T.R.
TURKISH NAVAL ACADEMY
NAVAL SCIENCE AND ENGINEERING INSTITUTE
DEPARTMENT OF OPERATIONS RESEARCH

A SIMULATION MODEL FOR A GENERIC UNIT-LOAD WAREHOUSE

A MASTER THESIS

ARDA CEYLAN

Advisor: Asst. Prof. M.Murat Günal

İSTANBUL, 2011

T.R.
TURKISH NAVAL ACADEMY
NAVAL SCIENCE AND ENGINEERING INSTITUTE
DEPARTMENT OF OPERATIONS RESEARCH

A SIMULATION MODEL FOR A GENERIC UNIT-LOAD WAREHOUSE

A MASTER THESIS

ARDA CEYLAN

Advisor: Asst. Prof. M.Murat Günal

İSTANBUL, 2011

A SIMULATION MODEL FOR A GENERIC UNIT-LOAD WAREHOUSE

ARDA CEYLAN

Submitted in partial fulfillment of the requirement for degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

Turkish Naval Academy
Naval Science and Engineering Institute

Author:----------
Arda Ceylan

Defence Date: 13th June 2011

Approved by:

Asst. Prof. M.Murat Günal (Thesis Advisor)

Prof. Agostino Bruzzone (Defense Committee Member)

Asst. Prof. A. Özgür Toy (Defense Committee Member)

Assoc. Prof. Bülent Çatay (Defense Committee Member)

Asst. Prof. Gürdal Ertek (Defense Committee Member)

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to the following persons who have made the completion of this thesis possible:

My lovely precious wife Francesca, for her everlasting patience, continuous support and encouragement, my beloved daughter Noemi, for being our sunshine and the meaning of our life, my dearest parents, for their eternal love and affection to me and my unique sister Ahu, for her invisible struggle and moral support,

My distinguished thesis advisor, Asst.Prof. M.Murat Günal, for inspiration, motivation and sage guidance,

Prof.Agostino Bruzzone, *per la sua professionalitá, disponibilitá e i preziosi consigli,*

My academic advisor, Asst.Prof. A.Özgür TOY, for his devotion and insightful criticism,

Asst.Prof. Gürdal Ertek, for his invaluable assistance and enlightening remarks,

Asst.Prof. Hakan Tozan, Asst.Prof. Serol Bulkan and Asst.Prof. Alp Üstündağ for their exceptional contributions to my scientific development,

Assoc.Prof. Bülent Çatay for his worthy comments as a committee member,

Dr. Mustafa KARADENİZ, for his collaboration and understanding,

Dr. Kadir Alpaslan Demir, for his help throughout my study,

Cenk Şentürk and Hümeyra İslam, for their efforts on administrative issues,

and finally my colleagues Mehmet Tezcan and Latif Yanar for their sincere fellowship.

Güler yüzlü güzel insan, sevgili babam Kâmil Ceylân'ın aziz hatırasına…

DISCLAIMER STATEMENT

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Turkish Naval Forces, Turkish Naval Academy and Naval Science and Engineering Institute.

PALET BAZLI ÇALIŞAN JENERİK BİR DEPONUN BENZETİM MODELİ

Arda Ceylan

Harekat Analizi Yüksek Lisans Tezi, 2011

Danışman: Yrd.Doç.Dr. M.Murat GÜNAL

Anahtar Kelimeler: Depo, Benzetim, Olay Çizelgeleme

Depolama tedarik zincirinin en önemli halkalarından birini oluşturmaktadır. Bu alanda gerçekleştirilen çalışmalarda analitik ve benzetim odaklı yöneylem araştırması teknikleri yaygın olarak kullanılmaktadır. Doğası gereği karmaşık bir yapıya sahip olan depo sistemlerinin analitik yöntemler ile modellenmesinde ileri seviyede sistem detayına girilememekte, buna karşın benzetim ile karmaşık depo sistemlerinin modellenmesi mümkün olmaktadır.

Bu çalışmada palet bazlı olarak çalışan depolarla ilgili kaynak tahsis, yerleşim planlama ve depolama politikaları ile ilgili problemlerin incelemesine yönelik bir analiz yazılımı geliştirilmesi hedeflenmiştir. Bu maksatla, öncelikle, olay çizelgeleme yaklaşımını temel alarak SharpSim adı verilen bir benzetim kütüphanesi, müteakiben genel bir depoya ilişkin nesnel yapıyı bünyesinde barındıracak bir depo kütüphanesi (WareLib) oluşturulmuştur. Daha sonra olay çizgeleri yöntemi ile ana

depo fonksiyonlarını içeren genel bir kavramsal model geliştirilmiş ve bu kavramsal model, ilk aşamada geliştirilen benzetim ve depo kütüphaneleri kullanılarak bir benzetim modeline dönüştürülmüştür. Model ile sunulan arayüz, 3D animasyon ve istatistikî yapı özellikleri ile araştırmacılarla uygulamacılar arasındaki karşılıklı iletişimin artırılması amaçlanmıştır. Çalışmanın son bölümünde, depo mal giriş ve çıkış kapısı yerleşimlerinin forklift kullanım oranları üzerindeki etkilerinin ölçülmesine ilişkin bir analiz uygulaması yapılmıştır.

A SIMULATION MODEL FOR

A GENERIC UNIT-LOAD WAREHOUSE

Arda Ceylan

Operations Research, Master of Science Thesis, 2011

Advisor: Asst.Prof. M.Murat Günal

Key Words: Warehouse, Simulation, Event Scheduling

Warehousing is one of the most important elements of a supply chain. It is common to use analytical and simulation-based operations research techniques in the studies held on this topic. Warehousing systems are complex in their nature and analytical techniques lack modeling high level of system detail. On the other hand, simulation is more convenient to model complex warehouse systems with high level of detail.

The objective of the thesis is to develop a general analysis tool for unit-load warehouses to study some warehousing problems such as resource allocation, layout design and storage policies. For this reason, first we have developed a general purpose simulation library (SharpSim) adopting Event Scheduling approach, and then we have created a warehouse library (WareLib) which maintains the data structure required to simulate a warehouse. We have, then, developed a general

conceptual model which includes main warehouse functions and this conceptual model is implemented by using simulation and warehouse libraries created in previous stages. The analysis tool provides an interface, 3D animation and some statistical features by which we hope to contribute to communication among researchers and practitioners. At the final phase of the study, we have analyzed the effects of dock positioning on forklift utilizations.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODEBOXES

ABBREVIATIONS

DES : Discrete Event Simulation

ES : Event Scheduling

EGs : Event Graphs

FEL : Future Event List

SKU : Stock Keeping Unit

JIT : Just In Time

FCFS : First Come First Serve

FIFO : First In First Out

COI : Cube per Order Index

IFU : Inbound Forklift Utilization

OFU : Outbound Forklift Utilization

CA : Crossdocking Adopted

CNA : Crossdocking Not Adopted

1 INTRODUCTION

1.1 Overview

In today's highly competitive markets, the use of Operations Research (OR) methods which makes improvements in Supply Chain Systems (SCSs) has much importance than it has in the past. Improvements in any stage of SCSs have individual and whole system effects. In this context, warehouses are accepted as one of the key elements which have crucial impacts on SCSs. Plenty of researches in the literature accordingly have sought ways to improve warehousing systems.

In this thesis, we aimed at developing a general analysis tool for unit-load warehouses operating with single putaway/picking policies to study some warehousing problems related with resource allocation, layout design and storage policies. We have used discrete event simulation (DES) method and event scheduling (ES) approach in our study. The reasons behind the decision of adopting DES and ES will be explained in detail later in this chapter.

We developed our model in three phases. First, we have developed a general purpose simulation library (SharpSim). SharpSim is a DES library written in Visual CSharp (C#) and created to implement Event Graph models. By creating our own simulation library, we have gained total control of the software to be developed at the first phase. This library has provided us an environment to implement our model.

Next, we have developed a warehouse library (WareLib) that maintains the basic warehouse objects and data structures required to simulate a warehouse. This library provides us a generic virtual warehouse for our analysis tool. Furthermore, WareLib provides some 3D animation features for the models referring to it.

Later, we have developed a general conceptual model for a unit-load warehouse operating with single putaway/picking policy. This conceptual model has been implemented by the use of SharpSim and WareLib. In addition to being a foundation for the analysis tool, we believe that the conceptual model can also be used as a template by modelers who intend to develop conceptual warehouse simulation models adopting ES approach. Modelers can simply generate adaptive conceptual models using this template. The model includes (1) receiving, (2) putaway, (3) crossdocking, (4) picking and (5) shipping functions. Finally, we have implemented our conceptual model using SharpSim and WareLib libraries and created our analysis tool.

1.2 Objectives

The objective of the thesis is to develop a general analysis tool for unit-load warehouses operating with single putaway/picking policies to study some warehousing problems related with resource allocation, layout design and storage policies. To reach our objective, we have determined a set of sub-objectives for the thesis. These sub-objectives are; first, to develop a general purpose discrete event simulation library, next, to create a generic warehouse library and finally, to develop a general component based conceptual model for unit-load warehouses and implement this conceptual model to create the analysis tool by the use of simulation and warehouse libraries.

Problems to be studied with the model can be categorized into three main groups. These are resource allocation problems, storage policy problems and layout design problems. Resource allocation problems include manual tasks (e.g. inspection of truck load, pre-receiving operations) and forklift tasks (e.g. putaway and picking

functions). For example, the number of resources required to satisfy system needs can be determined for a given input set (layout, item profiles, demand profiles etc.) with the model. Second group includes the problems related with random and dedicated storage policies. These policies can be implemented with the model and thus can be compared with each other. The effects of these policies on system performance criteria can also be measured. Finally, warehouse layout problems such as sizing and dimensioning, aisle orientation, determining required number of storage locations etc., can be studied with the model. The layout is a user input for the model and represented with a set of spreadsheets in a very detailed manner. The detail level includes the position of each storage locations, positions of docks and aisles. Furthermore, all these problem types can be examined together with the model.

On the other hand, the model has some limitations. The model is based on single command policy (single putaway/picking policy). Putaway and picking transactions are executed separately. Furthermore, only single deep racks are used in the model.

1.3 Structure

This thesis has six chapters. This first chapter is an introductory chapter including an overview of the thesis, objectives, and a review of warehouse systems, tools and the literature. In the second chapter, we explain the specifics of SharpSim which is a general purpose DES library and show how to implement a simple event graph model with SharpSim. In the third chapter, WareLib, the virtual data structure required to simulate a warehouse is introduced. In the fourth chapter, we present our general conceptual model for a unit-load warehouse operating with single putaway/picking policy and its implementation. In the fifth chapter, we made some experiments to measure the effects of dock positioning on forklift utilizations by

using our model. The last chapter includes the conclusions of the thesis.

1.4 Warehouse Systems

Warehouse is a facility where commodities are stored. It can be defined as a buffer in a supply chain system. The presence of such a buffer is required because of variances in supply chain systems. Frazelle (2002) indicates that in spite of arising initiatives such as e-commerce, supply chain integration and just-in-time (JIT) delivery, imbalances in SC systems will always be experienced and this will naturally bring the concept of warehousing forth.

There are a variety of warehouse types. These are raw material warehouses, semi-finished product warehouses, finished product warehouses, distribution centers, local warehouses etc. Warehouses may vary also in procedures, however most warehouses share some operations, literally receiving, putaway, picking and shipping operations. Receiving includes the receipt of incoming material into warehouse, the control of quantity/quality and distributing of materials to other functions. Putaway is to carry and place the materials into storage locations. Picking is to collect materials from storage locations and bring them into shipping area to satisfy demands. Finally shipping includes control of quantity/quality and loading materials into trucks. Another popular function, crossdocking, is used to satisfy demands directly from offloading area to incur less storage cost.

To create and maintain a world-class warehousing and material handling system, Frazelle (2002) suggests some set of principles. These are “identifying causes of bottlenecks and melioration opportunities by analyzing order and item profiles”, “determining performance gaps by benchmarking warehouse practices with world-class standards”, “reconfiguring processes by focusing on material and information

handling activities, computerizing and mechanizing the system to improve warehouse throughput" and finally "setting goals and implementing ergonomic improvements in manual activities". This set of principles which provide a better understanding of warehousing systems guides to improvement in warehouses.

The last word is about unit-load warehouses. These warehouses store pallets in storage locations. Third-party transshipment warehouses, beverage and grocery distributors, and appliance manufacturers are given as examples of unit-load warehouses in Pohl et al. (2008). It is also indicated in the study that other warehouses working with less than unit-load units has some unit-load level activities.

1.5 Simulation

Simulation is one of the many methods to study a system. Law et al. (2000) decomposes the ways to study a system into two at the uppermost level. These are experimenting with the real system and experimenting with a model of a real system. Since warehousing systems have high level of complexity, experimenting with a real system would be much time consuming and costly as mentioned in Abu-Taieh (2008). In the same decomposition, Law et al. (2000) separates model of the real system as physical model and mathematical model. Again, to develop a physical model of a warehouse does not seem to be much reasonable in terms of efficiency and flexibility. Accordingly, most of the academic researches are based on mathematical models in the literature. Finally, mathematical models are categorized into analytical solution and simulation.

In Law et al. (2000), it is indicated that analytical methods should be preferred unless the solution is complex and require vast computing resources. But, in case of complexity, the model should be studied by means of simulation. In Banks (1998), it

is implied that, by simulation, internals of the model can be represented in detail differently from analytical models and studying warehouses through computer simulation is an effective way to increase the efficiency of warehouse in terms of performance and design. Senko et al. (1990) indicates that improperly designed warehouses inevitably face some problems such as low service level and high costs. The efficiency of a warehouse is strictly bound to whether the design reflects the nature of the activity. The main reason of any problems encountered in a warehouse is the lack of information about system constraints and bottlenecks; however simulation analysis of the system can avoid bad scenarios. (Bruzzone et al, 2009).

