A MULTI-MODAL DISCRETE-EVENT SIMULATION MODEL FOR MILITARY DEPLOYMENT

A THESIS
SUBMITTED TO THE DEPARTMENT OF
INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Uğur Ziya Yıldırım

January 2009

I certify that I have read this thesis and have found that it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Prof. Barbaros Ç. Tansel (Supervisor)

I certify that I have read this thesis and have found that it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Prof. İhsan Sabuncuoğlu (Supervisor)

I certify that I have read this thesis and have found that it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Prof. Cevdet Aykanat

I certify that I have read this thesis and have found that it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Prof. Berna Dengiz

I certify that I have read this thesis and have found that it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Assoc.Prof. Oya Ekin Karaşan

Approved for the Institute of Engineering and Sciences:

Prof. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

A MULTI-MODAL DISCRETE-EVENT SIMULATION MODEL FOR MILITARY DEPLOYMENT

Uğur Ziya Yıldırım
Ph.D. in Industrial Engineering
Supervisors: Prof. Barbaros Ç. Tansel
Prof. İhsan Sabuncuoğlu
January 2009

This study introduces a logistics and transportation simulation as a tool that can be used to provide insights into potential outcomes of proposed military deployment plans. More specifically, we model a large-scale real-world military Deployment Planning Problem (DPP) that involves planning the movement of military units from their home bases to their final destinations using different transportation assets on a multimodal transportation network. We apply, for the first time, the *Event Graph* methodology and *Listener Event Graph Object* framework to create a discrete event simulation (DES) model of the DPP. We use and extend Simkit, an open-source Java Application Programming Interface for creating DES models. The high-resolution approach that we take in most part, allows us to estimate whether a given plan of deployment will go as intended, and determine prospective problem areas in a relatively short time compared to other existing simulations because of the absence of the need to use several models of differing resolutions in succession as often done in literature. For a typical deployment scenario for four battalions, run times are between 25 to 27 minutes for 60 runs of the model on a 1.6 GHz Pentium(R) M PC with 512 MB RAM. That is less than 30 seconds per run.

To accurately incorporate real and detailed transportation network data into the simulation, we use *GeoKIT*, a state-of-the-art, Java-based Geographical Information System. The component-based approach adopted in development of our simulation model enables us to easily integrate future additions to our model. The DES developed as part of this dissertation provides a test bed for currently existing deployment scenarios. While our DES model is not a panacea for all, it allows for testing the feasibility and sensitivity of deployment plans under stochastic conditions prior to committing members of the military into harm's way.

Our main contribution is to develop a comprehensive, multi-modal, high-resolution, loosely-coupled and modular, extendable, platform independent, state-of-the-art GIS based simulation environment that views the deployment operations as end-to-end processes. Such a simulation environment for multi modal deployment planning and analysis does not exist.

Additionally, we simulate and analyze a typical real-world case study by using conventional methods and the rather novice *Nearly Orthogonal Latin Hypercube Sampling (NOLHS)* technique. We use a space-filling nearly orthogonal design of 29 factors and 257 runs to determine the factors that impact most on a deployment plan. We make 15 replications of each of the 257 runs (scenarios) to reach a total of 257x15=3855 computer runs compared to an experiment with 29 factors, each with only 2 levels and 15 replications per run, for a complete enumeration experiment (2^{29} x15= 8,053,063,680 computer runs!).

Keywords: discrete-event simulation; military deployment; event graphs; java; geographical information system, nearly orthogonal latin hypercube sampling.

ÖZET

ASKERİ BİRLİKLERİN İNTİKALİ İÇİN ÇOK-MODLU BİR KESİKLİ OLAY SİMÜLASYONU

Uğur Ziya Yıldırım
Endüstri Mühendisliği Bölümü Doktora
Tez Yöneticisi: Prof. Barbaros Ç. Tansel
Prof. İhsan Sabuncuoğlu
Ocak 2009

Bu tezde, askeri intikal planlarının muhtemel sonuçlarına ışık tutabilecek bir lojistik ve ulaştırma simülasyon modeli tanıtılmaktadır. Daha detaylı olarak bahsedecek olursak, gerçek hayatta karşılaşılan ve çok modlu bir ulaştırma ağını kullanarak kışlalarından hedef bölgelerine intikal eden büyük çaplı askeri birliklerin intikal problemini modellemekteyiz. İlk defa olarak, *Event Graphs (Olay Grafikleri)* ve *Listener Event Graph Objects (Dinleyen Olay Grafiği Nesneleri)* altyapısını kullanarak intikal problemininin simülasyon modelini oluşturduk. Kesikli olay simülasyon modelimizi geliştirirken, bir açık kaynak Java Uygulama Programlama Arayüzü olan *Simkit*'i kullandık ve ilaveler yaptık. Çoğunlukla kullandığımız çok çözünürlü yaklaşım sayesinde, bir intikal planının arzu edildiği şekilde yürüyüp yürümeyeceğini ve muhtemel problem sahalarını ne olabileceğini, literatürde genelde kullanılmakta olan farklı çözünürlü modellerin ardışık çalıştırılması yaklaşımına nazaran daha çabuk bir şekilde belirleyebilmekteyiz. Dört taburun intikalini içeren tipik bir intikal senaryosu modelinin 1.6 GHz Pentium(R) M özelliklerinde ve 512 MB RAM içeren bir bilgisayarda 60 defa çalıştırılması 25 ile 27 dakika arasında sürdü. Bu çalıştırma başına 30 saniyeye tekabül etmektedir.

Ulaştırma ağına ait gerçek ve detaylı bilgileri simülasyon modelinde kullanabilmek maksadıyla, teknolojinin son yeniliklerini içeren Java tabanlı bir coğrafi bilgi sistemi yazılımı olan *GeoKIT*'i kullandık. Simülasyon modelimizi geliştirirken kullandığımız modüler yaklaşım gelecekte yeni modüllerin kolayca eklenmesini sağlamaktadır. Bu tezin bir parçası olarak geliştirilen kesikli olay simülasyonu mevcut intikal senaryoları için bir test ortamı yaratmaktadır. Her ne kadar geliştirdiğimiz kesikli olay simülasyon modeli askeri intikale ait tüm sorunların panzehiri olmasada, askeri personeli tehlikeye atmadan önce intikal planlarının fizibilite ve duyarlılık analizlerini stokastik bir ortamda deneme fırsatı sunmaktadır.

En büyük katkımız kapsamlı, çok modlu, yüksek çözünürlü, esnek ve modüler, geliştirilebilir, değişik donanımlar üzerinde çalışabilen, teknolojinin son yeniliklerine sahip bir coğrafi bilgi sistemi yazılımı üzerinde çalışan ve intikali baştan sona modelleyen bir simülasyon ortamı yaratmamızdır. Çok modlu intikal planlaması ve analizi için böyle bir simülasyon ortamı bulunmamaktadır.

Ayrıca, tipik bir gerçek dünya probleminin simülasyon modelini klasik ve henüz nispeten yeni ortaya konmuş bir yöntem olan *Nearly Orthogonal Latin Hypercube Sampling (Yaklaşık Dikey Latin Hiperküp Örneklemesini)* kullanarak analiz ettik. 29 faktör ve 257 farklı faktör seviyesi (senaryo) içeren, örneklem uzayını dolduran ve yaklaşık dikey bir tasarım kullandık. Her senaryo için 15 tekrar yaparak modeli bilgisayarda 257x15=3855 defa çalıştırdık. Bunu tüm alternatifleri deneyen ve sadece ikişer seviye içeren 29 faktörlü bir tasarımda onbeşer tekrarlı bir deneyle kıyaslarsak, o zaman modeli bilgisayarda 2^{29} x15= 8,053,063,680 defa çalıştırmamız gerekecekti.

Anahtar sözcükler: kesikli olay simülasyonu; askerî intikal; olay grafikleri; java; coğrafi bilgi sistemi; yaklaşık dikey latin hiperküp örneklemesi.

ACKNOWLEDGEMENTS

I thank my advisors Prof. Barbaros Tansel and Prof. İhsan Sabuncuoğlu for their guidance, expertise, and patience throughout this dissertation research. With their support and encouragement, this study has been an enjoyable and invaluable learning experience for me. They are what kept me going as I struggled with details of this research in peculiar places of the globe such as Baku-Azerbaijan, Constanta-Romania, and Kabul-Afghanistan, and during deployment orders to northern Iraq. They are true role models, both as an academic and as a person, and there is still so much I can learn from them. It has really been an honor to work with these consummate academics.

I am indebted to members of my dissertation committee, Prof. Cevdet Aykanat, Prof. Berna Dengiz, and Assoc. Prof. Oya Ekin Karaşan, for showing keen interest in the subject matter and for accepting to read and review this thesis. Their remarks and recommendations have been invaluable. I am grateful to Prof. Mehmet Baray for his confidence in me.

I would like to express my thanks to Major İbrahim Akgün for his true friendship, support, and encouragement on this similar topic we have worked on.

I am also grateful to Erhan Çınar, Murat Durmaz, and Tolga Akçelik for their true friendship and help in integrating GeoKIT to our simulation software.

Last but not least, I thank my parents and sister for their unconditional love, support, understanding, and patience. Without them, this study would not have been possible. This is what kept me away from them over the years during Bayrams and holidays.

TABLE OF CONTENTS

LIST OF FIGURES .... **X**

LIST OF TABLES .... **XI**

ABBREVIATIONS .... **XII**

1. INTRODUCTION **1**

1.1. OVERVIEW 1
1.2. PROBLEM AND SYSTEM DEFINITION 3
1.3. BACKGROUND, PURPOSE, AND RATIONALE 7

2. BACKGROUND **13**

2.1. DEPLOYMENT 13
2.1.1. Mobilization, Deployment, Redeployment, Employment, and Demobilization Processes 13
2.2.1. Deployment Phases 14
2.3.1. Deployment Planning 28

3. THE MODELING ARTIFACTS **31**

3.1. THE PROGRAMMING LANGUAGE 31
3.1.1. Java Features and Terms 31
3.1.2. Why Java2™ ? 31
3.2. SIMKIT 32
3.2.1. Event List Implementation in Simkit 33
3.2.2. Starting and Stopping 33
3.2.3. Tasks in Running a Simkit Model 34
3.2.4. Listener Patterns 34
3.2.5. PropertyChangeListener Pattern 36
3.2.6. Collecting Statistics 36
3.2.7. Generating Random Numbers and Variates 37
3.2.8. Obtaining Simkit 37
3.2.9. The Reasons for Selecting Simkit and Event Graphs 38
3.2.10. The Listener Event Graph Objects (LEGOs) 41
3.3. THE GEOGRAPHICAL INFORMATION SYSTEM (GIS) 43

4. THE MODELING DETAILS **47**

4.1. COMPONENTS OF THE DEPLOYMENT SIMULATION 47
4.2. THE LAND COMPONENT 47
4.2.1. HomeBaseDeparture Class 47
4.2.2. LandMaintDelay Class 53
4.2.3. LandArrivalDestination Class 55
4.2.4. Exogenous (Input) and Endogenous (Output) Variables for the Land Component 56
4.3. THE SEA COMPONENT 59
4.3.1. LandArrivalAtSPOE Class 59
4.3.2. SeaMaintDelay Class 60
4.3.3. SeaArrivalAtSPOD Class 62
4.3.4. Exogenous (Input) and Endogenous (Output) Variables for the Sea Component 63
4.4. THE RAIL COMPONENT 66

4.4.1. LandArrivalAtRPOE Class ... *66*
4.4.2. RailMaintDelay Class ... *67*
4.4.3. RailArrivalAtRPOD Class ... *69*
4.4.5. Exogenous (Input) and Endogenous (Output) Variables for the Rail Component ... *70*
4.5. THE AIR COMPONENT ... 73
4.5.1. LandArrivalAtAPOE Class ... *73*
4.5.2. AirDelay Class ... *74*
4.5.3. AirArrivalAtAPOD Class ... *75*
4.5.4. Exogenous (Input) and Endogenous (Output) Variables for the Air Component ... *76*
4.6. THE SIMULATION EVENT LISTENER PATTERNS ... 78
4.6.1. The Simulation Event Listener Pattern for Land and Sea Components ... *80*
4.6.2. The Simulation Event Listener Pattern for Land and Rail Components ... *82*
4.6.3. The Simulation Event Listener Pattern for Land and Air Components ... *83*
4.7. ANIMATION AND GRAPHICAL USER INTERFACE (GUI) ... 84
4.8. EXPLANATIONS OF SOME OF THE OTHER JAVA CLASSES (MODULES) ... 91
4.8.1. Model Package (Component) ... *91*
4.8.2. GUI Package (Component) ... *92*
4.8.3. Statistics Package (Component) ... *92*
4.8.4. Network Package (Component) ... *92*
4.8.5. Explorer Package (Component) ... *93*

5. VERIFICATION AND VALIDATION ... 94

6. SIMULATION ANALYSIS OF A TYPICAL REAL-WORLD SCENARIO ... 100

6.1. SIMULATION ANALYSIS OF A TYPICAL REAL-WORLD CASE STUDY ... 100
6.2. INITIAL RESULTS ... 104
6.3. FURTHER ANALYSIS USING NOLH SAMPLING ... 107
6.3.1. Nearly Orthogonal Latin Hypercube Sampling and Its Application ... *108*
6.3.2. Further Results ... *110*
6.4. CONCLUSIONS AND FUTURE WORK ... 114

APPENDIX A ... 116

REFERENCES ... 132

LIST OF FIGURES

FIGURE 1. INTRA-THEATER MOVEMENT OF UNITS (ADAPTED FROM FM 4.01-011). 16
FIGURE 2. INTER-THEATER UNIT MOVEMENT (ADAPTED FROM FM 4.01-011). 17
FIGURE 3. A NOTIONAL SEAPORT OF EMBARKATION. (ADAPTED FROM FM 4.01-011) 19
FIGURE 4. A NOTIONAL AERIAL PORT OF EMBARKATION. (ADAPTED FROM FM 4.01-011) 20
FIGURE 5. A NOTIONAL MARSHALING AREA AT A SPOE (ADAPTED FROM FM 4.01-011) 22
FIGURE 6. A NOTIONAL SEAPORT OF DEBARKATION (ADAPTED FROM FM 4.01-011). 25
FIGURE 7. A NOTIONAL AERIAL PORT OF DEBARKATION. 26
FIGURE 8. RECEPTION AND ONWARD MOVEMENT. (ADAPTED FROM FM 4.01-011) 27
FIGURE 9. SIMEVENTLISTENER PATTERN. 36
FIGURE 10. EVENT SCHEDULING USING EVENT GRAPHS. 40
FIGURE 11. EVENT CANCELLING USING EVENT GRAPHS. 40
FIGURE 12. EVENT GRAPH FOR MULTIPLE-SERVER QUEUE. 41
FIGURE 13. SIMPLIFIED EVENT GRAPH FOR THE LAND COMPONENT'S HOMEBASEDEPARTURE CLASS. 49
FIGURE 14. A MORE DETAILED EVENT GRAPH FOR THE HOMEBASEDEPARTURE CLASS. 50
FIGURE 15. SIMPLIFIED EVENT GRAPH FOR THE LAND COMPONENT'S LANDMAINTDELAY CLASS. 54
FIGURE 16. SIMPLIFIED EVENT GRAPH FOR THE LANDARRIVALATDESTINATION CLASS. 56
FIGURE 17. SIMPLIFIED EVENT GRAPH FOR THE SEA COMPONENT'S LANDARRIVALATSPOE CLASS. 60
FIGURE 18. SIMPLIFIED EVENT GRAPH FOR THE SEA COMPONENT'S SEAMAINTDELAY CLASS. 61
FIGURE 19. SIMPLIFIED EVENT GRAPH FOR THE SEA COMPONENT'S ARRIVALATSPOD CLASS. 63
FIGURE 20. SIMPLIFIED EVENT GRAPH FOR THE RAIL COMPONENT'S LANDARRIVALATRPOE CLASS. 67
FIGURE 21. SIMPLIFIED EVENT GRAPH FOR THE RAIL COMPONENT'S RAILMAINTDELAY CLASS. 69
FIGURE 22. SIMPLIFIED EVENT GRAPH FOR THE RAIL COMPONENT'S ARRIVALATRPOD CLASS. 70
FIGURE 23. SIMPLIFIED EVENT GRAPH FOR THE AIR COMPONENT'S LANDARRIVALATAPOE CLASS. 74
FIGURE 24. SIMPLIFIED EVENT GRAPH FOR THE AIR COMPONENT'S AIRDELAY CLASS. 75
FIGURE 25. SIMPLIFIED EVENT GRAPH FOR THE AIR COMPONENT'S AIRARRIVALATAPOD CLASS. 76
FIGURE 26. THE SIMULATION EVENT LISTENER PATTERN FOR LAND AND SEA COMPONENTS. 80
FIGURE 27. THE SIMULATION EVENT LISTENER PATTERN FOR LAND AND RAIL COMPONENTS. 83
FIGURE 28. THE SIMULATION EVENT LISTENER PATTERN FOR LAND AND AIR COMPONENTS. 84
FIGURE 29. A SAMPLE SCREEN SHOT OF ANIMATION SHOWING A DEPLOYMENT USING ROAD (THIN BLACK LINES) AND RAIL NETWORKS (THICK BROWN LINES) AND SEA LINES (BEST VIEWED IN COLOR). 85
FIGURE 30. A SCREEN CAPTURE OF THE HOME BASE CREATION SCREEN. 86
FIGURE 31. A SCREEN CAPTURE OF THE TRUCK CREATION SCREEN. 87
FIGURE 32. A SCREEN CAPTURE OF THE TRUCK BREAKDOWN PROBABILITIES AND MAINTENANCE, LOAD/UNLOAD TIME DISTRIBUTIONS CREATION SCREEN. 88
FIGURE 33. A SCREEN CAPTURE OF LOAD CREATION. 89
FIGURE 34. A SCREEN CAPTURE OF PAX CREATION. 89
FIGURE 35. A SCREEN CAPTURE OF SPOE CREATION. 90
FIGURE 36. A SCREEN CAPTURE OF NETWORK MODULE. 90
FIGURE 37. SIMPLIFIED VERSION OF THE MODELING PROCESS (ADAPTED FROM SARGENT, 2001). 94
FIGURE 38. ONE OF THE NETWORK MODELS USED FOR VERIFICATION PURPOSES. 96
FIGURE 39. PERCENTAGE OF AVERAGE ON-TIME ARRIVALS VS. REDUCTION IN TMAX BY 12 HOURS FOR UNIT A. 106
FIGURE 40. PERCENTAGE OF AVERAGE ON-TIME ARRIVALS VS. REDUCTION IN TMAX BY 12 HOURS FOR UNIT D. 107
FIGURE 41. REGRESSION TREE FOR UNITS B AND C. 113

LIST OF TABLES

Table 1. Mobility planning, logistics and transportation models and simulations commonly used by military planners. 8
Table 2. Relationship between Event Graphs and Simkit (Adapted from (Jose, 2001)). 39
Table 3. Optimal numbers, movement routes, and movement times of loaded trucks. 98
Table 4. Units listed by their deployment components and major equipment. 101
Table 5. Time windows for units to be deployed. 102
Table 6. Initial delay times for each deployment component. 103
Table 7. Factors and their levels. 110
Table 8. 10 Runs for First 5 Factors of 257-Run Matrix. 111

ABBREVIATIONS

ADAMS	Allied Deployment and Mobility System
ALD	available-to-load date
APC	armored personnel carrier
APOE	air port of embarkation
APOD	air port of debarkation
DPP	deployment planning problem
DES	discrete-event simulation
EAD	earliest arrival date
EG	Event Graph
ELIST	the Enhanced Logistics Intra-Theater Support Tool
FEL	future event list
GUI	graphical user interface
JFAST	the Joint Flow and Analysis System for Transportation
LAD	latest arrival date
LEGO	Listener Event Graph Object
MOE	measures of effectiveness
MSR	main supply route
NATO	North Atlantic Treaty Organization
NOLHS	nearly-orthogonal Latin Hypercube Sampling
POE	port of embarkation
POD	port of debarkation
PORTSIM	the Port Simulation
PSA	port support area
RDD	required delivery date
RLD	ready-to-load date

RoRo	roll-on roll-off
RPOE	rail port of embarkation
RPOD	rail port of debarkation
SPOE	sea port of embarkation
SPOD	sea port of debarkation
TA	transportation asset
TLoaDS	the Tactical Logistics Distribution System
TPFDD	time-phased force deployment data
TRANSCAP	Transportation System Capability
SIMULOGS	Simulation of Logistics Systems
UTM	Universal Transverse Mercator
WGS	World Geodetic System

1. INTRODUCTION

In this chapter, we give an overview of the problem, define the problem and the system, present the background information, and give the purpose and rationale behind our work.

1.1. OVERVIEW

Regional and asymmetric threats and the increase in worldwide terrorist activity have made logistics and mobility increasingly important in our rapidly changing world. This study focuses on logistics and transportation simulations or computer-based planning tools that are used to provide insight into the potential outcomes of proposed logistical courses of actions prior to and after committing members of the military into harm's way. Specifically, we deal with the Deployment Planning Problem (DPP), defined and thoroughly described first by Akgün and Tansel (2007). DPP involves positioning of many military units to carry out a mission. During peace time, military units move from their home bases to their designated destinations using different transportation assets. This movement usually takes place on a multimodal (land, rail, sea, air, and inland waterways) transportation network. During a crisis, where time is of essence, it becomes critical to move troops and equipment with limited resources and on a short notice. The movement of the units must conform to a preplanned time-table called time-phased force deployment data (TPFDD). The TPFDD describes, among other things, the initial departure times of military units from their home bases, and their earliest and latest arrival times at their designated destinations. When many units need to deploy, the TPFDD is intended to coordinate their movement in order to efficiently use the existing transportation assets and network. It is also meant to prevent congestion at destinations and transfer points, where mode changes are necessary. Yet, creating TPFDD requires joint work of well-trained logistical and operational planners, and is very time consuming. Military deployment planners need a fast and accurate tool that takes into account the stochastic nature of events to analyze a military deployment plan.

A deployment plan may not always go as initially planned. Unexpected breakdown of transportation assets, road traffic accidents, and congestion at transfer points are some of the events that may disrupt a plan. A deployment involves simultaneous movement and utilization of many entities, resources, and transportation assets. Thus, a stochastic model is more suitable for analyzing this truly hard real-world problem that deals with expensive military hardware and irreplaceable human life.

Existing models and simulations are of varying resolutions. Most of the time, higher-resolution models provide input to the lower-resolution models. This makes it necessary to run several models in succession for analysis. But such a set-up takes a long time. Almost all of the deployment related models require a specific hardware system to run on. Yet, military usually employs different hardware systems, and thus it would be useful to have models or simulations which can run on multiple platforms. A detailed and accurate representation of the transportation network and infrastructure is necessary for realistic analysis. Thus, there is a need for a geographical information system to be used with deployment models or simulations. Currently existing models or simulations either do not have this capability or have a limited representation of geographical information. Furthermore, not all transportation modes are modeled in all deployment models or simulations, which makes it necessary to run at least two models in succession for a large deployment scenario. This again increases set-up and run times. Thus, it is desirable to have a multi-modal simulation model.

For these reasons, we have decided to develop a multi-modal, platform-independent, discrete-event simulation model of military deployment with accurate transportation network and infrastructure data, and a high-resolution except at transfer points, allowing planners to develop and analyze plans in a relatively short time.

1.2. PROBLEM AND SYSTEM DEFINITION

The DPP is fully described in Akgün and Tansel (2007). Most of the following is borrowed from their description and rephrased as necessary.

The DPP deals with the movement of many military units from their Areas of Responsibilities (home bases) to their Tactical Assembly Areas (final destinations). The movement could be either an intra-theater or an inter-theater type. Intra-theater movement can be regarded as the movement of units using different modes of transportation (e.g., land, sea, air, and rail) inside a country's borders. Inter-theater movement refers to the movement of units between countries using air and sea assets (strategic deployment). Once the units reach the destination country, then other available modes of transportation can be utilized inside that country. In this context, the terms "theater" and "country" are used synonymously.

During intra-theater movement, a unit may go directly from its home base to its final destination throughout the entire journey using a single mode of transportation assets (TAs) on a given mode of transportation network that supports the movement of the TAs under consideration. It may also use in succession any of land, rail, sea, or air transportation networks and the TAs dedicated to them, making mode changes as necessary along the way. However, the fewer the mode changes are at transfer points, the easier is the deployment. If a transfer is necessary, the initial movement from home bases is by ground transportation to a transfer point (a location where the movement switches from one mode of transportation to another). Main transfer points are harbors, train stations, and airports. At these locations, the pax (troops) and cargo (weapon systems, material, equipment, and supplies) of a unit are transferred from one set of TAs to another set that operate on a different network. This location is also called a Port of Embarkation (POE). The next mode change location, where the items are offloaded and loaded onto another set of TAs is called a Port of Debarkation (POD). These may be sea, rail, and air POEs or PODs. Inter-theater movement differs from intra-theater movement only by its use of strategic lift (air and sea) assets to reach the next theater of operations (Akgün and Tansel, 2007).

At transfer points, units usually queue up before being loaded on vessels. This location is called a *staging area* where units wait and prepare for shipment. A staging area can be regarded as a service point, i.e. one with a certain capacity of material handling equipment and load/unload docks. When there is not enough capacity at a staging area to hold large number of deploying units, a *marshalling area* is operated. A marshalling area can be regarded as a waiting/parking place prior to entering the staging area. It helps provide an uninterrupted flow of items through their transfer points. Staging/Marshalling areas are also operated at home bases and destinations (Akgün and Tansel, 2007).

A unit may be divided into three components (forward party, pax party, and cargo party) during deployment. Ground movement is usually conducted in convoys to maintain the unity of the component, and the size of the convoys may vary depending on operational/tactical objectives and limitations. The synchronization of departures of these components from their home bases and their arrival at their designated destinations is dictated by operational requirements, threat level, availability and capacity (lanemeter, seat, volume, weight) of lift assets, and the current conditions of transportation infrastructure (Akgün and Tansel, 2007).

A unit will usually use its own (organic) TAs to conduct a deployment. However, for heavy lift requirements (for example tanks and artillery pieces) over long distances, TAs of other military transportation units may have to be used. In addition, outsourcing of TAs from national civilian companies or other nations may be required depending on the distances and numbers and sizes of units involved in the deployment.

While time is of essence during a crisis, cost may be of main concern during peace time. The source of TAs used affects the cost and timing issues of unit movements. For example, outsourced TAs may not be available on time and leasing costs are associated with them. In addition, unpredictable stochastic events (breakdowns, accidents, delays etc.), load/unload/idle times at home bases/destinations/transfer points, convoy speeds, and speeds of transportation assets need to be taken into account to determine if a plan of deployment may be realized in actuality.

The planning for a particular deployment may take place beforehand. This is called *deliberate* planning when time is not a critical factor. When the time available for planning for actual deployment of armed forces is short, this is called *crisis* planning or *time-sensitive* planning where the planning process must be quick and flexible to adapt to changing situations. Deliberate planning may contribute to time-sensitive planning. Whether deliberate or not, each deployment plan has a TPFDD which at least includes units' transportation requirements by type and quantity, and movement data by mode, earliest times of departures from home bases, and earliest and latest times of arrivals at POEs / PODs / destinations. It divides a unit's components by transportation mode, ports of embarkation or debarkation, and movement dates.

1.3. BACKGROUND, PURPOSE, AND RATIONALE

There exist deployment planning models and simulations with varying levels of detail and purpose. For a more comprehensive survey of military planning systems and a review of strategic mobility models supporting the defense transportation system, the interested reader is referred to Boukhtouta et al. (2004) and McKinzie and Barnes (2003). It is possible to classify military deployment models and simulations into two groups depending on their level of resolution and the purpose of use.

The first group includes relatively low-resolution models and simulations that may be used to model deployment of military units between theaters of operation (e.g., from Turkey to Afghanistan) or inside a theater of operation (e.g., inside Turkey). Deployments between different theaters of operation using only air and sea transportation assets are referred to as *strategic deployment*. The most frequently used modeling tools (software) for modeling strategic deployments are NATO's *ADAMS* (*Allied Deployment and Movement System*) and U.S. Military's *JFAST* (*The Joint Flow and Analysis System for Transportation*). A technical guide for ADAMS is provided by Heal and Garnett (2001) and general information for JFAST is available at <http://www.jfast.org>. An example of simulations modeling deployment inside a theater of operation is *ELIST* (*The Enhanced Logistics Intra-Theater Support Tool,* Groningen et al., 2005).

The second group includes higher-resolution models and simulations that may also be used to provide input to the models in the first group. Examples of these models and simulations are *TLoaDS* (*The Tactical Logistics Distribution System,* 1999) and *PORTSIM* (*The Port Simulation,* 2004). Other important examples of such models are TRANSCAP (Transportation System Capability, 2006), SIMULOGS (Simulation of Logistics Systems, 2006), and Simulation of Transportation Logistics (2002). Table 1 gives a comparison of the aforementioned models in terms of their characteristics for multi-modality, platform independence, GIS-support, input/output analyzer module, and discrete-event simulation modeling capability.

TABLE 1. Mobility planning, logistics and transportation models and simulations commonly used by military planners.

Simulations/ Models	**Multi Modal (Yes/No)**	**Air (A), Rail (R), Sea (S), Land(L)**	**Platform Independence (Yes/No)**	**GIS Support**	**Input/ Output Analyzer Module**	**Discrete Event Simulation (Yes/No)**	**Comments**
ADAMS	Yes	A,S	No	Limited	None	No	A NATO model for strategic deployment
JFAST	Yes	A,S	Yes	Limited	Both	Both	A classified, joint US model for strategic deployment
ELIST	Yes	A,R,S,L	Yes	Limited	Both	Yes	For intra-theater support planning
TLoaDS	Yes	A,R,S,L	No	Limited	Both	Yes	Built Using EXTEND™ and SDI Industry Pro
PORTSIM	Yes	R,S,L	No	Limited	Both	Yes	Simulates seaport operations and determines throughput at the port
TRANSCAP	Yes	L,R	Partially	No	None	Yes	Models deployment from US Army installations
SIMULOGS	No	L	Partially	Limited	None	Yes	Runs on Sun Unix Workstation and PCs with Windows NT
Simulation of Transp. Logistics	Yes	A,S	No	No	Both	Yes	Based on ARENA, uses VBA and Excel
The Simulation Model proposed in this thesis	Yes	A,R,S,L	Yes	Yes	None	Yes	Written in Java and uses GeoKIT for GIS support.

Except for the input/output analyzer module, none of the models or simulations relevant to this study in Table 1 meets all of the other four criteria fully. Our proposed simulation model is listed at the bottom of Table 1. We have developed a simulation model of military deployment that meets all of these four criteria and has a high-resolution except at transfer points. The modeling level of detail at transfer points for infrastructure is low resolution in that, we use aggregate capacities for each individual transfer point and time delays at these locations for problems such as maintenance breakdowns etc. Even though the level of resolution at transfer points can be increased, we refrained in the dissertation from doing so as the detailed modeling at each transfer point can be separately done without significantly affecting the overall structure of the deployment simulation. The detailed model at a transfer point determines the overall delay induced at that location, which in our case is reflected into our model by appropriate distributions. We use high-resolution modeling for the modeling of transportation assets, cargo, and resources, which includes specific dimensions of cargo, and capacities, speeds, and dimensions of resources and transportation assets. Furthermore, the GIS data we use in the simulation is also of high-resolution. It specifies, among other things, the capacities and classifications of bridges, roads, and other detailed data of related transportation infrastructure. As for the input/output analyzer modules, an input analyzer may be obtained through commercial-off-the-shelf products. An output analyzer module is in development and left as future work to the study at hand.

Our high-resolution except-at-transfer-points approach to modeling the DPP allows us to obtain quick first-cut insights into potential outcomes of a deployment plan without compromising reality. For example, for a typical deployment scenario for four battalions, run times are between 25 to 27 minutes for 60 runs of the model on a 1.6 GHz Pentium(R) M PC with 512 MB RAM. That is less than 30 seconds per run. While our high-resolution except-at-transfer-points simulation model is not a panacea for all, it provides a realistic and quick litmus test for the applicability of existing deployment plans and allows a quick construction of contingency deployment plans. It is an all-encompassing model for all modes of transportation. Yet, our model's extendable architecture is flexible enough to accommodate future addition of higher-resolution sub-modules intended especially for detailed modeling at transfer points. Moreover, our model is generic enough to be used in commercial logistics applications after some problem-specific modifications.

The aim of this dissertation is to develop a logistics and transportation simulation for use in the analysis of military DPP. We apply, for the first time, the *event graph (EG)* methodology and *listener event graph object (LEGO)* framework to create a multi-modal discrete-event simulation model of the DPP. EGs and LEGOs provide a simple yet powerful and elegant way of representing discrete event simulation (DES) model of deployment, and enable easy creation of component-based models of a real-world military problem. The high-resolution approach that we take in most part, allows us to estimate whether a given plan of deployment will go as intended, and determine prospective problem areas in a relatively short time compared to other existing simulations because of the absence of the need to use several models of differing resolutions in succession. The short run times achieved demonstrate this. The very accurate and detailed GIS data, and the detailed data used in modeling of entities, resources, and military equipment in the simulation permit us not to exchange reality in favor of shorter run times. We extend Simkit by writing additional Java classes that are specific to military deployment. The component-based approach adopted in development of our simulation model enables us to easily integrate future additions to our model. The DES developed as part of this dissertation provides a test bed for currently existing deployment scenarios.

Our main contribution is to develop a comprehensive, multi-modal, high-resolution, loosely-coupled and modular, extendable, platform independent, state-of-the-art GIS based simulation environment that views the deployment operations as end-to-end processes. To the extent of our knowledge, such a simulation environment for multi modal deployment planning and analysis does not exist in the literature.

