made rollback able to comply with the SPEEDES framework and initialization, number generation mechanism in new distributions was also made similar to SPEEDES. Testing and validation of new distributions was performed using Arena Input analyzer.

Table 7: List of available Distributions and new Distributions in SPEEDES.

Distributions Available in SPEEDES	**New Distributions Developed**
Uniform, Uniform Int, Uniform Double	Lognormal Distribution
Exponential Distribution	Triangular Distribution
Laplace Distribution	Weibull Distribution
Rayleigh Distribution	Gamma Distribution
Triangle Up, Triangle Down	Johnson Distribution
Beta Distribution	Discrete Distribution
Gaussian Distribution	Continuous Distribution
Density function Distribution	
Cauchy Distribution	

The major task in version 7 was to use these distributions in the model and generate random numbers similar to the Arena model. By addition of new random number generators in this version, version 7 was exactly representing the Arena shuttle model; hence 100% conversion was achieved.

To run any model of Discrete-Event Simulation, some runtime parameter need to be specified. To specify these parameters an input parameter file was created. In this file simulation parameters like Simulation Time, Speed were specified as well as shuttle model specific parameters like Number of Shuttles, Number of Pads, Number of Engines, and Number of Replications were specified. By changing shuttle specific parameters, new scenarios could be created without actually going into the code. The testing of generated model was performed and results are documented in next chapter.

3.10 Summary

In this chapter the stages of creation of the shuttle model were explained. Developing the shuttle model in SPEEDES was an iterative process. The next chapter describes the validation and testing performed with created the SPEEDES shuttle model.

CHAPTER 4: VALIDATION AND TESTING OF SHUTTLE MODEL

4.1 Introduction

This chapter presents the validation and verification results of the SPEEDES shuttle model against the Arena shuttle model. The validation was conducted at each stage of development. Different scenarios were run with the validated model and the results are presented here.

4.2 Verification and Validation

In each stage of the development, the SPEEDES shuttle model was validated with the corresponding Arena model. This helped to debug the system from the start of development. Due to validation of the model from the beginning, very few errors were encountered at the end of development.

Each model was validated against the variable ‘No of flights completed’, because the original Arena model was validated against same variable. SPEEDES does not provide all the random number generators which Arena use. To work around this problem a stream of 100 random numbers, of required distribution, was generated and used in the SPEEDES model. The Generated SPEEDES model and the Arena model for each stage were run for 10 years with 30 replications. The averages of the above mentioned variable were compared for the Arena and SPEEDES models. Following table documents the results.

Table 8: Validation results in different stages

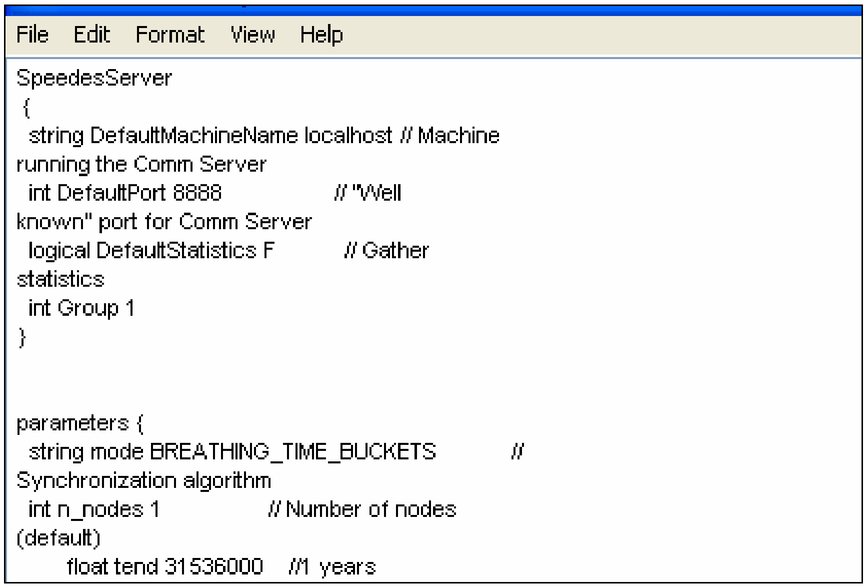
	Run Time	Base Time Units	Replications	Arena	SPEEDES
Stage 2 (with 1 shuttle)	10 years	Days	30	40.16	42.88
Stage 3	10 year	Days	30	954.30	958.39
Stage 4	10 years	Hours	30	2279.7	2285. 57
Stage 5	10 years	Hours	30	1748.97	1753.45
Stage 6	10 years	Hours	30	62.633	64.56

4.3 Tools Used for Testing and Validation

Different tools were created and used to conduct extensive testing of the SPEEDES Shuttle Model. These tools included Parameters files and Perl scripts used to create and test different scenarios. These tools made it possible to run several replications, with changing parameters, without user interruption.