Paul et al. (1998) defines simulation as a process of designing a model of a real system and conducting experiments with this model for the purpose of either understanding the behavior of the system or of evaluating various strategies for the operation of the system. In Cassandras et al. (1999), it is defined as a process of numerically evaluating a system model and estimating variables of interest.

In Fishman (2001), technical attractions of simulation are listed as ability to compress and expand time, to control sources of variation, to avoid errors in measurement, to stop and review, to restore system state, to facilitate replication and to control level of detail.

In Law et al. (2000), simulation is classified along three different dimensions. These are static vs. dynamic models, deterministic vs. stochastic models and continuous vs. discrete models. Since the passage of time holds a key for the representation of a complex warehouse system, warehousing concept includes stochastic factors and state changes occur at discrete points in time, our model should be a dynamic, stochastic and discrete model. Such a model refers a DES model.

All simulation implementations adopt a modeling approach. The determination of the DES modeling approach is another important strategic decision for the project. In literature, these approaches are categorized into Process Interaction, Activity Scanning, Three Phase and Event Scheduling.

Process Interaction focuses on processes which can be described as set of events. In this approach, entity flows play the main role where flows include all states of objects. The process is described as “a time-ordered sequence of events, activities and delays that describe the flow of a dynamic entity through a system” in Carson (1993). Process Interaction is popular and widely used since it is easier to conceive and implement, but “deadlock problem”, as mentioned in Pidd (1998), stands as the weak point. This approach is used commonly by DES software, among them Automod. Another common approach, named Flow Transaction, is a derivative of Process Interaction. Arena, ProModel and Witness are some of popular software using this approach as implied in Abu- Taieh (2008).

Activity Scanning worldview concentrates on activities of a model and the conditions that allow an activity to begin. In this view, all activities are scanned in each time step and initiated up to their conditions. It is also called as Two Phase. First-STEP is one of the software in this category as mentioned in Abu-Taieh (2008). Slow runtime is a handicap of this approach. Three Phase approach is a variant of Activity Scanning. It is a bit more tedious to model, but faster since only conditional activities are scanned at each step.

Event Scheduling requires the identification of events and the impacts of those events on system’s state variables. This approach is more efficient than other approaches but can be complicated to model up to size of model. Some software using Event Scheduling approach are Sigma and SIMKIT. Since it is known as

computationally efficient, Event Scheduling approach was preferred in our study in spite of some modeling complications mentioned in the literature.

1.6 Literature Review

In this section, we examine some of the studies held on warehousing systems. First, we present a general picture of warehousing problems by examining two complementary literature reviews; then, evaluate some warehouse models.

In Gu et al. (2007), operation problems are classified according with the main warehouse functions; namely receiving, shipping, storage and order picking. Regarding to receiving and shipping; the problem structure is presented in a circumstantial manner in the study. Information about incoming shipments, such as their arrival time and contents, information about customers demands, such as orders and their expected shipping time and information about warehouse dock layout and available material handling resources are presented as given information for these problems. With this information set, allocations and assignments of resources are determined subject to some constraints such as layout, relative location of docks, management policies and throughput requirements etc. On the storage side; fundamental questions such as, what should be the inventory level for an SKU and where should an SKU be kept in the warehouse, come forward. In terms of storage location assignment problem, there are three main storage policies, namely random policy, dedicated policy and class-based policy. If there is no assignment for the storage locations, it is called random policy. In dedicated policy, each storage location can be assigned to an item. In class-based policy, items are grouped into classes, let call product class, and storage locations are assigned to these classes. With dedicated storage policy, material handling systems can be used in a more

effective way, since fast-moving items can be placed close to docks. On the other hand, it suffers from higher storage space requirement, because, for each product a number of storage locations which equals to its maximum inventory level should be reserved. Class-based is between these two. There are three frequently used criteria for assignment. These are popularity, max inventory and cube-per-order index (COI). In popularity criterion, items or product classes are ordered according with their number of retrievals per unit time. Items having highest popularity are assigned to the most preferential locations. In maximum inventory criterion, items or product classes are ordered according with maximum storage space requirements. Items having lowest space requirements are assigned to the most preferential locations. COI criterion combines former two. It is the ratio of maximum inventory to popularity. Items having lowest COI values are assigned to the most preferential locations. Lastly, order picking is categorized into batching, sequencing/routing and sorting. Batching is a planning problem and the decision variable for this problem is to partition of orders for assignment to pickers. In sequencing/routing, the aim is to determine the best sequence and route of locations. Sorting is required for the warehouses having accumulation conveyor. It is used to separate picked items according with shipping needs.

The design problems are categorized into overall structure, sizing and dimensioning, department layout, equipment selection and operation strategy in Gu et al. (2010). The determination of the quantity of departments and flows among departments are in overall structure problems. Warehouse sizing and dimensioning problems deal with the storage capacity and operating costs in terms of floor space respectively. Department layout problems consider door location, aisle orientation, length and width of aisles, and number of aisles. The determination of storage and

material handling systems considering performance aspects is dealt with in equipment selection category. Lastly, operation strategies are related with the decisions about storage and order picking operations.

A model to obtain best trailer-door assignment to minimize total cost which is a function of travel time between doors and waiting time up to congestion is developed by Bartholdi et al. (2000). In the study, it is found that activity tends to concentrate in the center. Real case implementation of the model provided a %11 improvement in productivity.

Bartholdi (2004) asks if there is a best shape for the crossdock warehouses. The study reports that the most common crossdock warehouse shapes are I, L and T. According to Bartholdi (2004), I-shaped is preferred in general for smaller crossdock warehouses. The door with smallest average distance is indicated as the best door and center doors of an I-shaped warehouse are given as the most convenient doors in the study. Two important definitions are diameter and centrality. The diameter refers to the distance between doors and the centrality is the number of doors required to increase the diameter by one door offset. It is indicated that I-shape lose more centrality as the number of doors increase comparing to other designs. On the other hand, those designs having some corners incur some other costs because of labor efficiency up to safety regulations and congestion. As the result of computational experiments, it is reported that alternative shapes other than I-shape can be preferable when the number of doors exceeds 150.

Inter-leaving in unit-load warehouses are examined in Pohl et al. (2008). The study is dedicated to find out the best design for dual-command operations out of three common warehouse designs. For the comparison of chosen warehouse designs, expected travel distance expressions for dual-command operations are developed. As

a conclusion, they show that the least commonly found in practice is the best design for inter-leaving.

Ertek et al. (2007) dedicate their study to analyze how the number of cross aisles affects picking travel time in a rectangular warehouse operating with multi-picking tours. For this reason, two types of cross aisles, literally equally spaced and unequally spaced, have been considered. The computational experiments resulted that equally spaced cross aisles aid to reduce picking travel time more, comparing to unequally spaced ones. They provide a table for the number of equally spaced cross aisles that should be applied when length of the warehouse and the number of main aisles are given. Furthermore, they indicate that the number of cross aisle should be set to three when length of warehouse and number of main aisles are not determined.

A conceptual warehouse simulation model is introduced by Zhou et al. (2005). The model includes three main flow pattern, namely inbound, outbound and truck-docking flow patterns. A set of activities such as receiving, putaway, picking and shipping are executed in each pattern. Patterns are presented with flow charts as shown in .

Hoyur et al. (2006) developed an Arena-based simulation model for a unit-load warehouse to help the warehouse managers in their decisions such as determining the required number of storage locations, docks and material-handling systems. The model considers two patterns, namely inbound and outbound patterns. Inbound pattern includes receiving and putaway functions, while outbound process includes picking and shipping functions. The experimentations have been made considering forklift utilizations for a given warehouse layout for five customers and three item types. As a conclusion, number of required resources in terms of storage location and material handling system has been determined with the model.

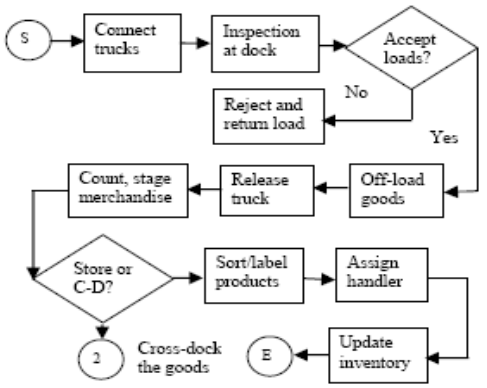

Figure 1: In-bound Flow Pattern
(Zhou et al., 2005)

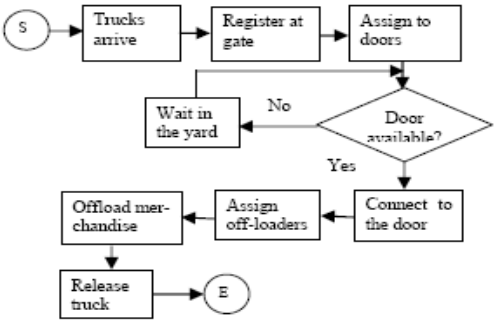

Figure 3: Truck-docking Flow Pattern
(Zhou et al., 2005)

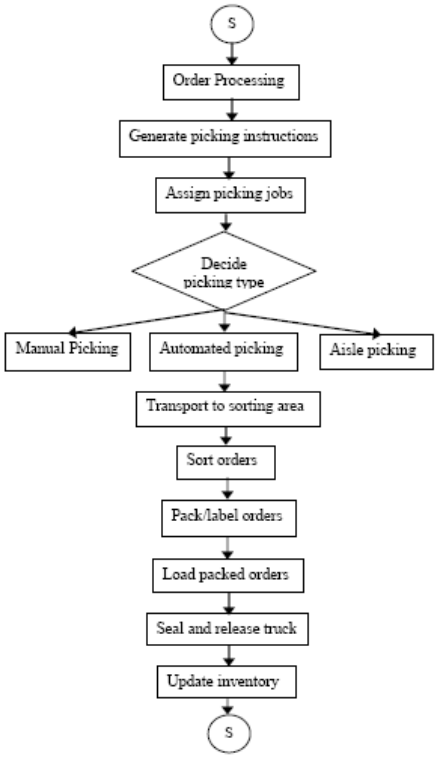

Figure 2: Outbound Flow Pattern
(Zhou et al., 2005)

Figure 1-1: Conceptual Simulation Modeling of Warehousing Operations

A simulation model developed with Fortran IV based simulation language GASP II is introduced by Bafna (1973). The model aims to evaluate alternative designs of unit-load warehouse systems using stacker cranes. In the study, it is presented a cost model including floor space, construction, material handling systems etc. It concludes that height has a decreasing impact on warehousing cost.

One of the examples of simulation models developed for analyzing material handling in warehouses is Macro et al. (2002). The main objectives of the study are to determine the effects of different rack types on size of two different warehouses and to measure storage method and layout efficiency. A different conceptual model was designed for each warehouse. First warehouse was simulated with a configuration where low performing rack types were converted with alternative ones.

Second warehouse was assumed to have an additional space operating with different rack types. Both cases were evaluated in terms of storage utilization and operating cost and as a result selective and push-back options are presented as favorable choices.

Gagliardi et al. (2007) developed a DES model based on Visual Basic to analyze a warehouse having some operational inefficiency. The goal of the model is to reduce stockouts by the use of some storage policy criteria. As the conclusion of the study, popularity rule has been indicated as the best performing one for the warehouse examined.

Zoning may have some pros and cons such as positive effects of picker's increasing acquaintance in the zone or requirement for a larger storage area. A simulation research on zoning by Petersen (2002) introduces a model which focuses on aisle orientation of picking zones and concludes that zone size has a crucial effect on operational cost.

As a result of our review, we have concluded that although detailed warehouse simulation models are very useful in measuring the effects of planned strategic and tactical changes on the system, the number of simulations which examine warehouses with all details is limited in the literature (Bruzzone et al, 2001).

2 DEVELOPING A GENERAL PURPOSE SIMULATION LIBRARY: SHARPSIM

2.1 Introduction

SharpSim is a general purpose DES library written in Visual CSharp (C#) and created to implement models developed with Event Scheduling (ES) approach. In the first chapter, we explained the reasons behind the decisions of choosing DES and Event Scheduling. This chapter covers the specifics of SharpSim. At the end of the chapter, implementation aspects of SharpSim will be explained with an M/M/N queuing system simulation example.

2.2 Event Graphs

Among all conceptual modeling methods, Event Graphs (EGs) are the best in terms of simplicity to design models adopting Event Scheduling worldview. First step in model building in SharpSim is to create an EG of the system to be modeled and therefore it makes sense to review EGs.

EGs are a way of conceptual representation of Event Scheduling world-view. Buss (2001) indicates that "Event Graphs are a way of representing the future event list logic for a discrete-event model". There are two main components of an EG. These components are events and edges. Nodes are used to represent events while arcs are used to represent edges. Figure 2-1 is useful to comprehend the basics of EGs. In Buss (2001), the basic structure is shown as follows:

"If condition (i) is true at the instant event A occurs, then event B will immediately be scheduled to occur t time units in the future with variables k assigned the values j."

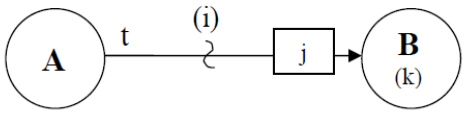

Figure 2-1: A Basic Event Graph

This basic structure can be used to model events and their relations in a system such as a queuing system. An M/M/N model which is presented in Figure 2-2 will provide us a better understanding of EGs. Note that this model does not include any entity object. Entity objects are moving items in a system such as patients in a hospital simulation, parts in a plant. Entities are represented with counter variables in this model. Parameters passed on edges are integer values holding entity identification numbers.