The rest of this dissertation is organized as follows: Chapter II gives background on terminology and basics of deployment planning. Chapter III provides information on the programming language of choice, Java2™, the simulation library written in Java2™ called Simkit, and the Geographical Information System, GeoKIT, used with the transportation simulation. Chapter IV provides the event graphs (the conceptual and logical models) of the simulation software and its modules developed. Verification and Validation issues are discussed in Chapter V and analysis of a typical real-world case-study scenario is explained and analyzed in Chapter VI.

2. BACKGROUND

This chapter provides basic and necessary background information about deployment planning.

Our aim here is two folds: The first one is to acquaint the reader with military-specific terminology used in deployment of military units. The second is to lay out some of the requirements modeled by the transportation simulation being developed as part of this dissertation to assist in planning of military deployments. The concepts and material contained in this chapter is a summary of relevant parts of Field Manual (FM) 4-01-11, FM 55-10, FM 55-65, FM 100-17, FM 100-17-3. For further details on military-specific terminology, the interested reader is referred to these references which most of the material contained here follows. The reader familiar with military-specific terminology used in transportation and deployment may skip to Chapter 3.

2.1. DEPLOYMENT

2.1.1. Mobilization, Deployment, Redeployment, Employment, and Demobilization Processes

Force projection involves *mobilization*, *deployment*, *redeployment*, *employment of forces*, and *demobilization.*

Mobilization is the process by which all or parts of the Armed Forces are brought to a state of readiness for war or other national emergencies such as natural disasters.

Deployment means preparing and moving the force and its supplies to the area of operations in response to a crisis or natural disaster.

Redeployment is the preparation for and movement of the force and its material deployed from an area of operation to another or to its designated home base.

Employment is the use of forces in their areas of operations to carry out the mission.

Demobilization is the act of returning the force and materiel to a premobilization posture or to some other approved posture.

2.2.1. Deployment Phases

Military units may be deployed between theaters of operation (e.g., from Turkey to Afghanistan) or inside a theater of operation (e.g., inside Turkey). Deployments between different theaters of operation (*inter-theater*) using only air and sea transportation assets are referred to as *strategic deployment*. The phases of a strategic (inter-theater) deployment process are:

- Predeployment activities
- Movement to the Port of Embarkation (POE)
- Strategic lift
- Reception at the Port of Debarkation (POD)
- Theater onward movement

If the deployment is inside a theater of operation, it is referred to as an *intra-theater* deployment which may involve all modes of transportation. The following two figures show possible *intra-theater* and *inter-theater* movement steps of a unit from an origin location to a tactical assembly area (TAA - i.e., the final destination), respectively. Intra-theater movement can be regarded as the movement of units using different modes of transportation (land, sea, air, and rail) inside a country (theater) as shown in Figure 1. On the other hand, inter-theater movement of units can be regarded as the movement of units using only air and sea assets (strategic lift) between countries (theaters). Once the units reach the destination country (theater), then other available modes of transportation can be utilized inside that country (theater) as depicted in Figure 2.

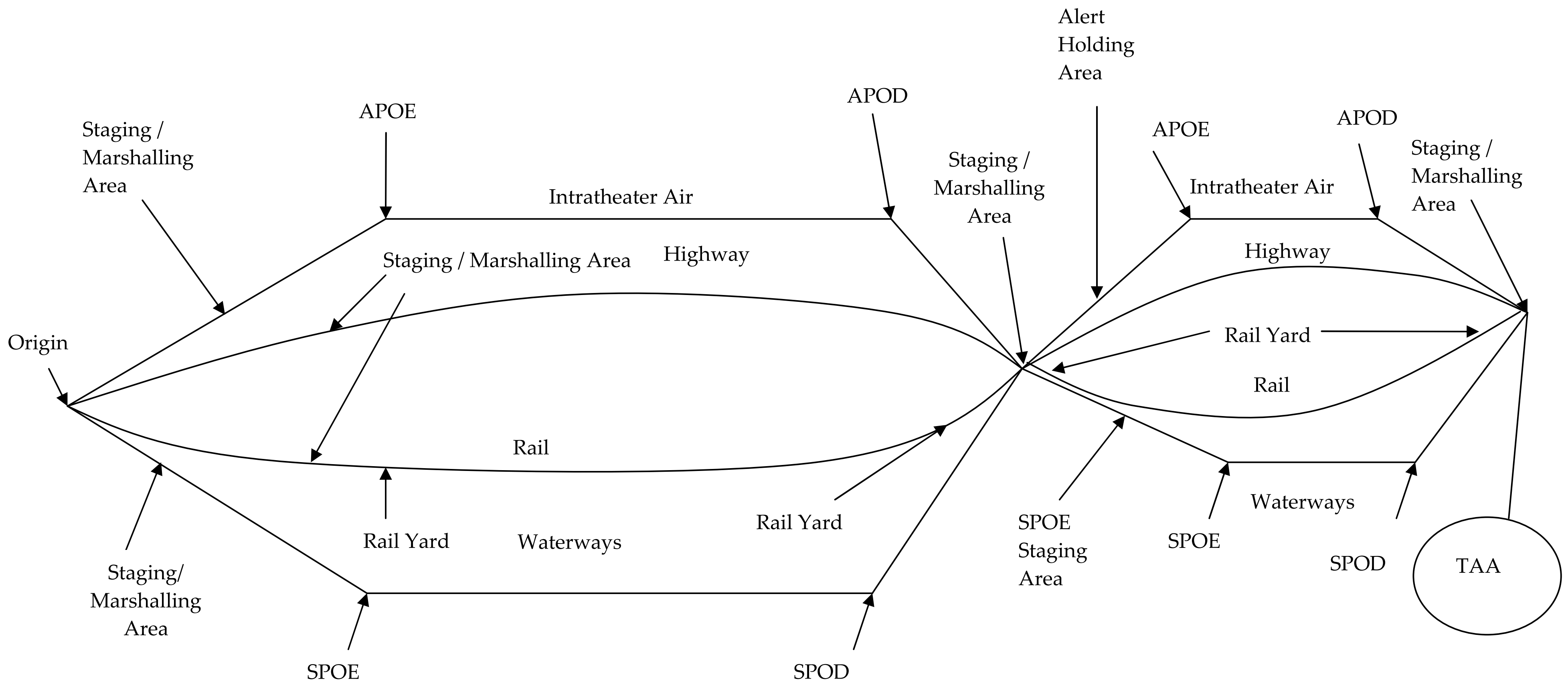


Figure 1. Intra-theater movement of units (Adapted from FM 4.01-011).

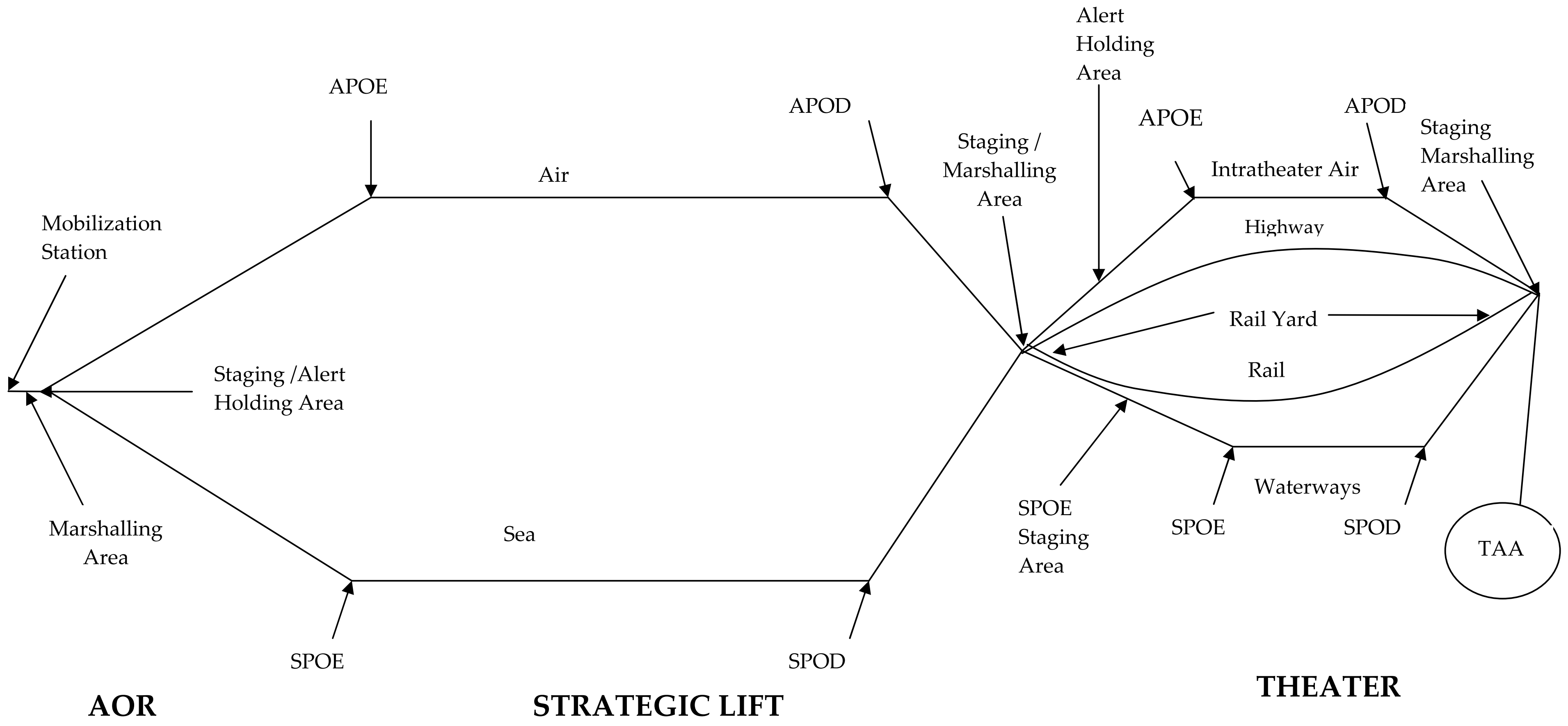


Figure 2. Inter-theater unit movement (Adapted from FM 4.01-011).

2.2.1.1. Predeployment Activities

Predeployment activities are those that units carry out based on initial notification, warning orders, and alert orders for operations. They are aimed at preparing forces for deployment. Those that are relevant to the simulation model being developed include echeloning (organizing units for movement, i.e., dividing echelons into advance party, pax party, and cargo party), tailoring (adding or subtracting units to/from a planned task organization based on the mission and available lift), deployment planning, and equipment maintenance.

2.2.1.2. Movement to the POE

The movement to the POE is initiated upon receipt of a movement directive. Units are validated and configured for movement. Units may move from their home stations to the POE using ground and/or rail transportation. If ground transportation is used, movement is carried out in convoys. Convoy is the preferred method of moving wheeled vehicles to ports and other facilities that are within one day's distance while rail is the preferred method for moving all wheeled vehicles over one day's driving distance from the port. The accepted deployment method for rotary wing aircraft is by self-deployment to the POE. Fixed wing aircraft are normally self-deployable to the area of operation. There are two types of POE, sea and aerial. Figures 3 and 4 depict a notional seaport of embarkation SPOE and an aerial port of embarkation APOE, respectively.

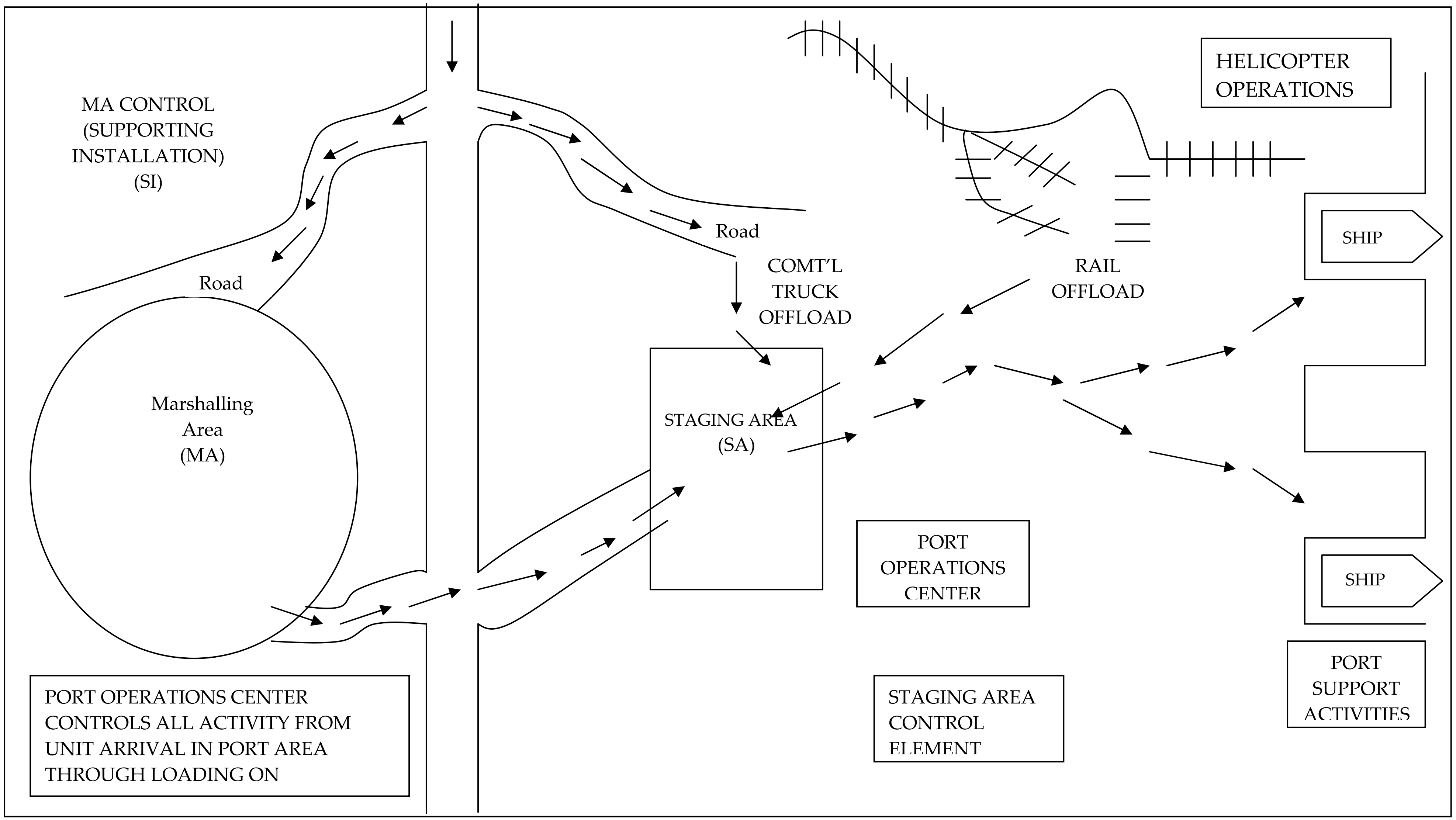


Figure 3. A notional seaport of embarkation. (Adapted from FM 4.01-011)

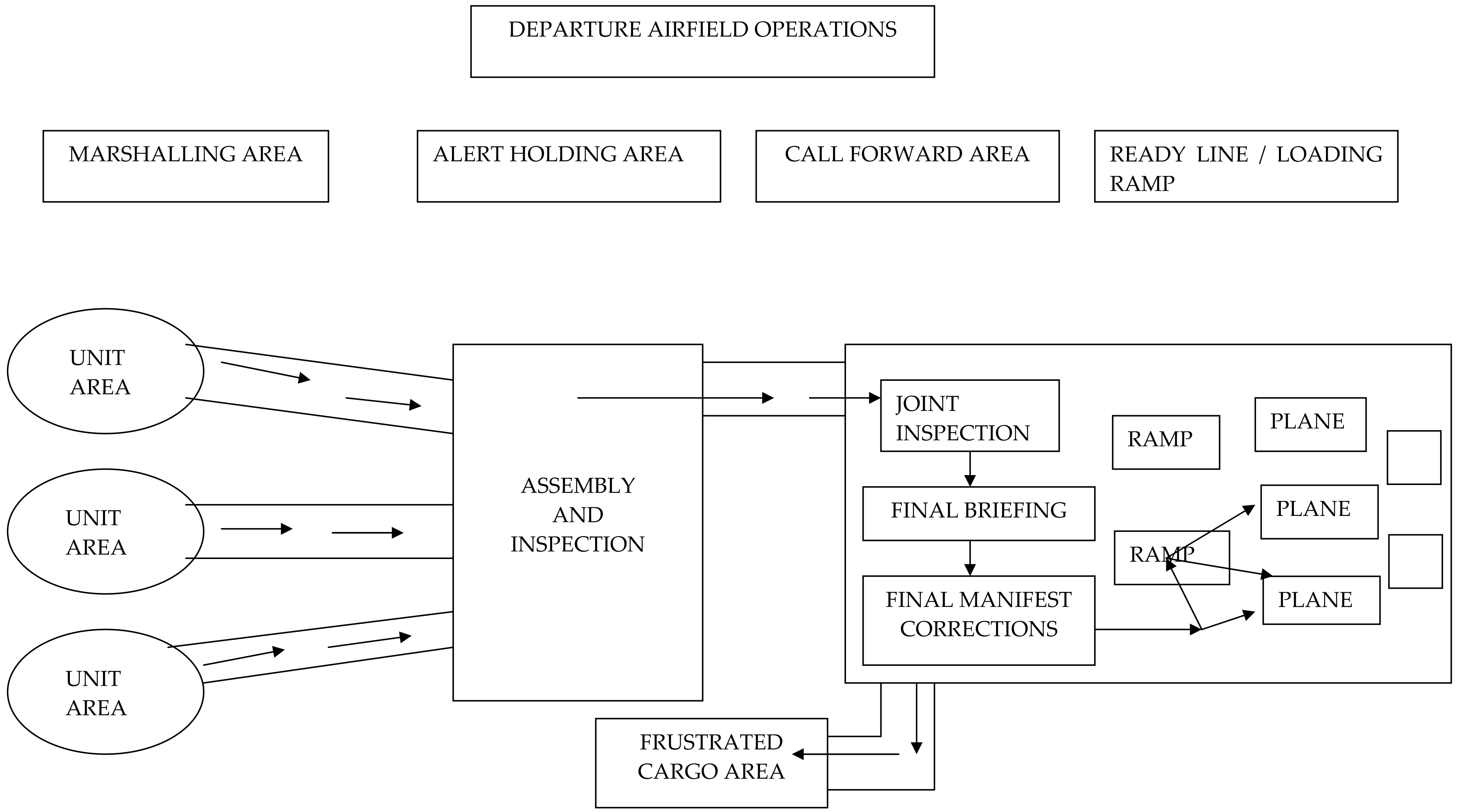


Figure 4. A notional aerial port of embarkation. (Adapted from FM 4.01-011)

Units that use a SPOE are held in the port *staging area* to prepare for shipment before being loaded on vessels. However, in many cases, there is not enough room at the sea terminal to stage the entire unit or large numbers of units scheduled to move at the same time. In such cases, a *marshaling area* is operated. Figure 5 shows a notional marshaling area at a SPOE. The primary purpose of a marshaling area is to provide a location to receive unit personnel, equipment and supplies, and configure them for overseas movement by sea prior to entering the staging area. As the vessel gets ready, the units are called from the marshaling area to the staging area based on a call forward plan.

The two areas, staging area and marshaling area, serve much the same service. The distinction between them is that the owning command retains responsibility and accountability for the shipment in the marshaling area while port commander assumes the custody of equipment and supplies in the staging area.

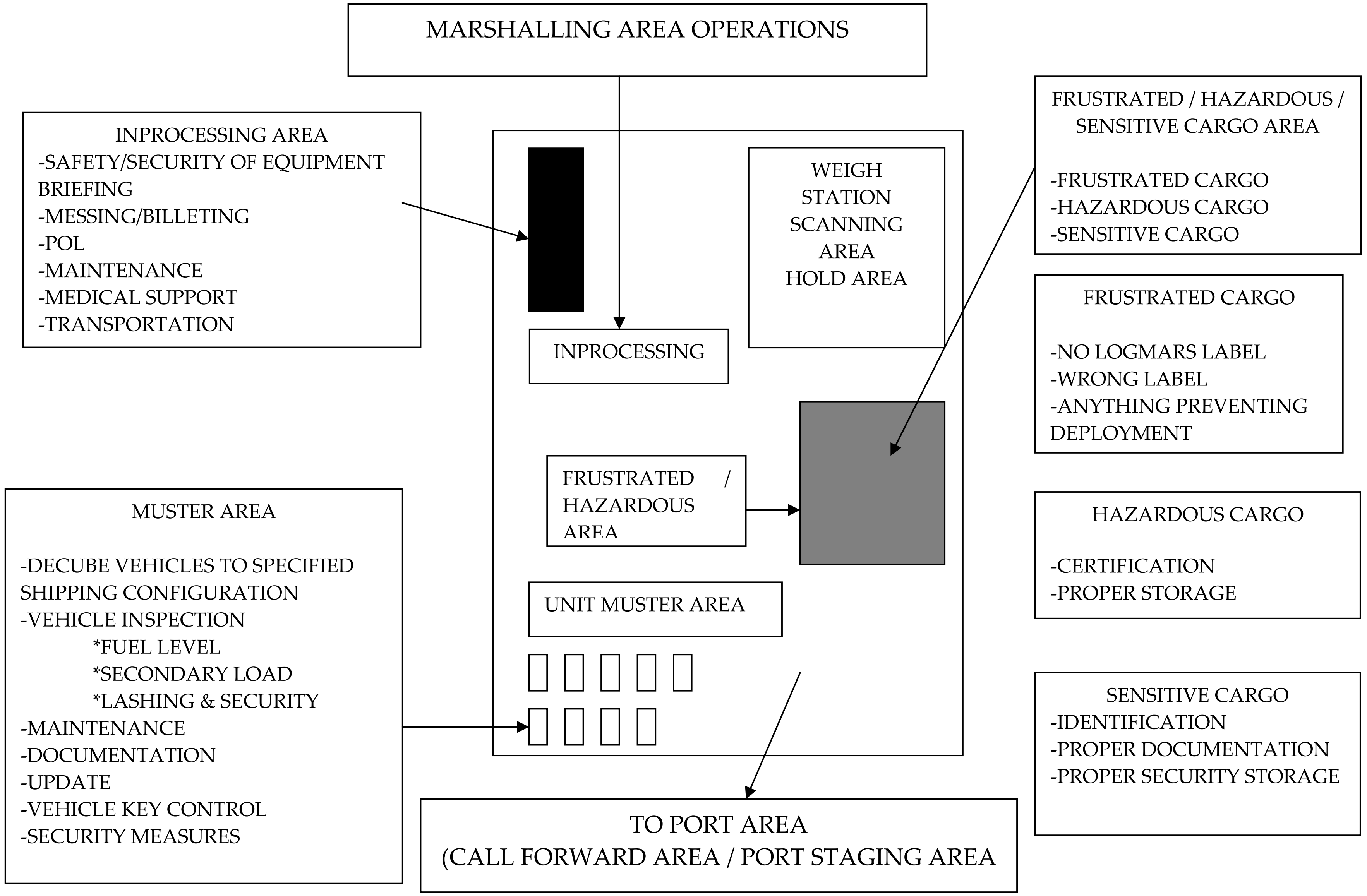


Figure 5. A notional marshaling area at a SPOE (Adapted from FM 4.01-011)

2.2.1.3. Strategic Lift

Strategic lift begins with departure from the POE and ends with arrival at the POD, which is an APOD or a SPOD. Normally troops are deployed by air and equipment by sea. The estimated arrival of equipment at the SPOD normally dictates when personnel are airlifted to the theater. Synchronizing the arrival by air or sea into the area of responsibility at the SPOD and personnel arriving at the APOD is necessary.

Sealift capability involves a variety of vessels, such as RoRo (Roll on \ Roll off) ships, fast sealift ships, ready reserve ships, and chartered ships. Airlift is used primarily to transport personnel, selected vehicles, and unit equipment.

2.2.1.4. Reception at the POD

This implies the arrival of forces at the POD in the area of operation and ends with the departure of the forces from the POD.

Figure 6 shows a notional seaport of debarkation. At SPODs, discharged unit equipment, materiel, and supplies are held in the *staging area* which is established for the transshipment and accounting of equipment. Figure 7 depicts a notional aerial port of debarkation. At APODs, units go through off-load ramp area, holding area, and marshaling area to prepare for onward movement.

2.2.1.5. Theater Onward Movement

Figure 8 shows the relationship between reception and onward movement processes. During this critical phase of deployment, the availability of transportation again takes an important role to keep units and supplies moving forward directly to the area of employment.

Theater onward movement takes place utilizing a movement program that incorporates convoy, rail, and contracted assets to ensure the forward and concurrent movement of troops and supplies. Truck terminal and trailer transfer points are established for use in line-haul or relay operations. Rail transport, when available, will also be used to transport heavy tracked vehicles and other large items of equipment as far forward as possible. From those points on, heavy equipment transporters complete the movement to destination.

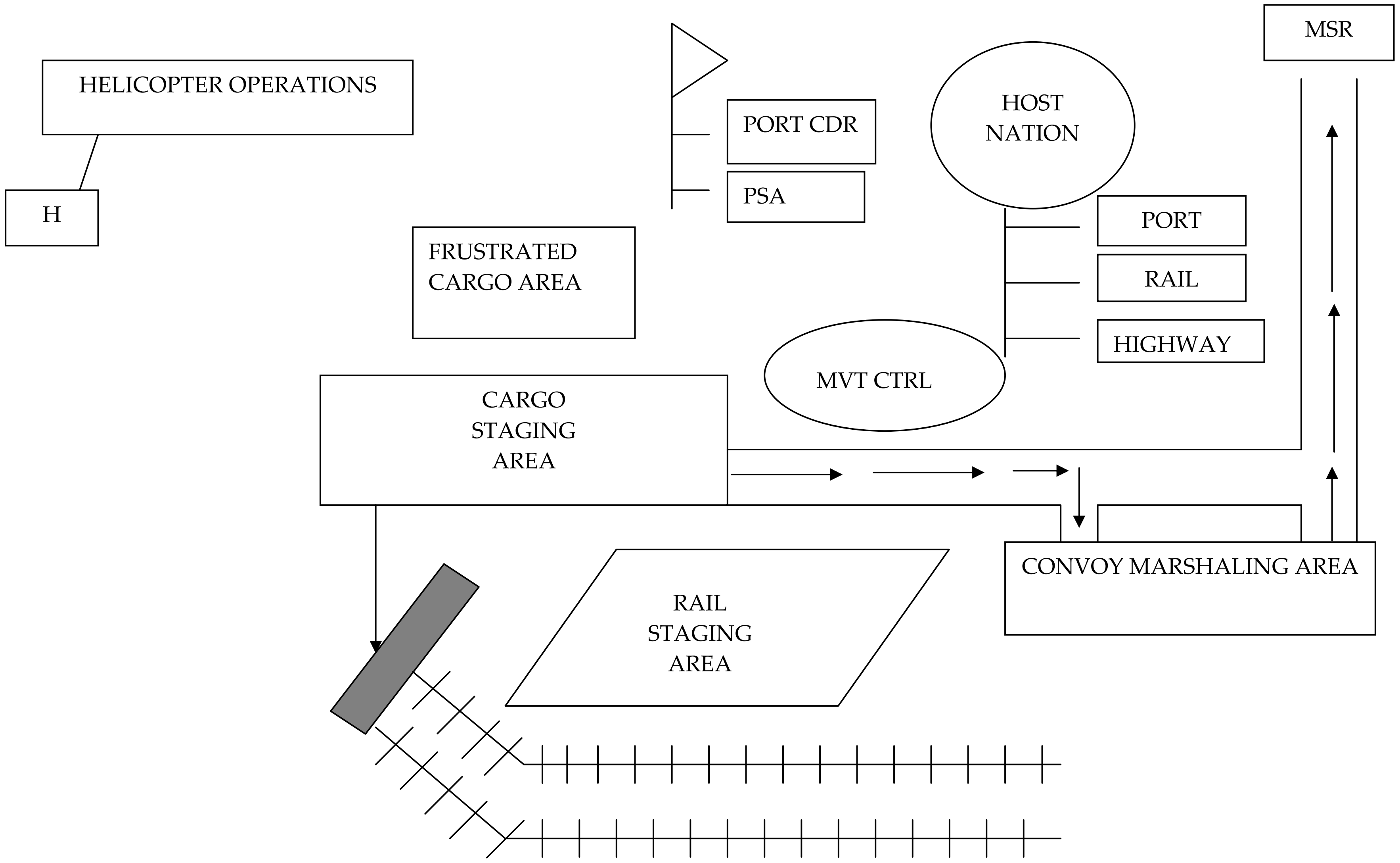


Figure 6. A notional seaport of debarkation (Adapted from FM 4.01-011).

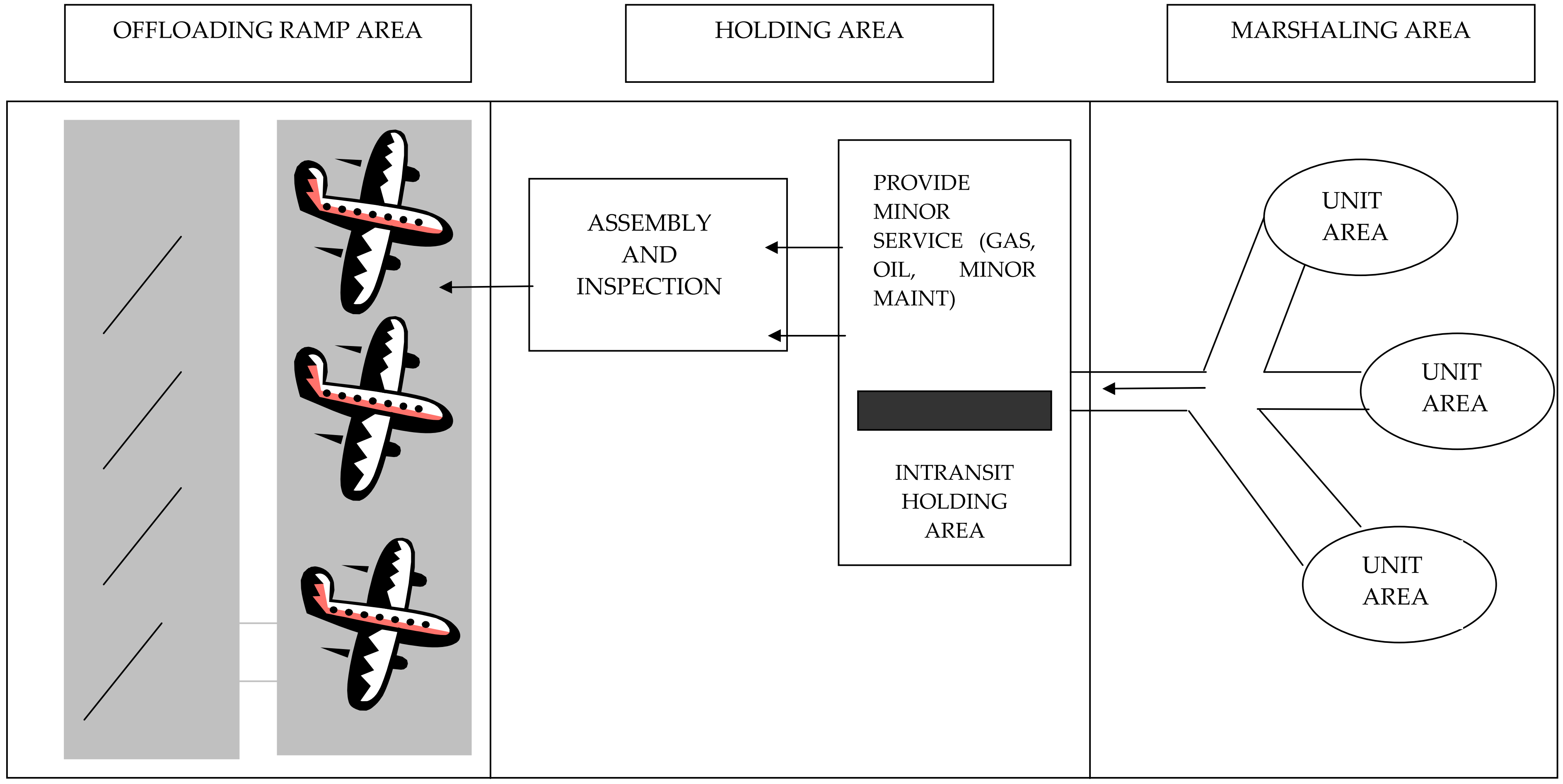


Figure 7. A notional aerial port of debarkation.