4.3.1 Using Parameter Files

The figure 15 shows the parameters file used for configuration setting of the SPEEDES Shuttle Model. The file is divided into two sets of parameters. The first set called ‘SpeedesServer’ is used to specify the parameters for server configuration. It specifies the default port number, turns the statistics ON and assigns a group name to the simulation. Port number is used to open the channel for communication between the processors during the distributed simulation. Group name helps to run different models at the same time on same nodes. The second set of parameter sets general parameters required for simulation run such as synchronization algorithm, number of nodes in the simulation, and simulation time in seconds. These parameters help to run different scenarios of the simulation model.

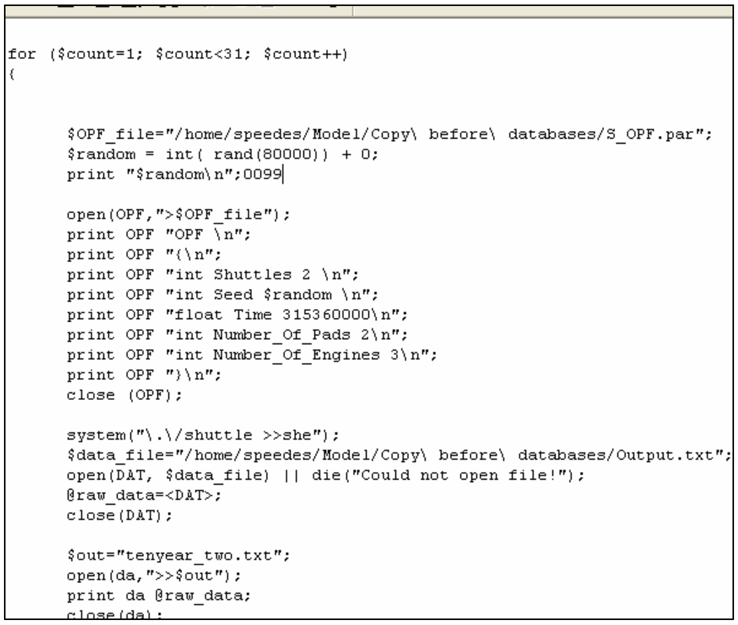


```
SpeedesServer
 {
  string DefaultMachineName localhost // Machine
running the Comm Server
  int DefaultPort 8888                // "Well
known" port for Comm Server
  logical DefaultStatistics F         // Gather
statistics
  int Group 1
}


parameters {
  string mode BREATHING_TIME_BUCKETS          //
Synchronization algorithm
  int n_nodes 1               // Number of nodes
(default)
      float tend 31536000    //1 years
```

Figure 15: Parameter file to set configuration of server

4.3.2 Using Perl Scripts for Automating the Modal Execution

Perl script is used to automate the model execution with multiple replications. Different Shuttle Model variables such as Number of shuttle, PADS, Engines can be specified in this script. The script generated a random number for seed for each replication. Warm-up period for statistics collection can also be specified in this script. The output of this script is a par (parameter) file which is used by SPEEDES Shuttle Model. Using Perl script the model execution could be automated. For each simulation run/replication a separate par file is generated with new seed, by the Perl script



```
for ($count=1; $count<31; $count++)
{


      $OPF_file="/home/speedes/Model/Copy\ before\ databases/S_OPF.par";
      $random = int( rand(80000)) + 0;
      print "$random\n";0099

      open(OPF,">$OPF_file");
      print OPF "OPF \n";
      print OPF "{\n";
      print OPF "int Shuttles 2 \n";
      print OPF "int Seed $random \n";
      print OPF "float Time 315360000\n";
      print OPF "int Number_Of_Pads 2\n";
      print OPF "int Number_Of_Engines 3\n";
      print OPF "}\n";
      close (OPF);

      system("\.\/shuttle >>she");
      $data_file="/home/speedes/Model/Copy\ before\ databases/Output.txt";
      open(DAT, $data_file) || die("Could not open file!");
      @raw_data=<DAT>;
      close(DAT);

      $out="tenyear_two.txt";
      open(da,">>$out");
      print da @raw_data;
      close(da);
```

Figure 16: Perl Script for Automating the Model execution

.

4.4 Final Verification and Validation

The Model is verified to make sure that it represents the logic correctly. Generally verification is performed with Animation if provided. Since SPEEDES does not have animation capabilities, the created shuttle model had to be verified by walking through the complete model step by step. The model was run in debug mode and many output statements were place to track the execution path. The Model was verified to represent correct logic.