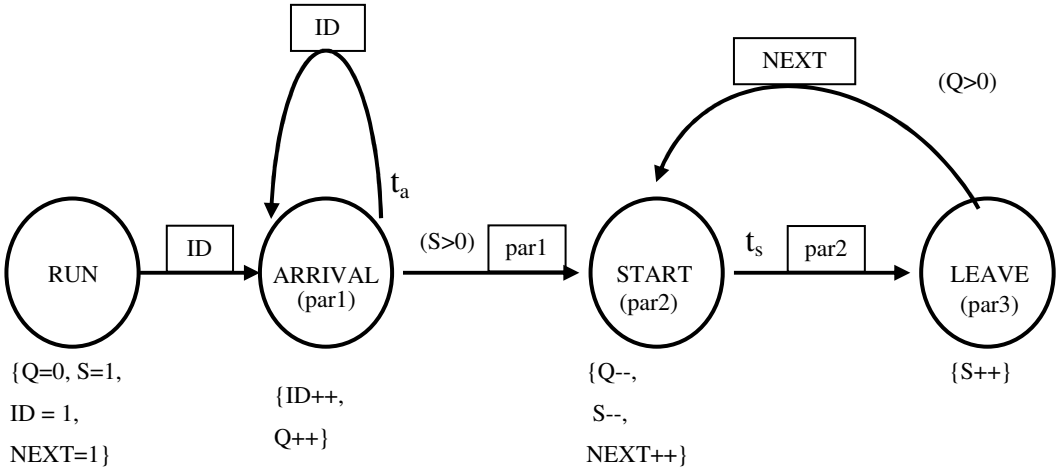

Figure 2-2: M/M/1 Model

In this EG model, there are four events, namely *Run*, *Arrival*, *Start* and *Leave*. Arcs (edges) determine the relations among events and constitute the system flow. *Q* and *S* are system state variables. *Q* holds the number of customers waiting in queue for service. The server queue operates with First-Come-First-Serve (FCFS) policy. *S* holds the available number of servers. *ID* and *NEXT* are other system variables. *ID* is the identification number of customer and *NEXT* is first customer to be served. *Par*

stands for parameter and represents the parameter of the event. According to ES, edge attributes are set to event parameters automatically in scheduling phase. This mechanism is called as parameter passing.

According to ES algorithm, the first event in future event list (FEL) is executed, relevant state changes defined to this event occurs, following events are scheduled to FEL and executed event is removed from FEL. Finally, the algorithm controls if the simulation is terminated and if not continue to iterate with the first event in FEL. Iterations continue till the simulation termination time or there is no event in FEL.

At the beginning of the simulation, only a *Run* event is scheduled to future event list. At time zero, *Run* event is executed. Consequently, system variables *Q, S, ID* and *NEXT* are set to their initial values. After state changes occur, scheduling procedure take part in. *Arrival* event follows *Run* event in the EG model. Therefore, scheduling a new *Arrival* event to FEL should be decided at this time. Obviously, an *Arrival* event will be scheduled to FEL because no condition is defined on the edge linking *Run* and *Arrival* events in EG model. Note that linking edge carry an attribute and this attribute holds the value of *ID*. This value automatically set to following event's parameter. Therefore, the *Arrival* event of the first customer (*ID*: 1) is scheduled to FEL at the same simulation time. Since *Run* event is executed at time 0, *Arrival* event is also scheduled to time 0. Finally, *Run* event is removed from FEL.

The first event in FEL is an *Arrival* event at this time. *Arrival* event is executed and state changes defined to *Arrival* event occur. *ID* and *Q* are incremented. This means that the first customer arrived to the system and is added to server queue. Next, the loop edge's attribute is set to *ID*. Since *ID* is incremented to 2, next *Arrival* event is scheduled with the *ID* value "2" after a delay of t_a (interarrival time). It means the arrival of the first customer schedules the arrival of the second customer.

Accordingly, the arrival of the second customer schedules the arrival of the third customer and so on. Furthermore, *Arrival* event schedules a *Start* event up to server availability with *Arrival* event's parameter value. This value holds the first customer *ID*. Therefore, the arrival of the first customer schedules the start of the service for the first customer if the server is available. Since, at the beginning of the simulation, the server is available, it schedules a *Start* event at the same simulation time. Finally, *Arrival* event is removed from FEL.

At this time, there are two events in FEL, namely an *Arrival* and a *Start* event. *Start* event is scheduled to time 0 and *Arrival* event is scheduled to time t_a. Therefore, next event to be executed is *Start* event. *Start* event is executed and relevant state changes occur. *Q, S* and *NEXT* takes their new values. This means that first customer is served at the moment therefore it is no longer in queue and server is busy. At the same time, *ID* of next customer to be served is "2" and *NEXT* holds this value. Later on, for the first customer, a *Leave* event is scheduled to FEL after a delay of t_s (service time). Finally, *Start* event is removed from FEL.

At the moment, there is an *Arrival* event and a *Leave* event. If *Leave* event occur first, it doesn't schedule any *Start* event because there is no customer waiting in server queue. Suppose that, *Arrival* event is scheduled before *Leave* event. In this case, *Arrival* event schedules another *Arrival* event for the third customer but fail to schedule a *Start* event for the second customer since the server is busy. In this case, the second customer has to wait till the first customer leaves the system. After the execution of *Arrival* event, if *Leave* event is executed, then a *Start* event is scheduled automatically since there is a customer waiting for the service. Scheduled *Start* event takes the value of *NEXT* as the customer identification number and it is 2. Suppose that, the third customer arrived before the first customer left the system. Again no

Start event is scheduled and *Q* incremented to 2. When the first customer leaves the system, it schedules a *Start* event for the second customer because *NEXT* still holds the value of 2.

We believe that, this example is satisfactory to demonstrate EGs method. At the end of the chapter, we present a tutorial to show how to implement an M/M/N model including entity objects with SharpSim. Here, we continue with some specifics of SharpSim.

2.3 SharpSim

SharpSim has a object-oriented structure. It involves a set of classes as shown in Figure 2-3.

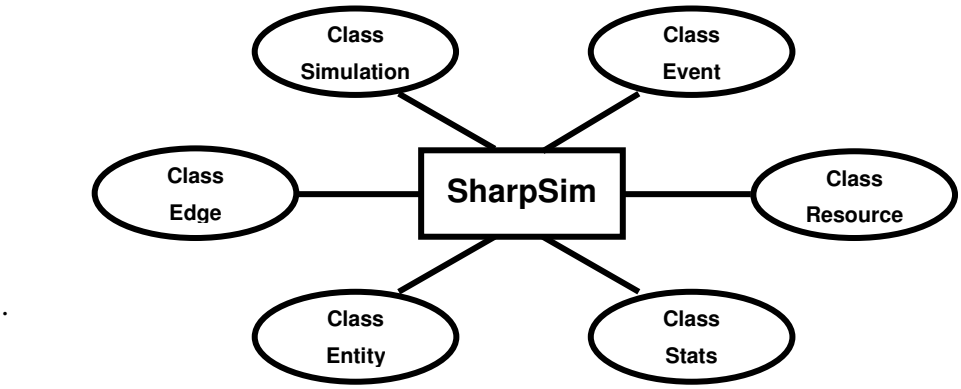

Figure 2-3: Structure of SharpSim

2.3.1 Simulation Class:

Simulation class is the core of SharpSim. It provides a platform where all elements of the model can interact with each other. This class includes the main properties of the simulation such as *FEL*, simulation time (*clock*), the mechanism which handles event scheduling algorithm and the thread that executes the model. The class presented in Figure 2-4 is made using Star UML software and in its

standards - (private)/+ (public) signs are used as modifiers. (see StarUML, 2011)

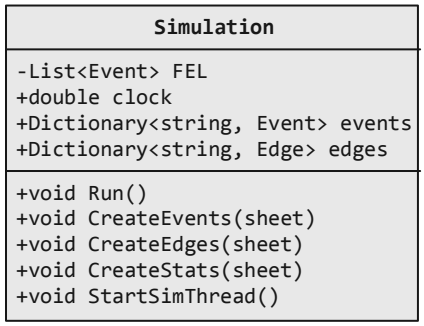

Figure 2-4: Class Simulation

2.3.1.1 Properties

- *FEL*:

This collection is a dynamic list involving the set of *Event* objects. Each Event object in this collection represents an event which is going to be executed in future. Newly scheduled *Event* object are inserted in *FEL* based on their execution times and priorities. Upon execution, each *Event* object is removed from the list.

- *clock*:

This global variable holds the time of the simulation. It is handled in *Run* method. It proceeds to the execution time of the next event at each event execution.

- *events*:

This collection involves the set of events instantiated at the beginning of simulation. This list can be thought as the static *Event* objects depot of the model. *Event* class will be explained later in this chapter.

- *edges*:

This collection involves the set of *Edge* objects instantiated at the beginning of the simulation. *Edge* class will be explained later in this chapter.

2.3.1.2 Methods

- *Run*:

The Event Scheduling algorithm presented below is handled in this method.

  - Step 1: Advance clock to imminent event time
  - Step 2: Execute imminent event and update system state change entity attributes
  - Step 3: Generate future events (if necessary) and place the event notices on future events list (FEL), ranked by event time
  - Step 4: Remove event notice for imminent event from future events list FEL

- *CreateEvents*:

This method is used to instantiate the *Event* objects of the model. The method takes a spreadsheet as an argument and this spreadsheet involves the arguments required to instantiate *Event* objects.

- *CreateEdges*:

This method is used to instantiate the *Edge* objects of the model. The method takes a spreadsheet as an argument and this spreadsheet involves the arguments required to instantiate *Edge* objects.

- *CreateStats*:

This method is used for the collection of statistics. The method takes a spreadsheet as an argument and this spreadsheet includes the names of the statistical information that are going to be collected with the model. When this method is

invoked, SharpSim creates two dictionaries. These dictionaries are called as *dictionary* and *global dictionary*. The first dictionary is used to collect statistics during a replication. The second dictionary is used for holding statistics of all replications. *CreateStats* method has an overload that takes an argument of string type. Different from first overload, latter method is invoked for each variable that is going to be added to the dictionaries.

- *StartSimThread*:

Threads are used to execute multi-tasks in a single application. This is what we need, because while simulation is running, we may need to do some other tasks, such as writing to a user control, or animating objects on the screen. SharpSim provides a simulation thread (*simThread*) which is associated with *Run* method. Thus, some other tasks can be achieved simultaneously while the simulation runs on a separate thread. When we need to run the simulation, we can use directly *Run* method or simply use *StartSimThread* method to start the thread which invokes *Run* method. Using multi-threading is an extra feature provided by SharpSim and it is not mandatory to use. The down side of using threads is that they may easily cause a chaos. When more than one thread is used in a program, the threads must be synchronized. Synchronization is required especially when same resources are used by more than one thread. For example, if we have a warehouse simulation model and we are animating objects on the screen, we need two separate threads which read from and write to shared data structures. One thread is responsible for simulation algorithm, and the other thread is for animation. Imagine that we have a data structure, such as *List*, which stores pallet objects in the warehouse and animation thread is using this list to get the position of each pallet. At the same time, the simulation algorithm generated a new pallet which arrived to the warehouse and it

should be added to this list. Since there is no synchronization, *simThread* will try to add this pallet to the list while animation thread is using this list for getting the pallet positions to draw on the screen. Naturally the model will generate an exception in this case. To avoid this exception to occur, these two threads must be synchronized so that no such a coincidence occurs. There are a set of mechanisms for synchronization in CSharp, namely interlocked, lock and monitor. SharpSim uses lock mechanism for synchronization and provides a read-only *threadlock* object. *Run* method iterates for each event in *FEL*. *Threadlock* is locked at the beginning of each repetition and it is released when the iteration ends. Therefore, no other thread can start while *threadlock* is locked. But simulation thread may start while another thread is in action. If another thread is defined in the model and this new thread uses the same resources which *simThread* use, same mechanism should be implemented for the method which this new thread invokes. If no thread is defined, or the defined thread does not use the same resources, there is no need for synchronization.

2.3.2 Event Class:

An event is represented by a node in an Event Graph model. It is an occurrence which causes a state change in a system. For each node in the EG model, an *Event* objects is created at the start of the simulation. This set of *Event* objects is used as a static depot for the simulation. During the simulation, any time a new event should be scheduled, relevant *Event* object in this depot is cloned and added to *FEL*. The clone of the *Event* object holds all attributes of the base event. Those attributes are not only properties or methods, but also event handlers. SharpSim utilizes event-handling of C# for two cases. These two cases are explained below.

Consider the model in Figure 2-1. In that model, there are two events and a

linking edge between these two events. When event A is executed, event B is scheduled to FEL if edge condition is satisfied. This means that event B should always follow event A. Event-handling of C# is used in this case by defining an event execution handler. In the model, event B subscribes to event A's event execution handler and associates a delegate method to this subscription. This subscription is set implicitly when linking edge between any event couple is instantiated. When event A is executed, it publishes its execution and invokes the delegate method of event B automatically. The delegate method of event B controls the linking edge's condition and schedules a copy of event B to FEL if condition allows. Recently scheduled event's parameter is set to linking edge's attribute value. This flow is presented in Figure 2-5.