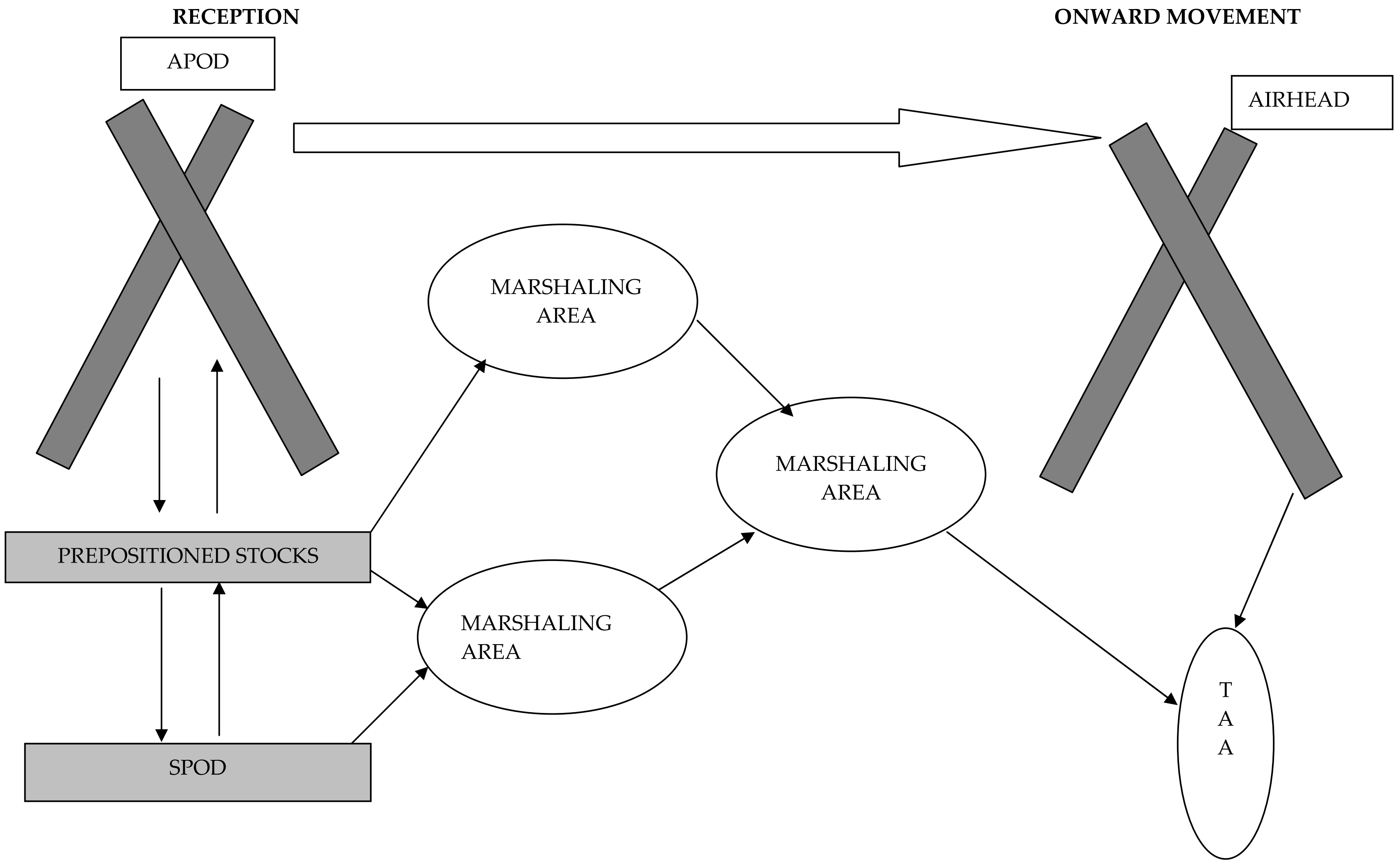


Figure 8. Reception and onward movement. (Adapted from FM 4.01-011)

2.3.1. Deployment Planning

The planning for a particular deployment contingency may take place beforehand. This is called deliberate planning when time is not a critical factor. When the time available for planning for actual deployment or employment of armed forces is short, the planner uses crisis-action procedures. The overall process of crisis-action planning mimics that of deliberate planning, but is much more flexible to accommodate requirements to respond to changing events.

2.3.1.1. Deliberate and Crisis-Action Planning

The highest military commands (the services or general staffs) develop deliberate plans during peace time, for potential contingencies within their areas of responsibility. The following definitions are relevant to the work at hand in terms of deliberate and crisis-action planning:

Each plan has *time-phased force and deployment data* (*TPFDD*) which includes personnel requirements, equipment requirements by type and quantity, and movement data by mode. It divides the unit by transportation mode, ports of embarkation or debarkation, and movement dates. For example, it identifies the advance party of a unit going by air when the unit's main body and equipment are going by sealift. During a crisis, the responsible command may update the plan with current information and in conjunction with supporting organizations and create operations orders for execution.

Following are the definitions of dates that are used to time-phase the movement plans according to the TPFDD.

Ready-to-load date (RLD): The RLD is the TPFDD date when the unit must be prepared to depart its origin.

Available-to-load date (ALD): The ALD is the TPFDD date when the unit must be ready to load on an aircraft or ship at the POE.

Earliest arrival date (EAD): The EAD is the earliest date that a unit, a re-supply shipment, or replacement personnel can be accepted at a POD during a deployment. The supported combatant commander specifies the EAD.

Latest arrival date (LAD): The LAD is the latest date when a unit, a re-supply shipment, or replacement personnel can be accepted at a POD to support the concept of operations. The EAD and LAD are used to describe a delivery window for transportation planning.

Required delivery date (RDD): The RDD is the date when a unit, a re-supply shipment, or replacement personnel must arrive at a POD and complete off-loading to support the current of operations.

Responsible command, transportation planners and the deploying forces use the RDD to determine critical interim dates, such as the date the unit must do the following:

- Depart the origin installation or home station.
- Arrive at the POE.

- Arrive at intermediate support bases if necessary.
- Arrive at the POD.

For crisis-action planning, where time is a critical factor, the above definitions are still valid. However, the planning could involve plan development from scratch or modifiying an existing deliberate plan already developed. In either case, the requirement for a quick analysis methodology that can generate a feasible TPFDD is crucial.

Before closing this chapter, we should note that for inter-theater operations, the concepts of a rail port of embarkation (RPOE) and a rail port of debarkation (RPOD) need to be added to POEs and PODs already mentioned. A RPOE and a RPOD are simply railyards where a mode change occurs between land and rail transportation modes.

Chapter III presents the tools that are used and extended to develop the DES for deployment planning.

3. THE MODELING ARTIFACTS

Section 3.1 of this chapter explains and justifies the reasons for using Java2™ as the programming language of choice. Section 3.2 introduces the discrete event simulation library, Simkit, written in Java2™. Section 3.3 introduces the reader to the basic concepts of a GIS system and, in particular, to the GIS system of choice, GeoKIT, used as part of the simulation.

3.1. THE PROGRAMMING LANGUAGE

The programming language of choice for the proposed dissertation is Java2™.

3.1.1. Java Features and Terms

Introductory information on Java™ features and terms can be found in Cornell and Horstmann (1999), Lewis and Loftus (2000), Narasimhan and Winston (2001) and Chapman (2000). More advanced material can be found in Cornell and Horstmann (1999 and 2000) and at www.javasoft.com.

3.1.2. Why Java2™ ?

For use on computers connected to the Internet, Java2™ programs run on a wide variety of hardware platforms. They can be loaded dynamically via a network and provide features that facilitate robust behavior.

For applications that have nothing to do with networks, it is a truly completely object-oriented programming language. One justification for our choice of Java2™ as the programming language of choice for this dissertation is expressed plainly in (Nance and Rioux, 2002). That is; the "...development methodologies of simulations for the purpose of analysis continue to transition from a procedural to an object-oriented paradigm due to an attempt to obtain the premises of object oriented programming such as improvements in reliability and reusability." Object-oriented (O-O) paradigm provides a consistent means of handling these problems by viewing the world as a set of autonomous modules that interact to solve a complex task, where each object is responsible for a specific part of the task. The O-O paradigm simplifies computer programming tasks and promotes reusability (Joines and Roberts, 1999).

Another justification is that Java programs can work on multiple tasks simultaneously and automatically recycle memory. Moreover, Java programs are independent of the operating system (Narasimhan and Winston, 2001). The military usually employs different hardware and operating systems, and thus it would be useful to have models or simulations which can run on multiple platforms. Furthermore, the geographical information system and the simulation library used as part of this research are also both written in Java2™.

3.2. SIMKIT

Following information on Simkit is mostly adapted from Buss (2001), Buss (2002), Buss and Sanchez (2002), Buss and Ruck (2004).

Simkit is an Application Programming Interface (API) for implementing Discrete Event Simulation (DES) models. It is written in Java2™ and runs on any operating system with Java2™ installed. Simkit adopts DES as its fundamental world view and does not directly implement other world views such as process/resource. Although this makes certain simple models slightly more complex, a pure DES world view provides more flexibility and modelling power than a pure process-oriented world view. In particular, every model that can be represented in the process world view can also be represented in a pure DES world view; the reverse is not true.

3.2.1. Event List Implementation in Simkit

All DES frameworks require an implementation of a Future Event List (FEL) to operate. Simkit implements a FEL in a class called *simkit.Schedule* that consists entirely of static methods and variables. Simkit attempts to hide the details of the FEL from the simulation modeller. Instead of directly placing events on the FEL, the programmer invokes the *waitDelay()* method on an instance of *simkit.SimEntityBase*.

When using Simkit as the basis for a Web Service, it becomes desirable to have multiple simulations running independently. As of version 1.2.14, there is the possibility of creating more than one FEL.

3.2.2. Starting and Stopping

The simulation run is controlled by the Schedule class, which also houses the FEL. Schedule initiates the run when there is a call to its startSimulation() method, which executes the FEL. Simulation continues executing until the FEL is empty. There are essentially four ways in Simkit by which this can occur:

(1)The FEL empties naturally of its own accord;

(2)There is an explicit call to Schedule.stopAtTime(double) before Schedule.startSimulation() is invoked;

(3)There is a call to Schedule.stopOnEvent(String, Class[], int) before Schedule.startSimulation() is invoked; and

(4)There is a call to Schedule.stopSimulation() anywhere in the program.

3.2.3. Tasks in Running a Simkit Model

There are mainly 7 tasks involved in running a Simkit model: (1) Instantiate desired objects, (2) Register SimEventListener objects, (3) Register PropertyChangeListener objects for statistics collection purposes, (4) Set stopping time or stopping criteria, (5) Set the mode of the run (verbose/quiet, single-step/continuously running), (6) Reset all SimEntityBase instances to initialize statistics collecting variables, and (7) Start the simulation

3.2.4. Listener Patterns

Simkit uses two "Listener" patterns, SimEventListener and PropertyChangeListener, to implement its component interoperability. The SimEventListener pattern is used to connect simulation components in a loosely coupled manner.

The listener design pattern is extensively used in modern software design, e.g. the graphical user interface design using Java's Swing components. The event source, the event listener and the event are the three actors of the listener design. Event listener objects register interest in an event source's events. When the source object fires the event, all registered listeners are notified and a reference to the event is passed to the listeners. The listener pattern inherently has loose coupling, thus allowing a generic design and implementation of both the listener and the source (e.g. using interfaces in Java, or an equivalent construct in other object-oriented languages). As a result, neither the source nor the listener need to be "aware" of the other in their design. Note that this is in contrast to the so-called Observer pattern, in which a callback is made from the listener to the source, a pattern that couples the two more tightly than the listener pattern.

In Simkit, a special kind of listener pattern called SimEventListener is the driving mechanism for creating simulation components. In the SimEventListener pattern, the event is in fact a SimEvent that had been previously scheduled by a SimEventSource object. When the scheduled event occurs, the scheduling object is notified by the Event List and is passed a reference to the SimEvent in question. A SimEventListener processes a heard SimEvent as if it had scheduled it itself. This is implemented in Simkit using Java's reflection. The SimEventListener pattern is shown in Figure 9. Any number of SimEventListeners may be registered for a given SimEventSource.

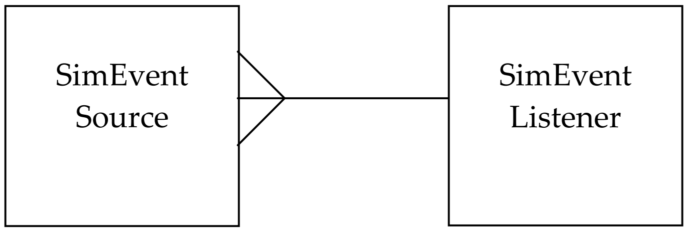


Figure 9. SimEventListener pattern.

3.2.5. PropertyChangeListener Pattern

This is used whenever a state variable changes value. In that case, a PropertyChangeEvent is dispatched to registered PropertyChangeListener objects. The purpose of PropertyChangeEvents is to support the generic observation of the simulation state trajectories, as well as any function thereof.

3.2.6. Collecting Statistics

Simkit uses the PropertyChangeListener pattern for collecting statistics from a simulation model. This pattern provides a great deal of flexibility for what gets collected, how it is collected, and which measures of performance are estimated.

This approach also enables a clean separation between implementing the dynamics of the model and gathering data. Thus, the model can be created without any concern over which statistics are to be estimated, and the model classes themselves will not contain any code involved with statistics. All a model class has to do is make sure it fires a PropertyChangeEvent whenever a state variable changes its value.

3.2.7. Generating Random Numbers and Variates

The available random number types include Antithetic, Congruential, MersenneTwister, Mother, NSSrng, PooledXORGenerator, PooledGeneratorBase, Sequential, and Tausworthe.

Random number generation for random variates, through RandomVariateFactory class, uses the MersenneTwister as the deafult random number type. Also, by default, a single instance of random number is used to give each RandomVariate its Uniform(0,1) random numbers.

Simkit's design permits much flexibility for generating random variates used in the simulation models. It enables the modeller to change any random variate in a model to any desired probability distribution without having to recompile the model.

Simkit uses a combination of a RandomVariate interface and an abstract factory that is called to produce instances of the desired implementation using only "generic" data–that is, Strings, Objects, and numbers. More detailed information on this is available in Simkit documentation.

3.2.8. Obtaining Simkit

The latest version of Simkit can be downloaded from the World Wide Web, free of charge, and with its source code at http://diana.gl.nps.navy.edu/Simkit/

3.2.9. The Reasons for Selecting Simkit and Event Graphs

There are many practical reasons for selecting Simkit. Simkit is an object-oriented, component-based API that can be used to create discrete event simulation models using the Event Graph (EG) methodology. Event graphs were first introduced by Schruben (1983). Schruben (1995), Sargent (1988) and Seila, Ceric and Tadikamalla (2003) give a more detailed discussion of event graph modeling. In the EG methodology, nodes represent events and directed arcs represent the scheduling relationships between events. A tilde in an arc represents the conditional scheduling of the event at the head of the arc whenever the stated Boolean condition is satisfied. Dashed arcs represent canceling relationships. A special event, referred to as "Run", is used to initially populate the event list as well as to initialize the state variables. In a given module, the Run event may be connected to another event via a directed arc or may appear as an isolated event. In the former case, it performs both of its functions (i.e., scheduling the initial event as well as initializing the state variables). In the latter case, the only function performed by the Run event is initialization of the state variables. EGs can be used to, simply and elegantly, represent any DES model. There are no theoretical limitations to the DES models that can be implemented in Simkit. The relationship between event graphs and Simkit can be seen in Table 2.

Event Graph	**Simkit**
State Variables	Class instance variable (protected)
Simulation Parameters	Class instance variable (private)
Events	Class "do" method
Event Parameters	Parameter of a "do" method
Event Actions	Code lines of a "do" method
Scheduling Edges	"waitDelay" call
Canceling Edges	"interrupt" call
Edge Delay Times	Time argument of "waitDelay" call
Edge Conditions	"if condition block", wrapping a "waitDelay" or "interrupt" call
Edge Arguments	"waitDelay" or "interrupt" call arguments

Table 2. Relationship between Event Graphs and Simkit (Adapted from (Jose, 2001)).

In the EG methodology, state variables are a collection of variables that do change (or have the possibility of changing) in a single simulation run. Simulation parameters are a collection of variables each of which stays fixed throughout a given simulation run. Nodes represent the (instantaneous) state transitions or events and the edges represent scheduling and cancelling relationships between events.

Event Graph methodology is sufficiently powerful by itself to represent any model that can be captured by the DES framework.

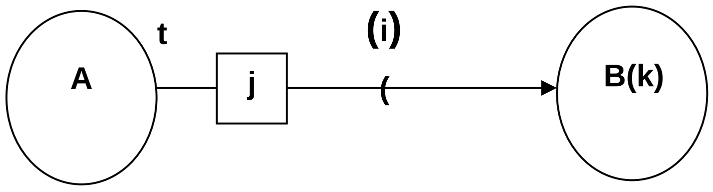


Figure 10. Event scheduling using event graphs.

Figure 10 implies the following: "When event A occurs, then if boolean condition (i) is true, then event B is placed in the future event list with a delay of time t. When event B occurs, its formal arguments, denoted by k, are passed the values in the expression denoted by j on the scheduling edge." (Buss and Ruck, 2004)

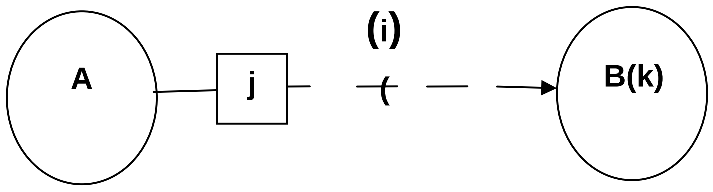


Figure 11. Event cancelling using event graphs.

Figure 11 implies the following: "When event A occurs, then the first scheduled event named B whose parameters exactly match the expression j is removed from the Future Event List. If no such event is scheduled when A occurs, then nothing happens and there is no error." (Buss and Ruck, 2004)

A more complicated event graph is as follows:

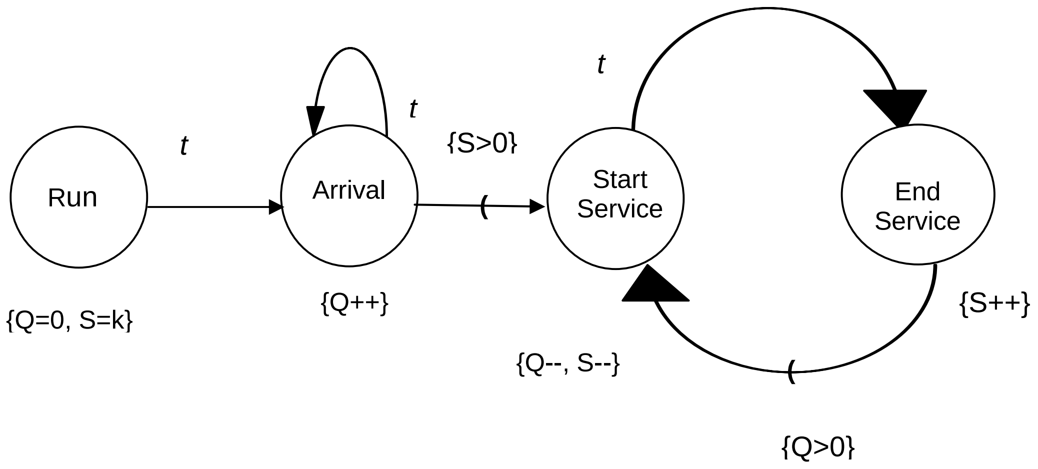


Figure 12. Event graph for multiple-server queue.

The Q and S represent the state variables of the simulation. They stand for the number in queue and the number of available servers respectively. At least one Run event is necessary in every model to initialize the state variables. State variables are updated, displayed in curly brackets, as each event is fired. Here the initial Arrival event is scheduled by the Run event. After that, each Arrival event will schedule a StartService event if the number of available servers S is greater than zero. A StartService event will decrease the number in queue, Q. EndService event will trigger a StartService event if Q is greater than zero.

The loose coupling in Simkit's component architecture facilitate a significant degree of reusability of simulation components.

3.2.10. The Listener Event Graph Objects (LEGOs)

LEGO framework (Buss and Sanchez, 2004) is used to connect components of our simulation. LEGOs are an extension to basic EGs which allow small models to be encapsulated in reusable modules. These modules can be treated as components of other modules. This modular structure is depicted by drawing a box or rectangle around the EG. Modules or components are linked using the listener pattern of Object Oriented Programming which enables production of larger and more complex modules. LEGOs register interest in other LEGOs and take appropriate actions when they "hear" state changes. The LEGO that is listened to is not affected and is not responsible for any actions taken. This connection is enabled by having events with the same name and signature in both components. This "listener" and "listened" relationship is depicted via an arc with a reversed triangle at one end, resembling a stethoscope. The object at the end of the arc with the reversed triangle is the "listened" object as depicted in Figure 9.

This loose-coupling of objects in the simulation allows a great amount of flexibility. EGs and LEGOs can be programmed using Simkit. Simkit has also been selected as the simulation engine for Combat XXI, the US Army's next-generation premier ground combat simulation. It integrates object-oriented design with a listener pattern to create a truly reusable component architecture for model components. The listener pattern implementation in Simkit is called Simulation Event Listener or "SimEventListener" pattern using Simkit's interface name (Buss and Sanchez, 2002), (Buss, 2002). Simkit, utilizing the same ideas of listener patterns, has a very loose coupling of model components. This permits a clean separation of model constructs from data gathering. Simkit's use of the "PropertyChangeListener" pattern for collecting statistics from a simulation model ensures that any Measures of Effectiveness (MOEs) can be estimated.

The listener patterns used to implement the loose coupling give the modeller a great degree of flexibility in adding new features to existing models without comprehensive changes to the source code.

Simulation models using Simkit can be built and executed on any Java2™ enabled platform.

3.3. THE GEOGRAPHICAL INFORMATION SYSTEM (GIS)

To satisfy the requirement for accurate animation based on real transportation network data (such as the capacities of roads, railways and bridges), our simulation model uses a state-of-the-art Java2™ based, licensed geographical information system (GIS) named GeoKIT, as part of the transportation simulation developed. GeoKIT is an Application Programming Interface (API) for manipulating and visualizing 2D/3D raster and vector spatial data. It is written in the Java2™ programming language and provides a comprehensive set of components to embed GIS functionality into the applications. GeoKIT is open to all types of geographical data and is independent of any particular data format. It achieves high performance mapping and precise geodetic calculations, coordinate transformations and map projections. With its object-oriented design in Java, it allows an intrinsically perfect interaction with Simkit and the EG paradigm. GeoKIT and the available GIS data allow high-resolution representation of transportation infrastructure (e.g., roads, railroads, bridge capacities, slope information etc.).

GeoKIT has been designed to support distributed architectures (client-server and 3-tier architecture) and in particular for developing stand-alone, Internet / Intranet based applications. All GeoKIT components implement a Java2™ interface, so a developer can customize all aspects of GeoKIT by replacing a component with a new implementation. All GeoKIT components are JavaBean compliant. GeoKIT is particularly well suited for distributed computing where multiple views access multiple data sources on different servers.

GeoKIT components are full Java Bean API. Their methods and events follow the Java Bean API standard. GeoKIT components can be manipulated graphically and through property editors in a Java Bean compliant Integrated Development Environment (IDE) such as JBuilder and Symantec Visual Cafe. It was developed using Sun Java SDK 1.4, Sun JFC (Swing) GUI tools, Sun Java 3D libraries, Apache xerces packages for (XML/GML parsing), Oracle JDBC drivers, Borland`s JBuilder 7.0 and Suse Linux 8.0. GeoKIT API supports both 3D and 2D data. GeoKIT has the following interfaces: HTML, SQL2, XML, JDBC.

The following data sources are available in GeoKIT: Vector Formats: ESRI Shape, MapInfo MIF/MID, MapInfo TAB, Genamap Exchange, VPF, DCW, OGC GML 2.0, INTERGRAPH DGN, AUTOCAD DXF, ENC S-57, Oracle Spatial Extension (9.2), GeoKIT Overlay Format, Nato ATCCIS Baseline 2.0 V.50 Data Model Overlay Storage, BİLGİ Vector Server, MS Excell CSV, ASCII, Genamap Exchange.

Raster Formats: CADRG, ADRG, CIB, TIFF, World TIFF (tfw), GeoTIFF, IKONOS Imagery, JPG, PNG, GIF, BMP, Bilgi Raster Server.

Cell/Matrix Data Formats: DTED Level 0,1,2, 16 Bit TIFF, USGS DEM.

Web Objects: GML Objects, GeoKIT WEB Objects. More information on GeoKIT can be found at <http://geokit.bilgigis.com>.

Our simulation model, developed using the tools briefly described above, has three main components; *Graphical User Interface (GUI)*, *network,* and *model*. The *GUI* component allows, among other things, an accurate animation using real geospatial transportation infrastructure data, point-and-click operations of route selection, adding/deleting/changing Home Bases/ Destinations/ POEs/ PODs and entities, and on-the-fly projection of the entire network from World Geodetic System (WGS) 84 to Universal Transverse Mercator (UTM) coordinates. WGS 84, developed in 1984 and updated in 2004, is an ellipsoidal reference frame for the earth for use in geodesy and navigation. UTM coordinate system was developed by NATO in 1947 and is used by most military maps in the world. The scenario information can be saved in XML format.

The *network* component allows shortest path selection in route planning. In addition, it allows the listing and selection of all routes on land and rail networks whose length (cost) is a user-specified percentage more than the shortest path's.

The model component has four components; Land, Sea, Rail, and Air. Each of these has three subcomponents. The subcomponents are connected via LEGOs and SimEventListener frameworks.

In addition, there are components, or Java classes, modeling the transportation assets (e.g., trucks, ships, airplanes) of different capacities and modes. The loads of different sizes and military hardware that need to be transported (such as tanks, generators, and field artillery guns) are also modeled. These do not exist in Simkit and have been added as extensions.

4. THE MODELING DETAILS

This chapter presents information on the event graphs (the conceptual and logical models) and listening patterns between components of the simulation. Some information on the use of animation, the graphical user interface (GUI), and additional Java classes is also included.

4.1. COMPONENTS OF THE DEPLOYMENT SIMULATION

There are four main components or modules of the deployment simulation: Land, Sea, Air, and Rail. These are described in more detail in Sections 4.2 through 4.5 The listener patterns that provide communication between the components are explained in Section 4.6. Details of the implementation of animation and GUI are included in Section 4.7. Explanations of some of the important Java classes used in the simulation are provided in Section 4.8.

4.2. THE LAND COMPONENT

The Land Component consists of three classes. These are named HomeBaseDeparture, LandMaintDelay, and LandArrivalAtDestination.

4.2.1. HomeBaseDeparture Class

Home base departure class simulates the loading of vehicles at their home bases, forming of convoys, and their departure. The movements are conducted in convoys either to the destination or the next mode change location (an intermediate location) which can be a SPOE, an APOE, or a RPOE (a rail port of embarkation). Vehicles that do not need to make additional trips simply park at their home bases.

The simplified event graph for this class is in Figure 13. Note that the arrival events at SPOE, RPOE, and APOE are omitted from this event graph to keep it simple for presentation purposes. The Run event in Figure 13 is used to schedule the first events (initial population of the event list) such as assigning movers (trucks, ships, trains etc.) to their respective initial locations and also to specify earliest times of departures for the movers from their home bases. All arcs in Figure 13 except two of them are marked by a tilde indicating that the events at the head of the marked arcs are scheduled conditionally. For example, the arc connecting "ArriveAtHomeBase" to "StartLoadingVehicle" schedules the event "StartLoadingVehicle" in the event list only if the Boolean condition "number of load docks available > 0" is satisfied. The Boolean conditions are omitted from the figure for the sake of simplicity. We also omit the parameters and state variables from the figure for the same reason. However, the sample lists of parameters and state variables for the entire Land Component are provided at the end of Section 4.2 in a section that presents Exogenous (Input) and Endogenous (Output) Variables for the Land Component. The lists of parameters and state variables are expandable depending on the statistics and measures of performance that are desired to be calculated. The parameters and state variables are also presented for a specific scenario in the case study in Section 6.

The sample Java code for the home base departure class is provided in Appendix A. The complete code for all classes is not provided here due to space limitations and the length of the code. The simulation has a total of *97 Java classes (modules)* and *over 16000 lines of code*. The requests for obtaining the entire code for research and other purposes may be considered on a case by case basis by the author and the thesis advisors of this dissertation.

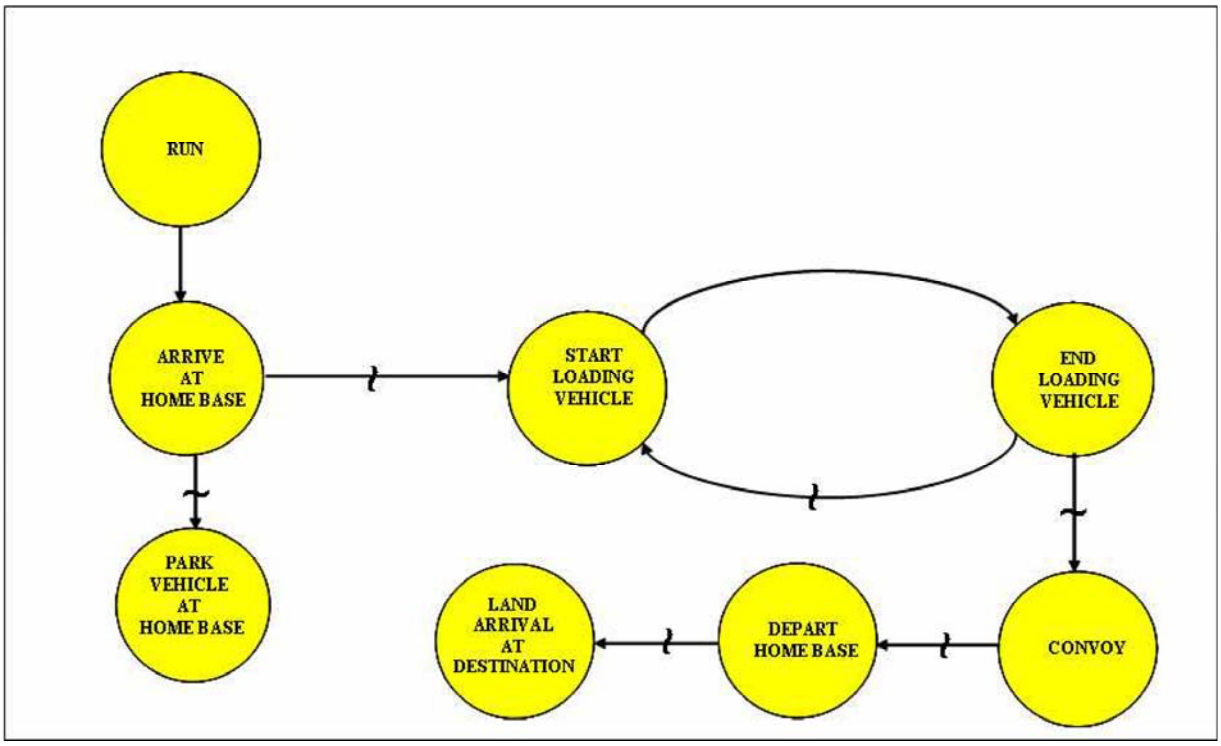


Figure 13. Simplified Event Graph for the Land Component's HomeBaseDeparture class.

In addition to the simplified version of the event graph above, we provide in Section 4.2.1.1 below a more complicated version only for the home base departure module. This is to help clarify some of the details of the event graphs (modules) used in the simulation. The details of the other event graphs are omitted in the following sections for the sake of brevity.

4.2.1.1. A More Detailed Event Graph for the HomeBaseDeparture Class

A more detailed version of the event graph for HomeBaseDeparture class is provided below. The sample Java code for this class is in Appendix A.

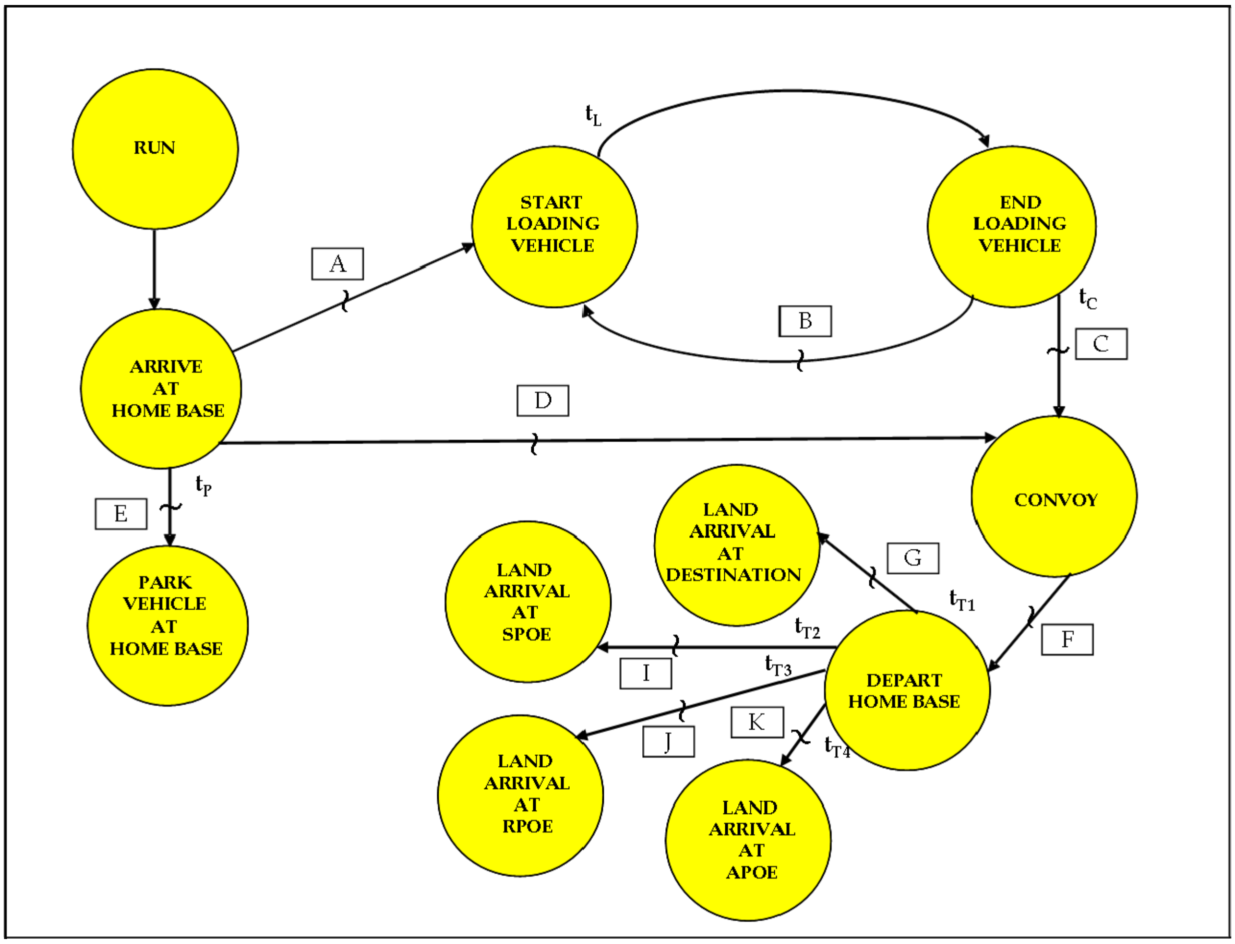


Figure 14. A More Detailed Event Graph for the HomeBaseDeparture class.