Verification of any simulation model involves walking through the complete model to make sure that it represents the correct logic.

During development of this model a library of required random numbers was created. The library had all the required distributions for the Space Shuttle Model. This random number library was included in final version 7 and the model was validated against the original Arena model. For final validation, the SPEEDES and Arena model were run for 10 years and 100 replications. A confidence interval around the no. of flights variable was built and the mean of SPEEDES model fit in that interval [10].

Table 9: Validation result of final SPEEDES model (10 years)

	Arena	**SPEEDES**
No. of replications	30	30
Run time	10 years	10 years
Mean	68.93	69.78
Std. Deviation	1.51	2.02
Conf. Intl Upper limit	71.89	-
Conf. Intl Lower limit	65.96	-

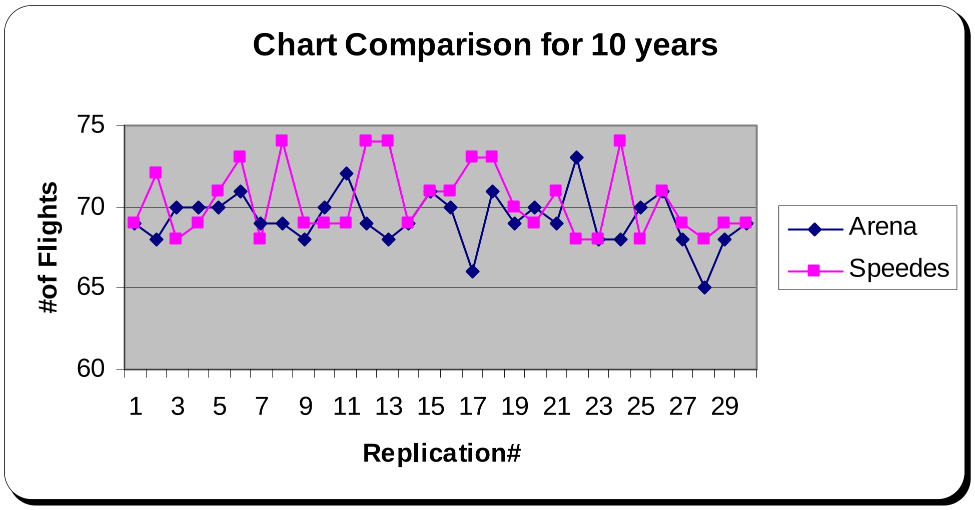


Figure 17: Comparison Chart 1

In the table in can be seen that for 10 years, the Arena model gives an average of 68.93 and SPEEDES model gives the average of 69.78. Also, the mean from SPEEDES is falling within 95% confidence interval of the Arena model.

Table 10: Validation result of final SPEEDES model (5 years)

	Arena	**SPEEDES**
No. of replications	30	30
Run time	5 years	5 years
Mean	35.53	37.33
Std. Deviation	1.008	0.66
Conf. Intl. Upper limit	33.55	-
Conf. Intl Lower limit	37.50	-

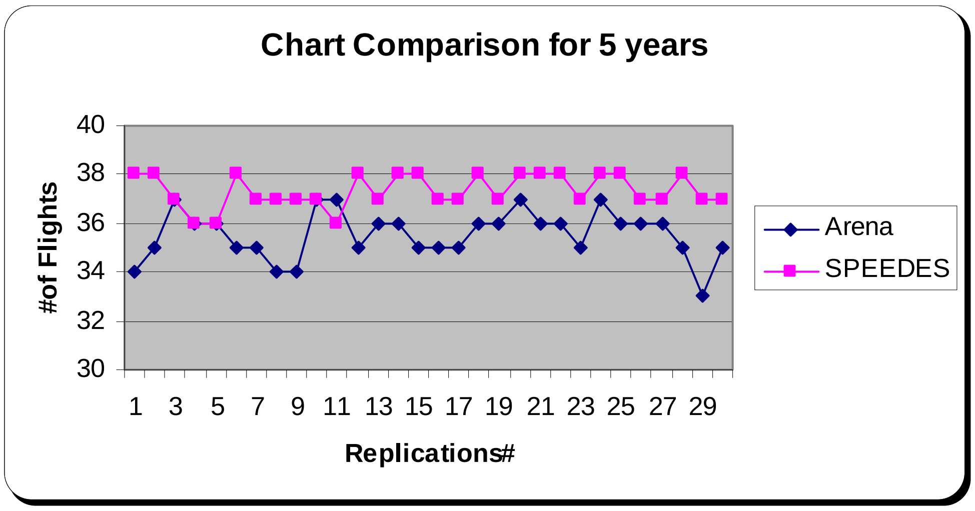


Figure 18: Comparison Chart 2

Similarly in this test, the mean of no. of flights from the SPEEDES model falls in between the upper and lower confidence interval of the Arena model. From these testing examples it can be said that the SPEEDES model had been validated and can model the shuttle operations. Different scenarios can be created with the verified SPEEDES model, which will be covered in next section.