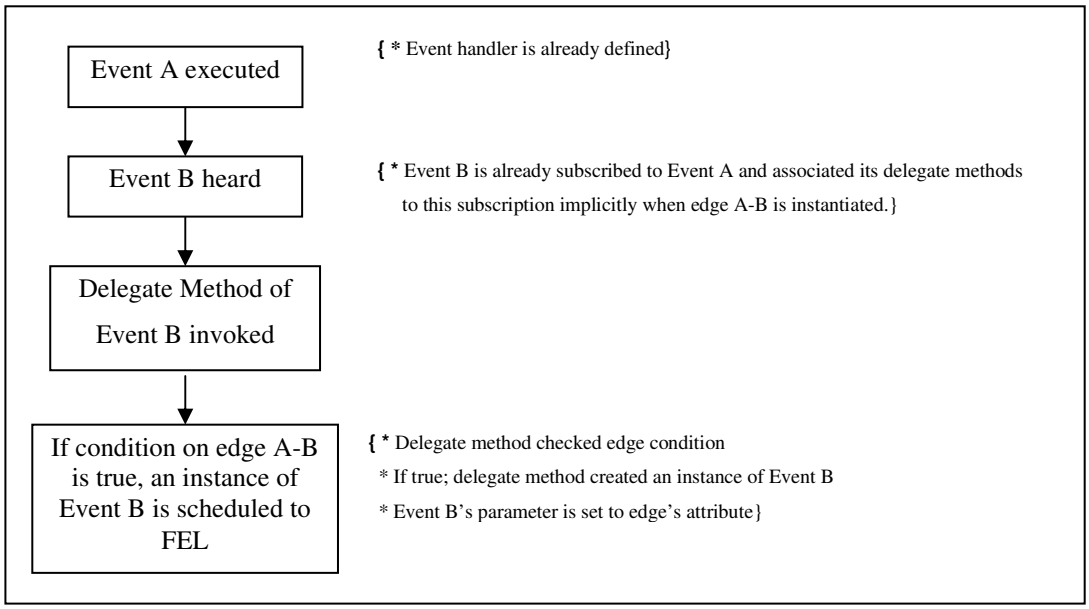

Figure 2-5: SharpSim Scheduling Flow

In the first case (simulation event scheduling), events subscribe to each other as indicated by edges and associate their delegate method with this subscription. In the second case (state change handling), model developing form subscribe to e*vents* and associate state change delegate methods with this subscription. These methods

include relevant state changes for each event. Therefore, a state change method is created for each event in the main model code. These methods are anonymous methods getting an argument of *Event* type. They are used to subscribe the simulation output form to the *Event* object which is given to the method as an argument. Furthermore, state changes are also included in this method. When an event is executed, relevant state change method is invoked automatically and state changes are executed.

Since simulation output form and following events use the same event handler, the order of handler subscription is important. According to ES, first state changes of an event are executed, and then following events are scheduled. Therefore, subscription of state change handlers should be before event scheduling procedure. This will be indicated in the M/M/N implementation example.

There are two notes on events in SharpSim. First, event-handling mechanism is about inner workings of SharpSim and therefore it is not a “must-be-known” issue for modelers. Second point is an event in event-handling is not the same thing with an event in DES.

The *Event* class structure is presented in Figure 2-6.

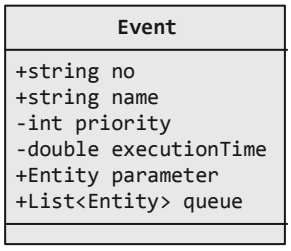

Figure 2-6: Class Event

2.3.2.1 Properties

- *no*:

The identifier number for the event.

- *name*:

The name of the event.

- *priority*:

Newly scheduled *Event* objects are inserted in future event list according with their planned execution time. When more than one event has the same execution time, ES algorithm needs a second parameter to decide which event will be executed earlier. Priority provides this secondary regulation for event execution order. It is crucial to assign priorities on events in order to avoid unordered event execution. For example, in a simple M/M/1 model, there are four events: *Run*, *Arrival*, *Start_Service*, *End_Service*. Consider that some time in model run, we have two *Arrival* events scheduled to this moment and the server is available. In this case, first of two *Arrival* events will be executed and schedule a *Start_Service* event to the same moment. Therefore, now we have one *Arrival* and one *Start_Service* events in FEL. If *Arrival* event is executed before *Start_Service* event, then another *Start_Service* event will be scheduled to FEL. But, this resource is not available actually. It seems available, because reducing the number of available servers is handled in the state change method of *Start_Service* and *Start_Service* event was not executed yet. Therefore, for an M/M/1 model, *Start_Service* event should have the priority on *Arrival* event. All event priorities should be decided with this approach at the conceptual modeling phase when an EG model is drawn.

- *executionTime*:

Each Event object has an execution time. The execution time of an event is mostly set during the simulation. For example, the execution times of Run and Terminate events can be set when the model is built.

- *parameter*:

This property along with attribute property of Edge class provides parameter passing. The value of edge's attribute is set to event's parameter. Entity objects can be transferred among events with parameter passing.

- *queue*:

In SharpSim, we have entities moving throughout the model. Since the model is made of events, we can say that entities visit events throughout the model. Sometimes, entities should wait for a while to visit an event, because the resource associated with that event is busy. This results in a queue. In this case, we can associate queues with events. In SharpSim each event has a queue property. This property is used as state variable when the system to be modeled has a queue. When a condition is not met, the entity can be kept in the relevant.

2.3.2.2 Methods

- *DelegateExecuteEvent*:

Event objects follow each other according with relations set by Edge objects (event handling). When an Event object executes, each following Event object's Delegate Execute Event method is invoked automatically. In case Edge object's condition between executed Event object and following Event object permits to schedule, following Event object is cloned and inserted to FEL.

2.3.3 Edge Class:

An *Edge* object is represented by an arc in an EG model. It links two events and defines relations between these events and constitutes the flow of the system. An event is scheduled to occur in the future if the condition on the edge is true. Furthermore, execution times of newly scheduled events are set according with edge's inter event time value. The *Event* class structure is presented in Figure 2-7.

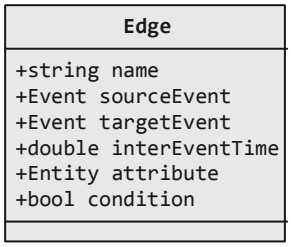

Figure 2-7: Class Edge

- *name*:

The name of the edge.

- *sourceEvent*:

Each *Edge* object links couple of *Event* objects. This is the event where the edge starts from.

- *targetEvent*:

This is the event where the edge arrives to. It subscribes to source events and when a *sourceEvent* is executed, following *targetEvent* is scheduled up to linking edge's situation.

- *interEventTime*:

These can be deterministic or stochastic. If no *interEventTime* is set, it is set to 0.

- *attribute*:

This property along with *parameter* property of Event class provides parameter passing. The value of edge's *attribute* is set to event's *parameter*. Object entities can be transferred among events with parameter passing.

- *condition*:

When an event is executed, every exiting edge's *condition* is controlled and accordingly following events are scheduled.

2.3.4 Entity Class

Moving items throughout the model can be defined as Entity objects. An Entity object for an M/M/1 model is a customer. SharpSim provides an abstract Entity class which is important for parameter passing. Therefore, Event parameter and Edge attribute is Entity type. The class structure is presented in Figure 2-8.

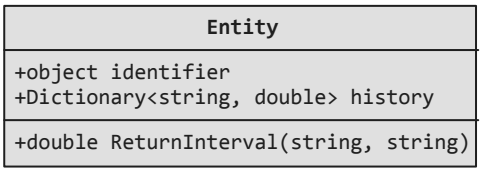

Figure 2-8: Class Entity

2.3.4.1 Properties

- *identifier*

Each *Entity* object has a distinctive identifier.

- *history*

When an event occurs, the execution time of *Event* object is added to entity's history. Thus, each *Entity* object keeps record of the time of its activities.

2.3.4.2 Methods

- *ReturnInterval*

This method is invoked for an *Entity* object. Two visited events' identification numbers are given to the method as arguments and this method returns passed time between these two events.

2.3.5 Resource Class

Resources are the limited capabilities of the system. A *Resource* object for an M/M/1 model is a server. SharpSim provides an abstract *Resource* class. Each resource collects its own statistics. The class structure is presented in Figure 2-9.

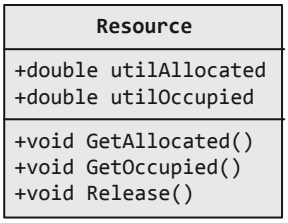

Figure 2-9: Class Resource

2.3.5.1 Properties

- *utilAllocated*

The resources can be allocated and occupied at the same moment. However, sometimes the allocation and occupation take place at different moments. Accordingly, two different statistics should be collected. This value holds the utilization according with allocated time.

- *utilOccupied*

This value holds the utilization according with occupied time.

2.3.5.2 Methods

- *GetAllocated*

Relevant resource gets allocated and the allocation time is saved by the resource itself.

- *GetOccupied*

Relevant resource gets occupied and the occupation time is saved by the resource itself.

- Release

Allocation and occupation utilization values are updated. Utilization value is computed with the division of total utilization to system time.

2.3.6 Stats Class

Stats class hosts some static properties and methods which are used to collect statistics of the model. In the center of all statistical activities, there are two static dictionaries. These dictionaries are created and deployed through *CreateStats* method of *Simulation* object. This was already explained earlier in this chapter. During the model run, some static methods of *Stats* class are used to collect statistics of the model, and collected information is processed through these dictionaries. The class structure is presented in Figure 2-10.

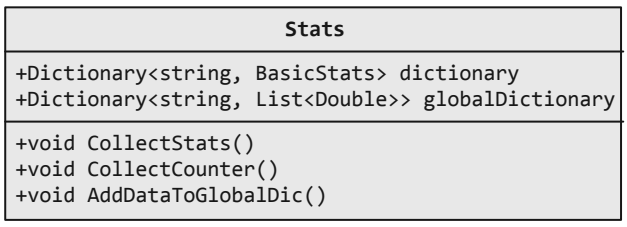

Figure 2-10: Class Stats

2.3.6.1 Properties

- *dictionary*

This dictionary is used to collect statistics during a replication. The dictionary key is string type and value is *BasicStats* type. *BasicStats* is a sub-class of *Stats* class. *BasicStats* has properties of total, mean, standard deviation, variation etc. Each key returns a *BasicStats* type value.

- *globalDictionary*

This dictionary is used for holding statistics of all replications. The dictionary key is string type and value is list of double type. Each key returns a list of double type value.

2.3.6.2 Methods

- *CollectStats*

CollectStats method takes a string (key) and a double (value) argument. The method updates *BasicStats* type value of the key by processing new value. For example, let's have an entry which is called *queue_wait_time* in our dictionary. For each entity leaving the queue, we collect waiting time of each entity. The method takes the key and the value, and updates *BasicStats* type value for key "queue_wait_time". For each invocation of the method, *BasicStats* type value is updated. This means mean and standard deviation etc. are updated.

- *CollectCounter*

CollectCounter method takes a string (key) and a bool argument. The method creates a counter variable and increments or decrements the value of the counter according with second argument.

- *AddDataToGlobalDic*

At the end of each replication, some values of each statistics are added to *globaldictionary*. Global statistics are computed implicitly and exhibited at the end of all replications on *Simulation Output Form*.

2.4 A Tutorial for M/M/N Example

The conceptual M/M/N model designed with Event Graphs method is presented in Figure 2-11. This model differs from M/M/1 model presented in Figure 2-2 in two senses. First, apparently this model has n servers. Second, there are Entity objects in this model. Therefore, entities are passed among events.

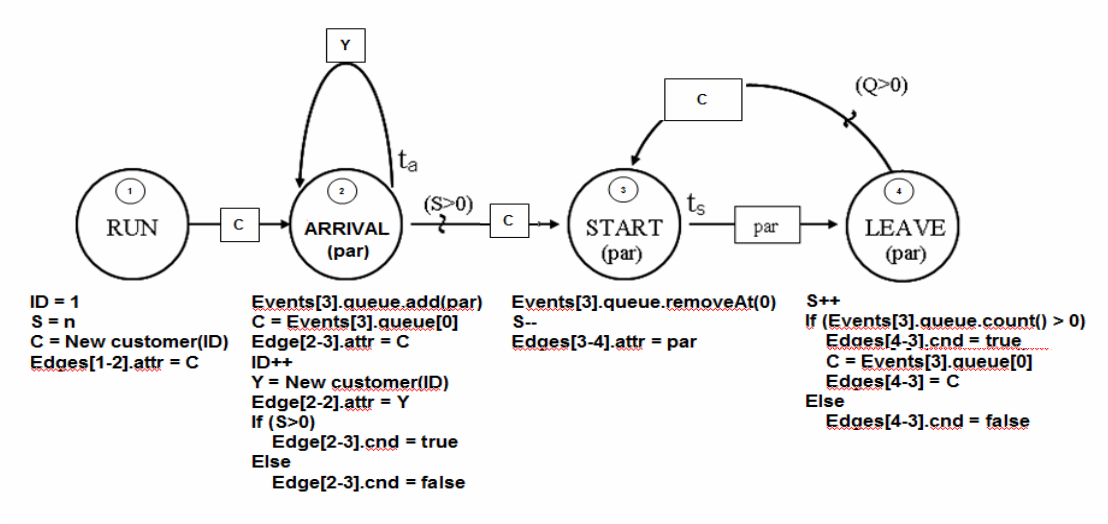

Figure 2-11: M/M/N Model

To build the M/M/N model in SharpSim, first it is required to create a C# project and refer SharpSim library to this project. Note that a third party library NExcel has to be referred to the project as well in case Excel input is going to be used for the instantiation of *Event* and *Edge* objects. To set our form, a *richtextbox* and a *button* object are added to the project's form. The use of these controls will be explained later. In the implementation of the model, first, *Simulation*, *Event* and *Edge* variables

that will constitute the model are defined. *Simulation* variable forms a platform where model elements interact with each other. *Event* and *Edge* variables are taken directly from conceptual model. In addition to events shown in Figure 2-11, also a *Terminate* event is defined. *Terminate* event is an occurrence which stops the simulation.