<u>*Events:*</u>

Run: Initializes the state variables and also used for intial population of the event list.

Arrive at Home Base: Arrival of vehicles at a home base.

Start Loading Vehicle: Start loading vehicles that have a carrying capacity for pax or cargo.

End Loading Vehicle: Ends loading a vehicle after a loading time t_L.

Convoy: Forms a convoy of vehicles that are finished loading or that will depart without loading.

Depart Home Base: Departure of vehicles that have reached a user specified convoy size.

Land Arrival at Destination: Vehicles arrive at their destination after a time of t_{T1} which may include travel time, time spent due to breakdowns and delays at blackspots.

Land Arrival at SPOE: Vehicles arrive at a SPOE after a time of t_{T2}.

Land Arrival at RPOE: Vehicles arrive at a RPOE after a time of t_{T3}.

Land Arrival at APOE: Vehicles arrive at an APOE after a time of t_{T4}.

Park Vehicles at Home Base: Vehicles park at home base if there is no need for an additional trip.

<u>*Boolean Conditions:*</u>

There are four possibilities for A.

A: (mover type is a "Bus" & there are pax) or (mover type is a carrier, e.g. a truck, & there are loads & number of available load docks is greater than zero) or (mover type is a "Bus" & there are pax & this is not a first trip) or (mover type is a carrier, e.g. a truck, & there are loads & number of available load docks is greater than zero & this is not a first trip).

B: (if Bus queue is not empty and if there are available pax & there is an available loading dock) or (if loading queue is not empty and if there are available loads & there is an available loading dock).

C: (if loading queue is empty & no more available loads) or (if bus queue is empty & no more available pax).

D: (mover type is not a carrier & mover type is not a "Bus" & this is a first trip)

There are four possibilities for E.

E: (mover type is a "Bus" & there are no pax) or (mover type is a carrier, e.g. a truck, & there are no loads) or (mover type is a "Bus" & there are no pax & this is not a first trip) or (mover type is a carrier, e.g. a truck, & there are no loads & this is not a first trip).

F: (Depart when convoy size reaches a user-specified size).

G: (Arrival at destination after a travel time of t_T = distance/convoy speed + delay due to breakdown and maintenance + delays due to Blackspots (if any)).

I: (Arrival at SPOE after a travel time of t_T = distance/convoy speed + delay due to breakdown and maintenance + delays due to Blackspots (if any)).

J: (Arrival at RPOE after a travel time of t_T = distance/convoy speed + delay due to breakdowns and maintenance + delays due to Blackspots (if any)).

K: (Arrival at APOE after a travel time of t_T = distance/convoy speed + delay due to breakdowns and maintenance + delays due to Blackspots (if any)).

State Variables:

State variables are the protected variables provided in Appendix A. They are not shown on the event graph in Figure 14.

Parameters:

Parameters are the instance (private) variables provided in Appendix A and they are not shown on the event graph.

4.2.2. LandMaintDelay Class

Land Maintenance Delay class simulates possible causes of delay that may take place during the land movement of a convoy from its home base to its destination. In addition, the delays that may occur during land movement from the transfer points SPOE, APOE or RPOE, (SPOD, APOD, or RPOD) to the home base (destination) are also taken into account in the same class. The simplified event graph for this class is provided in Figure 15.

The probabilities of breakdown are different for each type of land vehicle and they are obtained from the Turkish Army Logistics Directorate. The most probable locations of road traffic accidents are determined according to the historical data of so-called "black spots" on Turkey's roads. The data is obtained from <http://www.kgm.gov.tr/asps/trafik/karanokta.htm> and incorporated into the GIS. The probabilities of traffic accidents for these locations are obtained from a report prepared by the Turkish General Directorate of Highways for the years 1997-2002.

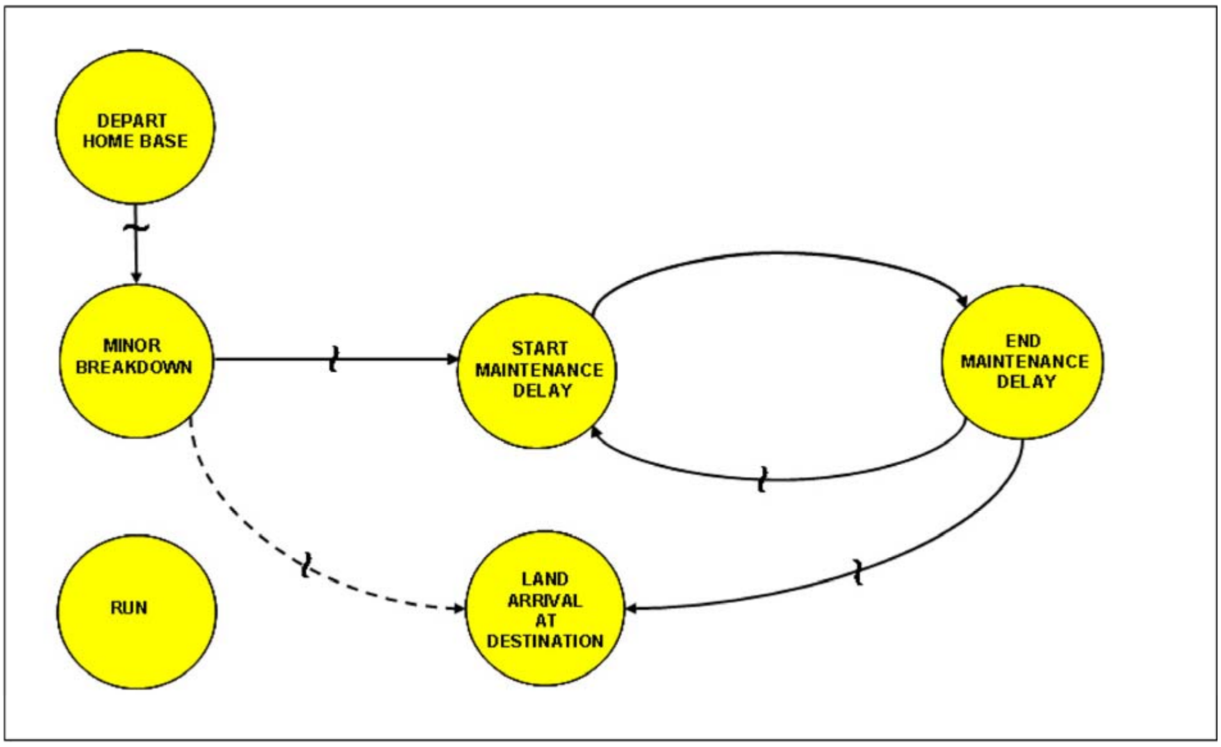


Figure 15. Simplified Event Graph for the Land Component's LandMaintDelay class.

The simplified event graph in Figure 15 states that if a minor breakdown occurs during travel from a home base to a destination, then the arrival event to destination is canceled (the dashed arc) and the arrival at destination is rescheduled after a delay time of maintenance obtained from a probability distribution. In a similar way, there are also possibilities of additional types of breakdown (medium and severe) and accidents when a convoy is on its way from its home base to its destination, or when it is on its way from an intermediate location such as SPOE, APOE or RPOE (SPOD, APOD, or RPOD) to its home base (destination). Certainly, there is the possibility of not having any problems while en route, and it is also incorporated into our model.

Anytime land transportation assets are used, breakdowns and road traffic accidents are possible to occur. However, the events of LandArrivalAtSPOE(RPOE, APOE), LandArrivalAtSPOD(RPOD, APOD), DepartSPOE(RPOE, APOE), DepartSPOD(RPOD, APOD), DepartDestination, ArriveAtHomeBase, Medium(Major) Breakdown, RoadAccident and the canceling arcs for LandArrivalAtSPOE(RPOE,APOE), LandArrivalAtSPOD (RPOD,APOD) and ArriveAtHomeBase are omitted from Figure 15 to keep the presentation simple.

4.2.3. LandArrivalDestination Class

Land arrival destination class simulates the arrivals of convoys at their designated destination locations, unloading of the vehicles at destinations and forming of convoys and their departures to bring the remaining items of the unit (from their designated home bases, SPODs, RPODs or APODs) if additional trips are required. Otherwise, the vehicles park and stay at their destination locations until a next order for movement is given. The vehicles will also park and stay there upon reaching the destination if they are not carriers (e.g. tanks). The event graph for the land arrival destination subcomponent is provided in Figure 16. The VehicleArrivalAtSPOD(RPOD, APOD) events scheduled by DepartDestination event are not included in Figure 16 to keep the presentation simple.

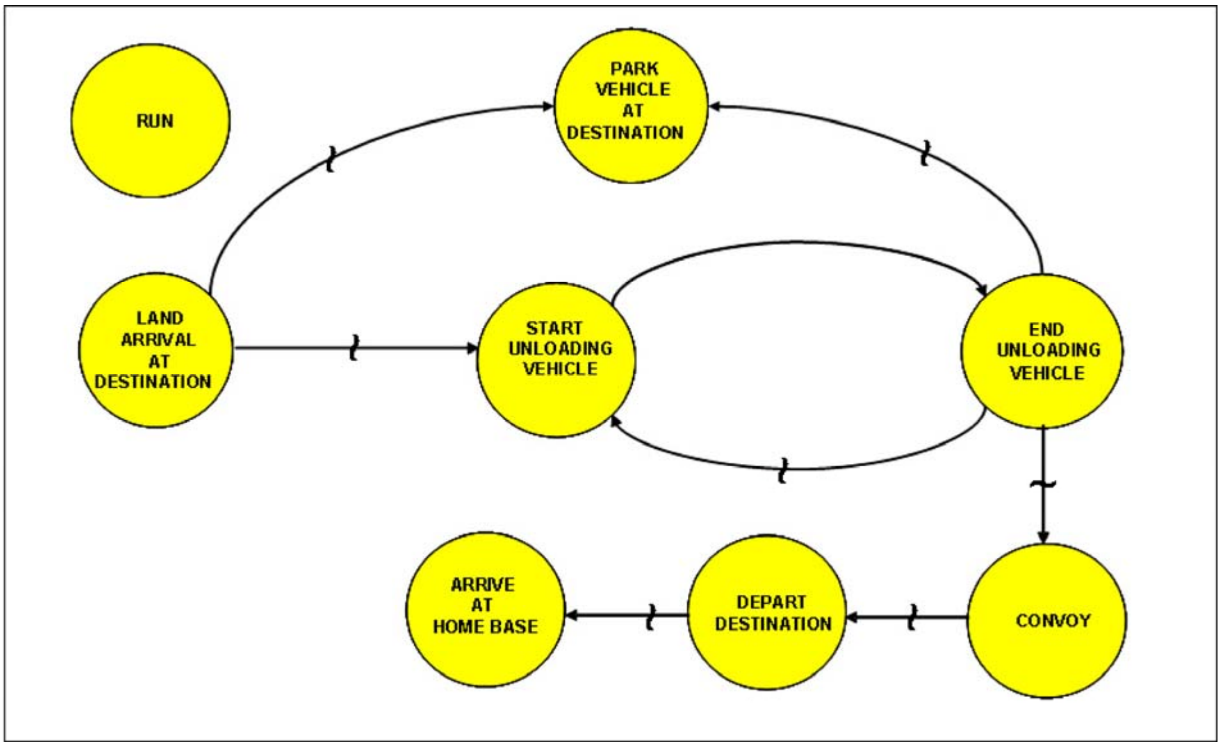


Figure 16. Simplified Event Graph for the LandArrivalAtDestination class.

4.2.4. Exogenous (Input) and Endogenous (Output) Variables for the Land Component

The events and activities for the land component are as shown in the event graphs above. The entities for the land component are trucks of different sizes and capacities such as a 5-ton truck, a 2.5-ton truck, and a tank/armored vehicle-carrying truck depending on the vehicles required/used for deployment. This section provides a sample list of possible exogenous (input) and endogenous (output) variables for the land component. This list is expandable depending on the statistics and measures of performance that are desired to be calculated.

4.2.5.1. Exogenous Variables (Input Variables) for the Land Component

The exogenous (input) variables include decision variables (controllable variables) and parameters (uncontrollable variables) of the simulation model.

4.2.5.1.1. Decision Variables (Controllable Variables)

- The number of loading docks at each home base.
- The speeds of Land Component vehicles.
- The speeds of the convoys.
- The number of vehicles in a convoy.
- The locations of "black spots" for road accidents.
- The number of unload docks at each destination.

4.2.5.1.2. Parameters (Uncontrollable Variables)

- The loading times for each type of vehicle at their home bases.
- The unloading times for each type of vehicle at their destinations.
- The maintenance times for each type of vehicle for a minor breakdown.
- The maintenance times for each type of vehicle for a medium breakdown.
- The maintenance times for each type of vehicle for a severe breakdown.
- The probability of a minor breakdown for each type of vehicle.
- The probability of a medium breakdown for each type of vehicle.
- The probability of a severe breakdown for each type of vehicle.

4.2.5.2. Endogenous Variables (Output Variables) for the Land Component

The endogenous (output) variables include state variables and performance measures.

4.2.5.2.1. State Variables

- The number of different types of vehicles in loading dock queues at home bases.
- The number of different types of vehicles in unloading docks at their destinations.
- The state of loading docks at home bases (busy vs. idle).
- The state of unloading docks at destinations (busy vs. idle).

4.2.5.2.2. Performance Measures

- Average loading time for each type of vehicle at the loading docks at home bases.
- Average waiting time for each type of vehicle in loading dock queues at home bases.
- The utilization of loading docks at home bases.
- Average maintenance time for minor breakdown for each type of vehicle.
- Average maintenance time for medium breakdown for each type of vehicle.
- Average maintenance time for severe breakdown for each type of vehicle.
- Average delay time for road accidents.

- The number of (minor/medium/severe) breakdowns for different types of vehicles.
- The number of road accidents.
- Average unloading time for each type of vehicle at the unloading docks at destinations.
- Average waiting time for each type of vehicle in unloading dock queues at destinations.
- The utilization of unloading docks at destinations.

4.3. THE SEA COMPONENT

The sea component consists of three classes. These are named LandArrivalAtSPOE, SeaMaintDelay, and SeaArrivalAtSPOD.

4.3.1. LandArrivalAtSPOE Class

The land arrival at SPOE class simulates the arrival of land convoy to its intermediate point, SPOE, for loading into a sea vessel. The vehicles of the convoy along with their loads can be directly loaded into the sea vessel if it is of RoRo type (Roll-on Roll-off, i.e., ferries designed to carry wheeled cargo such as cars, trucks etc.). Otherwise, the items have to be unloaded from the vehicles and then loaded into the sea vessel. If the ship type is not of RoRo and thus the vehicles do not go with the ship, then the unloaded vehicles may be loaded (if any loadable items are at the port) and depart for their next destination. They may also depart the SPOE empty and go to their next destination or original location (home base) if a return trip is required. These events are described in the event graph in Figure 17.

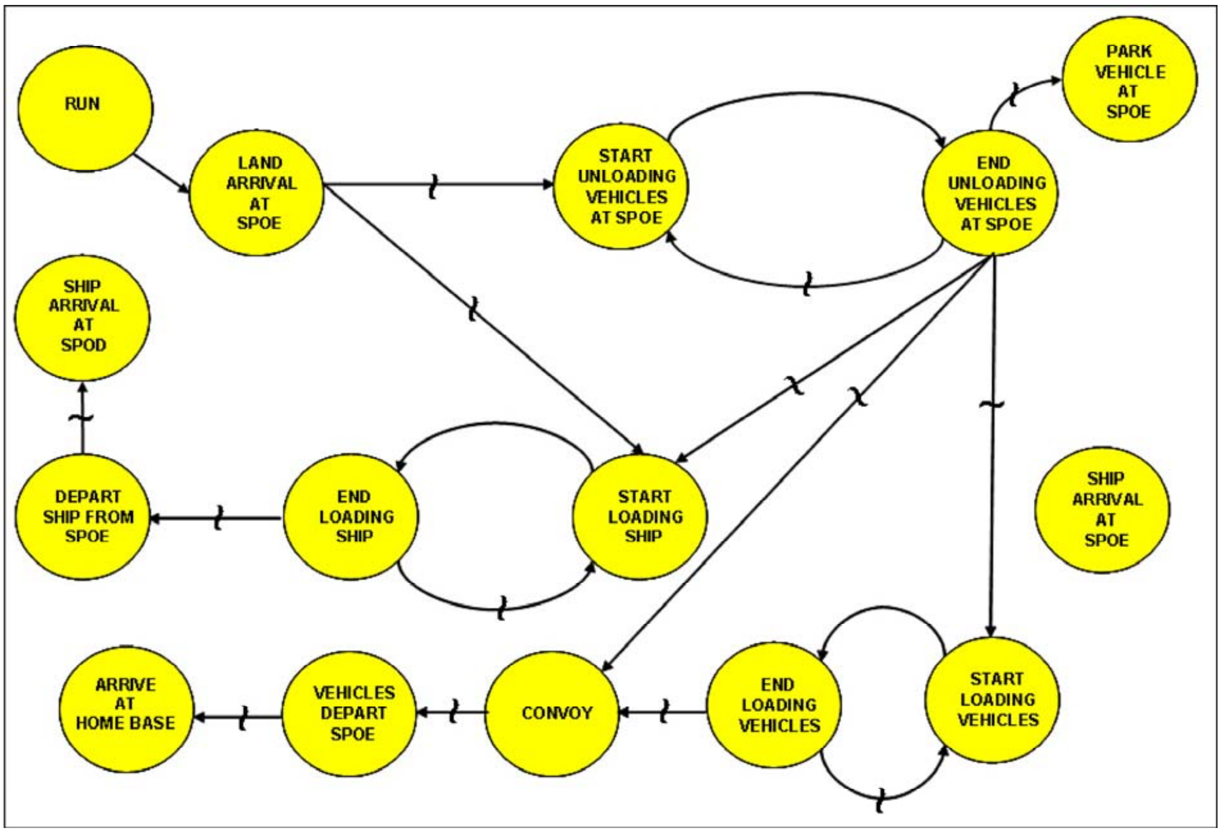


Figure 17. Simplified Event Graph for the Sea Component's LandArrivalAtSPOE class.

4.3.2. SeaMaintDelay Class

Sea maintenance delay class simulates possible different causes of delay that may take place during the movement of a sea vessel from its SPOE to SPOD and vice versa. The information on different causes of a delay and maintenance times for a sea vessel were obtained through interviews with experts at the Turkish Navy Command Headquarters.

Propeller breakdowns and steer breakdowns are more likely to occur but they can be fixed by the personnel aboard the ship. On the other hand, gyro breakdowns, main machine breakdowns, and radar breakdowns are less likely to occur but they cannot be repaired by the personnel aboard. These types of breakdowns would require the towing of the ship to a nearest port where repair facilities exist. The event graph for this class is in Figure 18.

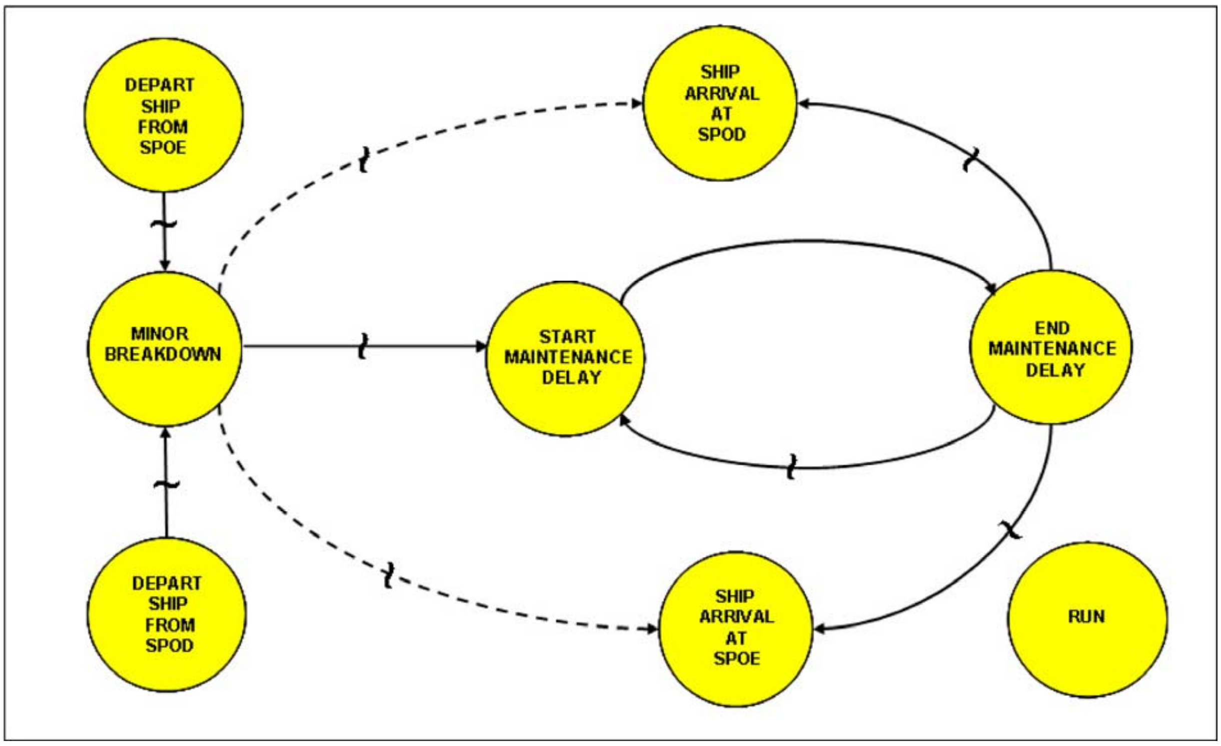


Figure 18. Simplified Event Graph for the Sea Component's SeaMaintDelay class.

Certainly, there is the possibility of departing the SPOE (SPOD) and arriving at the SPOD (SPOE) without any problems while enroute.

Note that only the minor breakdown case is depicted in Figure 18 to keep the presentation simple. The simplified event graph in Figure 18 states that if a minor breakdown occurs during travel from a SPOE (SPOD) to a SPOD (SPOE), then the arrival event to SPOD (SPOE) is cancelled (the dashed arc) and after a delay time of maintenance obtained from a probability distribution, the arrival at SPOD (SPOE) is rescheduled.

4.3.3. SeaArrivalAtSPOD Class

Sea arrival at SPOD class simulates the arrival of a ship to its SPOD and unloading of the ship. If the ship is of RoRo type, then the vehicles that debark may immediately form convoys and depart the SPOD for their destination. If the ship is not of RoRo type, then the vehicles present at the port for transporting the unloaded items and personnel to their destination have to be loaded onto the vehicles, form up convoys and depart for their destination.

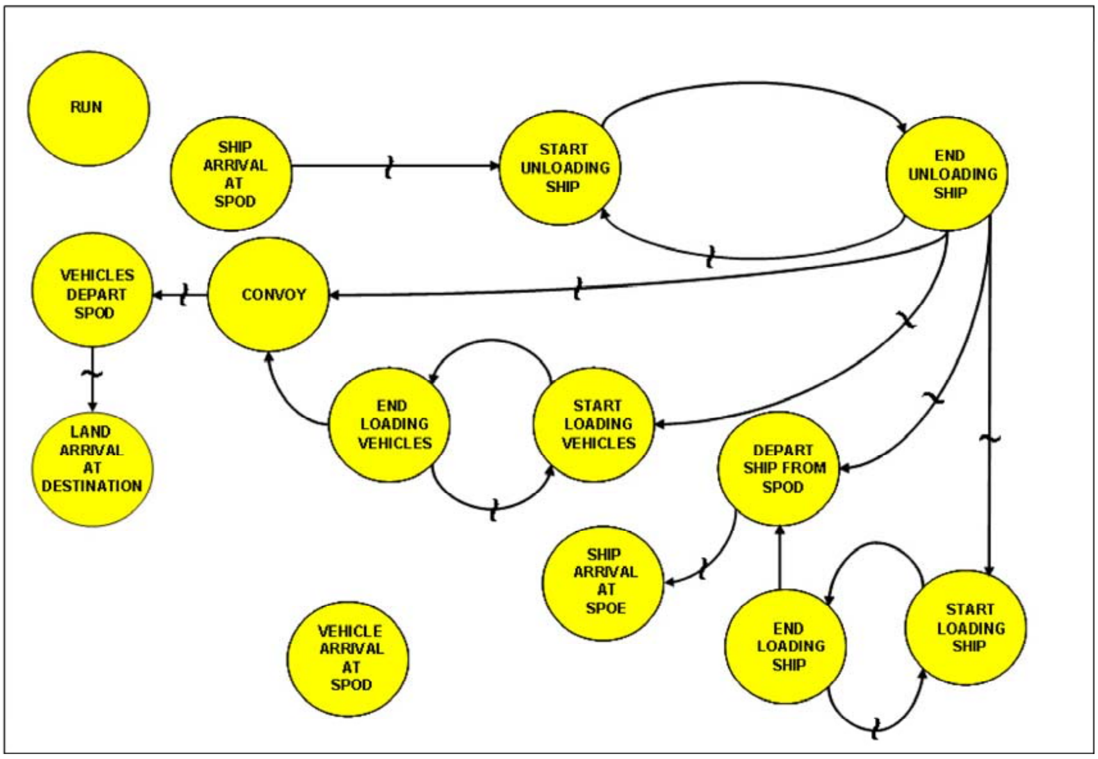


Figure 19. Simplified Event Graph for the Sea Component's ArrivalAtSPOD class.

The ship may immediately leave after discharging or sail after loading any available items/personnel to its SPOE if a return trip is required or if the ship is ordered to return. The simplified event graph for this class is in Figure 19.

4.3.4. Exogenous (Input) and Endogenous (Output) Variables for the Sea Component

The events and activities for the sea component are as shown in the event graphs above. The entities for the sea component are sea vessels of different sizes and capacities such as a RoRo ship or a ship that only carries pax. This section provides a sample list of possible exogenous (input) and endogenous (output) variables for the sea component. This list is expandable depending on the statistics and measures of performance that are desired to be calculated.

4.3.5.1. Exogenous Variables (Input Variables) for the Sea Component

The exogenous (input) variables include decision variables (controllable variables) and parameters (uncontrollable variables) of the simulation model.

4.3.5.1.1. Decision Variables (Controllable Variables)

- The number of loading docks at each SPOE.
- The speeds of sea vessels.
- The number of unloading docks at each SPOD.

4.3.5.1.2. Parameters (Uncontrollable Variables)

- The loading times for each type of sea vessel.
- The unloading times for each type of sea vessel.
- The maintenance times for each type of sea vessel for a steer breakdown.
- The maintenance times for each type of sea vessel for a propeller breakdown.
- The maintenance times for each type of sea vessel for gyro/radar/main machine breakdowns.
- The probability of a steer breakdown for each type of sea vessel.
- The probability of a propeller breakdown for each type of sea vessel.
- The probability of gyro/radar/main machine breakdowns for each type of sea vessel.

4.3.5.2. Endogenous Variables (Output Variables) for the Sea Component

The endogenous (output) variables include state variables and performance measures.

4.3.5.2.1. State Variables

- The number of different types of sea vessels in loading dock queues at each SPOE.
- The number of different types of sea vessels in unloading dock queues at each SPOD.
- The state of sea vessels at SPOEs (busy vs. idle).
- The state of sea vessels at SPODs (busy vs. idle).

4.3.5.2.2. Performance Measures

- Average loading time for each type of sea vessel at the loading docks at SPOEs.
- Average waiting time for each type of sea vessel in loading dock queues at SPOEs.
- The utilization of sea vessels.
- Average maintenance times for a steer breakdown for each type of sea vessel.
- Average maintenance times for a propeller breakdown for each type of sea vessel.
- Average maintenance times for gyro/radar/main machine breakdowns for each type of sea vessel.

- Average delay time for mine encounters.
- The number of (steer/propeller/radar/gyro/main machine) breakdowns for different types of sea vessels.
- Average unloading time for each type of sea vessel at the unloading docks at SPODs.
- Average waiting time for each type of sea vessel in unloading dock queues at SPODs.
- The utilization of loading docks at SPOEs.
- The utilization of unloading docks at SPODs.

4.4. THE RAIL COMPONENT

The rail component consists of three classes. These are named LandArrivalAtRPOE, RailMaintDelay, and RailArrivalAtRPOD.

4.4.1. LandArrivalAtRPOE Class

The land arrival at RPOE class simulates the arrival of a land convoy to its intermediate point, RPOE, for loading into a train. The vehicles of the convoy along with their loads can be directly loaded onto the train. Otherwise, the items have to be unloaded from the vehicles and then loaded into the train.

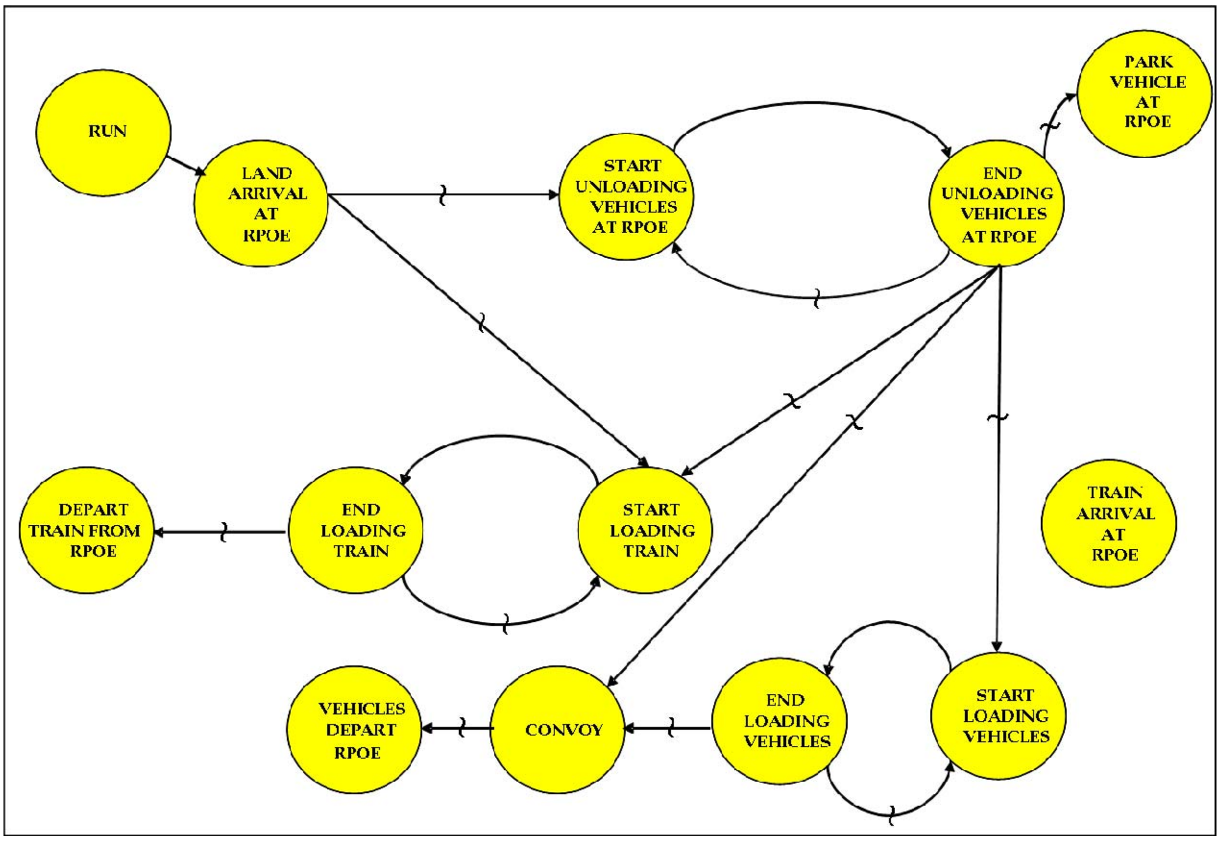


Figure 20. Simplified Event Graph for the Rail Component's LandArrivalAtRPOE class.

If the vehicles are not loaded onto the train and thus the vehicles do not go with the train, then the unloaded vehicles may be loaded (if any loadable items are at the train station) and depart for their next destination. They may also depart the RPOE empty and go to their next destination or original location (home base) if a return trip is required. These events are described in the event graph in Figure 20.

4.4.2. RailMaintDelay Class

Rail maintenance delay class simulates possible different causes of delay that may take place during the movement of a train from its RPOE to RPOD and vice versa.