4.5 Testing Different Feature of SPEEDES

After verification and validation of the SPEEDES Shuttle model, more observations were done to check whether the Shuttle Model developed in SPEEDES performs better and provide the same results when configured with advanced features. These observations are not in the scope of this thesis, and more analysis should be done in the future work. These features included load balancing, object distribution algorithms, and rollback feature

4.5.1 Testing the Block and Scatter Algorithms

In a distributed simulation environment, objects which require high CPU usage should be distributed across available CPUs to achieve a balanced CPU load and better run-time performance. The placement of simulation object on respective nodes can have a huge impact on processor or node load balancing. SPEEDES provides two automatic decomposition methods called block and scatter. Scatter algorithm decomposition distributes the simulation objects with consecutive kind ID's located on different consecutive nodes. On the other hand, the block algorithm starts off by calculating the number of simulation objects that can be placed on each node evenly [5].

The generated Shuttle Model was tested to implement the bock and scatter algorithms. The following snippet of code shows the method of implementing the scatter and block algorithms.

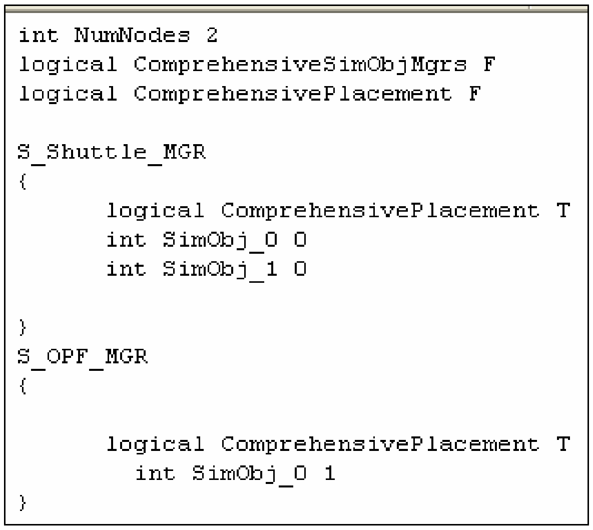
```
DEFINE_SIMOBJ(S_OPF, S_OPF::GetNumObjs(), SCATTER);

DEFINE_SIMOBJ(S_OPF, S_OPF::GetNumObjs(), BLOCK);
```

Figure 19: Implementing Scatter and Block Algorithms

The block and scatter algorithms were tested and expected results were obtained.

4.5.2 Testing the Load balancing with File Driven Algorithm

In a simulation that contains simulation objects which are tightly coupled and will tend to roll each other back if they reside on different nodes, manual placement can achieve better simulation performance results. The load balancing can be performed to a good extent using a file driven algorithm. In this algorithm, the object placement can be specified on each node manually. Figure above shows the parameter file used for testing this feature using the created SPEEDES Space Shuttle Model. It shows different simulation objects can be placed on desired nodes using their simulation managers.



```
int NumNodes 2
logical ComprehensiveSimObjMgrs F
logical ComprehensivePlacement F

S_Shuttle_MGR
{
      logical ComprehensivePlacement T
      int SimObj_0 0
      int SimObj_1 0

}
S_OPF_MGR
{

      logical ComprehensivePlacement T
        int SimObj_0 1
}
```

Figure 20: File Driven Object Placement

4.5.3 Testing the Roll backing Feature

When processing a simulation in parallel, one or more of the CPUs on which the simulation is distributed may run ahead in time relative to other nodes. An event on a simulation object being processed on a slower node may schedule an event for a simulation object being processed on one of the faster nodes. Events that were processed on the faster simulation object need to be reprocessed when an event in the past occurs. The event that was processed may have to change the state of the simulation object on the fast node, and those changes must be undone [5].

Rollback in SPEEDES is achieved with rollback variables. But to efficiently run the simulation with rollback, SPEEDES recommends using the rollback variables only to store state variables. To test this feature the SPEEDES Simulation Model was run with excessive rollback variables (used between function) and optimum rollback variables. The result was, the optimum simulation model rollback variable took less execution time than the other. Hence, different SPEEDES features were successfully tested with the developed SPEEDES Shuttle Simulation Model.