After defining, variables are instantiated under *button_click* method with the same order. Here, there is an important detail. Before the instantiation of *Edge* objects, state change handler methods should be associated with relevant events. This order is what ES algorithm obliges. Thus, state changes defined to events will be executed before scheduling following events. First state change handling methods should be created. These methods are anonymous methods which combines two functions. These methods include code snippets that subscribe the form to relevant event's handler and state change codes that will be executed automatically when the subscribed event is executed. After creating these anonymous methods, just after the instantiation of *Event* objects, these methods are getting associated with relevant *Event* objects. Lastly, *Edge* objects are instantiated. After the instantiation of each *Edge* object, inter event time attributes are set in case required by the model. We would like to collect statistics for the total system time of entities. For this reason, we are using *CreateStats* method to create required statistics data structure. Finally, simulation is run by the use of *Run* method of *Simulation* object. *Button_click* method is presented in CodeBox 2-1.

We have already explained the state change handling methods. In CodeBox 2-2, CodeBox 2-3, CodeBox 2-4, CodeBox 2-5 and CodeBox 2-6, state change handling methods of the M/M/N model are presented. Note that state changes indicated in Figure 2-11. are included in these methods.

Thus we have created all the state change handlers. Final step is to create a *Customer* class. We create the *Customer* class inheriting from abstract *Entity* class provided by SharpSim. Note that *Entity* class can't be used to instantiate *Entity* objects since it is an abstract class. The *Customer* class is created as shown in CodeBox 2-7.

```
private void button1_Click(object sender, EventArgs e)
{   //Create simulation platform
    sim = new Simulation(true, 10, false);
    //Create events
    eRun = new Event("1", "Run", 1, 0);
    eArrival = new Event("2", "Arrival", 4);
    eStart = new Event("3", "Start", 2);
    eLeave = new Event("4", "Leave", 3);
    eTerminate = new Event("5", "Terminate", 5, 50);
    //Associate State change handlers to relevant events
    Run(eRun);
    Arrival(eArrival);
    Start(eStart);
    Leave(eLeave);
    Terminate(eTerminate);
    //Create edges
    edge1_2 = new Edge("1-2", eRun, eArrival);
    edge2_2 = new Edge("2-2", eArrival, eArrival);
    edge2_2.dist = "exponential";
    edge2_2.mean = 5.0;
    edge2_3 = new Edge("2-3", eArrival, eStart);
    edge3_4 = new Edge("3-4", eStart, eLeave);
    edge3_4.dist = "exponential";
    edge3_4.mean = 5.0;
    edge4_3 = new Edge("4-3", eLeave, eStart);
    //collecting statistics
    sim.CreateStats("2-4");
    //RUN THE SIMULATION
    sim.Run();
    }
```

CodeBox 2-1: Button_Click Source Code-1

```
public void Run(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        ID = 1;
        S = 2;
        Customer customer = new Customer(ID);
        edge1_2.attribute = customer;
    };
}
```

CodeBox 2-2: Run Event State Change Handler

```
public void Arrival(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        eStart.queue.Add(e.evnt.parameter);
        edge2_3.attribute = eStart.queue[0];
        ID++;
        Customer cust = new Customer(ID);
        edge2_2.attribute = cust;
        if (S > 0)
            edge2_3.condition = true;
        else
            edge2_3.condition = false;
    };
}
```

CodeBox 2-3: Arrival Event State Change Handler

```
public void Start(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        eStart.queue.RemoveAt(0);
        S--;
        edge3_4.attribute = e.evnt.parameter;
    };}
```

CodeBox 2-4: Start Event State Change Handler

```
public void Leave(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        S++;
        if (eStart.queue.Count() == 0)
            edge4_3.condition = false;
        else
        {
            edge4_3.condition = true;
            edge4_3.attribute = eStart.queue[0];
        }
        Stats.CollectStats("2-4", e.evnt.parameter.ReturnInterval("2", "4"));
    };}
```

CodeBox 2-5: Leave Event State Change Handler

```
public void Terminate(Event evt)
{
    evt.EventExecuted += delegate(object obj1, EventInfoArgs e)
    {
        richTextBox1.Text += "Replication No : " + Simulation.replicationNow +
       "ended." + "\n";
        Stats.AddDataToStatsGlobalDictionary("2-4", Stats.Dictionary
        ["2-4"].mean);
    };}
```

CodeBox 2-6: Terminate Event State Change Handler

```
class Customer : Entity
{
   public Customer(int id)
          : base (id)
   {
      this.identifier = id;
   }
}
```

CodeBox 2-7: The Customer Class

We have implemented our model with SharpSim. When we run the model, we will get two forms. One is the Example form and the other is the *Simulation Form* which exhibits executed events' list and output.

Note that Event and Edge objects can also be instantiated by the use of Excel Input. In this case, NExcel library should be referred to the project and CodeBox 2-1 should be altered with CodeBox 2-8.

```
private void button1_Click(object sender, EventArgs e)
{
    this.openFileDialog1.FileName = "*.xls";
    this.openFileDialog1.InitialDirectory = Application.StartupPath;
    if (this.openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        //read the excel file
        string filename = openFileDialog1.FileName;
        Workbook workbook = Workbook.getWorkbook(filename);
         //Create simulation platform
        sim = new Simulation(true, 10, true);
         //Create events
        sim.CreateEvents(workbook.getSheet("Events"));
         //Associate State change handlers to relevant events
        Run(sim.events["1"]);
        Arrival(sim.events["2"]);
        Start(sim.events["3"]);
        EndService(sim.events["4"]);
        Terminate(sim.events["5"]);
        //Create events
        sim.CreateEdges(workbook.getSheet("Edges"));
   }
    //RUN THE SIMULATION
   sim.Run();
}
```

CodeBox 2-8: Button_Click Source Code-2

In this case, we need two spreadsheets holding argument information for the instantiation of Event and Edge objects. The spreadsheets required for this model is

presented in Figure 2-12 and.Figure 2-13.

	A	B	C	D
1	Event No	Name	Priority	Exec.Time
2	1	Run	1	0
3	2	Arrival	4	
4	3	Start	2	
5	4	Leave	3	
6	5	Terminate	5	50

Figure 2-12: Spreadsheet “Events”

	A	B	C	E	F	G	H	I	J
1	Edges	Source	Target	Inter.Time	Dist	Mean	Std.Dev.	src	trg
2	1-2	Run	Arrival					1	2
3	2-2	Arrival	Arrival		exponential	5		2	2
4	2-3	Arrival	StartService					2	3
5	3-4	Start	Leave		exponential	5		3	4
6	4-3	Leave	Start					4	3

Figure 2-13: Spreadsheet “Edges”

2.5 Verification

As indicated in Pidd (1998), verification is defined as a set of tests which controls the implementation of the model to understand whether the code represents the conceptual model. In the first type of these tests, the code is controlled line by line. The second type focuses on the modules and procedures of the implementation and controls these components if they produce expected outputs. The third type takes the implementation as a whole and checks the input/output relations of the model.

Since SharpSim is the implementation of ES approach, the algorithm of ES can be thought as the conceptual model of SharpSim. Therefore, these tests can be applied to SharpSim to understand whether it implements ES algorithm. For this reason, the implementation has been controlled line by line, method by method, class by class and as a whole to verify SharpSim.

2.6 Validation

For the validation of SharpSim, an M/M/1 model has been implemented with SharpSim. First, the validation of the model has been controlled with sensitivity analysis and extreme condition tests by observing model responses to input changes

and extreme input values. The input set and simulation settings presented below have been used in this test.

Input / Setting	Value
Inter Arrival Time	~ Exponential (5)
Service Time	~ Exponential (3)
Number of Servers	1
Number of Replications	20
Run Length	500000

Table 2-1: SharpSim Validation Test Input Set and Settings

This input set is assumed to be the base configuration. In the experiments, average queue wait and average system time have been observed. The results of sensitivity analysis are presented in Table 2-2.

No	Input Change over Base Conf.	Average Queue Wait	Average System Time
1	-	4.5017	7.4994
2	Number of Servers : 2	0.297	3.2952
3	Number of Servers : 3	0.0304	3.027
4	Inter Arrival Time ~ Exp (6)	3.0266	6.029
5	Service Time ~ Exp (4)	16.0606	20.0637

Table 2-2: M/M/1 Model Sensitivity Analysis

The results of extreme condition tests are presented in Table 2-3.

No	Input Change over Base Conf.	Average Queue Wait	Average System Time
1	Number of Servers : 100000	0	3
2	Number of Servers : 0	-	-

Table 2-3: M/M/1 Model Extreme Condition Tests

For further validation, M/M/1 model has been implemented with a commercial simulation package “MicroSaint Sharp (MSS) 3.0”. The model implemented with MSS 3.0 has been experimented with the base configuration and same simulation settings. The results of both implementations have been presented in Table 2-4.

Environment	Average Queue Wait	Average System Time
SharpSim	4,5017	7,4994
MSS 3.0	4,4934	7,4906

Table 2-4: The Comparison of SharpSim and MSS 3.0 Outputs

2.7 Comparison of SharpSim with Other Simulation Software

There are two types of simulation software; (1) Commercials-Off-The-Shelf (COTS) software which generally provides a graphical user interface (GUI) for the users having no or limited programming skills and (2) application programming interfaces (APIs) which is developed for the users having coding experience. To start with COTS software, we evaluated four of them; Arena, based on SIMAN, embedded with Visual Basic, supports discrete and continuous systems and offers 2D and 3D animations. AnyLogic, written in Java, supports DES, Agent-Based and System Dynamics concepts. Sophisticated 3D animation makes it a powerful analysis tool. Another popular software package, MicroSaint Sharp, offering 2D and 3D animations, is also a general purpose DES tool. MicroSaint Sharp is written in C#. Finally Sigma, written in C, based on Event Scheduling approach, allows the modeler to draw Event Graph on its palette. COTS software reaches a wider user community comparing to APIs since the former requires only the knowledge of how the software is used. In the application programming interface (API) group, SIMKIT, written in Java, is a good example since it is also based on Event Scheduling approach.

SharpSim shares a similar concept with Sigma and SIMKIT, in terms of event scheduling modeling approach. SharpSim is a simple API which is presented as a dynamic link library (DLL) and does not offer any GUI at the time being. It is developed with Visual CSharp and accordingly adopts object oriented phenomenon,

therefore SharpSim is easy to manipulate. To the best of our knowledge, SharpSim is the first and the only Visual CSharp implementation of a discrete event simulation library adopting ES.

In this section of our comparison, we made a performance evaluation between SharpSim and a commercial package MicroSaint Sharpp 3.0 in terms of runtime. As mentioned before, SharpSim adopts ES approach and ES is accepted as a computationally effective approach especially in congested models. On the other hand, MSS 3.0 adopts an approach other than ES which suffers from congestions. In this context, the durations of experiments made with SharpSim and MSS 3.0 are presented in Table 2-5. The input set and simulation settings described in validation section are used in all experiments.

Input Change over Base Conf.	**Duration (sec)**	
	SharpSim	**MSS 3.0**
-	14	150
Service Time ~ Exp (5)	15	2860
Service Time ~ Exp (6)	40	Significantly high

Table 2-5: The Experiment Durations of SharpSim and MSS 3.0

In the tests, any feature which may affect runtime performance of MSS was disabled to obtain results for a fair benchmark. MSS is professional simulation software which provides plenty of capabilities. It uses some external libraries and accordingly requires more memory. These factors obviously affect the performance of MSS negatively and this may justify the difference presented in the first line of Table 2-5. However, the rising gap between MSS and SharpSim in the second and third line where traffic intensity gets higher reveals that SharpSim and MSS are affected by the congestion in different ways. The higher the traffic intensity is, the better SharpSim performs, comparing to MSS.

3 CREATING A WAREHOUSE LIBRARY: WARELIB

3.1 Introduction

WareLib is a class library which maintains the basic warehouse objects and data structures required to simulate a warehouse. WareLib hosts a *Warehouse* object along with *Resource* and *Entity* objects. There are two types of *Resource* objects; locations and material handling systems. These resources interact with *Entity* objects, such as a pallet and a truck on *Warehouse* platform.

3.2 Resources

All *Resource* type classes in WareLib inherit from SharpSim *Resource* class. There are two main *Resource* types in WareLib. The first type is related to locations and the second type is related to material handling.

3.2.1 Grid Structure (Location Resources)

Warehouse is thought to be a grid which is composed of *Cell* objects. Each cell can be used for storage or passage purposes. Imagining a spreadsheet that represents a warehouse layout can be useful to conceive the grid based layout concept. The spreadsheet is divided into cells having the size of a pallet for storage cells and appropriate size for passage cells.

Figure 3-1 shows the hierarchy for grid based warehouse. This hierarchy is useful for object oriented thinking. *Cell* class is the base class and others down in the hierarchy inherit from *Cell* class. *Cell* type classes will be explained here till the second level in the hierarchy.

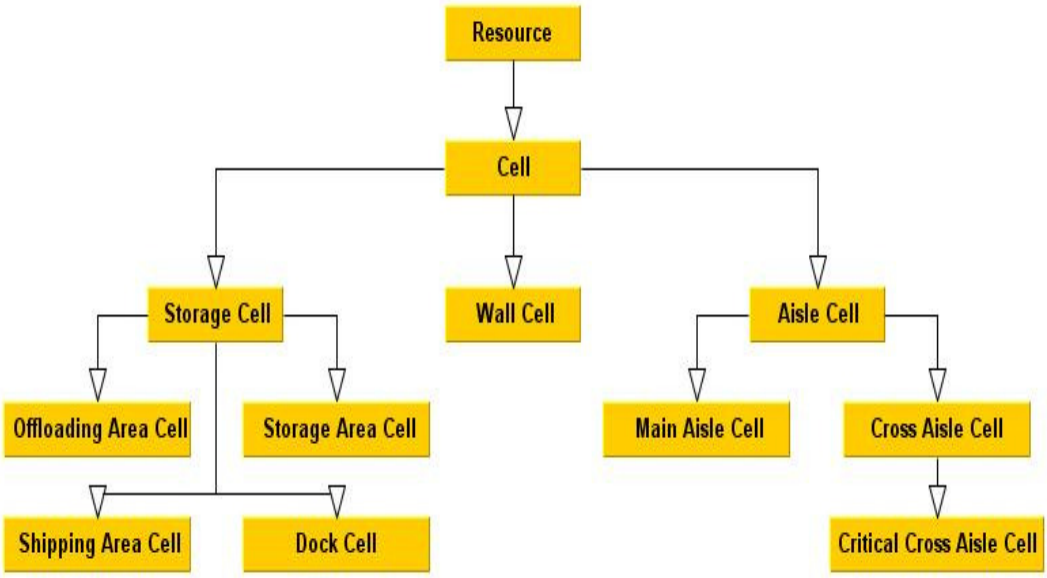

Figure 3-1: Grid Structure Hierarchy

3.2.1.1 Cell Class

Cell class is the base for other classes in the grid structure. This class inherits from *Resource* class.