The possible causes of maintenance delay are minor, medium, and severe breakdowns in rail transportation assets. The probabilities of breakdown are different for each certain type of train. These different types of delay are possible when a train is on its way from a RPOE (RPOD) to its RPOD (RPOE). Certainly, there is the possibility of not having any delays while enroute. The event graph for this class is in Figure 21.

Note that only the minor breakdown case is depicted in Figure 21 to keep the presentation simple. The simplified event graph in Figure 21 states that if a minor breakdown occurs during travel from a RPOE (RPOD) to a RPOD (RPOE), then the arrival event to RPOD (RPOE) is canceled (the dashed arc) and after a delay time of maintenance obtained from a probability distribution, the arrival at RPOD (RPOE) is rescheduled.

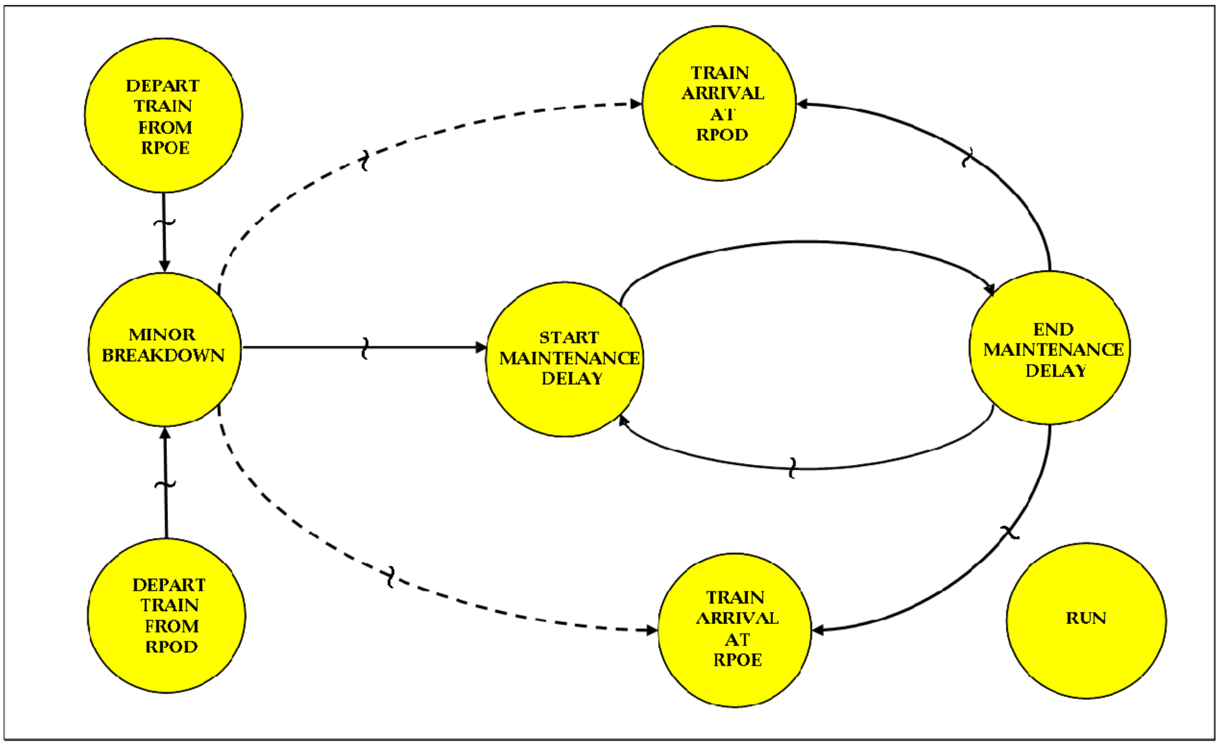


Figure 21. Simplified Event Graph for the Rail Component's RailMaintDelay class.

These different types of delay are possible when a train is on its way from a RPOE (RPOD) to its RPOD (RPOE).

4.4.3. RailArrivalAtRPOD Class

Rail arrival at RPOD class simulates the arrival of a train to its RPOD and its unloading. If the vehicles were loaded onto the train with their loads, then the vehicles that debark may immediately form convoys and depart the RPOD for their destination. If only the items and personnel were loaded at the RPOE, then the vehicles present at the RPOD for transporting the unloaded items and personnel to their destination have to be loaded. Then the vehicles form up convoys and depart for their destination.

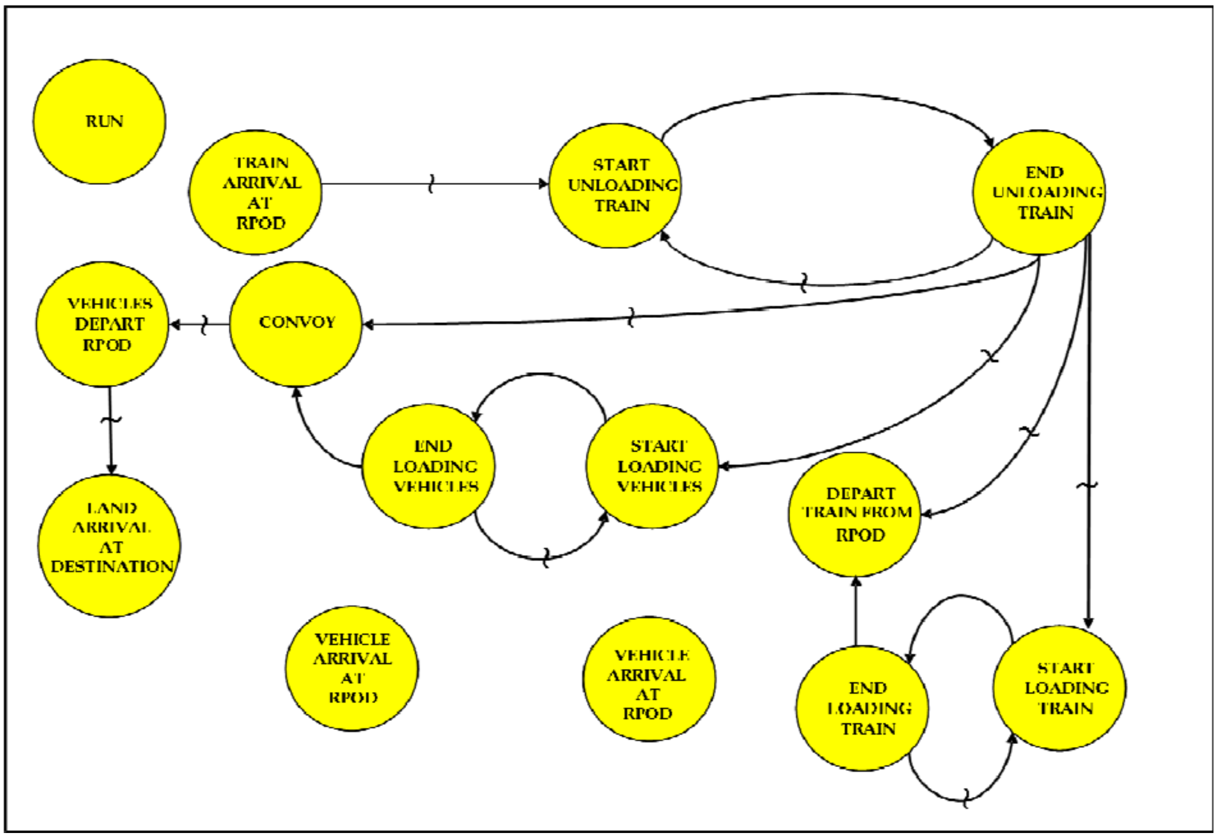


Figure 22. Simplified Event Graph for the Rail Component's ArrivalAtRPOD class.

The unloaded train may immediately leave after discharging or leave after loading any available items/personnel to its RPOE if a return trip is required or if the train is ordered to return. The event graph for this class is in Figure 22.

4.4.5. Exogenous (Input) and Endogenous (Output) Variables for the Rail Component

The events and activities for the rail component are as shown in the event graphs above. The entities for the rail component are trains of different sizes and capacities. This section provides a sample list of possible exogenous (input) and endogenous (output) variables for the sea component. This list is expandable depending on the statistics and measures of performance that are desired to be calculated.

4.4.5.1. Exogenous Variables (Input Variables) for the Rail Component

The exogenous (input) variables include decision variables (controllable variables) and parameters (uncontrollable variables) of the simulation model.

4.4.5.1.1. Decision Variables (Controllable Variables)

- The number of loading docks at each RPOE.
- The speeds of trains.
- The number of railroad cars in a train.
- The number of unload docks at each RPOD.

4.4.5.1.2. Parameters (Uncontrollable Variables)

- The loading times for each type of train.
- The unloading times for each type of train.
- The maintenance times for each type of train for a minor breakdown.
- The maintenance times for each type of train for a medium breakdown.
- The maintenance times for each type of train for a severe breakdown.
- The probability of a minor breakdown for each type of train.
- The probability of a medium breakdown for each type of train.
- The probability of a severe breakdown for each type of train.

4.4.5.2. Endogenous Variables (Output Variables) for the Rail Component

The endogenous (output) variables include state variables and performance measures.

4.4.5.2.1. State Variables

- The number of trains in loading dock queues at RPOEs.
- The number of trains in unloading docks queues at RPODs.
- The state of loading docks at RPOEs (busy vs. idle).
- The state of unloading docks RPODs (busy vs. idle).

4.4.5.2.2. Performance Measures

- Average loading time for each train at RPOEs.
- Average waiting time for each train in loading dock queues at RPOEs.
- The utilization of loading docks at RPOEs.
- Average maintenance time for minor breakdown for each type of train.
- Average maintenance time for medium breakdown for each type of train.
- Average maintenance time for severe breakdown for each type of train.
- The number of (minor/medium/severe) breakdowns for different types of trains.
- Average unloading time for each type of train at the unloading docks at RPODs.
- Average waiting time for each type of train in unloading dock queues at RPODs.
- The utilization of unloading docks at RPODs.
- The utilization of trains.

4.5. THE AIR COMPONENT

The air component consists of three classes. These are named LandArrivalAtAPOE, AirDelay and AirArrivalAtAPOD.

4.5.1. LandArrivalAtAPOE Class

The land arrival at APOE class simulates the arrival of land convoy to its transfer point, APOE, for loading into an airplane. The vehicles of the convoy along with their loads can be directly loaded into airplane if it is of a certain type (C-5 or Antonov). Otherwise, the items have to be unloaded from the vehicles and then loaded into the airplane. If the airplane type is not suitable to load the vehicles, thus the vehicles do not go with the airplane, then the unloaded vehicles may be loaded (if any loadable items are at the airport) and depart for their next destination. They may also depart the APOE empty and go to their next destination or original location (home base) if a return trip is required. These events are described in the event graph in Figure 23.

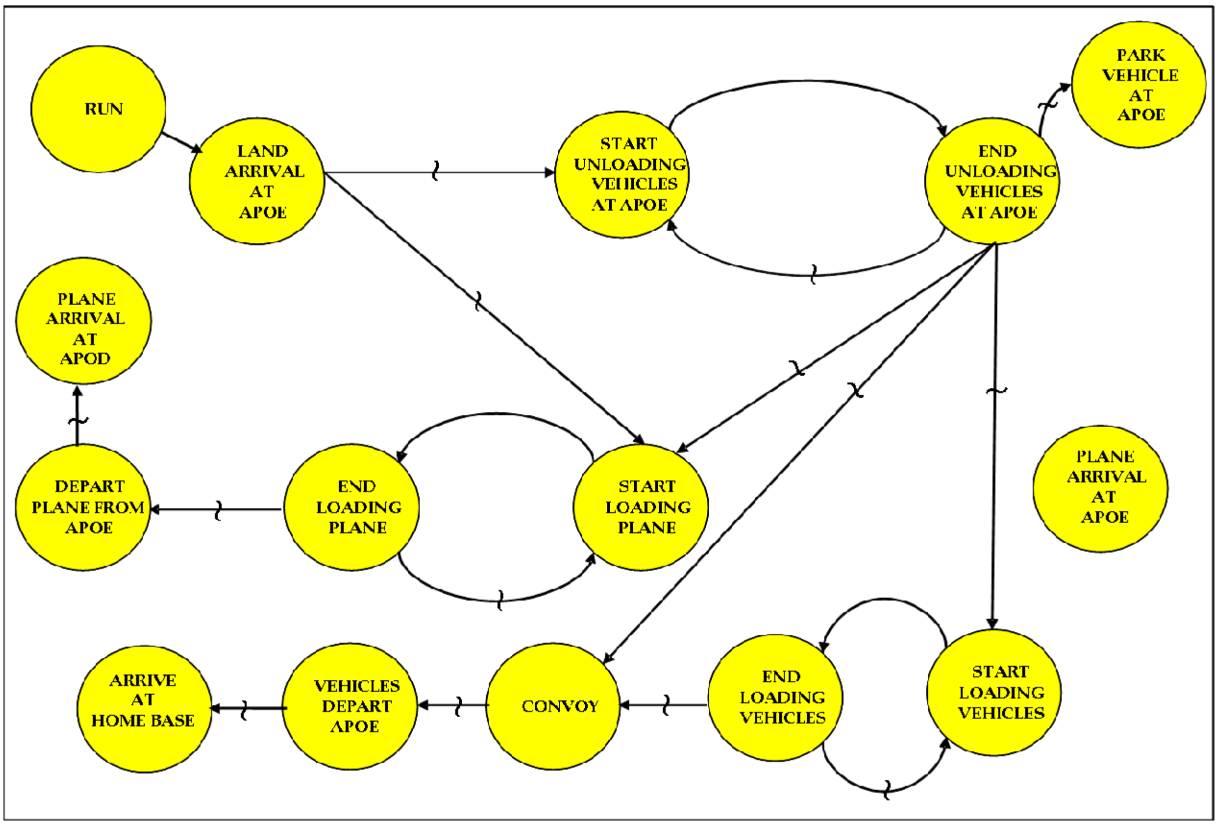


Figure 23. Simplified Event Graph for the Air Component's LandArrivalAtAPOE class.

4.5.2. AirDelay Class

Air delay class simulates possible random delay of circling in air upon arrival at destination airport due to air traffic congestion. Delays can also occur due to nonfatal breakdowns that do not cause a crash or by foreign object damage (usually by birds during takeoff). These could require the airplane to land at the nearest or the original airport where the plane took off. A fatal breakdown of the aircraft would cause a crash and require a search and rescue effort to be started. Delays may also be due to inclement weather conditions. These are not simulated at this point and left as future work. The simplified event graph for this class is in Figure 24.

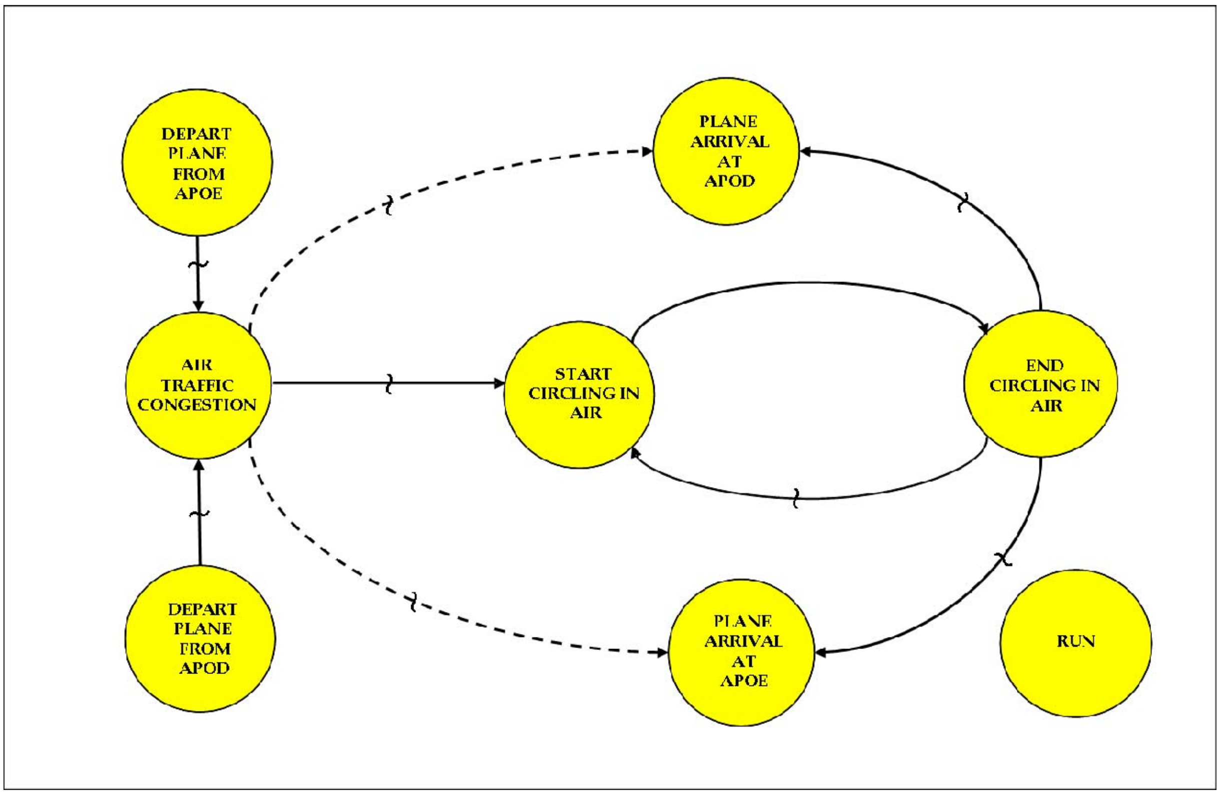


Figure 24. Simplified Event Graph for the Air Component's AirDelay class.

4.5.3. AirArrivalAtAPOD Class

Air arrival at APOD class simulates the arrival of an airplane to its APOD and unloading of the airplane. If the airplane is of a certain type (i.e., C-5 or Antonov) then the vehicles that debark may immediately form convoys and depart the APOD for their destination. If the airplane is not of a certain type, then the vehicles present at the airport for transporting the unloaded items and personnel to their destination have to be loaded onto the vehicles, form up convoys and depart for their destination. The airplane may immediately leave after discharging or take off for its APOE after loading any available items/personnel if a return trip is required or if the airplane is ordered to return. The simplified event graph for this class is in Figure 25.

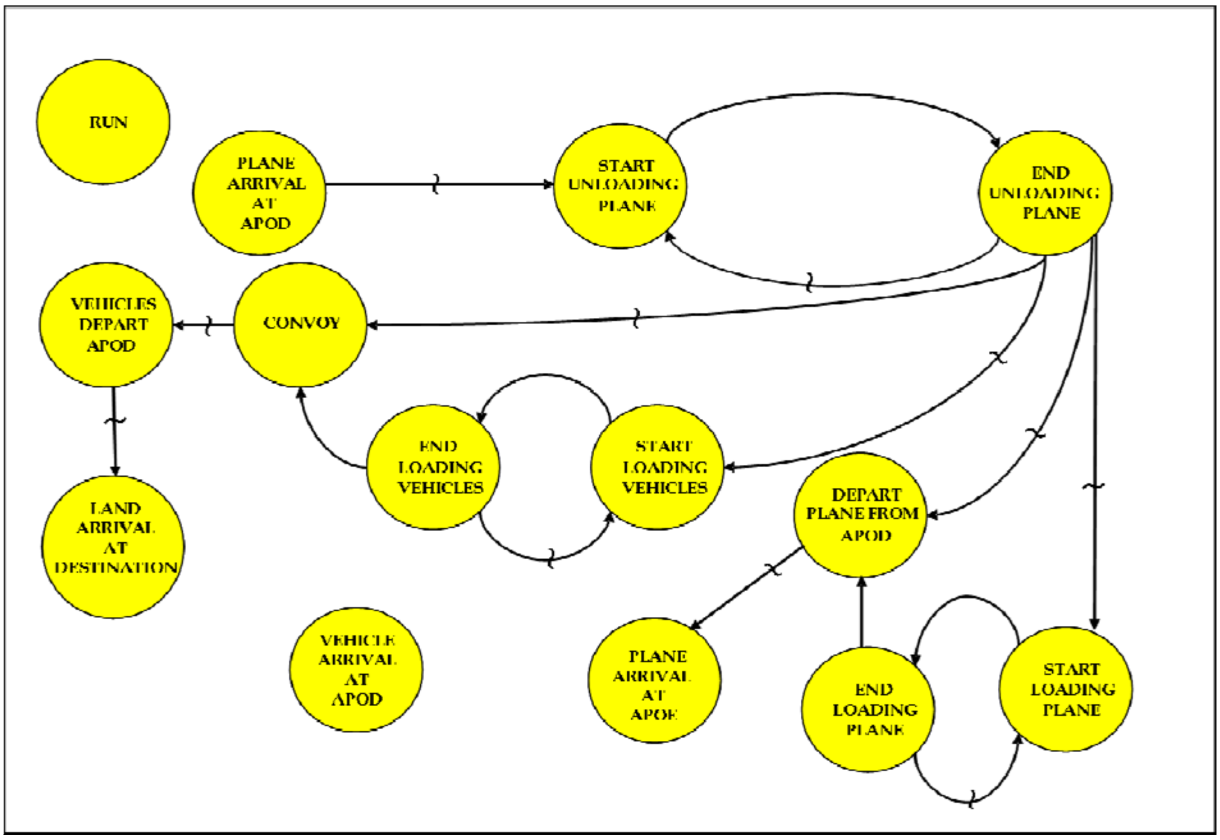


Figure 25. Simplified Event Graph for the Air Component's AirArrivalAtAPOD class.

4.5.4. Exogenous (Input) and Endogenous (Output) Variables for the Air Component

The events and activities for the air component are as shown in the event graphs above. The entities for the air component are aircraft of different sizes and capacities such as a C-5, Antonov, or C-130. This section provides a sample list of possible exogenous (input) and endogenous (output) variables for the air component. This list is expandable depending on the statistics and measures of performance that are desired to be calculated.

4.5.4.1. Exogenous Variables (Input Variables) for the Air Component

The exogenous (input) variables include decision variables (controllable variables) and parameters (uncontrollable variables) of the simulation model.

4.5.4.1.1. Decision Variables (Controllable Variables)

- The number of loading docks at each APOE.
- The speeds of aircraft.
- The number of unloading docks at each APOD.

4.5.4.1.2. Parameters (Uncontrollable Variables)

- The loading times for each type of aircraft.
- The unloading times for each type of aircraft.
- The delay times for circling in air due to traffic congestion at destination airport.
- The probability of a traffic congestion at destination airport.

4.5.4.2. Endogenous Variables (Output Variables) for the Air Component

The endogenous (output) variables include state variables and performance measures.

4.5.4.2.1. State Variables

- The number of different types of aircraft in loading dock queues at each APOE.

- The number of different types of aircraft in unloading dock queues at each APOD.
- The state of aircraft at APOEs (busy vs. idle).
- The state of aircraft at APODs (busy vs. idle).

4.5.4.2.2. Performance Measures

- Average loading time for each type of aircraft at the loading docks at APOEs.
- Average waiting time for each type of aircraft in loading dock queues at APOEs.
- The utilization of aircraft.
- Average delay in air due to traffic congestion.
- Average unloading time for each type of aircraft at the unloading docks at APODs.
- Average waiting time for each type of aircraft in unloading dock queues at APODs.
- The utilization of loading docks at APOEs.
- The utilization of unloading docks at APODs.

4.6. THE SIMULATION EVENT LISTENER PATTERNS

Listener Event Graph Object (LEGO) framework (Buss and Sanchez, 2004) is used to connect components of our simulation. LEGOs can be programmed using Simkit. The listener pattern implementation in Simkit is called Simulation Event Listener or "SimEventListener" pattern using Simkit's interface name (Buss and

Sanchez, 2002 and Buss, 2002). The simulation event listener patterns are used to ensure communication between components of the simulation. This pattern (SimEventListener pattern) is used to connect simulation components in a loosely coupled manner. Simulation objects register interest in certain other simulation objects' events. SimEvents are always invoked by a callback from the FEL to the scheduling object that ultimately invokes the corresponding "do" method. The SimEvent is then dispatched to every *SimEventListener* that has explicitly registered interest in that object's SimEvents.

There are four components or modules of the transportation simulation: Land, Sea, Air and Rail. In order for the two components to communicate with one another, their subcomponents or submodules must have at least one event with the same name. The same is true for two subcomponents to communicate with each other.

The Land Component consists of three subcomponents or submodules called classes in Java. These are named HomeBaseDeparture, LandMaintDelay, and LandArrivalAtDestination. For example, for the HomeBaseDeparture and LandMaintDelay subcomponents to communicate with each other there needs to be events with the same name in each one. The "DepartHomeBase" event is common to both HomeBaseDeparture and LandMaintDelay submodules as seen in Figures 13 and 15. Thus, when an event named "DepartHomeBase" is fired in HomeBaseDeparture component, it can be heard by the LandMaintDelay subcomponent if this component registers interest in that event by adding itself as a SimEventListener to "HomeBaseDeparture" subcomponent.

4.6.1. The Simulation Event Listener Pattern for Land and Sea Components

The simulation event listener pattern for the land and sea components is in Figure 26. The land component has three subcomponents: Home Base Departure, Land Maintenance Delay, and Land Arrival At Destination. The sea component has also three components: Land Arrival At SPOE, Sea Maintenance Delay, and Sea Arrival At SPOD.

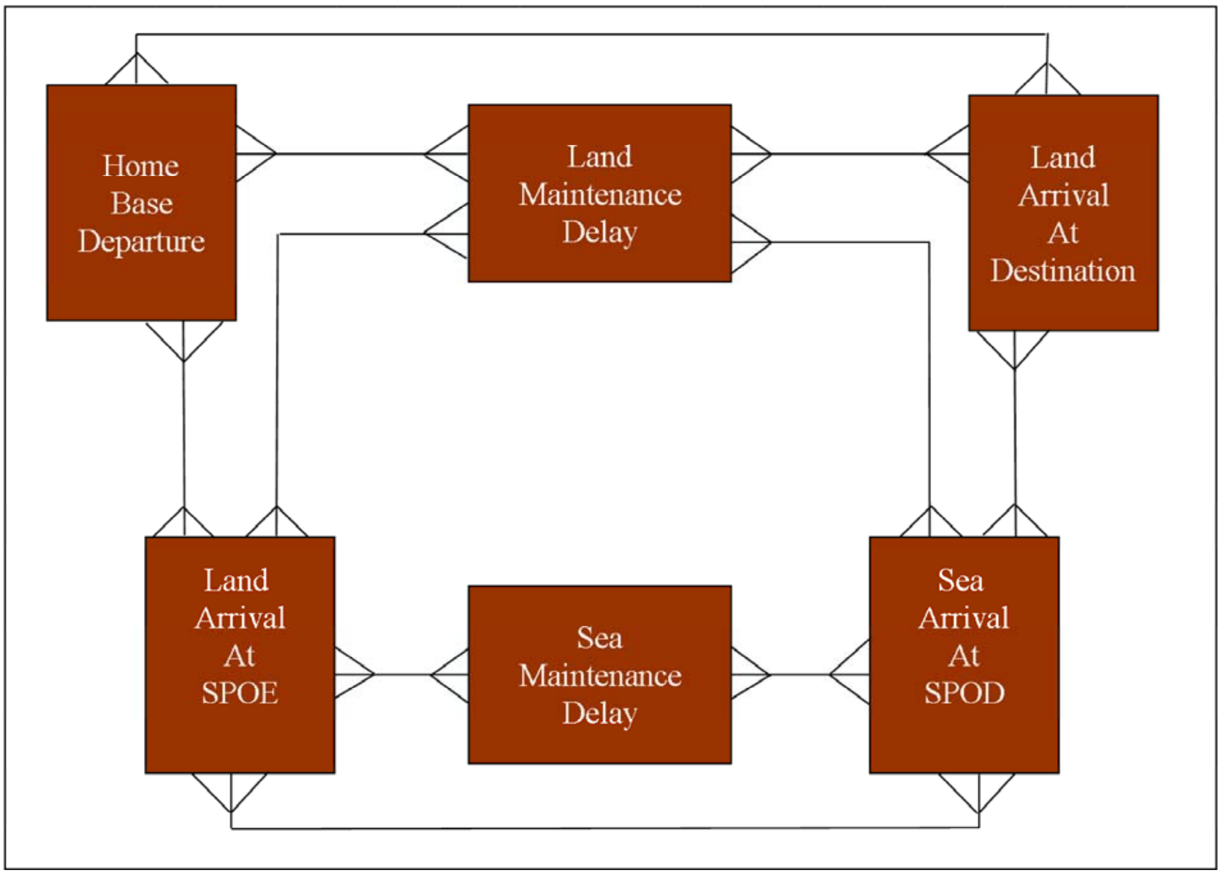


Figure 26. The Simulation Event Listener Pattern for Land and Sea Components.

The Land Maintenance Delay subcomponent listens to the "DepartHomeBase" events of departing convoys from the Home Base Departure component. Land Arrival At Destination subcomponent listens to the "LandArrivalAtDestination" events of the Land Maintenance Delay subcomponent. Land Maintenance Delay subcomponent also listens for "DepartDestination" events at Land Arrival At Destination subcomponent for convoys returning to the home base for multiple trips. For those convoys going to a SPOE, Land Arrival At SPOE subcomponent listens for "LandArrivalAtSPOE" events of the Land Maintenance Delay subcomponent. The Land Maintenance Delay subcomponent listens to "VehiclesDepartSPOE" events of the Land Arrival At SPOE subcomponent for the vehicles returning home base for multiple trips to the SPOE. Sea Maintenance Delay subcomponent listens to the "DepartShipFromSPOE" events of the Land Arrival At SPOE subcomponent. Sea Arrival At SPOD subcomponent listens to the "ShipArrivalAtSPOD" events of the Sea Maintenance Delay subcomponent. Similarly, the Sea Maintenance Delay subcomponent listens to the "DepartShipFromSPOD" events of the Sea Arrival At SPOD subcomponent for ship departures back to the SPOE. The Land Component Arrival At SPOE subcomponent listens to the "ShipArrivalAtSPOE" event of the Sea Maintenance Delay subcomponent.

The Land Maintenance Delay subcomponent listens to "VehiclesDepartSPOD" events of the Sea Arrival At SPOD subcomponent for vehicles going to the destination and thus may be subjected to maintenance delays.

4.6.2. The Simulation Event Listener Pattern for Land and Rail Components

The simulation event listener pattern for the land and rail components is explained here. The land component has three subcomponents: Home Base Departure, Land Maintenance Delay, and Land Arrival At Destination. The rail component has also three components: Land Arrival At RPOE, Rail Maintenance Delay, and Rail Arrival At RPOD.

The listening pattern connections are similar to those in Figure 26, with class names replaced by those of Rail component's. The SimEventListener pattern for land and rail components is displayed in Figure 27.

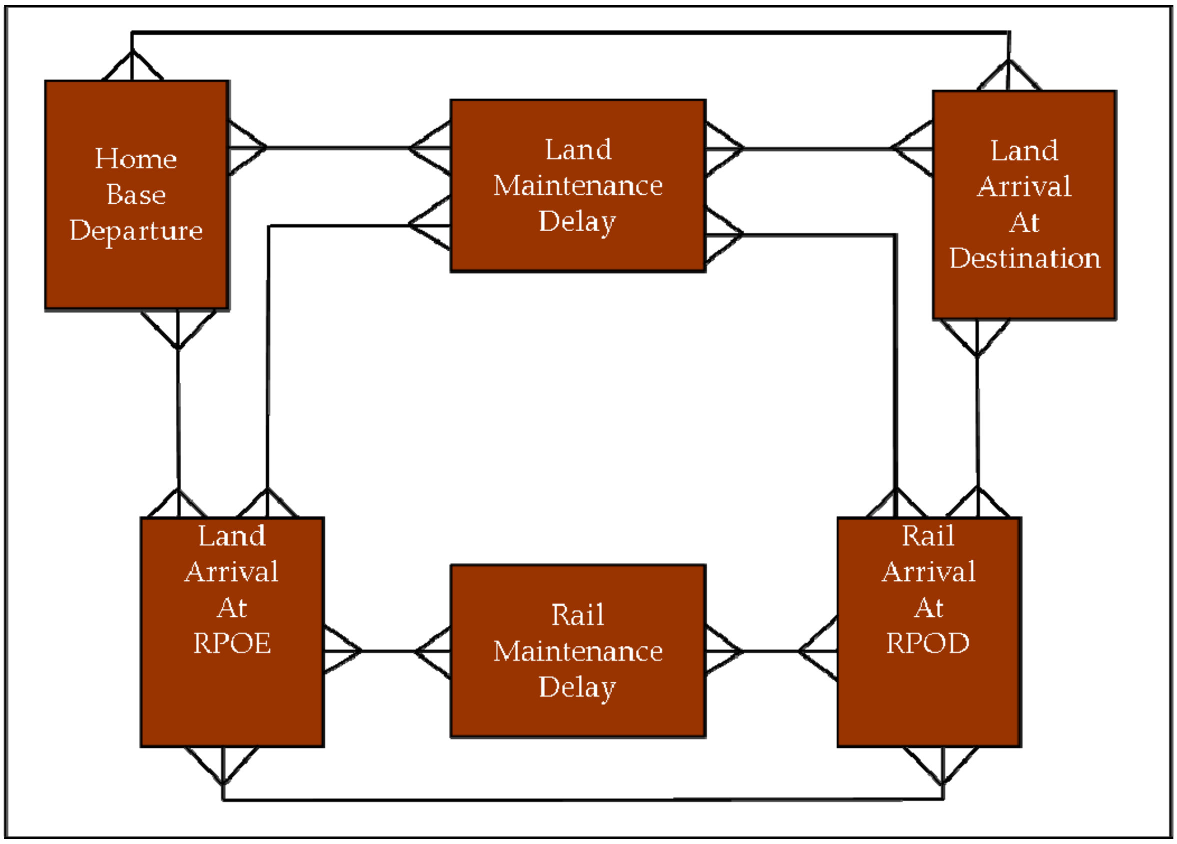


Figure 27. The Simulation Event Listener Pattern for Land and Rail Components.