4.6 Running Different Scenarios with SPEEDES Model

Different experiments were performed on the SPEEDES Space Shuttle Model. The distributed simulation feature of SPEEDES was tested in these experiments. The following section explains different experiments performed with single and multiple computers on the validated model.

4.6.1 Running SPEEDES Shuttle Model on a Single Computer and Multiple Computers

An experiment was performed on the SPEEDES Shuttle Model by running the model with 2 and 3 shuttles separately. The Model was run for 10 years and 30 replications. Following table presents the result of testing.

Table 11: Running SPEEDES model with 2 and 3 shuttles

	2 Shuttles	**3 Shuttles**
Mean	32.96	54.6
Std. dev	0.182	0.723

Same experiments were conducted by running the SPEEDES shuttle model on multiple computers. These computers were connected through LAN network with one server and multiple nodes and same results were obtained.

4.6.2 Testing with NASA AMES

In an experiment in NASA ARC, the distributed simulation capabilities of SPEEDES were tested by running different experiments by distributing the 41 objects of the STS Process Flow in different numbers of computers. These computers were located in different geographical locations:

1. The SPEEDES server was installed at the University of Central Florida (Orlando, Florida)
2. Two simulation nodes in two different computers at NASA ARC (Ames, California) were set up.

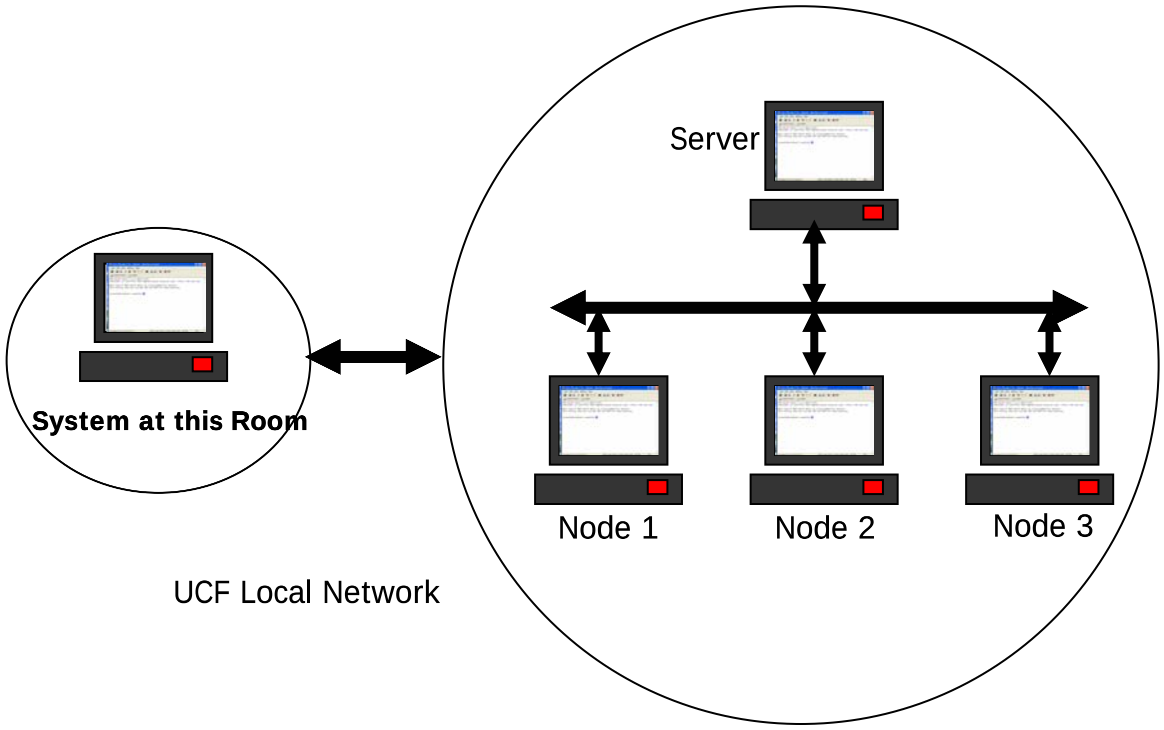


Figure 21: Parallel/Distributed Simulation at NASA ARC

The simulation interactions among the different objects produced the original results using a single computer. The gains are in speed and the utilization of unique resources attached to each node. The environment proved even more useful with the future additions to the original simulation model. These additions allowed for different resolution levels and the study of safety and human-behavior modeling issues. This experiment demonstrated the Distributed Simulation feature of SPEEDES to NASA.

4.7 Summary

The generated SPEEDES Shuttle Model was validated against the Arena Space Shuttle Model. Results of the validation are presented in this chapter. After validation, different scenarios were created to test the features of SPEEDES. The SPEEDES Shuttle Model was run with different no. of shuttles, distributions, and run length. This validation and testing was one of the major accomplishments in this project.