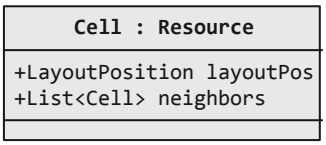

Figure 3-2: Class Cell

- *layoutPosition*:

This property holds the 3D position of the *Cell* object in the warehouse.

- *neighbors:*

This set of properties hold the neighbor *Cell* objects. These properties constitute the warehouse graph by defining relations among cells. This graph is used in computation of distances in the warehouse.

3.2.1.2 Storage Cell Class

StorageCell class inherits from Cell class. *StorageLocationAreaCell*, *OffloadingAreaCell* and *ShippingAreaCell* classes inherit from this class.

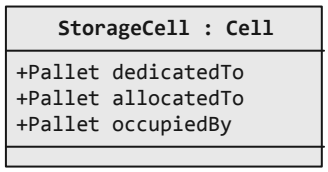

Figure 3-3: Class StorageCell

- *dedicatedTo*:

Each storage cell can be dedicated to a type of item.

- *allocatedTo*:

This property is set to allocated entity and provides a control to impede allocation of same location for more than one entity at a time.

- *occupiedBy*:

When an entity (e.g. a pallet) arrives to a location, this property is set to this entity. Thus, at any moment, content of each location is known.

3.2.1.3 AisleCell Class

AisleCell class inherits from *Cell* class and *CrossAisleCell* and *MainAisleCell* classes inherit from this class. *CriticalCrossAisleCell* class inherits from *CrossAisleCell* class.

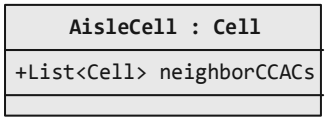

Figure 3-4: Class AisleCell

- *neighborCCACs*

With this property, *CriticalCrossAisleCell* objects are linked to each other. *These objects* are used as the main nodes of the warehouse graph. Since, relations among each *StorageLocationCell* object and *CriticalCrossAisleCell* objects are already defined; this graph is used in the computation of all the shortest routes among any storage locations with Floyd-Warshall algorithm at the start of the simulation. (About Floyd-Warshall algorithm, see Taha, 2007). Thus, no computation is made during execution. Warehouse graph representation is presented in Figure 3-5.

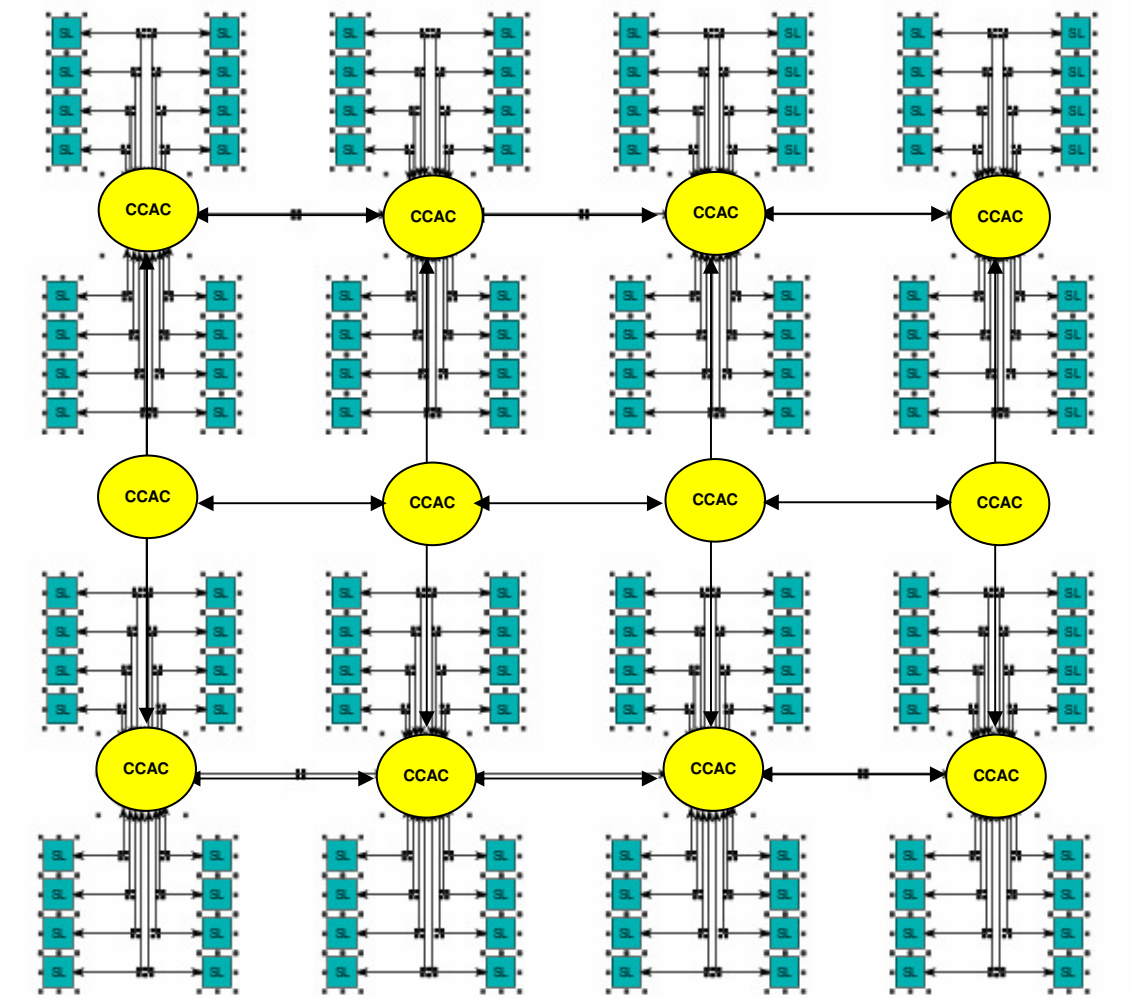

Figure 3-5: Warehouse Graph

Here SL and CCAC stand for storage location critical cross aisle cells.

WallCell class inherits from *Cell* class. No storage in and no passage through this cell is allowed.

3.2.2 Handling Resources

In order to reduce the complexity of the model, handling resources are preferred to be used as counting variables as much as possible. The *Forklift* objects are exceptionally evaluated as objects in the library.

3.3 Entities

There are four main entities in the warehouse library which are truck, pallet, demand and order. These classes inherit from abstract *Entity* Class.

3.3.1 Truck Class

Truck class inherits from abstract *Entity* class and *Arriving Truck* and *Departing Truck* classes inherit from this class.

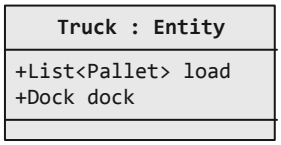

Figure 3-6: Class Truck

- *load*

This is a collection that keeps the truck load information. For each *Truck* object, a set of *Pallet* objects are instantiated and set to relevant *Truck* object's load property.

- *dock*

Upon arrival, *Truck* objects are assigned to a dock for receiving. When a *Truck* object is assigned to a dock, this *Dock* object is set to *Truck* object's *Dock* property.

3.3.2 Pallet Class

Pallet class inherits from abstract *Entity* class. They move along the warehouse.

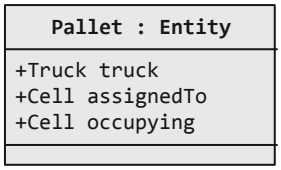

Figure 3-7: Class Pallet

- *truck*

Each *Pallet* object keeps associated *Truck* object information.

- *assignedTo*

When a *Pallet* object is assigned to a *Cell* object such as *Dock* object *or StorageAreaCell* object, *Pallet* object's *assignedTo* property is set to assigning *Cell* object.

- *occupying*

When a Pallet object arrives to a *Cell* object position, *Pallet* object's *occupying* property is set to occupied *Cell* object.

3.3.3 Demand Class

Demand class inherits from abstract *Entity* class. Demand is the list involving orders. Each order is satisfied with a pallet. Each demand invokes a departing truck for the system.

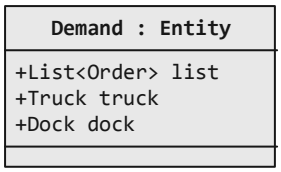

Figure 3-8: Class Demand

- *list*

Each *Demand* object holds a list of *Order* object.

- *truck*

Each demand is shipped with a separate truck in the model. This property is set to relevant shipping *Truck* object.

- *dock*

When a shipping *Truck* object associated with a demand is assigned to a shipping dock, this dock is set to *Demand* object's dock property.

3.3.4 Order Class

Order class inherits from abstract *Entity* class. If it can be satisfied, an order is associated with a *Pallet* object in the model.

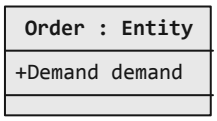

Figure 3-9: Class Order

- *demand*

This property is set to demand which the order belongs to.

3.4 Area Class

Warehouses are composed of some areas, literally storage area, offloading area and shipping area. All areas are the collection of some locations such as storage locations. With the grid structure, we have divided our warehouses into location cells. With area class, we make some list of same type of objects and define an area. For example, storage area is composed of storage area locations.

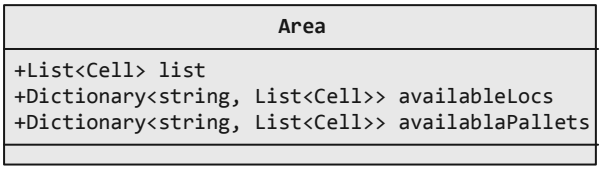

Figure 3-10: Class Area

- *list*

This property holds the list of all cells defined to the area. An example is the list of storage locations for the storage area.

- *availableLocs*

This dictionary holds all idle locations according with their key values. Key values are dedications of items. This means that at any time during model run, it is possible to get the list of available storage locations dedicated to any item. This collection is updated momentarily during model run according with pallet moves.

- *availablePallets*

This dictionary holds all available *Pallet* objects in an area to satisfy orders. Key values of this dictionary are items' identifiers. This means that at any time during model run, it is possible to get the list of available list of any stored item. This collection is updated momentarily during model run according with pallet moves.

3.5 Warehouse Class

Warehouse object is a platform which all elements of the warehouse are stored and interact with each other. There are plenty of properties and methods in this class, but only some important ones are presented below.

3.5.1.1 Properties

Warehouse is composed of areas and aisles. Each area (e.g. storage area,

offloading area) is composed of Cell objects.

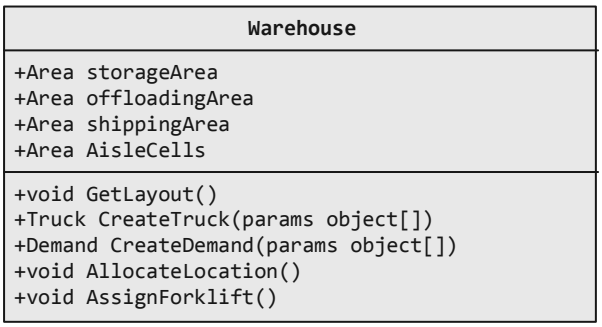

Figure 3-11: Class Warehouse

3.5.1.2 Methods

- *GetLayout*

Cellular layout information is taken from a spreadsheet, cell objects are created and interrelations among these objects are defined. There are some other methods to get traffic information, dedication plan and initial pallet positions.

- *CreateTruck*

This method creates a *Truck* object which is used as an entity in the model. Furthermore, according with the argument, a set of *Pallet* entities are created as the load of the *Truck* object.

- *CreateDemand*

This method creates a *Demand* object which is used as an entity in the model. Furthermore, according with the argument, a set of *Order* objects are created as the list of the *Demand* object.

- *AllocateLocation*

This method represents a set of allocation methods. With these methods, location resources are allocated for a *Pallet* object.

- *AssignForklift*

This method is used to assign a forklift for an operation. Assigned forklift can not be assigned to another job until it is released.

3.6 Animation

Animation features of WareLib are developed through XNA Game Studio. XNA is an animation library developed to create games (See Microsoft, 2011). In WareLib, XNA's animation form is used as a display control. Warehouse models developed with WareLib are automatically supported with 3D animation. Therefore animation is a natural output of the WareLib models. As one of the validation methods, animation gives to modelers a better understanding of the model.

The display control provided by XNA library is placed in Warehouse form. When animation is enabled, warehouse form including display control is exhibited on the screen. Display control has bars to tune camera position and view.

Display control updates animation momentarily. Since time evolution is irregular in discrete event simulation, the jumps of the objects on the screen is inevitable To avoid these jumps, a *Ping* event which is executed in regular time steps is required. In each *Ping* event, moving objects' positions are updated. When inter event time between any two *Ping* events decreases, a smoother move of *Pallet* objects is obtained. However, this causes the simulation to run slower since the number of *Ping* events to be executed increases.

In WareLib, only *Pallet* objects are animated on display control. Each item type is represented with a different color.