4.6.3. The Simulation Event Listener Pattern for Land and Air Components

The simulation event listener pattern for the Land and Air components is explained here. The Land component has three subcomponents: Home Base Departure, Land Maintenance Delay, and Land Arrival At Destination. The Air component has three components: Land Arrival At APOE, Air Delay and Air Arrival At APOD.

The listening pattern connections are similar to those in Figure 26, with class names replaced by those of Air component's. The SimEventListener pattern for land and air components is displayed in Figure 28.

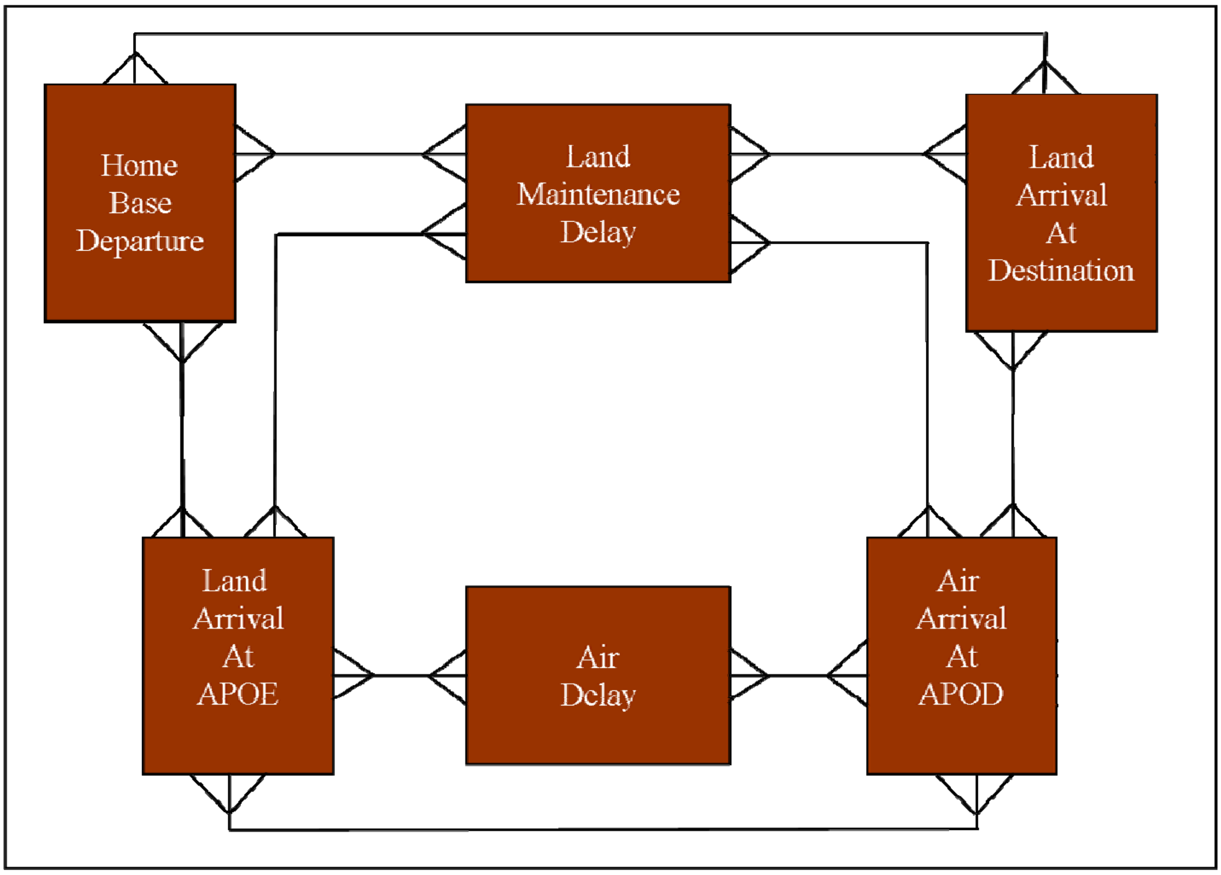


Figure 28. The Simulation Event Listener Pattern for Land and Air Components.

4.7. ANIMATION AND GRAPHICAL USER INTERFACE (GUI)

The implementation of animation in our simulation is performed by periodically scheduling a single recurring event called "Ping". A component called "PingThread" simply puts "Ping" events into the event list with deterministic time between occurrences. When the "Ping" event is heard by "Simulation" subcomponent of *model* component, it updates the locations of mover objects and the "GUIMain" subcomponent of *GUI* component repaints the icons of mover objects on the map view. A screenshot of a simple implementation of this is in Figure 29.

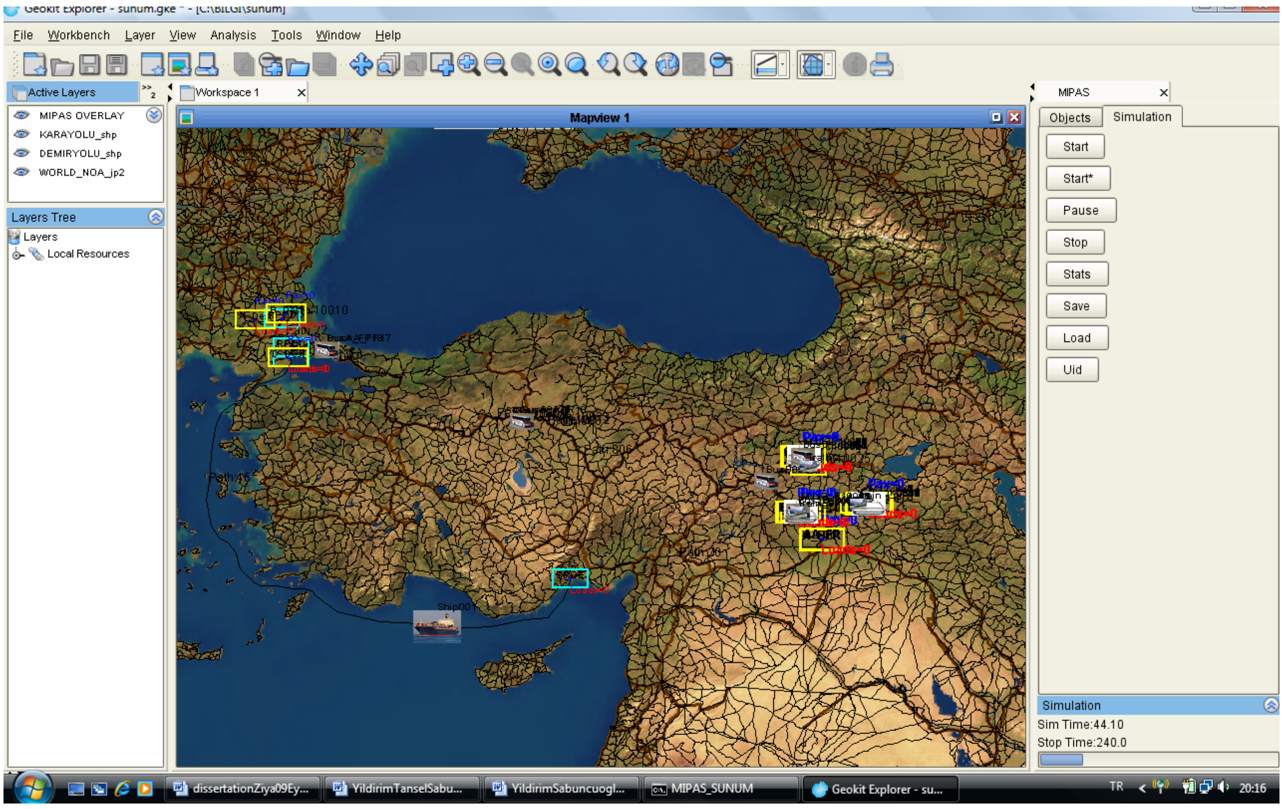
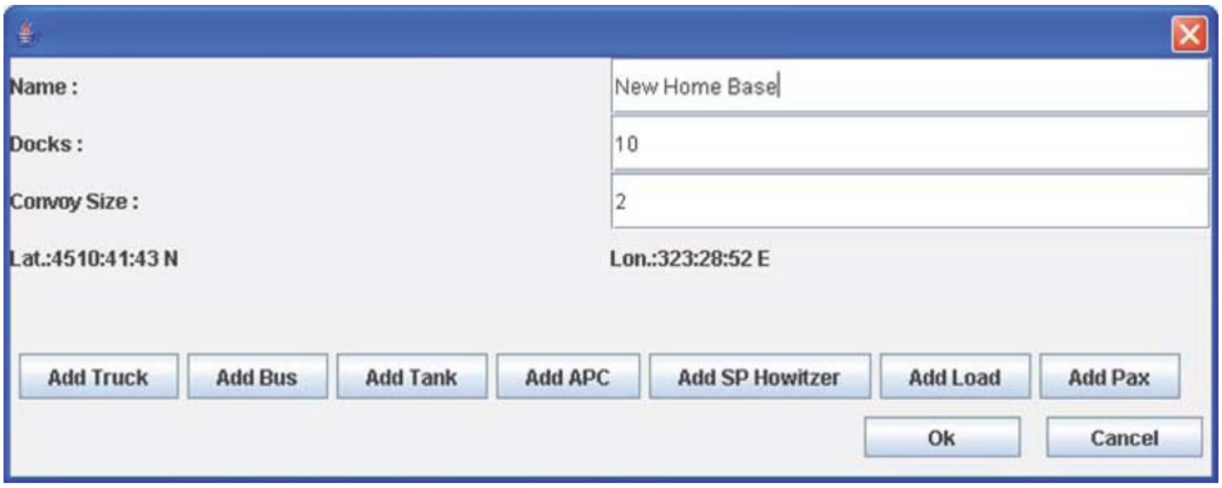


Figure 29. A sample screen shot of animation showing a deployment using road (thin black lines) and rail networks (thick brown lines) and sea lines (best viewed in color).

This sample screen shot of animation shows a deployment using road (thin black lines) and rail networks (thick brown lines) and sea lines (best viewed in color). A detailed treatment of simple movement and animation in DES is presented in (Buss and Sanchez, 2005).

We also provide a brief description of the GUI and provide a few sample screen captures to show the general structure of the GUI and the steps that would be required to define and run a simulation. Figure 30 is a screen capture of the home base creation screen from which the user can add appropriate transportation assets, loads, and pax to the home base created.


Name : New Home Base

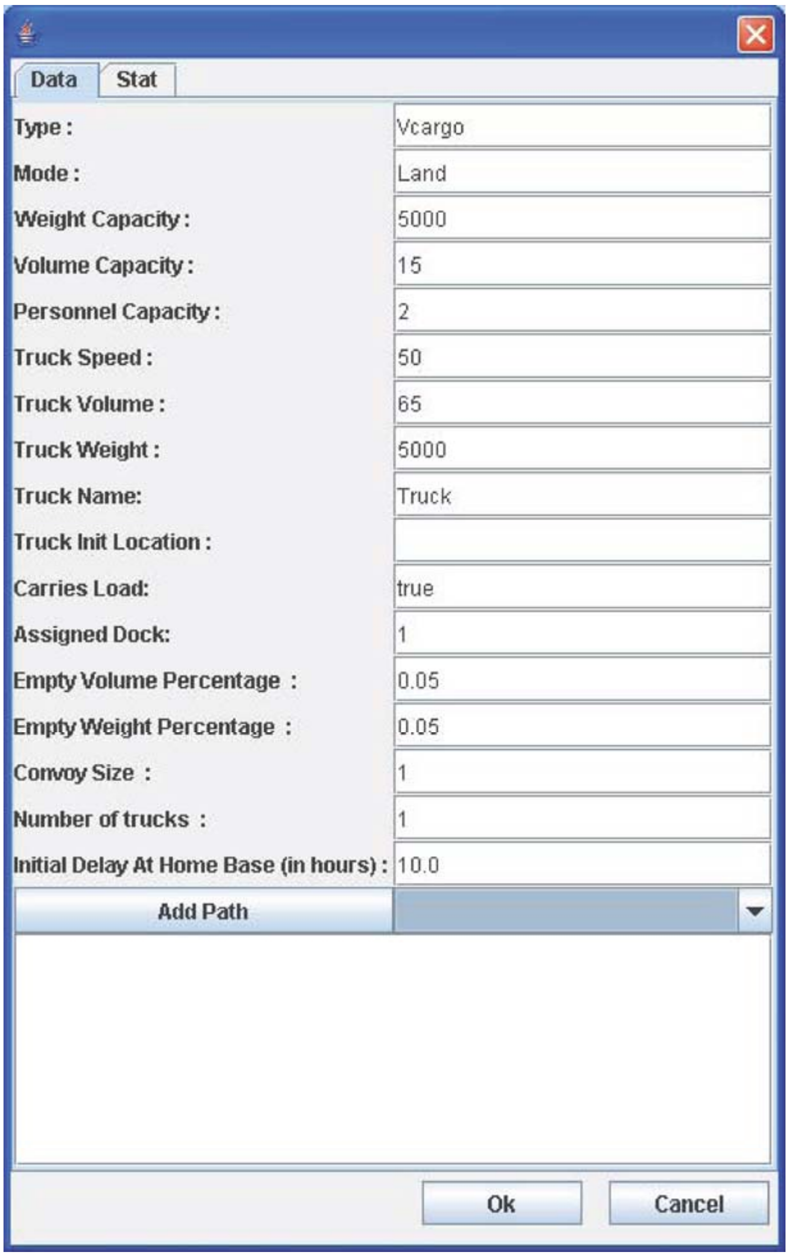
Docks : 10

Convoy Size : 2

Lat.:4510:41:43 N Lon.:323:28:52 E

Figure 30. A screen capture of the home base creation screen.

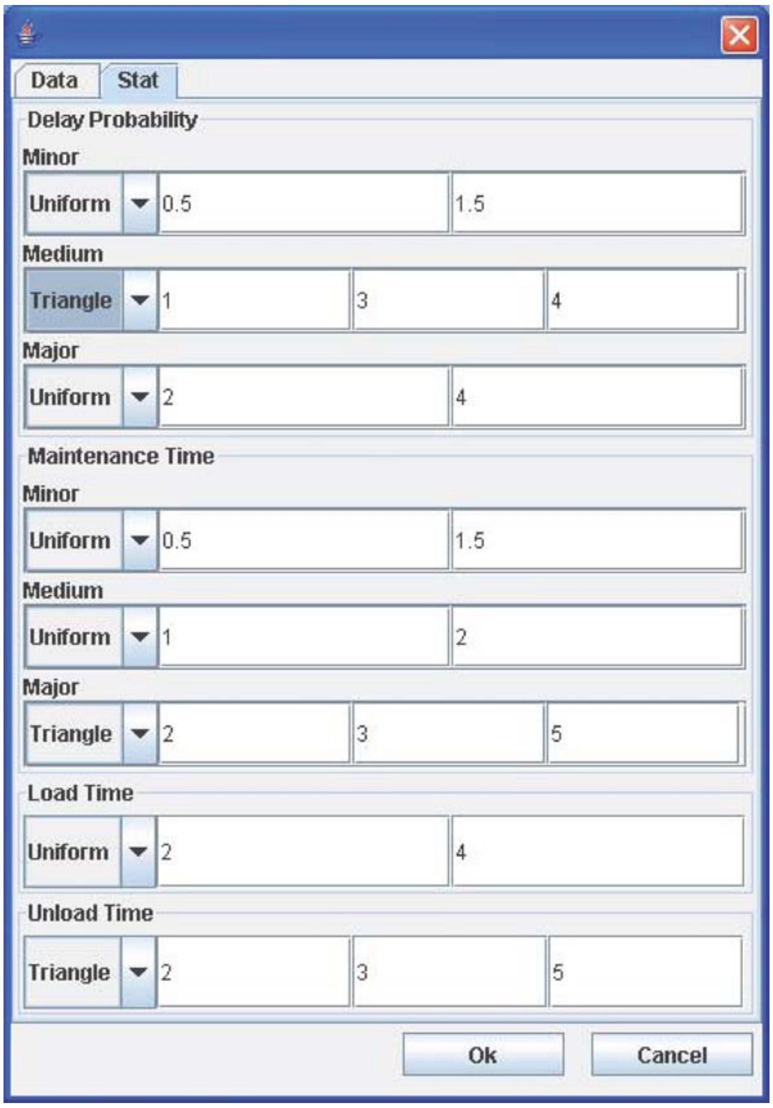
Figure 31 depicts the detailed data used in creation of a truck. The user can create trucks of different capacities and sizes, determine according to the deployment plan their initial delay times at their respective home bases, and assign routes (paths) to follow. It is clear from Figure 31 that the modeling level of resolution for truck entity is high.



Type :	Vcargo
Mode :	Land
Weight Capacity :	5000
Volume Capacity :	15
Personnel Capacity :	2
Truck Speed :	50
Truck Volume :	65
Truck Weight :	5000
Truck Name:	Truck
Truck Init Location :	
Carries Load:	true
Assigned Dock:	1
Empty Volume Percentage :	0.05
Empty Weight Percentage :	0.05
Convoy Size :	1
Number of trucks :	1
Initial Delay At Home Base (in hours) :	10.0

Figure 31. A screen capture of the truck creation screen.

Figure 32 is a screen capture of the GUI where the user can select breakdown probabilities, their maintenance delay times and load/unload times for trucks. All standard probability distributions are supported. Creation of other entities such as buses, tanks, self-propelled howitzers, trains, and ships used in the simulation are done using similar GUIs. They are not all included here.



Delay Probability

Minor

Uniform	0.5	1.5	

Medium

Triangle	1	3	4

Major

Uniform	2	4	

Maintenance Time

Minor

Uniform	0.5	1.5	

Medium

Uniform	1	2	

Major

Triangle	2	3	5

Load Time

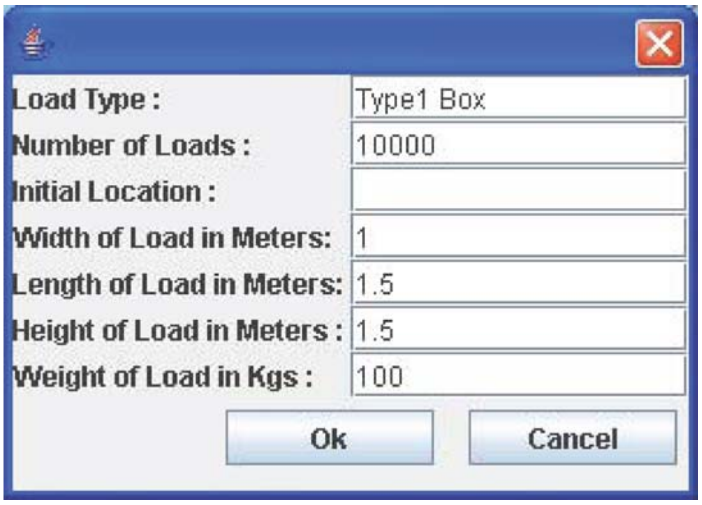
Uniform	2	4	

Unload Time

Triangle	2	3	5

Figure 32. A screen capture of the truck breakdown probabilities and maintenance, load/unload time distributions creation screen.

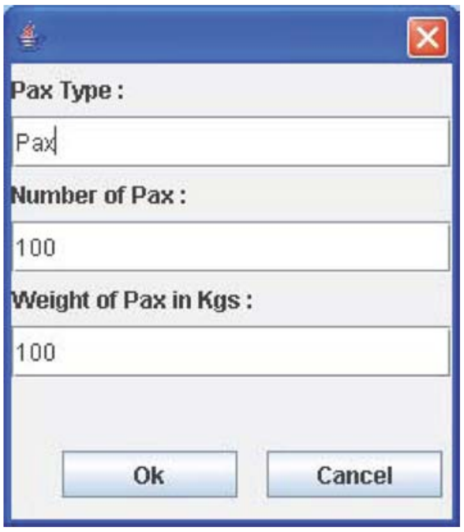
Figure 33 depicts the GUI used in detailed (height, length, width, weight, and volume) creation of loads.



Load Type :	Type1 Box
Number of Loads :	10000
Initial Location :	
Width of Load in Meters:	1
Length of Load in Meters:	1.5
Height of Load in Meters :	1.5
Weight of Load in Kgs :	100

Figure 33. A screen capture of load creation.

Figure 34 depicts the GUI used in creation of pax.



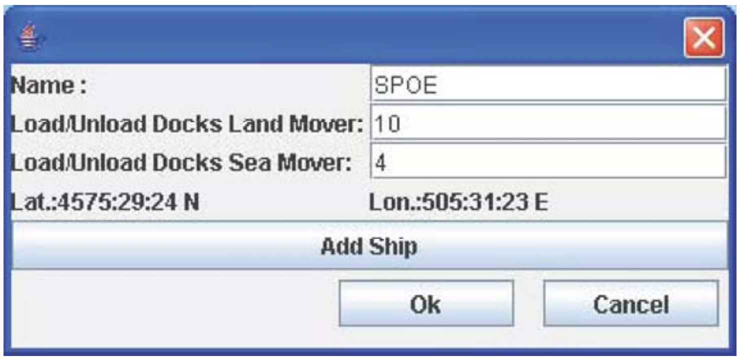
Pax Type :
Pax

Number of Pax :
100

Weight of Pax in Kgs :
100

Figure 34. A screen capture of pax creation.

Figure 35 is a screen capture of the seaport of embarkation (SPOE) creation screen from which the user can determine the number of load/unload docks at the port and add ships of appropriate characteristics to the SPOE created. The GUI for creation of a SPOD is similar and thus not presented here.



Name :	SPOE
Load/Unload Docks Land Mover:	10
Load/Unload Docks Sea Mover:	4
Lat.:4575:29:24 N	Lon.:505:31:23 E

Figure 35. A screen capture of SPOE creation.

Figure 36 depicts the GUI used by the network module of the simulation which allows shortest path selection in route planning. In addition, it allows the listing and selection of all routes on land and rail networks whose length (cost) is a user-specified percentage more than the shortest path's.

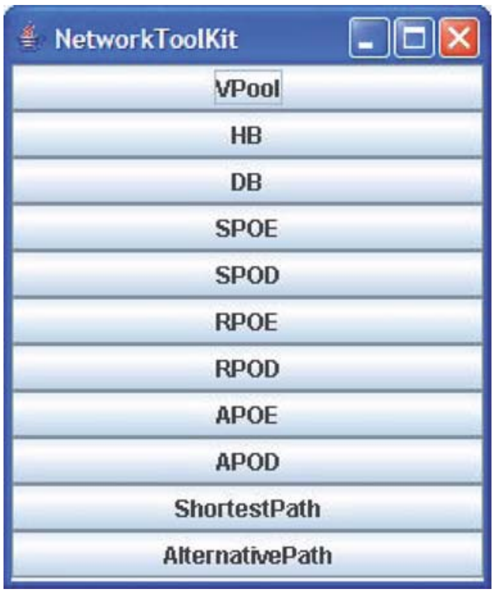


Figure 36. A screen capture of network module.

4.8. EXPLANATIONS OF SOME OF THE OTHER JAVA CLASSES (MODULES)

The simulation has a total of *97 Java classes (modules)* and *over 16000 lines of code*. Here we provide explanations of some of the Java classes or modules used in the simulation. The requests for obtaining the entire code for research and other purposes may be considered on a case by case basis by the author and the thesis advisors of this dissertation. The classes are organized under five packages or components. These are the model, gui, statistics, network, and explorer packages.

4.8.1. Model Package (Component)

Some of the classes (subcomponents) in this package and a brief explanation on what they do are provided below.

HomeBaseDeparture, LandArrivalDestination, LandArrivalAtAPOE, LandArrivalAtSPOE, LandArrivalAtRPOE, MaintenanceDelay, SeaArrivalAtSPOD, RailArrivalAtRPOD, AirArrivalAtAPOD are the classes explained previously using their event graphs.

Bus, APC (armored personel carrier), Ship, SPHowitzer (self-propelled howitzer), Tank, Truck, Plane, Train are classes or design blueprints to create carriers of different sizes and capacities.

Pax and Load classes create loads and troops to be transported. PaxStock and PaxContainer are helper classes for Pax class.

MipasMover, MipasPath, and MipasPathMoverManager are used for creation and movement of movers. Convoy class is used for creation of convoys.

PingThread, PingListener and PingEvent classes are used for animation purposes. MapViewProvider is an interface class for displaying map view.

4.8.2. GUI Package (Component)

Most of the classes used in the model component have their corresponding classes related to GUI in this package. MipasHomeBaseDialog creates a dialog box for creation of home bases. It is similar for other classes such as SPOE and MIPASSPOE etc. MIPASHomeBase wraps HomeBaseDeparture class so that it is displayable on the mapview. Dialog boxes, path creation handling, and BlackSpots are displayed using the classes in this package.

4.8.3. Statistics Package (Component)

This is actually a subpackage of the gui component and it provides the panels for creation of statistical distributions and delays. Some of the classes are DelayPanel, DistributionFactory which allows for creation of all standard distributions, StatDistributionPanel, StatInputPanel, and TriangularDist.

4.8.4. Network Package (Component)

This package has NetworkCreater, NetworkDirectionFunction, NetworkShortestPathCreationHandler, NetworkAlternativePathCreationHandler, NetworkCostFunction, NetworkUtil, and NetworkWizard classes used for shortest and alternative path creation and cost calculation.

4.8.5. Explorer Package (Component)

This package has ExplorerPlugin, ObjectControlPanel and SimulationControlPanel classes used for running the simulation as a plugin inside GeoKIT Explorer.

5. VERIFICATION AND VALIDATION

Verification (also referred to as debugging by Kleijnen in (Sargent et al., 2000)) is simply making sure that the computer program of the computerized model and its implementation is correct. A model is said to be valid if within its domain of applicability it posesses a satisfactory range of accuracy consistent with the intended application of the model (Sargent, 1999).

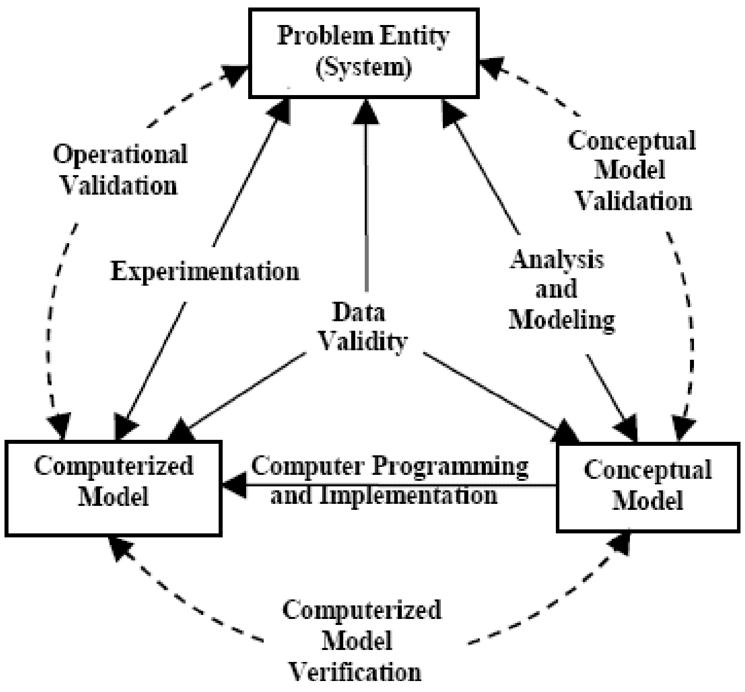


Figure 37. Simplified Version of the Modeling Process (Adapted from Sargent, 2001).

Sargent favors the simplified version of the modeling process displayed in Figure 37 and relates verification and validation to this.

Verification and validation were conducted by using appropriate methods explained in (Sargent, 2001 and Balcı, 1998). To mention a few more specifically, for *face validity*, we have discussed inputs and outputs of the model and its EGs with potential users of the model and personnel at Transportation Coordination Center of the General Staff. We used *assertion checking* to verify that the model functioned within its acceptable domain. Incrementally, *bottom-up testing* was performed, where each individual subcomponent was tested and integrated. *Fault (failure) insertion testing* was used to test whether the model responded by producing an invalid behavior given the faulty component. During *special input testing,* we used an arbitrary mixture of minimum and maximum values, and *invalid data* for the input variables, and tested for potential peculiar situations at the *boundary values*. In addition, we have tested the validity and behavior of the model under *extreme workload and congestion* at the load/unload docks and transfer points such as SPOEs, SPODs etc. Animation also helped in discovering errors during model development.

Furthermore, results of the deployment optimization model developed by Akgün and Tansel (2007) were used for verification purposes. Akgün and Tansel used three different networks with number of nodes 13, 19, and 25 and number of arcs 49, 77, and 109, respectively. For each network, they generated five problems corresponding to 4, 8, 16, 32, and 64 types of items. Of the 15 problem types generated by Akgün and Tansel, we have used for verification purposes three of them, corresponding to three different network sizes, and 32 item types. We report here only the scenario corresponding to the network with 13 nodes and 49 arcs depicted in Figure 38.

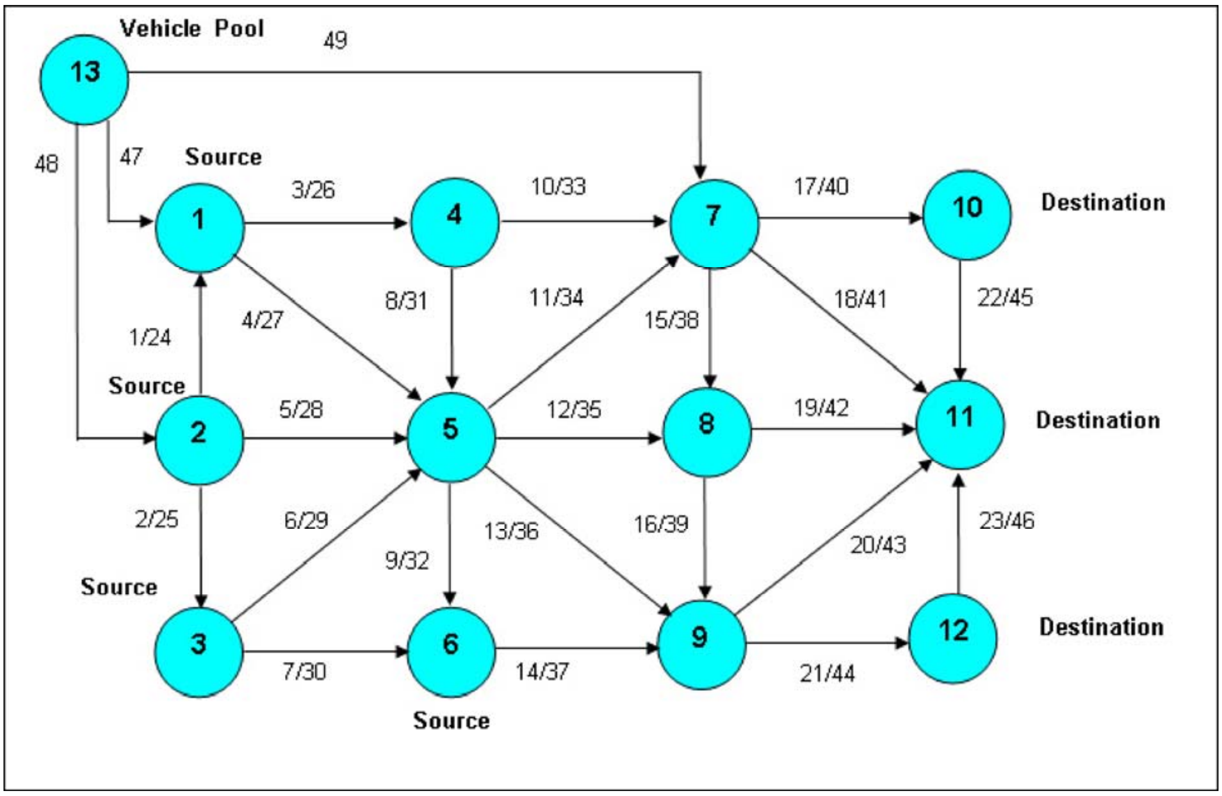


Figure 38. One of the network models used for verification purposes.

The first set numbers on the arcs are those of the directed arcs depicted in Figure 38. The second set of numbers belongs to the reverse arcs. In this network, source nodes (home bases) are 1, 2, 3, and 6. Demand nodes (destinations) are 10, 11, and 12. Transshipment nodes are 4, 5, 7, 8, and 9. There is no transfer node (POE or POD). Dummy node (vehicle pool) is 13. It provides transportation assets inorganic to the units and can be regarded as a home base for vehicles. In this scenario, 32 different item types of varying amounts are transported. Item types 1, 9, 17, and 25 are pax. The others are loads of different dimensions and weights. There are 6 different types of TAs allocated to this scenario. The total number of TAs allocated is 61 vehicles. Load and unload times for each TA is 1 hour. For all TAs, the minimum and maximum arrival times at destinations are 80 and 100 hours, respectively. The scenario had around 180,000 single variables, 4,300 discrete variables and 130,000 equations. The objective was to find the minimum cost deployment plan meeting the time-window deadlines to be at destination. This scenario had only one mode of transportation (land). The scenario was solved to optimality via ILOG CPLEX 9.0 on a 1.5 GHz PIV PC with 1.5 GB RAM. The optimal cost obtained is $5,727,912.00. This cost figure includes fixed cost of procurement of TAs and the variable cost of travel per unit time.