CHAPTER 5: CONCLUSION AND FUTURE RESEARCH

5.1 Conclusion

This thesis is the proof of concept of using SPEEDES for a large Distributed Simulation Model. Converting the model from Arena to SPEEDES was a stepwise process with a lot of challenges. The legacy Arena model was very detailed. This took 7 stages in SPEEDES to convert the complete model. The model was validated against the Arena model successfully. Different experiments were performed to test the features of the SPEEDES. These experiments proved the capabilities of SPEEDES for distributed simulation. Though SPEEDES was being used for defense simulations, this thesis proved its usability for industrial simulation and space simulation opening a new horizon for SPEEDES applications.

5.2 Contributions

This project had lot of contributions from teams in University of Central Florida and All Points Logistics. Karthik Narayanan contributed by developing the Random number Generation library. This library proved to be very much useful in modeling all the random numbers available in Arena. Mario Marin initiated this project by developing a mini-shuttle model in Arena. This model served as proof of concept SPEEDES. Amit

Wasadikar was consulted as an expert in Arena, and helped to further break down the model into different stages.

5.3 Future Research

This thesis has proved that SPEEDES has potential to be a dominant Discrete-Event Simulation Environment. Since SPEEDES support Distributed Simulation and High Level Architecture (HLA), it can be made web enabled. Following are the possible research topics which can help enhance SPEEDES applications.

5.3.1 Developing Predefined Components

SPEEDES can achieve wide acceptance if predefined components for simulation are built. While developing this model, the code for common functionalities was repeated due to unavailability of such simulation classes. Hence future research in this area can be done by developing a Discrete-Event Simulation library with classes for modules such as server, transporter, decision etc. Using these predefined classes, the simulation model in SPEEDES can be generated in a shorter period.

5.3.2 Developing the Visualization Architecture

SPEEDES is a powerful Discrete-Event Simulation environment but it lacks visualization capability. Visualization will help to understand the simulation models better. Following diagram shows the proposed architecture for visualization, for SPEEDES.