4 DEVELOPING A SIMULATION MODEL FOR A GENERIC UNIT-LOAD WAREHOUSE

4.1 Introduction

Warehouses vary in design and procedures. However, some patterns and functions are shared by most warehouses. Starting from this point, we have developed a general warehouse conceptual model and implemented it using SharpSim and WareLib libraries. Conceptual modeling and implementation aspects are explained later in this chapter. The purpose of the model is to create a general analysis tool for studying warehousing problems indicated in Chapter 1. Furthermore, we believe that our conceptual model can be used as a template for modelers who intend to develop conceptual warehouse simulation models adopting Event Scheduling (ES) approach. Modelers can generate an adaptive conceptual model using this template. The model includes main warehouse functions, literally receiving, putaway, picking and shipping. Besides, we adopted crossdocking function to the warehouse model. Since the conceptual model is component based, any component can be removed from the model easily without requiring any other modifications. Obviously some components are vital for the model.

The conceptual model is designed for unit-load warehouses operating with single-picking policy. It is developed with Event Graphs (EGs) method which is best fit for ES approach. Since the model tracks each pallet, it offers a high level of detail but does not suffer from prolong runtime.

The model separates warehouse operations into inbound and outbound processes as encountered in almost all warehouses. Each process decomposes into some sub-processes and components. According to this, inbound process involves truck

receiving, pallet receiving and putaway sub-processes; whereas outbound process comprises demand receiving component, demand satisfaction and shipping sub-processes. The total decomposition of warehouse operations used in the model is presented in Figure 4-1.

Below, the model is grouped and explained under three groups. These are inbound and outbound processes and general events. General section covers events required by simulation model logic. Other two comprise warehouse functions. The decomposition presented below is the categorization of processes into sub-processes and components.

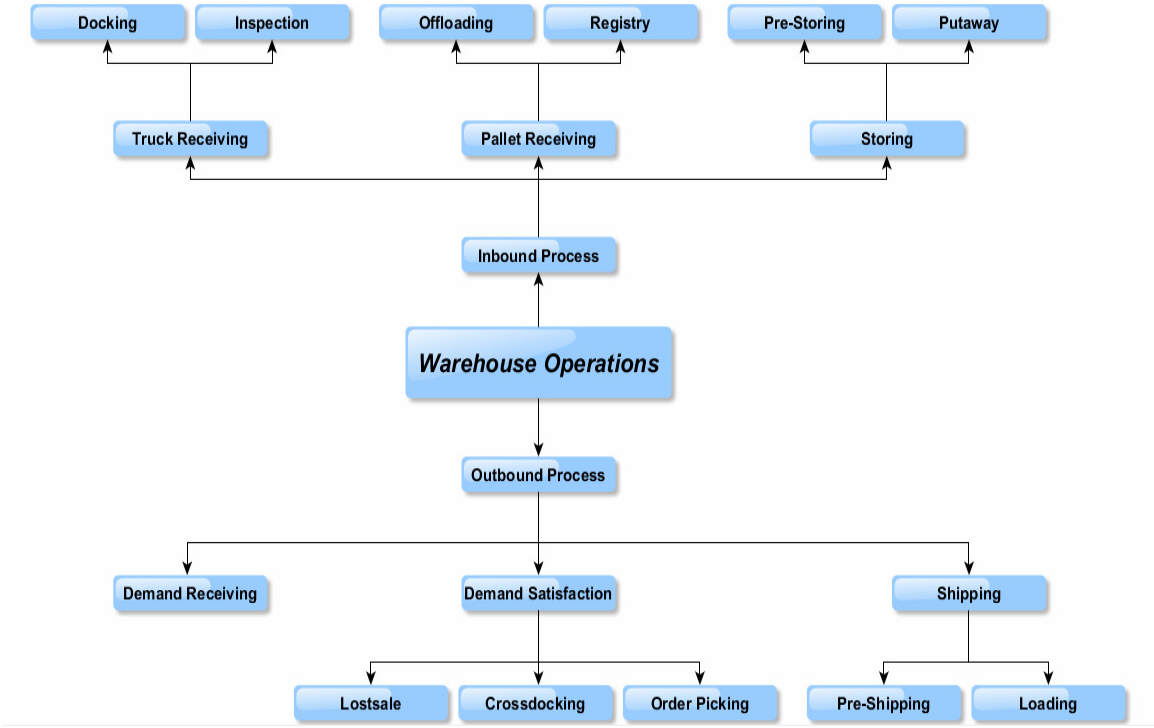

Figure 4-1: Decomposition of Warehouse Operations

4.2 Inbound Process

Inbound process is one of the main flows in warehouses. It starts with the arrival of the incoming trucks and ends when a pallet is laid on its storage location. Inbound process of our model is presented as an Event Graph in Figure 4-2.

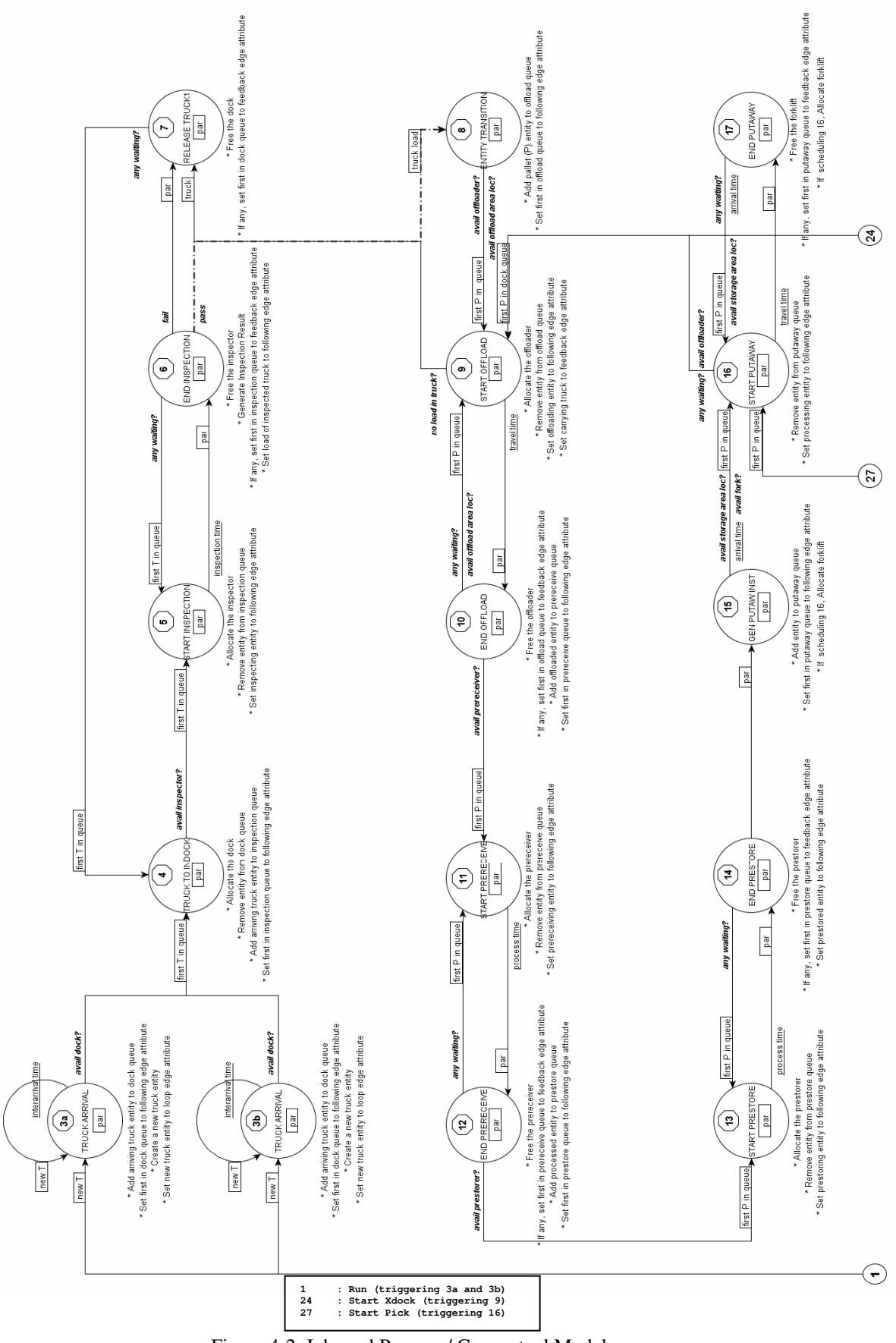

Figure 4-2: Inbound Process / Conceptual Model

In Figure 4-1, inbound process is categorized into some sub-processes. Those sub-processes are truck receiving, pallet receiving and storing. Truck receiving sub-process is grouped into two components, namely docking and inspection components.

4.2.1 Docking Component

Docking component is shown in Figure 4-3 and includes *Truck_Arrival* and *Truck_to_Receiving_Dock* events. In the model, two *Truck_Arrival* events are included to demonstrate that multi-arrival can be defined in the model to manipulate different kinds of arrival patterns.

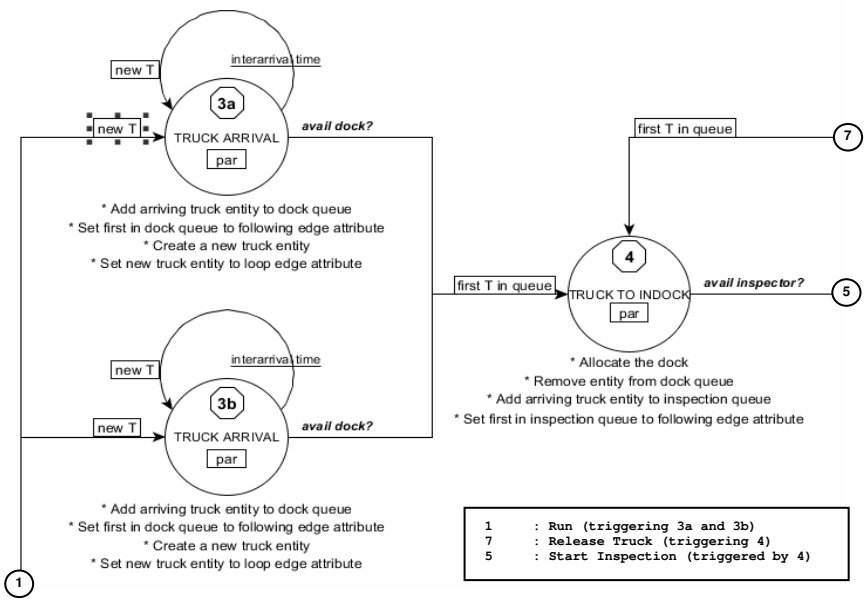

Figure 4-3: Docking Component / Conceptual Model

Truck_Arrival events are represented by the node number 3a and 3b in the EG model. *Truck_Arrival* event represents the arrival of a truck. *Truck_Arrival* event triggers *Truck_to_Receiving_Dock* event and itself with a self loop. There is a condition up to dock availability between arrival event and docking event. Truck entities waiting for an available dock are kept in the dock queue. The first truck

entity in the dock queue is passed to *Truck_to_Receiving_Dock* event. A new truck entity is created in this event and passed to loop for triggering a new arrival. Pallet entities that are carried by truck entities are created together with truck entities. This loop has a delay which represents inter arrival time. *Truck_Arrival* events can be multiplied according with arrival needs. Besides any *Truck_Arrival* events can be thought as sub-components. They can be removed from the model or new truck arrival events can be added to the model.

Truck_to_Receiving_Dock is represented by the node number 4 in the EG model. *Truck_to_Receiving_Dock* event triggers a *Start_Inspection* event up to inspector availability condition. Truck entities waiting for inspection are kept in inspection queue. The first truck entity in the inspection queue is passed to following event.

4.2.2 Inspection Component

Inspection component is shown in Figure 4-4 and includes start inspection, end inspection and release truck events.

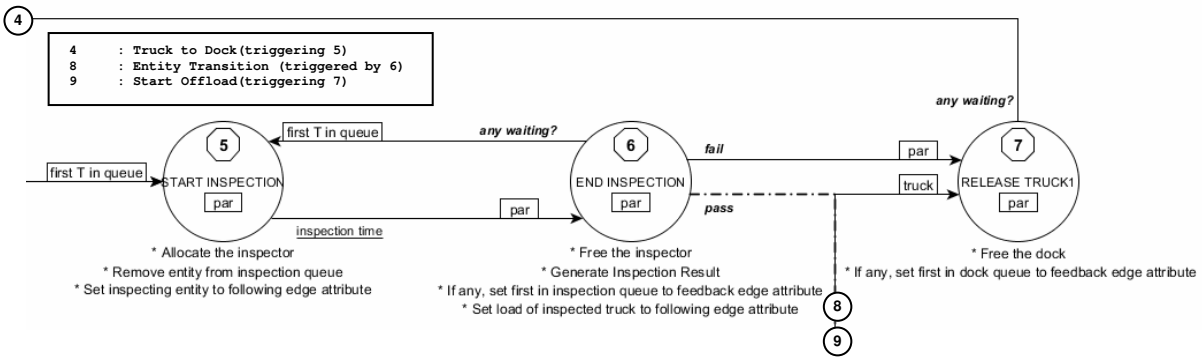

Figure 4-4: Inspection Component / Conceptual Model

Start_Inspection is represented by the node number 5 in the EG model. *Start_Inspection* event triggers an *End_Inspection* event by passing its parameter. Inter event time is set to inspection time.

End_Inspection is represented by the node number 6 in the EG model. If truck entity fails the inspection, then a *Release_Truck* event is triggered, otherwise a *Truck*_to_*Pallet_Transition* event is triggered for each pallet entity associated with that truck entity. This kind of passing is called "multi passing" and it is shown with a dashed line. Since an inspector is going to be available at *End_Inspection* event, a *Start_Inspection* event is triggered up to inspection queue condition. The first entity in the queue is passed to *Start_Inspection* event.