A part of the optimal solution of the model that shows movement routes and times of loaded trucks is in Table 3.

Convoy #	Departure Time	Source Node	TA #	TA Type	Through Nodes	Demand Node
1 (1st trip)	68	1	1	1	5-9	12
	68	1	16	6	5-9	12
	68	1	1	4	5-9	12
2 (1st trip)	68	3	10	3	5-7	10
	68	3	2	4	5-7	10
	68	3	4	6	5-7	10
2 (2nd trip)	89	2	10	3	5-9	11
	89	2	2	4	5-9	11
	89	2	4	6	5-9	11
1 (2nd trip)	90	6	1	1	5-7	10
	90	6	1	4	5-7	10
	90	6	16	6	5-7	10

Table 3. Optimal numbers, movement routes, and movement times of loaded trucks.

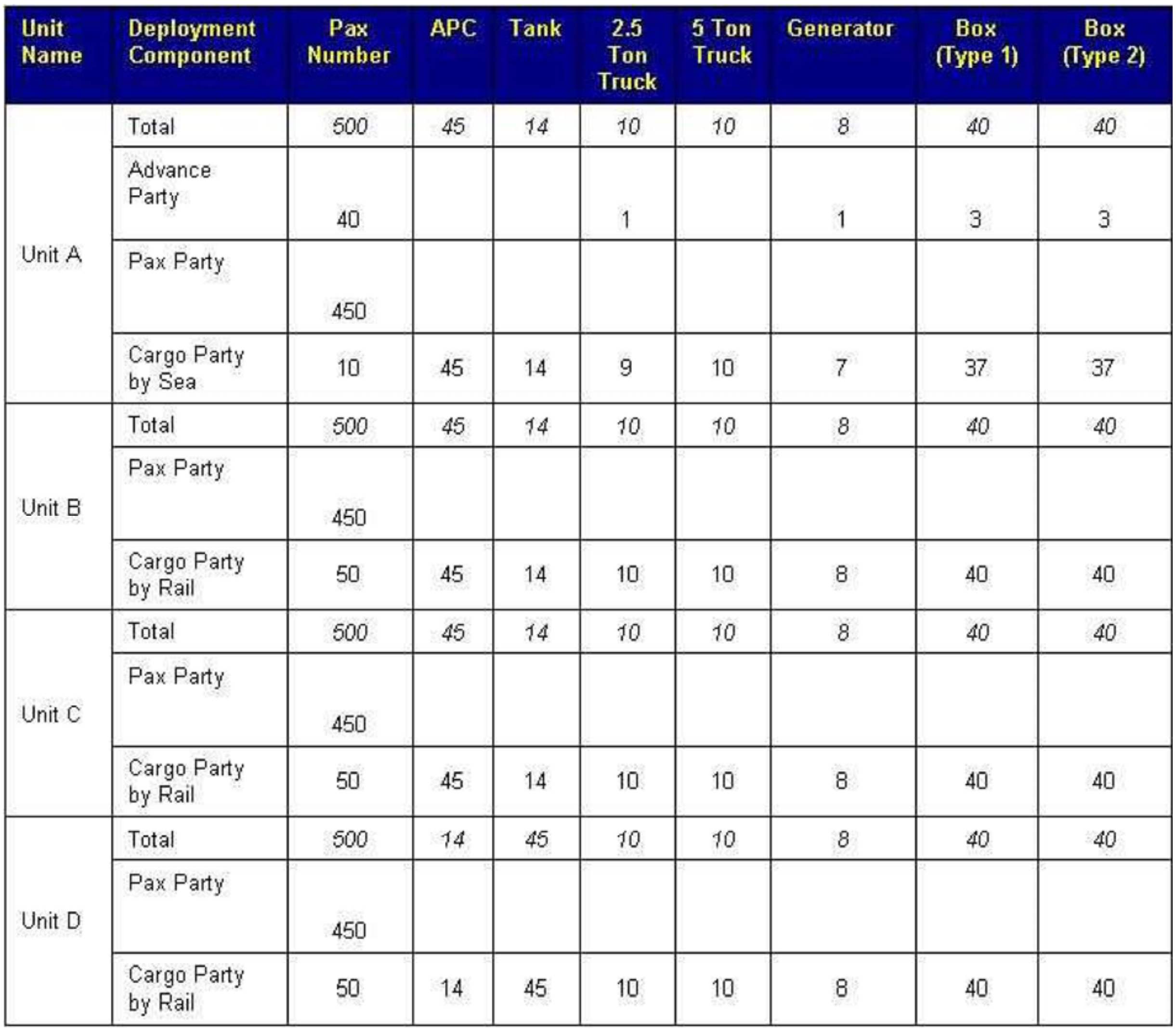
The first line in Table 3 shows that one TA of type 1 departs from source node 1 at time t=68, and travels through nodes 5 and 9 to reach the demand node 12. The first three lines of Table 3 indicate that a total of 18 (1+16+1) vehicles depart as a convoy from source node 1 at time t=68. Upon reaching demand node 12 (at time t=82, not shown), these vehicles unload and return to source node 6. As seen in the last three rows in Table 3, these vehicles in the first three lines of Table 3 that departed at t=68 for destination node 12, now depart at t=90 from source node 6 to demand node 10. Thus, in the optimal solution of network in Figure 38, multiple trips are made from source node 1 to destination node 12 and from source node 6 to destination node 10 by the same set of vehicles shown in the first and last three rows of Table 3. The remaining rows of Table 3 also depict multiple trips between source node 2 to destination node 11, and source node 3 to destination node 10. The optimization results for movement of empty TAs from dummy node (vehicle pool) 13, and after unloading at destination nodes are not included here. The results of the optimization model were simulated using deterministic values used in the optimization model. That is, load and unload times were taken as 1 hour each. Travel times used in arcs of the optimization network were transferred to distance figures using constant travel speeds (60 km. per hour) to be able to physically create the network in the simulation. The cost figure obtained under these deterministic conditions from the simulation was the same as the cost figure obtained from the optimization model.

In addition, simulation results were compared to the historical deployment data obtained from the Scientific Decision Support Center of the General Staff. The results of the comparison were satisfied in terms of validation of our model.

6. SIMULATION ANALYSIS OF A TYPICAL REAL-WORLD SCENARIO

6.1. SIMULATION ANALYSIS OF A TYPICAL REAL-WORLD CASE STUDY

A typical and most-encountered deployment scenario, which is a good benchmark to test our model, for deploying four battalions (three mechanized and one armored) during peace-time from southeastern to northwestern Turkey is analyzed. The scenario uses land, sea, and rail transportation networks and assets. These four units deploy from three different home bases to three unique destinations. Units C and D are co-located, and Units B and C deploy to the same destination location. The data related to the deployment of each unit and its components are listed in Table 4.

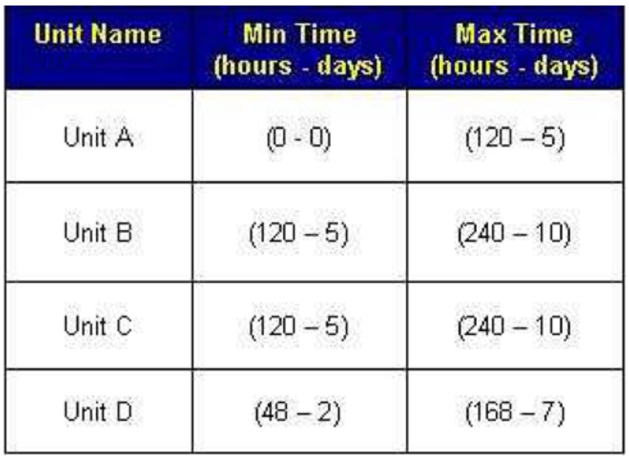


Unit Name	Deployment Component	Pax Number	APC	Tank	2.5 Ton Truck	5 Ton Truck	Generator	Box (Type 1)	Box (Type 2)
Unit A	Total	500	45	14	10	10	8	40	40
	Advance Party	40			1		1	3	3
	Pax Party	450							
	Cargo Party by Sea	10	45	14	9	10	7	37	37
Unit B	Total	500	45	14	10	10	8	40	40
	Pax Party	450							
	Cargo Party by Rail	50	45	14	10	10	8	40	40
Unit C	Total	500	45	14	10	10	8	40	40
	Pax Party	450							
	Cargo Party by Rail	50	45	14	10	10	8	40	40
Unit D	Total	500	14	45	10	10	8	40	40
	Pax Party	450							
	Cargo Party by Rail	50	14	45	10	10	8	40	40

Table 4. Units listed by their deployment components and major equipment.

As shown in the rows allocated to Unit A in Table 4, Unit A deploys in three components; *advance, pax,* and *cargo* parties. The *advance party* for Unit A has 40 pax, one 2.5-ton truck, one generator, three Type 1 boxes, and three Type 2 boxes. The *pax party* for Unit A has 450 personnel. The *cargo party* for Unit A will deploy by sea. It has 10 pax, 45 armored personnel carriers (APCs), 14 tanks, nine 2.5-ton trucks, ten 5-ton trucks, seven generators, 37 Type 1 boxes, and 37 Type 2 boxes. Unless otherwise indicated in Table 4, each component deploys by land. The total number of deployed personnel and equipment for Unit A are shown in the line named *total*. Units B, C, and D deploy in two components each. Their *cargo parties* deploy by rail as indicated in Table 4.

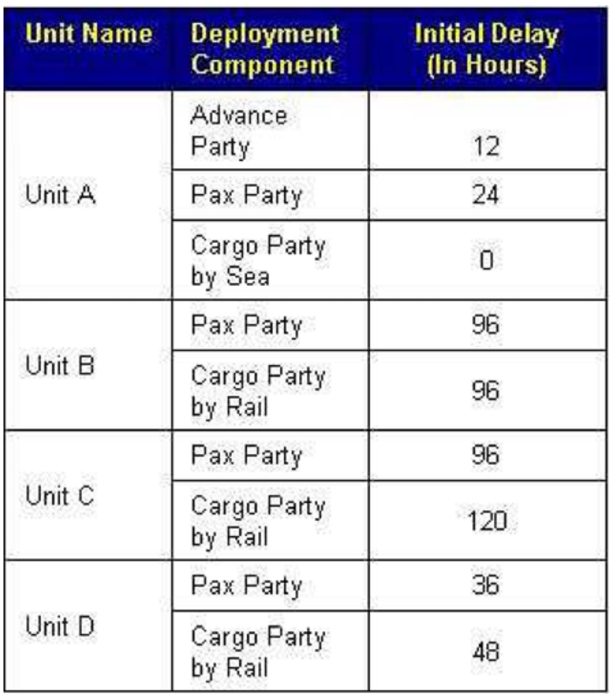
Table 5 shows the minimum and maximum time requirements for deployed units to be at their designated destination locations. For example, all components of Unit B must deploy not before 120 hours and not after 240 hours.



Unit Name	Min Time (hours - days)	Max Time (hours - days)
Unit A	(0 - 0)	(120 – 5)
Unit B	(120 – 5)	(240 – 10)
Unit C	(120 – 5)	(240 – 10)
Unit D	(48 – 2)	(168 – 7)

Table 5. Time windows for units to be deployed.

Table 6 shows the initial delay times for each deployment component. Delay times are used to ensure timely and coordinated arrivals at destinations for each deploying unit. For example, Pax component of Unit D departs after 36 hours of simulation time, and its Cargo component departs after 48 hours.



Unit Name	Deployment Component	Initial Delay (In Hours)
Unit A	Advance Party	12
	Pax Party	24
	Cargo Party by Sea	0
Unit B	Pax Party	96
	Cargo Party by Rail	96
Unit C	Pax Party	96
	Cargo Party by Rail	120
Unit D	Pax Party	36
	Cargo Party by Rail	48

Table 6. Initial delay times for each deployment component.

The preparation time for all units in this scenario is 10 days (240 hours) and is not listed in Table 6. The preparation time is also not simulated. The simulation for this scenario is a terminating one with a termination time of 240 hours. The transportation assets allocated to this scenario are not listed in detail here. They include one large RoRo ship, four trains with enough and appropriate rail cars, trucks, and tank carriers (one tank carrier can carry a single tank or 2 APCs). The armored vehicles must be carried by rail or sea to distances over 300 km. In this scenario, they deploy over 300 km. Other restrictions used in the simulation, such as the storage capacities of SPOE, SPOD, RPOE, and RPOD, are not presented here.

6.2. INITIAL RESULTS

We adjust the sample size by setting the number of replications. To achieve the desired accuracy of 90% relative precision (10% error), we first ran the model for 5 replications for a run-length of 240 hours. Using our performance measure of percentage of average on-time arrivals at each unit's destination, we calculated point and interval estimates. For each unit, the half-width as a measure of accuracy varied. In order to achieve the desired accuracy in the worst case, we used the maximum half-width for all units for the given run-length and decided that 10 runs would suffice.

Yet, since run-time is not expensive in our case, we ran the simulation for 60 replications (around 25 minutes) on a 1.6 GHz Pentium(R) M PC with 512 MB RAM to see the model's behavior. Our performance measure was the percentage of average number of TAs from each of the four units that arrived in their given time windows at their respective destinations. This performance measure was met for each unit. However, there were different numbers of vehicle breakdowns that lasted varying amounts of time. Even though all TAs from 4 units arrived during their given time windows, some had to deal with delays due to maintenance problems. To be more specific, on average, Unit A had two minor breakdowns per run. Unit B had one minor breakdown every four runs. Unit C had 9 breakdowns per 5 runs, and 17 medium breakdowns per 20 runs. Unit D had one minor breakdown per run, one medium breakdown per 5 runs, and one major breakdown per 4 runs.

That is, all deployed units arrived at their respective destinations with some maintenance problems. This is common for any military deployment scenario, yet it does not provide much insight other than to have spare equipment and personnel to fix broken down vehicles. In order to see how this plan would behave in an emergency situation, such as a quickly escalating military situation or a natural disaster, we decided to reduce allowed maximum arrival times (Tmax) at each unit's respective destinations by 12-hour increments and ran the simulation for each case. Even though a replication number of 10 is sufficient, we used 15 replications for each case just to be on the safe side. We made a total of 15x12 = 180 replications. We assumed that since the situation is urgent, the initial delay figures depicted in Table 4 are zero. Likewise, the allowed minimum arrivals times at each unit's respective destination are taken as zero. The results are displayed for each unit. These figures represent the robustness of a peace-time deployment scenario under urgent circumstances. For Unit A, all deployment components arrived on-time for the peace-time scenario with Tmax = 120 hours. Figure 39 depicts that the percentage of average on-time arrivals for Unit A after Tmax= 96 hours drops below 20 percent. This is unacceptable by any military standard.

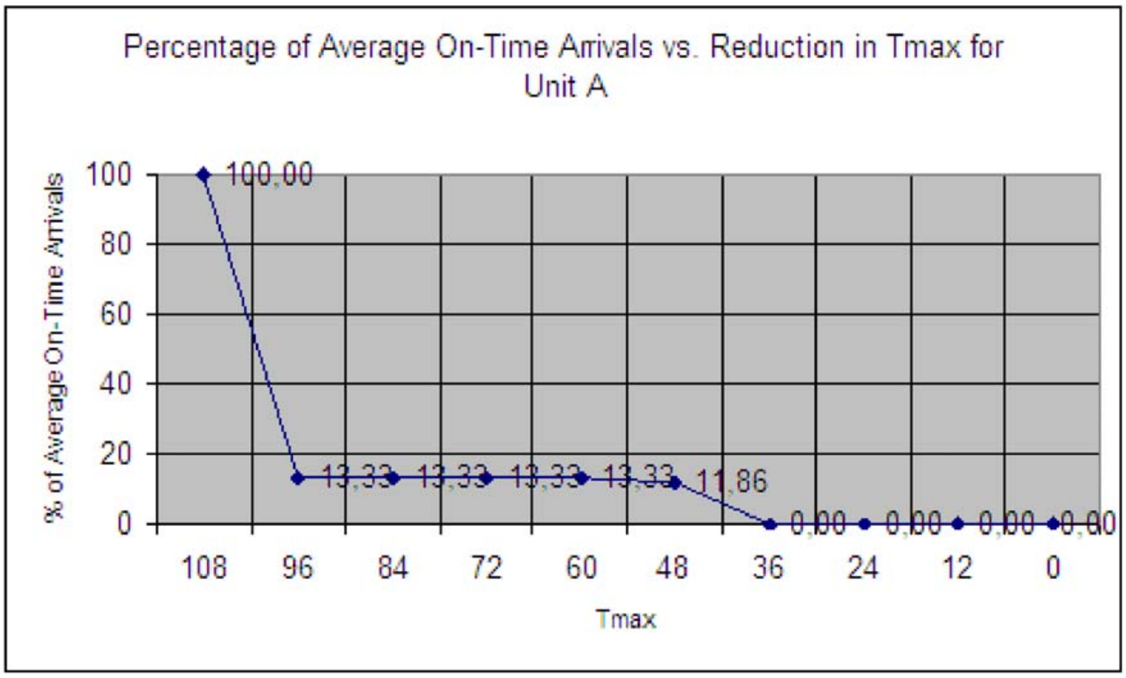


Figure 39. Percentage of Average On-Time Arrivals vs. Reduction in Tmax by 12 hours for Unit A.

Units B and C arrive at the same destination, and thus are considered together. For Units B and C, there is no change of average on-time arrivals in the range Tmax is considered. That is, this deployment scenario is robust for Units B and C. Figure 40 shows that the danger zone for Unit D lies between Tmax = 72 hours and 60 hours as the percentage of average on-time arrivals drops from 88 percent to 28 percent. 28 percent is certainly unacceptable.

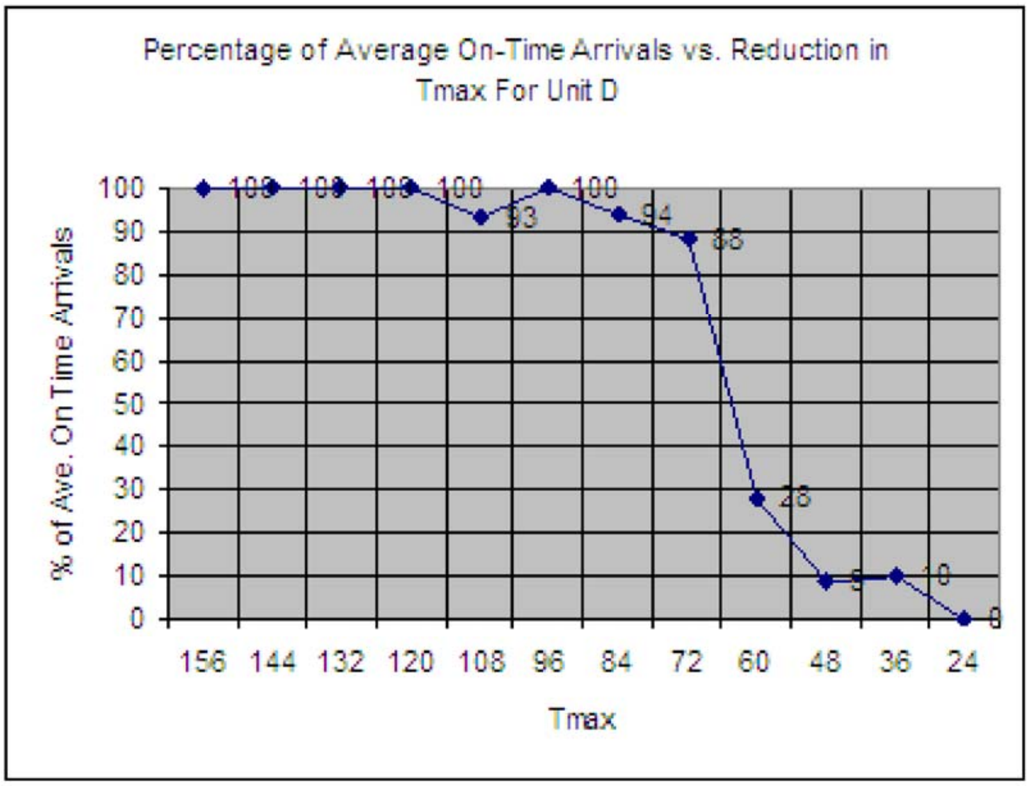


Figure 40. Percentage of Average On-Time Arrivals vs. Reduction in Tmax by 12 hours for Unit D.

These are very critical information as to what would happen if an existing plan had to be modified accordingly as unexpected events unfold. A military mission cannot be accomplished and lives could be lost if military units can not arrive at their designated destinations in a timely manner according to their plans. Our simulation proves to be an invaluable tool to create, and to determine the applicability and robustness of existing plans in a relatively short time.

6.3. FURTHER ANALYSIS USING NOLH SAMPLING

Even a small-scale military deployment scenario can have a large number of input variables (factors) that may impact the outcome of the plan. In classical design of experiments (DOE), one can explore only a handful of cases. With the use of computers and in a simulation setting, the user can vary a large number of input variables. Yet, even with today's computing power, a complete enumeration of all possible scenarios is an exhaustive task. To intelligently sample the state space of possible alternatives in a simulation setting, we use an approach developed for large-scale simulation experiments with many factors (Kleijnen et al. 2005, Cioppa and Lucas 2007).

6.3.1. Nearly Orthogonal Latin Hypercube Sampling and Its Application

DOE deals with different fields from medicine to farming, and provides procedures for efficient conduct of statistical experiments. Computer simulation provides a vast area in which to expand DOE. Some of the possible designs are presented in the Design Toolkit (Kleijnen et al. 2005). Latin Hypercube (LH) designs are recommended for simulation experiments with minimal assumptions and many factors. Ye (1998) developed an algorithm for orthogonal LH designs (Ye 1998). This was later extended by Cioppa and Lucas (2007), who gave up small amounts of orthogonality for better space-filling designs, and developed the Nearly Orthogonal Latin Hypercube (NOLH) designs. Orthogonal Latin Hypercube (OLH) and NOLH designs are special cases of LH designs. OLH designs have strict orthogonal properties, i.e., a matrix condition number of 1, and a maximum pairwise correlation of zero between any two columns in the design matrix. NOLH designs relax the requirements on the orthogonal properties. NOLH designs choose the most space-filling design among design matrices that satisfy near orthogonal thresholds. In a good space-filling design, the design points are required to be scattered throughout the experimental region with minimal unsampled regions. The limits used by Cioppa and Lucas (2007) are a condition number of less than 1.13 and a maximum pairwise correlation between all columns of the design matrix in the interval (-.03, .03). The NOLH design matrix is a compromise between complete enumerations of all possible scenarios, which is an exhaustive task even with today's computing power, and an OLH design. NOLH designs can also handle discrete variables, as opposed to LH designs which can only handle continuous variables, at a cost of orthogonality and space-filling properties if the levels of discrete variables are a few. Generating these designs is a time-consuming process. But a catalogue of ready-to-use designs are available online. This paper utilizes a 29-factor and 257-run design by Sanchez and Hernandez, 2005). Table 7 provides the factors and their levels for experimentation with our deployment simulation. The levels of these factors are entered into the spreadsheet provided by Sanchez and Hernandez (2005) to create a nearly-orthogonal and space-filling 257-run design matrix. The factors are the convoy speed, the number of load and unload docks at transfer points (such as SPOE/RPOE etc.) and minor/medium/major breakdown probabilities for transportation assets used in the deployment. These factors were chosen according to expert opinion. As more units are added, more factors with varying levels will have to be considered.

#	Factor Name	Low	High
1	Convoy speed	30	70
2	Unit A Sea Group # load docks	3	37
3	Unit A Destination # unload docks	3	10
4	Unit B Home # load docks	3	37
5	Unit B RPOE # load docks	1	10
6	Unit B RPOD # unload docks	1	6
7	Unit B Destination # unload docks	3	37
8	Unit C Home # load docks	3	20
9	Unit C RPOE # load docks	1	10
10	Unit D Home # load docks	3	37
11	Unit D RPOE # load docks	1	6
12	Unit D RPOD # unload docks	1	6
13	Unit D Destination # unload docks	3	37
14	Truck Minor Breakdown Probability	0	0.05
15	Truck Medium Breakdown Prob.	0	0.07
16	Ship Minor Breakdown Probability	0	0.1
17	Ship Medium Breakdown Prob.	0	0.008
18	Train Minor Breakdown Probability	0	0.06
19	Train Medium Breakdown Prob.	0	0.02
20	Train Major Breakdown Probability	0	0.001
21	Bus Minor Breakdown Probability	0	0.05
22	Bus Medium Breakdown Probability	0	0.025
23	Bus Major Breakdown Probability	0	0.008
24	Tank Minor Breakdown Probability	0	0.1
25	Tank Medium Breakdown Prob.	0	0.04
26	Tank Major Breakdown Probability	0	0.008
27	APC Minor Breakdown Probability	0	0.15
28	APC Medium Breakdown Prob.	0	0.025
29	APC Major Breakdown Probability	0	0.009

Table 7. Factors and their levels.

6.3.2. Further Results

Part of the design matrix (first 10 runs for the first 5 factors) created using the factors and their levels in Table 7 are presented in Table 8. For each of these input combinations (rows) of our 257-run design matrix (partly depicted in Table 8), we have written a script to modify the simulation's base-case XML scenario file.

low level	30	3	3	3	1
high level	70	37	10	37	10
decimals	0	0	0	0	0
factor name	1	2	3	4	5
1	46	33	7	24	7
2	35	17	9	36	7
3	36	23	4	22	10
4	46	8	6	25	8
5	45	22	7	17	9
6	48	15	10	8	9
7	32	24	6	7	6
8	45	4	6	17	9
9	33	21	10	24	4
10	49	6	9	27	1

Table 8. 10 Runs for First 5 Factors of 257-Run Matrix.

We made 15 replications (to achieve the desired accuracy of 90% relative precision) of each of the newly-created 257 scenarios to reach a total of 257x15=3855 computer runs. Compare this to an experiment with 29 factors each with only 2 levels and 15 replications per run for a complete enumeration experiment (2^{29} x15= 8,053,063,680 computer runs!). We have written a VBA script to extract and calculate the percentage of on-time arrivals for each run (averaged across replications) from the simulation output files written onto EXCEL spreadsheets. For Unit A, Averaged on-time Arrivals at its destination (AoA) ranged between 76.8% and 100% with a average of 97.93% and a standard deviation of 0.0374. That is, on average, 97.93 percent of all vehicles of Unit A arrived at their destination on-time according to the deployment plan. For Units B and C deploying to the same destination, AoA ranged between 96% and 100% with an average of 99.96% and a standard deviation of 0.0034. The deployment plans for Units B and C seem to be robust across the input combinations simulated. For Unit D, AoA ranged between 93.04% and 98.8% with an average of 98.57% and a standard deviation of 0.0118.

To identify which factors contribute more to our performance measure of AoA, we use a nonparametric approach, namely regression trees, to reveal the structure in the data in a more human-readable way. We append the AoA response variable for Units as 30th column to the 257-run design matrix for 29 factors, and import this newly formed 30-column matrix into JPM (SAS 2005). This is done separately for Unit A, Unit D, and Units B&C. The data are split into two leaves in such a way that the variability in the response within each leaf decreases and the variability in the response between leaves increases (Kang, Doerr, and Sanchez 2006). The split is continued until the point of diminishing returns in the value of R^2 (a measure of the amount of variance in the data that is explained by the given model) is reached. Only after 26 splits, an R^2 of 0.617 is reached for Unit A. For Units B and C, an R^2 value of 0.317 is reached only after JMP makes 6 splits, and no further splits can be made. However, a careful examination of the split columns (factors) reveal that two factors have nothing to do with the deployment of Units B and C , and thus can be pruned to achieve an almost the same level of R^2 of 0.314. Figure 41 shows the final regression tree for Units B and C. For (column) factor 11 (the number of load docks at RPOE for Unit D), the AoA is 100% for

60 scenarios when number of load docks is greater than 2, as opposed to an AoA of 99.4% for 6 scenarios when the number of load docks is less than 2.

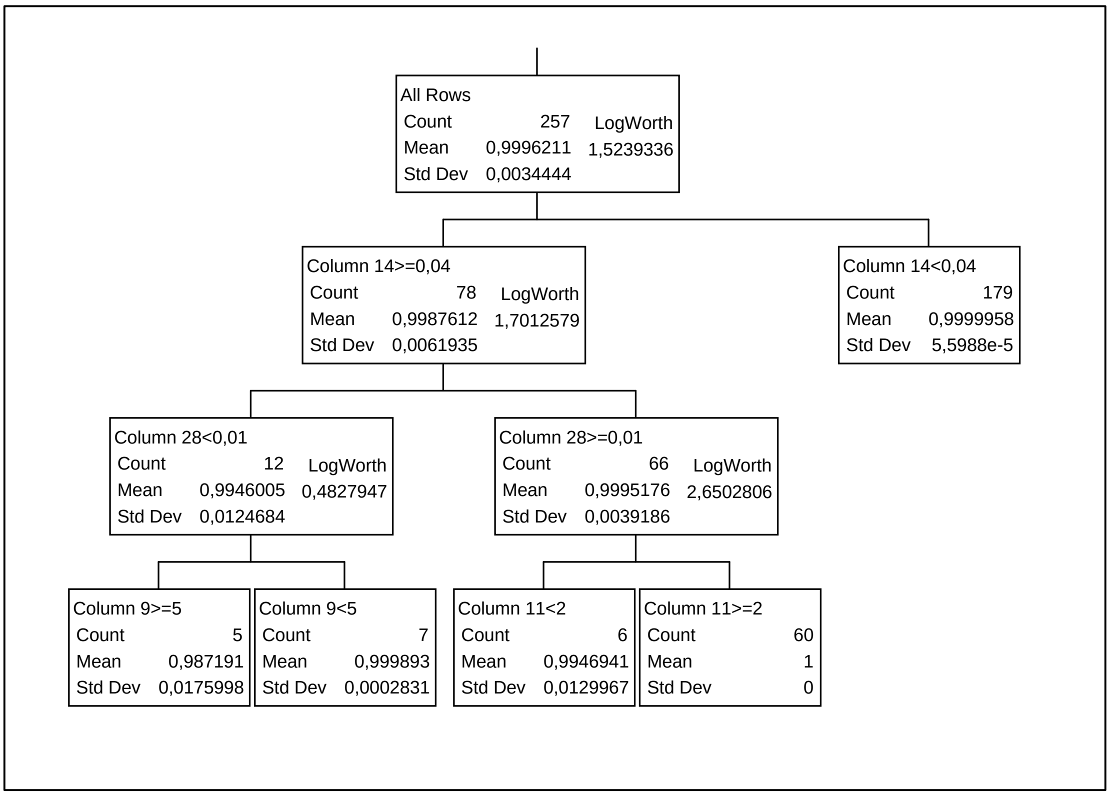


Figure 41. Regression Tree for Units B and C.

Although not a huge difference, this is due to the fact that Units C and D are co-located and share the same resources (loading docks) even though the design matrix is for deployment of Units B and C. Major factors of influence seem to be minor breakdown of trucks and medium breakdown of Armored Personnel Carriers (APCs). As for Unit D, JMP makes 43 splits of the data to achieve an R^2 of 0.417. A careful examination of splits and pruning of unnecessary and illogical leaves (i.e. factors not related to deployment of Unit D), reveals that the same R^2 value can be achieved with only 11 splits of the data.

6.4. CONCLUSIONS AND FUTURE WORK

In this dissertation, we have introduced a logistics and transportation simulation developed for use in the military DPP. We applied, for the first time, the EG methodology and LEGO framework to create a multi-modal discrete-event simulation model of the DPP. EGs and LEGOs provided a simple yet powerful and elegant way of representing DES model of deployment, and enabled easy creation of component-based models of a real-world military problem. The mostly high-resolution used allowed us to estimate whether a given plan of deployment will go as intended, and determine prospective problem areas in a relatively short time compared to other existing simulations because of the absence of the need to use several models of differing resolutions in succession. The short run times achieved demonstrated this. The very accurate and detailed GIS data, and the detailed data used in modeling of entities, resources and military equipment in the simulation permitted us not to exchange reality in favor of shorter run times. We had to extend Simkit by writing additional Java classes that are specific to military deployment. The component-based approach adopted in development of our simulation model enables us to easily integrate future additions to our model. These additions may be detailed modeling of infrastructure and resources at transfer points. Our model is generic enough to be used in commercial logistics applications after some problem-specific modifications. While some of these modifications may be minor ones, others could be major modifications depending on the domain of the problem desired to be modeled. Finally, we have simulated a real-world case study to see its robustness under urgent situations. We used a rather new and intelligent nearly-orthogonal design of experiments to further complement classical analysis techniques. Our simulation provided valuable insights as to when and what percentage of units would be at their designated destinations if the original plan had to be modified for more urgent deployment of military units. This is invaluable information for a commander since a military mission cannot be accomplished and lives could be lost if military units can not arrive at their designated destinations in a timely manner according to their plans.

As for future work, we plan on integrating our simulation software to the optimization model developed by Akgün and Tansel (2007) for use in simulation optimization applications. In addition, we consider adding web-based services for obtaining real-time or near real-time weather information and integrating this into the simulation environment. We desire to test more scenarios and will look for ways to automating the design of experiments part using NOLHS. This will be done in accordance with our on-going work in development of a detailed output analyzer module for the simulation.