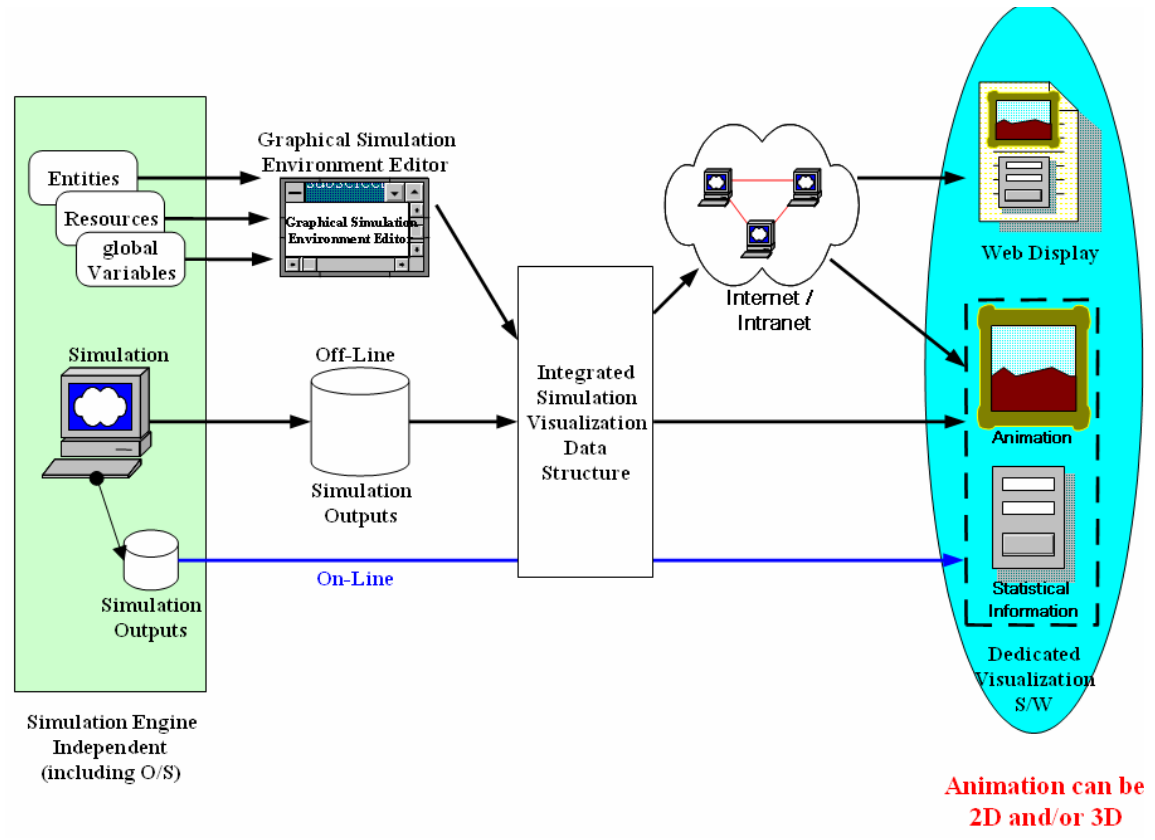

Graphical Simulation Environment Editor
Entities
Resources
global Variables
Graphical Simulation Environment Editor
Simulation
Off-Line
Simulation Outputs
Integrated Simulation Visualization Data Structure
Internet / Intranet
Web Display
Animation
Statistical Information
Dedicated Visualization S/W
On-Line
Simulation Outputs
Simulation Engine Independent (including O/S)
Animation can be 2D and/or 3D

Figure 22: DES Visualization Architecture

As shown in the figure, the proposed architecture is divided in three modules. The first module will include the SPEEDES simulation engine, the second module will be the Integrated Simulation Visualization Data Structure, and the third module will be animation software built using advanced graphical languages. The simulation animation could be run either On-Line or Off-Line. In On-Line approach, the SPEEDES model will write the simulation output in some files, which will be connected to the Integrated Simulation Visualization Data structure. This Data structure will be read by animation software to show online animation. In the Off-Line approach simulation output from the SPEEDES model will be stored on a database and then executed by the simulation software through an Integrated Simulation Visualization Data Structure. With this architecture, simulation and animation can be run on the same computer, different computers in network or on remote machines through a web browser [18].

5.3.3 Integrating the Spaceport Models

Since SPEEDES supports High Level Architecture (HLA), the existing Spaceport models can be integrated with SPEEDES. These Spaceport models include Model of Space Shuttle Operations, Spaceport Safety Modeling and Optimization, Generic Simulation Environment for Modeling the Future Launch Operations, Range Process Scheduling Tools etc. In future, if these models are built in SPEEDES, they can be integrated with each other using HLA [11].

5.3.4 Developing Java Graphical Interface

The SPEEDES models can be further enhanced by integrating with a Java Graphical Interface. As shown in the figure, the SPEEDES simulation models can be coupled with geographical animation. A SPEEDES simulation will talk to the central SPEEDES server. The SPEEDES server integrated with a Java State Manager and Java GUI will produce the animation with physical location of objects.