APPENDIX A

Sample Java code for home base departure class is provided below. The complete code for all classes is not provided here due to space limitations and the length of the code. The requests for obtaining the entire code for research and other purposes may be considered on a case by case basis by the author and the thesis advisors of this dissertation.

```
package mipas.model;
/******************************************************************************
 * author : Ugur Ziya YILDIRIM
 * Mipas
 * date : November, 2005
 * A class that implements Home Base Departures for land movers
 * They depart after loading and move to their destination, a SPOE, a RPOE
 * or an APOE.  Upon arrival they may unload and return for multiple
 * trips if necessary.  Those land movers that do not need to load anything
 * simply form a convoy and depart home base for where they have to go after
 * the convoy size reaches a prefixed convoy size. Those that return may park at a
 * parking lot if there is no load to transport.
 ******************************************************************************/
import java.awt.Image;
import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import javax.swing.JFrame;

import mipas.gui.BlackSpotsGroup;
import mipas.gui.MIPASBusDialog;

import simkit.Schedule;
import simkit.smdx.PathMoverManager;

import com.bilgi.geokit.kernel.feature.geom.model.GKCoordinate;
import com.bilgi.geokit.kernel.registry.GKRegistryNode;
```

```
public class HomeBaseDeparture extends AbstractLocationBase {
  // mover container
  protected LinkedList moverContainer;

  //***state variables***
  //load queue at home base
  protected LinkedList loadingQueue;
  protected LinkedList busQueue;

  // number of vehicles in the loading queue
  protected int numInLoadingQueue;
  protected int numInBusQueue;

  // number of available load docks at home base
  protected int numAvailableLoadDocks;

  //total time spent in loading docks
  protected double timeInLoadingDocks;

  //delay in loading queue
  protected double delayInLoadingQueue;

  //number of breakdown according to their types
  protected int numberMinorBreakdownsFromHomebase;
  protected int numberMediumBreakdownsFromHomebase;
  protected int numberMajorBreakdownsFromHomebase;

  //time spent during breakdowns according to their types
  protected double timeInMinorBreakdownFromHomebase;
  protected double timeInMediumBreakdownFromHomebase;
  protected double timeInMajorBreakdownFromHomebase;

  //time spent in "black spots"
  protected double blackSpotsDelayTime;

  // parking lot at home base to park vehicles that return and have no load to carry.
  // Just a storage
  protected LinkedList parkingLotHomeBase;

  // ***instance variables***
  //name of home base
  private String nameHomeBase;

  // location of home base
  private Point2D homeBaseLocation;

  //number of load docks at home base for land vehicles
  private int numLoadDocks;
  private LoadStock baseStock = new LoadStock();
  private PaxStock basePax = new PaxStock();

  //***constructor***
  public HomeBaseDeparture(int nld,GKCoordinate hbl, String name) {
    setNumLoadDocks(nld);
    setHomeBaseLocation(new Point2D.Double(hbl.x,hbl.y));
    setName(name);
    moverContainer = new LinkedList();
    stock = new LoadStock();
    pStock = new PaxStock();
    loadingQueue = new LinkedList();
    busQueue = new LinkedList();
    parkingLotHomeBase = new LinkedList();
  }

  public void reset(){
    super.reset();
    numAvailableLoadDocks = numLoadDocks;
    loadingQueue.clear();
    busQueue.clear();
    numInLoadingQueue=0;
    numInBusQueue=0;
    timeInLoadingDocks=0;
    delayInLoadingQueue=0;
    numberMinorBreakdownsFromHomebase = 0;
    numberMediumBreakdownsFromHomebase = 0;
    numberMajorBreakdownsFromHomebase = 0;

    timeInMinorBreakdownFromHomebase = 0;
    timeInMediumBreakdownFromHomebase = 0;
    timeInMajorBreakdownFromHomebase = 0;

    blackSpotsDelayTime =0;

    for (Iterator iter = moverContainer.iterator(); iter.hasNext();) {
                  MipasMover element = (MipasMover) iter.next();
                  element.setLocation(new Point2D.Double(getLocation().x,getLocation().y));
                  element.reset();
    }
    parkingLotHomeBase.clear();
    stock = cloneLoadStock();
    pStock = clonePaxStock();
  }
```

```
private LoadStock cloneLoadStock() {
            LoadStock stock = new LoadStock();

            for (Iterator iter = baseStock.getContainers(); iter.hasNext();) {
                    LoadContainer lc = (LoadContainer) iter.next();
                    LoadContainer nlc = new LoadContainer(lc);
                    stock.addLoadContainer(nlc);
            }
            return stock;
  }


  private PaxStock clonePaxStock() {
            PaxStock pStock = new PaxStock();
            for (Iterator iter = basePax.getContainers(); iter.hasNext();) {
                    PaxContainer pc = (PaxContainer) iter.next();
                    PaxContainer npc = new PaxContainer(pc);
                    pStock.addPaxContainer(npc);
            }
            return pStock;
  }
  //The run method for intialization of statistics and
 // Initial population of thesimulation  event list.
  public void doRun(){
    firePropertyChange("numAvailableLoadDocks",numAvailableLoadDocks);
    firePropertyChange("numInLoadingQueue",numInLoadingQueue);
    firePropertyChange("numInBusQueue",numInBusQueue);

    for(Iterator i = moverContainer.iterator() ; i.hasNext();){
      MipasMover mover = (MipasMover)i.next();
      mover.reset();
     waitDelay("ArrivalHomeBase", 0.0, mover);
    }
  }

  //arrival of movers to home base
  public void doArrivalHomeBase(MipasMover mover){
    mover.setArrivalTime();
    if (mover.getTransferPointIndex()==-1 || !mover.getMoverPath().getFrom().equals(this)){
            mover.moveToNextTransferPoint();
    }

    //mover type cannot carry load (e.g. a tank),  1st time departure, and mover is not a bus.
    if(mover.getCarryLoad()==false &&
mover.getMoverPath().getFrom().equals(this)&&!(mover.getType().equals("Bus"))){
            double initDelay = mover.getInitialDelayAtHB();
            waitDelay("FormConvoyAtHomeBase", initDelay, mover);
      return;
    }

    //mover type is bus, there are pax, and 1st time departure
    if (mover.getType().equals("Bus")&& !(pStock.isEmpty())){
           busQueue.add(mover);
      firePropertyChange("numInBusQueue", numInBusQueue, ++numInBusQueue);
          waitDelay("StartLoadAtHomeBase", 0.0);
      return;
    }

    //mover type is bus, no pax available and 1st time departure
    if (mover.getType().equals("Bus")&&(pStock.isEmpty())){
      waitDelay("ParkVehicleAtHomeBase", 0.1, mover);
      return;
    }

    //mover can carry load (e.g. truck), there is load to carry, load docks available and 1st time departure
    if ((mover.getCarryLoad()==true)  &&(this.getNumAvailLoadDocks()>0) && !(stock.isEmpty())&&
mover.getMoverPath().getFrom().equals(this)){
      loadingQueue.add(mover);
      firePropertyChange("numInLoadingQueue", numInLoadingQueue, ++numInLoadingQueue);
      waitDelay("StartLoadAtHomeBase", 0.0);
      return;
    }
    //mover can carry load, 1st time departure but no load is available
    if ((mover.getCarryLoad()==true)&& stock.isEmpty()
&&(mover.getMoverPath().getFrom().equals(this))){
      waitDelay("ParkVehicleAtHomeBase", 0.1, mover);
      return;
    }

    //not a 1st time departure (returning), mover type is bus and no pax to carry
    if
(mover.getType().equals("Bus")&&(pStock.isEmpty()&&!(mover.getMoverPath().getFrom().equals(this))))
{
      waitDelay("ParkVehicleAtHomeBase", 0.1, mover);
      return;
    }

    //returning, mover type is a bus and pax available
    if (
(mover.getType().equals("Bus"))&&!(pStock.isEmpty())&&!(mover.getMoverPath().getFrom().equals(this)
)){
           busQueue.add(mover);
      firePropertyChange("numInBusQueue", numInBusQueue, ++numInBusQueue);
                  waitDelay("StartLoadAtHomeBase", 0.0);
      return;
    }

    //mover can carry load, returning and  no load is available
    if ((mover.getCarryLoad()==true)&& stock.isEmpty()
&&!(mover.getMoverPath().getFrom().equals(this))){
        waitDelay("ParkVehicleAtHomeBase", 0.1, mover);
        return;
    }

//    mover can carry load (e.g. truck), there is load to carry and load docks available and a returning
departure
    if ((mover.getCarryLoad()==true)  &&(this.getNumAvailLoadDocks()>0 && !stock.isEmpty()&&
!(mover.getMoverPath().getFrom().equals(this))) ){
       loadingQueue.add(mover);
       firePropertyChange("numInLoadingQueue", numInLoadingQueue, ++numInLoadingQueue);
       waitDelay("StartLoadAtHomeBase", 0.0);
       return;
    }

  }

  //parking vehicles that are not going anywhere and staying (for the moment) at home base
  public void doParkVehicleAtHomeBase (MipasMover mover){
    addParkingLotHomeBase(mover);
  }


  // start loading movers (bus or any vehicle that can carry load) at home base
  public void doStartLoadAtHomeBase(){
          if (!busQueue.isEmpty())  {
                   MipasMover mover = (MipasMover) busQueue.removeFirst();
                   if (mover.getType().equals("Bus")){
                            double paxCapacity = mover.getPaxCap();
                            while (!pStock.isEmpty() &&(paxCapacity>=0)){
         Pax pax = pStock.getPax();
         mover.addPax(pax);
         paxCapacity = paxCapacity -1;
       }
        firePropertyChange("numInBusQueue", numInBusQueue, --numInBusQueue);
              waitDelay("EndLoadAtHomeBase", mover.getLoadTime().generate(), mover);
              return;
                   }
          }

    if ((!loadingQueue.isEmpty()) ){
      MipasMover mover = (MipasMover)loadingQueue.removeFirst();
      mover.setStartServiceTime();
      firePropertyChange("delayInLoadingQueue", Schedule.getSimTime()-mover.getArrivalTime());
```

```
        firePropertyChange("numAvailableLoadDocks", numAvailableLoadDocks,--
numAvailableLoadDocks);
        firePropertyChange("numInLoadingQueue", numInLoadingQueue, --numInLoadingQueue);

        double weight = mover.getWeightCap();
        double vol = mover.getVolumeCap();

        while (!stock.isEmpty() &&(vol >= mover.getEmptyVolPercentage()*mover.getVolumeCap()) &&
              (weight >= mover.getEmptyWeightPercentage()*mover.getWeightCap())){
          Load load = stock.getLoad();
          mover.addLoad(load);
          vol = vol - load.getVolume();
          weight = weight - load.getWeight();
         }
        waitDelay("EndLoadAtHomeBase", mover.getLoadTime().generate(), mover);
      }
    }

    // end loading movers at home base
    public void doEndLoadAtHomeBase(MipasMover mover){

      firePropertyChange("numAvailableLoadDocks",
numAvailableLoadDocks,++numAvailableLoadDocks);
      firePropertyChange("timeInLoadingDocks", Schedule.getSimTime() - mover.getStartServiceTime());

      if((loadingQueue.size() > 0)&&(!stock.isEmpty())&&(this.getNumAvailLoadDocks()>0)){
        waitDelay("StartLoadAtHomeBase", 0.0);
      }
      if((!busQueue.isEmpty())&& !(pStock.isEmpty())){
         waitDelay("StartLoadAtHomeBase", 0.0);
       }

      double initialDelay = mover.getInitialDelayAtHB();
      waitDelay("FormConvoyAtHomeBase", initialDelay, mover);
    }

    //movers form convoys departing for destination or spoe/rpoe/apoe
    //when appropriate convoy size is reached, convoy starts departing home base
    public synchronized void doFormConvoyAtHomeBase (MipasMover mover) {

              LinkedList convoy = getConvoy(mover.getMoverPath());
              if (!convoy.contains(mover))
                        convoy.add(mover);

      if (getConvoySize() == convoy.size()) {
                        LinkedList list = new LinkedList();
                        list.addAll(convoy);
                        convoy.clear();
                        Convoy cv = new Convoy(list);
                        waitDelay("DepartHomeBase", 0.25, cv);
      }
    }
    //depart as a convoy from the home base
    public void doDepartHomeBase(Convoy convoy) {
              Iterator movers = convoy.movers();
              double pMinor;
              double pMed;
              double pMajor;
              double minorDelayTime;
              double medDelayTime;
              double majorDelayTime;
              double maxDelayTime = 0;

              double maintenanceStartTime = 0;
              double rand = Math.random();

              while(movers.hasNext()) {
                        MipasMover m = (MipasMover)movers.next();

                        pMinor = m.getMinorDelayProbability().generate();

                        pMed = m.getMediumDelayProbability().generate();
                        pMajor = m.getMajorDelayProbability().generate();

                                    if (pMinor >= rand) {
                                                firePropertyChange("numberMinorBreakdownsFromHomebase",
++numberMinorBreakdownsFromHomebase);
                                                minorDelayTime = m.getMinorDelayTime().generate();

          firePropertyChange("timeInMinorBreakdownFromHomebase",minorDelayTime);
                                                maxDelayTime = Math.max(maxDelayTime, minorDelayTime);
                                    }
                                    if (pMed >= rand) {
                                                firePropertyChange("numberMediumBreakdownsFromHomebase",
++numberMediumBreakdownsFromHomebase);
                                                medDelayTime = m.getMediumDelayTime().generate();
          firePropertyChange("timeInMediumBreakdownFromHomebase",medDelayTime);
                                                maxDelayTime = Math.max(maxDelayTime, medDelayTime);
                                    }
                                    if (pMajor >= rand) {
                                                firePropertyChange("numberMajorBreakdownsFromHomebase",
++numberMajorBreakdownsFromHomebase);
                                                majorDelayTime = m.getMajorDelayTime().generate();

          firePropertyChange("timeInMajorBreakdownFromHomebase",majorDelayTime);
                                                maxDelayTime = Math.max(maxDelayTime, majorDelayTime);
                          }
              }
              double blackSpotsDelay = MipasSimulation.getBlackSpotDelayTime(
                      convoy.getMoverPath(), BlackSpotsGroup.ACTIVE_DISTANCE);
              firePropertyChange("blackSpotsDelayTime", blackSpotsDelayTime + blackSpotsDelay);
              maxDelayTime += blackSpotsDelay;

              if (maxDelayTime == 0) { // no delay
                        for (Iterator iter = convoy.movers(); iter.hasNext();) {
                                MipasMover mover = (MipasMover) iter.next();
                                mover.setSpeed(convoy.getSpeed());
                          waitDelay("DepartHomeBase", 0.5,mover);
                        }
              } else { // maintenance delay with maxDelayTime after maintenance start time
                          MipasPath p = convoy.getMoverPath();
            double totalDistance =
LandArrivalDestination.getTotalDistance(convoy.getLocation(),convoy.getMoverPath().getPath());
            double travelTime = (totalDistance/convoy.getSpeed());// need to make stochastic

                        maintenanceStartTime = Math.random()*travelTime;
                        ArrayList moverManagers = new ArrayList();
                        for (Iterator iter = convoy.movers(); iter.hasNext();) {
                                    MipasMover mover = (MipasMover) iter.next();
                        PathMoverManager myManager = new PathMoverManager (mover,p.getPath());
                        myManager.start();
                        moverManagers.add(myManager);
                        }

                      waitDelay(p.getTo().getArrivalMaintenanceDelay(),
"MaintenanceDelayBegin",maintenanceStartTime,new Object[]{convoy,new
Double(maxDelayTime),moverManagers});
              }
    }
    // movers depart home base
    public void doDepartHomeBase(MipasMover mover) {
      MipasPath mipasPath = mover.getMoverPath();


      ILocationBase base = mipasPath.getTo();
      if (base instanceof HomeBaseDeparture) {
         double totalDistance=0;
         totalDistance =
LandArrivalDestination.getTotalDistance(mover.getLocation(),mipasPath.getPath());
         double travelTime = (totalDistance/mover.getSpeed());
         PathMoverManager myManager = new PathMoverManager (mover,mipasPath.getPath());
         myManager.start();
         firePropertyChange("TotalDistance", totalDistance);
        firePropertyChange("TravelTime", travelTime);
waitDelay(mover.getMoverPath().getTo(),"ArrivalHomeBase",travelTime,myManager.getMover());
    }
    if (base instanceof LandArrivalDestination) {
        double totalDistance=0;
        totalDistance =
LandArrivalDestination.getTotalDistance(mover.getLocation(),mipasPath.getPath());
        double travelTime = (totalDistance/mover.getSpeed());
        PathMoverManager myManager = new PathMoverManager (mover,mipasPath.getPath());
        myManager.start();

        firePropertyChange("TotalDistance", totalDistance);
        firePropertyChange("TravelTime", travelTime);

waitDelay(mover.getMoverPath().getTo(),"LandArrivalAtDestination",travelTime,myManager.getMover(
));
    }
    if (base instanceof LandArrivalAtSPOE) {
        double totalDistance=0;
        totalDistance =
LandArrivalDestination.getTotalDistance(mover.getLocation(),mipasPath.getPath());
        double travelTime = (totalDistance/mover.getSpeed());
        PathMoverManager myManager = new PathMoverManager (mover,mipasPath.getPath());
        myManager.start();
        firePropertyChange("TotalDistance", totalDistance);
        firePropertyChange("TravelTime", travelTime);
waitDelay(mover.getMoverPath().getTo(),"LandArrivalAtSPOE",travelTime,myManager.getMover());
    }

    if (base instanceof LandArrivalAtRPOE) {
        double totalDistance=0;
        totalDistance =
LandArrivalDestination.getTotalDistance(mover.getLocation(),mipasPath.getPath());
        double travelTime = (totalDistance/mover.getSpeed());
        PathMoverManager myManager = new PathMoverManager (mover,mipasPath.getPath());
        myManager.start();
        firePropertyChange("TotalDistance", totalDistance);
        firePropertyChange("TravelTime", travelTime);

waitDelay(mover.getMoverPath().getTo(),"LandArrivalAtRPOE",travelTime,myManager.getMover());
    }
    if (base instanceof LandArrivalAtAPOE) {
        double totalDistance=0;
        totalDistance =
LandArrivalDestination.getTotalDistance(mover.getLocation(),mipasPath.getPath());
        double travelTime = (totalDistance/mover.getSpeed());
        PathMoverManager myManager = new PathMoverManager (mover,mipasPath.getPath());
        myManager.start();
        firePropertyChange("TotalDistance", totalDistance);
        firePropertyChange("TravelTime", travelTime);
waitDelay(mover.getMoverPath().getTo(),"LandArrivalAtAPOE",travelTime,myManager.getMover());
    }
  }

  //***setters and getters***
    public HomeBaseDeparture getHomeBase(){
    return this;
  }

  public LinkedList getLoadingQueue(){
    return (LinkedList)loadingQueue.clone();
  }

  public int getNumAvailLoadDocks(){
    return numAvailableLoadDocks;
  }

  public void setNumLoadDocks(int nld) {
    numLoadDocks = nld;
  }

  public int getNumLoadDocks() {
    return numLoadDocks;
  }

  public void setHomeBaseLocation(Point2D hbl) {
    homeBaseLocation = hbl;
  }

  public Point2D getHomeBaseLocation() {
    return homeBaseLocation;
  }

  public void setName(String nhb) {
    nameHomeBase = nhb;
  }

  public String getName() {
    return nameHomeBase;
  }

  // methods related to containers
  public void addMover(MipasMover mover){
    moverContainer.add(mover);
  }
```

```
public void addParkingLotHomeBase(MipasMover mover){
  parkingLotHomeBase.add(mover);
}

 public String getBaseName() {
    return this.nameHomeBase;
 }

 public GKCoordinate getLocation() {
    return new GKCoordinate(this.homeBaseLocation.getX(),homeBaseLocation.getY());
 }

 public double getVolumeCapacity() {
    return 0;
 }

 public double getWeightCapacity() {
    return 0;
 }

 public Image getSymbol() {
    return null;
 }

 public List getMoverContainer() {
    return (LinkedList)moverContainer.clone();
 }


 public GKRegistryNode getRegistryNode() {
    GKRegistryNode root
       = new GKRegistryNode("HomeBaseDeparture");

    GKRegistryNode node = new GKRegistryNode("NameHomeBase");
    node.setValue(this.nameHomeBase);
    root.addChild(node);

    node = new GKRegistryNode("LoadStock");
    for (Iterator it = this.baseStock.getContainers(); it.hasNext(); ) {
       LoadContainer loadContainer = ((LoadContainer)it.next());

       if (loadContainer != null) {
          node.addChild(loadContainer.getRegistryNode());
       }
    }
    root.addChild(node);
    node = new GKRegistryNode("PaxStock");
    for (Iterator it = this.basePax.getContainers(); it.hasNext(); ) {
        PaxContainer paxContainer = ((PaxContainer)it.next());

        if (paxContainer != null) {
            node.addChild(paxContainer.getRegistryNode());
        }
    }
    root.addChild(node);

    if (this.homeBaseLocation != null) {
        node = new GKRegistryNode("HomeBaseLocation");
        GKRegistryNode ele = new GKRegistryNode("X");
        ele.setValue(this.homeBaseLocation.getX() + "");
        node.addChild(ele);
        ele = new GKRegistryNode("Y");
        ele.setValue(this.homeBaseLocation.getY() + "");
        node.addChild(ele);
        root.addChild(node);
    }

    node = new GKRegistryNode("NumLoadDocks");
    node.setValue(this.numLoadDocks + "");
    root.addChild(node);

    node = new GKRegistryNode("ConvoySize");
    node.setValue(this.convoySize + "");
    root.addChild(node);
    return root;
}

public void loadFromRegistry(GKRegistryNode root) {
    GKRegistryNode node = root.getChild("NameHomeBase");
    if (node != null) {
        this.nameHomeBase = node.getValue();
    }
    node = root.getChild("LoadStock");
    if (node != null) {
        this.baseStock = new LoadStock();
        for (Enumeration enm = node.getChildren(); enm.hasMoreElements(); ) {
            GKRegistryNode ele = ((GKRegistryNode)enm.nextElement());
            if (ele != null) {
                LoadContainer loadContainer = new LoadContainer();
                loadContainer.loadFromRegistry(ele);
                this.baseStock.addLoadContainer(loadContainer);
            }
        }
    }
    node = root.getChild("PaxStock");
    if (node != null) {
        this.basePax = new PaxStock();
        for (Enumeration enm = node.getChildren(); enm.hasMoreElements(); ) {
            GKRegistryNode ele = ((GKRegistryNode)enm.nextElement());

            if (ele != null) {
                PaxContainer paxContainer = new PaxContainer();
                paxContainer.loadFromRegistry(ele);
                this.basePax.addPaxContainer(paxContainer);
            }
        }
    }

    node = root.getChild("HomeBaseLocation");
    if (node != null) {
        double x = Double.parseDouble(node.getChild("X").getValue());
        double y = Double.parseDouble(node.getChild("Y").getValue());

        setHomeBaseLocation(new Point2D.Double(x, y));
    }

    node = root.getChild("NumLoadDocks");
    if (node != null) {
        setNumLoadDocks(Integer.parseInt(node.getValue()));
    }
    node = root.getChild("ConvoySize");
    if (node != null) {
        setConvoySize(Integer.parseInt(node.getValue()));
    }
}
public void addLoad(Load load) {
        baseStock.addLoad(load);
}

public void addPax(Pax pax) {
        basePax.addPax(pax);
}
protected MaintenanceDelay createMaintenanceDelay() {
                return new MaintenanceDelay(this,"ArrivalHomeBase");
        }
//minor breakdown stats
        public int getNumberMinorBreakdownsFromHomebase() {
                return numberMinorBreakdownsFromHomebase;
        }
```

```
//minor breakdown duration
public double getTimeInMinorBreakdownFromHomebase() {
        return timeInMinorBreakdownFromHomebase;
}

//medium breakown stats
public int getNumberMediumBreakdownsFromHomebase() {
        return numberMediumBreakdownsFromHomebase;
}

//medium breakdown duration
public double getTimeInMediumBreakdownFromHomebase() {
        return timeInMediumBreakdownFromHomebase;
}

//major breakdown stats
public int getNumberMajorBreakdownsFromHomebase() {
        return numberMajorBreakdownsFromHomebase;
}

//major breakdown duration
public double getTimeInMajorBreakdownFromHomebase() {
        return timeInMajorBreakdownFromHomebase;
}

public double getBlackSpotsDelayTime() {
        return blackSpotsDelayTime;
}

public LinkedList getBusQueue() {
        return busQueue;
}

public double getDelayInLoadingQueue() {
        return delayInLoadingQueue;
}

public String getNameHomeBase() {
        return nameHomeBase;
}

public int getNumAvailableLoadDocks() {
        return numAvailableLoadDocks;
}

public int getNumInBusQueue() {
        return numInBusQueue;
}
```

```
	public int getNumInLoadingQueue() {
		return numInLoadingQueue;
	}
	public LinkedList getParkingLotHomeBase() {
		return parkingLotHomeBase;
	}
	public double getTimeInLoadingDocks() {
		return timeInLoadingDocks;
	}
	//test
	public static void main(String[] args) {
		JFrame frm = new JFrame();
		Bus bus = new Bus();
		MIPASBusDialog dlg = new MIPASBusDialog(frm,new ArrayList());
		dlg.showDialog(frm);
		System.out.println(bus.getLoadTime().getParameters());
		Object[] parameters = bus.getLoadTime().getParameters();
		for (int i = 0; i < parameters.length; i++) {
			System.out.println("Param = "+parameters[i]);
		}
	}
}
```

REFERENCES

Akgün, İ. and Tansel, B.Ç. "Optimization of Transportation Requirements in the Deployment of Military Units." Computers and Operations Research, Volume 34, Issue 4, p. 1158 – 1176, April 2007.

Balcı, Osman. Verification, Validation, and Testing. Handbook of Simulation. John Wiley & Sons, Inc., 1998. p. 335 – 393.

Boukhtouta, A., Bedrouni, A., Berger, J., Bouak, F., and Guitouni, A. "A Survey of Military Planning Systems." 2004. Available via <http://www.dodccrp.org/events/9th_ICCRTS/CD/papers/096.pdf> [accessed December 15, 2007].

Burke, J. F., Love, R.J., Macal, C.M., Howard, D.L., and Jackson, J. "Modeling Force Deployment from Army Installations Using the Transportation System Capability (TRANSCAP) Model." Last accessed at http://www.osti.gov/bridge/servlets/purl/757530-YBfenI/webviewable/757530.pdf on 15 Dec 2006.

Buss, Arnold. "Discrete Event Programming With Simkit". Published in Simulation News Europe, 2001.

Buss, Arnold H. and Sanchez, Paul J. "Building Complex Models With LEGOS (Listener Event Graph Objects)". Proceedings of the 2002 Winter Simulation Conference, E. Yücesan, C-H. Chen, J.L.Snowdon, and J.M.Charnes, eds.

Buss, Arnold. "Component Based Simulation Modeling With Simkit". Proceedings of the 2002 Winter Simulation Conference, E. Yücesan, C-H. Chen, J.L.Snowdon, and J.M.Charnes, eds.

Buss, Arnold and Ruck, John. "Joint Modeling and Analysis Using XMSF Web Services". Proceedings of the 2004 Winter Simulation Conference, R.G. Ingalls, M.D. Rossetti, J.S.Smith, and B.A. Peters, eds.

Chapman, Stephen J. "Java for Engineers and Scientists". Prentice-Hall Inc., 2000.

Cioppa, T. M., and Lucas, T. W. 2007. Efficient Nearly Orthogonal and Space Filling Latin Hypercubes. *Technometrics* 49(1):.

Cornell, Gary and Horstmann, Cay S. “Core Java 1.2 Volume I – Fundamentals.”. Sun Microsystems Press. A Prentice Hall Title, 1999.

Cornell, Gary and Horstmann, Cay S. “Core Java 2 Volume II – Advanced Features.”. Sun Microsystems Press. A Prentice Hall Title, 2000.

Field Manual 4.01-011, *Unit Movement Operations,* Headquarters, Department of The Army, Washington, DC, October 2002.

Field Manual 55-10, *Movement Control,* Headquarters, Department of The Army, Washington, DC, 9 February 1999.

Field Manual 55-65, *Strategic Deployment,* Headquarters, Department of The Army, Washington, DC, 3 October 1995.

Field Manual 100-17, *Mobilization, Deployment, Redeployment, and Demobilization,* Headquarters, Department of The Army, Washington, DC, 28 October 1992.

Field Manual 100-17-3, *Reception, Staging, Onward Movement, and Integration,* Headquarters, Department of The Army, Washington, DC, 17 March 1999.

GeoKIT User Manual. Contained in the GeoKIT Showcase CD as a pdf file. Courtesy of BilgiGIS.

GeoKIT v2 Tutorial. Contained in the GeoKIT Showcase CD as a pdf file. Courtesy of BilgiGIS.

Goldsman, David., Pernet, Sebastien and Kang, Keebom. “Simulation of Transportation Logistics”. Proceedings of the 2002 Winter Simulation Conference, E. Yücesan, C-H. Chen, J.L.Snowdon, and J.M.Charnes, eds.

Groningen, C.N.V., Blachowicz, D., Braun, M.D., Love, R.J., Simunich, K.L. and Widing, M.A. “Modeling Military Deployment in Theaters of Operations – Balancing Deployment Alternatives.”2005. Available via <http://www.dis.anl.gov/temps/LogisticsSpectrum.ELIST.MergedFigures.8Aug05.pdf> [accessed December 15, 2006].

Heal, G.N., Garnett, I.K., *Allied Deployment and Movement System (ADAMS) Version 3.0 Tutorial*, Technical Note 669, NATO Consultation, Command and Control Agency, The Hague, September 2001.

Howard, D.L., Bragen, M.J., Burke, J.F. Jr., and Love, R.J. "PORTSIM 5: Modeling from a Seaport Level". Mathematical and Computer Modeling, 2004.

JFAST 7.2 Handbook. "Joint Flow and Analysis System for Transportation". http://www.jfast.org. November 15, 2000.

Joines, Jeffrey A. and Roberts, Stephen D. "Simulation in an Object-Oriented World". Proceedings of the 1999 Winter Simulation Conference, 1999.

Jose, Angel San. "Analysis, Design, Implementation and Evaluation of Graphical Design Tool To Develop Discrete Event Simulation Models Using Event Graphs and Simkit." MSc. Thesis done at the Naval Postgraduate School, Monterey, Ca. September, 2001.

Kang, K., Doerr, K. H., and Sanchez, S. M. 2006. A Design of Experiments Approach to Readiness Risk Analysis. *Proceedings of the 2006 Winter Simulation Conference*, ed., L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 1332-1339.Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Kleijnen, J. P. C., Sanchez, S. M., Lucas, T. W., and Cioppa, T. M. 2005. A User's Guide to the Brave New World of Simulation Experiments. INFORMS Journal on Computing, 17(3): 263-289.

Krause, L.C. and Parsons, David J. "Tactical Logistics and Distribution System (TLOADS) Simulation". Proceedings of the 1999 Winter Simulation Conference.

Lewis, John and Loftus, William. "Java Software Solutions – Foundations of Program Design". 2nd Ed. Addison-Wesley, 2000.

Logistics Federates Homepage:
http://ms.ie.org/SIW_LOG/Log_Federates/FederateTable.shtml

McKinzie, K. and Barnes, W. J. "A Review of Strategic Mobility Models Supporting the Defense Transportation System." 2003. Available via <http://www.me.utexas.edu/~barnes/research/files/update_06_2003/RSMMSDTS.pdf> [accessed December 15, 2006].

Nance, Richard E. and Rioux, Glenn P. "Generalizing: Is It Possible To Create All-Purpose Simulations?" Proceedings of the 2002 Winter Simulation Conference, E. Yücesan, C-H. Chen, J.L.Snowdon, and J.M.Charnes, eds.

Narasimhan, Sundar and Winston, Patrick H. "On To Java 2. Third edition." . Addison-Wesley, 2001.

Sanchez, S. M. and Hernandez, A. S. 2005. NOLHdesigns Spreadsheet. Available online via <http://diana.cs.nps.navy.mil/SeedLab/> [accessed 12/19/2007].

Sargent, Robert G. "Event Graph Modelling for Simulation with an Application to Flexible Manufacturing Systems." Management Science 34(10). Pp.1231-1251, 1988.

Sargent, Robert G. "Validation and Verification of Simulation Models." Proceedings of the 1999 Winter Simulation Conference. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

Sargent, Robert G., Glasgow, P.A., Kleijnen Jack P.C., Law, Averill M., McGregor, Ian and Youngblood, Simone. "Strategic Directions in Verification, Validation and Accreditation Research". Proceedings of the 2000 Winter Simulation Conference.

Sargent, Robert G. "Some Approaches and Paradigms for Verifying and Validating Simulation Models." Proceedings of the 2001 Winter Simulation Conference. B. A. Peters, J.S. Smith, D.J.Mederios, and M.W.Rohrer, eds.

SAS. 2005. JMP User's Guide, Version 6: SAS Institute Inc. Cary, NC, USA.

Schruben, Lee. "Simulation Modeling with Event Graphs". Communications of the ACM 26(11):957-963, 1983.

Schruben, Lee. "Building Reusable Simulators Using Hierarchical Event Graphs". Winter Simulation Conference pp.472-475, 1995.

Seila, A.F., Ceric, V. and Tadikamalla, P. "Applied Simulation Modelling". Belmant, California: Brooks – Cole, 2003.

SIMULOGS (Simulation of Logistics Systems). Last accessed at http://ms.ie.org/SIW_LOG/Log_Federates/SIMULOGS.ppt on 15 Dec 2006.

Ye, K.Q. 1998. “Orthogonal Column Latin Hypercubes and Their Applications in Computer Experiments.” Journal of the American Statistical Association, 93: 1430-1439.

Yıldırım, U. Z., Sabuncuoğlu, İ. and Tansel, B. Ç. “A Simulation Model for Military Deployment.” *Proceedings of the 2007 Winter Simulation Conference,* eds., S.G.Henderson, B.Biller, M.-H.Hsieh, J.Shortle, J.D.Tew, and R.R.Barton, 2007.