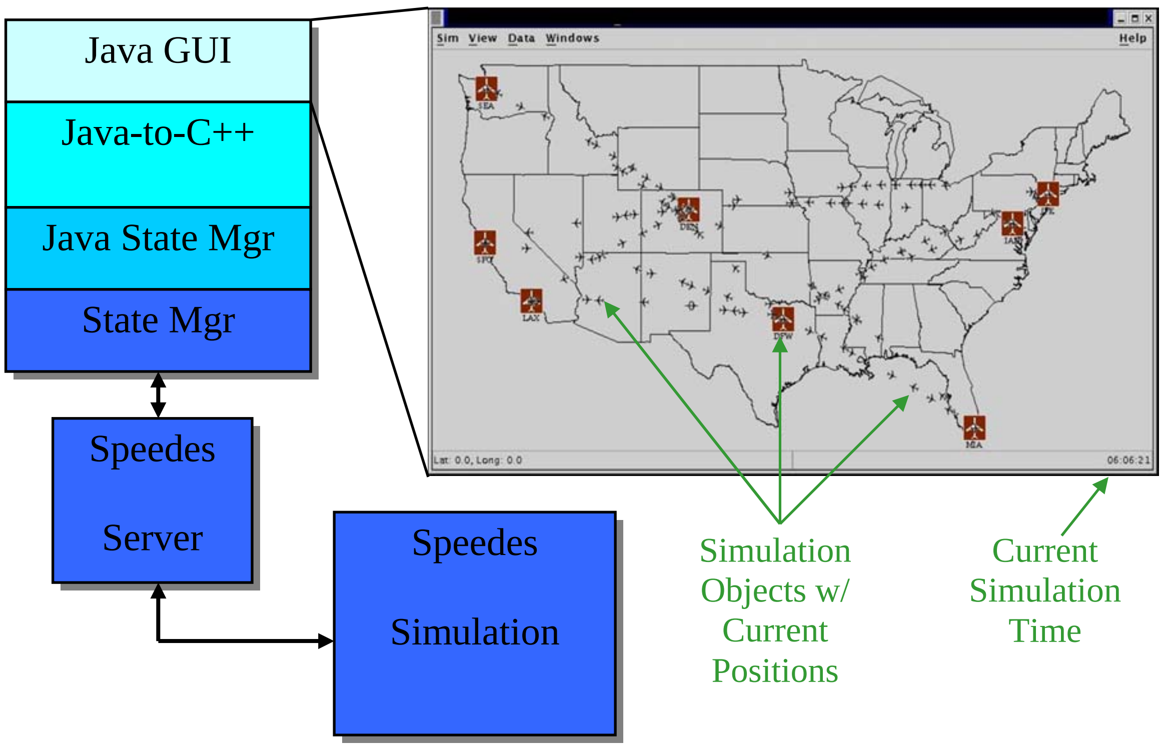


Figure 23: Java Graphical Interface

LIST OF REFERENCES

1. Paruchuri Amith, Wasadikar, Amit, Marin Mario F., Dr. Sepulveda Jose A., Dr. Rabelo Luis (2004) “Parallel Discrete Event Simulation of Space Shuttle Operations Design, Development and Performance using SPEEDES”
2. Cates, G., Mollaghasemi, M., Rabadi, G., and M. Steele (June 2002), “Simulation Modeling and Analysis of Space Shuttle Flight Hardware Processing,” World Automation Congress, Orlando, Florida.
3. Ball Peter (May 1996) ‘Introduction to Discrete Event Simulation,’,2nd DYCOMANS workshop on ‘Management and Control : Tools in Action’ http://www.dmem.strath.ac.uk/~pball/simulation/simulate.html
4. Wieland Frederick, Blair Eric, Zukas Tony (1995) “Parallel Discrete-Event Simulation (PDES): A Case Study in Design, Development, and Performance Using SPEEDES” MITRE Corporation 7525 Colshire Dr. McLean, VA 22102 8186-7120-3/95 $04.00 © IEEE
5. SPEEDES User’s Guide Prepared by Metron, Inc. (30 April 2003), Prepared for: The Joint National Integration Center
6. Alhir Sinan Si “What is the Unified Modeling Language (UML)?” (1998) http://home.comcast.net/~salhir/WhatIsTheUML.PDF
7. Compton Jeppie, Rabelo Luis C., Marin Mario F. (September 30, 2003) “Spaceport Models Assesment FY03 Report”, All Points Logistics (APL) Research Report.

8. Braun David, Sivils Jeff, Shapiro Alex, Versteegh Jerry (Retrieved on 12/01/04) "Unified Modeling Language (UML) Tutorial website". http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/class.htm
9. "Making Better Standards website" (Retrieved on 11/25/04) http://portal.etsi.org/mbs/Languages/UML/uml_example.asp
10. Law, A.M., and Kelton, D.W., (2000), "Simulation Modeling and Analysis", McGraw-Hill, Inc.
11. Compton J., Rabelo L., Marin M. (2003). "Spaceport Model Assessment FY03 Report" All Points Logistics (APL) group
12. SPEEDES API reference Manual, Prepared by, Metron Inc, Prepared for The Joint National Test Facility
13. Rabelo Luis, Sepulveda Jose, Marin Mario, Paruchuri Amith, Wasadikar Amit, Nayaranan Karthik (2004) "Parallel Discrete Event Simulation of Space Shuttle operations" Proceedings of the 2004 Winter Simulation Conference.
14. Sepúlveda José, Rabelo Luis, Park Jaebok, Riddick Frank, Peaden Cary (2004) "Implementing The High Level Architecture in the Virtual Test Bed" Proceedings of the 2004 Winter Simulation Conference.
15. Xia Zheng (2000) "Flight Simulation on Parallel Computers" Master's thesis at Unicersity of Central Florida
16. Berry Orna, Lomow Greg (September 1986) "Speeding up distributed simulation using the time warp mechanism" Proceedings of the 2nd workshop on making distributed systems work.

17. Lu Tainchi, Lee Chungan, Hsia Wenyang, Lin Mingtang (July 2000), "Supporting large-scale distributed simulation using HLA". ACM Transactions on modeling and computer simulation.

18. Perumalla Kalyan S., Park Alfred, Fujimoto Richard M., Riley George F. (June 2003), "Scalable RTI-based paralleled simulation of networks". Proceedings of the 17th workshop on parallel and distributed simulation.

19. Kreutzer W. (1986) Systems Simulation – "Programming Styles & Languages" Addison-Wesley

20. Loo Boon Thau, Huebsch Ryan, Harren Matthew, "Parallel discrete event simulation in Titanium"
http://www.cs.berkeley.edu/~boonloo/classes/cs267/final/cs267paper.pdf