Release_Receiving_Truck is represented by the node number 7 in the EG model. Since a dock is available with the release of a truck entity, a Truck to Receiving Dock event is triggered up to dock queue condition. The first entity in the dock queue is passed to Truck to Receiving Dock event.

Pallet receiving sub-process is also grouped into two components, namely offloading and registry.

4.2.3 Offloading Component

Offloading component is shown in Figure 4-5 and includes entity transition, start offload and end offlaod events.

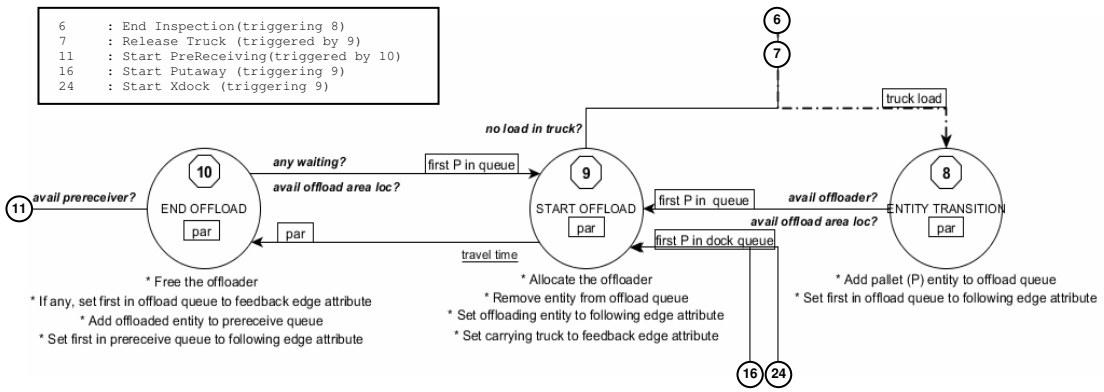

Figure 4-5: Offloading Component / Conceptual Model

Truck_to_Pallet_Transition is represented by the node number 8 in the EG model. By the use of multi-passing and this transition event, more than one event is scheduled to future event list in return of one event. This event triggers a *Start_Offload* event up to off-loader and offloading area availability conditions. It passes its pallet entity parameter to following event. Pallet entities waiting for offloading are kept in offload queue.

Start_Offload is represented by the node number 9 in the EG model. All pallets are offloaded from truck to offloading area. This event triggers an *End_Offload* event by passing its parameter. Inter event time is computed according to the distance between dock and allocated offloading location cell. *Start_Offload* event also triggers a *Release_Receiving_Truck* event in case offloaded pallet is the last pallet in the truck.

End_Offload is represented by the node number 10 in the EG model. When a pallet is laid on offloading area, pre-receiving operations start. Accordingly, this event triggers a *Start_PreReceiving* event up to pre-receiver availability condition. Besides, a *Start_Offload* event is triggered up to offload queue condition. The first entity in the offload queue is passed to *Start_Offload* event. Pallet entities waiting for prereceiving are kept in pre-receive queue.

4.2.4 Registry Component

Registry component is shown in Figure 4-6 and includes start pre-receiving and end pre-receiving events.

Start_PreReceiving is represented by the node number 11 in the EG model. Prereceiving function represents some primary operations such as labeling before putaway process. This event triggers an *End_PreReceiving* event with a delay of

process time and passes its pallet entity to *End_PreReceiving* event.

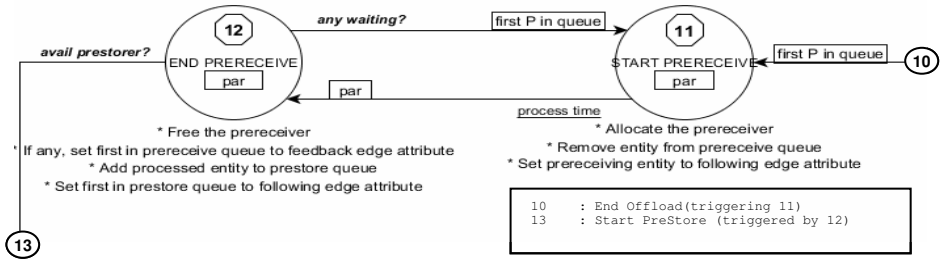

Figure 4-6: Registry Component / Conceptual Model

End_Prereceiving is represented by the node number 12 in the EG model. *End_PreReceiving* event triggers *Start_PreStoring* event up to pre-storer availability condition. Besides, a *Start_PreReceiving* event is triggered up to pre-receiving queue condition. The first entity in the pre-receiving queue is passed to *Start_PreReceiving* event. Pallet entities waiting for prestoring are kept in pre-storing queue.

Storing sub-process is grouped into pre-storing and putaway components.

4.2.5 Pre-storing Component

Pre-storing component is shown in Figure 4-7 and includes start pre-storing and end pre-storing events.

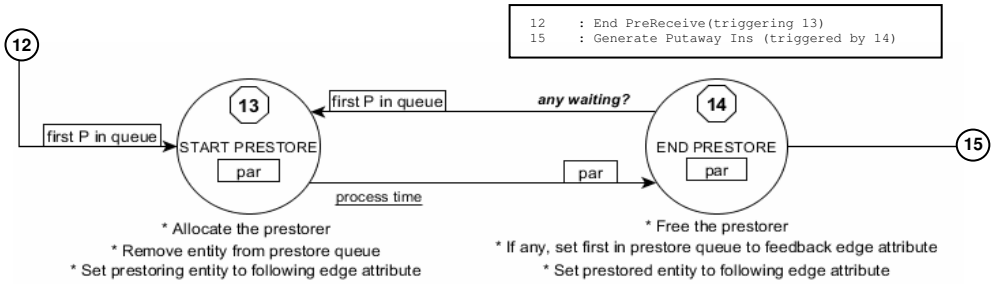

Figure 4-7: Pre-Storing Component / Conceptual Model

Start_Prestoring is represented by the node number 13 in the EG model. Pre-Storing process represents secondary operations in the offloading area before putaway process. This event triggers an *End_PreStoring* event with a delay of

process time and passes its pallet entity to *End_PreStoring* event.

End_Prestoring is represented by the node number 14 in the EG model. *End_PreStoring* event triggers *Generate_Putaway_Instructions* event. Besides, a *Start_PreStoring* event is triggered up to pre-storing queue condition. The first entity in the pre-storing queue is passed to *Start_PreStoring* event.

4.2.6 Putaway Component

Putaway component is shown in Figure 4-8 and includes generate putaway instructions, start putaway and end putaway events.

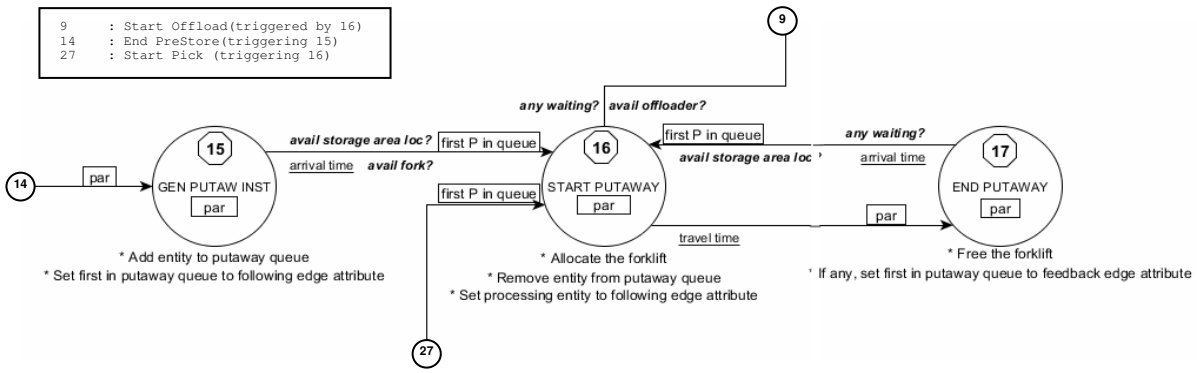

Figure 4-8: Putaway Component / Conceptual Model

Generate_Putaway_Instructions is represented by the node number 15 in the EG model. *Generate_Putaway_Instructions* event triggers a *Start_Putaway* event up to putaway forklift and storage location availability conditions with a delay of assigned forklift arrival time. It passes its pallet entity parameter to following event. Items are carried by forklifts one by one. Pallet entities waiting for putaway are kept in putaway queue.

Start_Putaway is represented by the node number 16 in the EG model. *Start_Putaway* event triggers an *End_Putaway* event with a delay of the transportation time between offloading area location and allocated storage location. It

passes its pallet entity parameter to *End_Putaway* event. Since starting to putaway will free one of an offloading area location, *Start_Putaway* event also triggers a *Start_Offload* event up to offloader availability and offloading queue conditions.

End_Putaway is represented by the node number 13 in the EG model. *End _Putaway* event is the last event in the inbound process. It triggers a *Start_Putaway* event up to putaway queue condition and the first entity in the putaway queue is passed to *Start_Putaway* event.

4.3 Outbound Process

Outbound process starts with the arrival of demand and includes some other internal processes to satisfy the demand. It is presented in Figure 4-9. In the decomposition presented in Figure 4-1, outbound process is categorized into demand receiving component and demand satisfaction and shipping sub-processes.

4.3.1 Demand Receiving Component

Demand receiving component is shown in Figure 4-10 and includes demand arrival, truck to shipping dock and entity transition events. In the model double demand arrival events are utilized to demonstrate that multi-arrival can be defined in the model to manipulate different type of demand arrival patterns.

Demand_Arrival is represented by the node number 18a and 18b in the EG model. *Demand_Arrival* event represents the arrival of a demand. It triggers a *Truck_to_Shipping_Dock* event and itself with a self loop. There is a condition up to dock availability between arrival event and docking event. Truck entities associated waiting for dock are kept in dock queue. The first demand entity in the dock queue is passed to *Truck_to_Shipping_Dock* event. A new demand entity is created in this

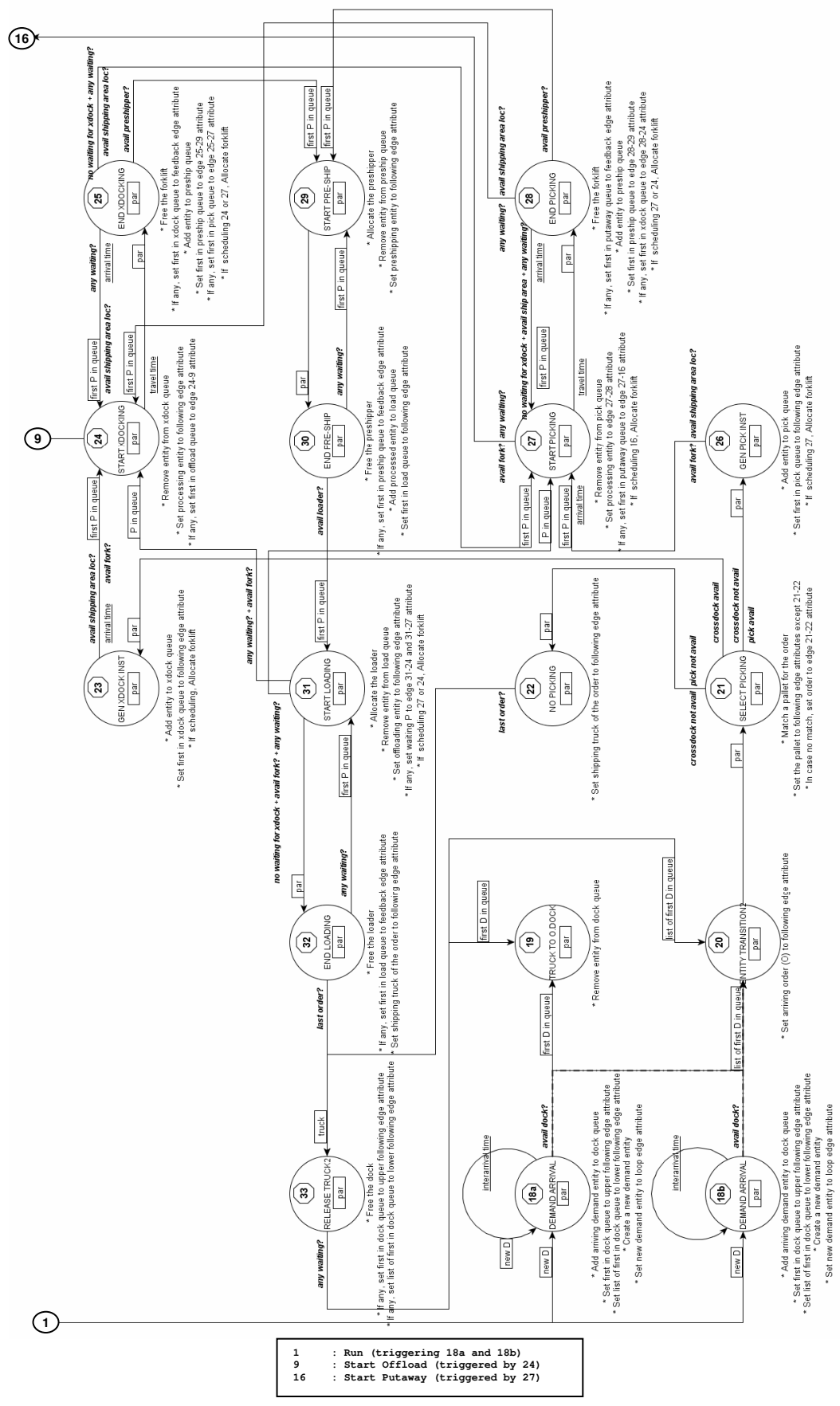

Figure 4-9: Outbound Process / Conceptual Model

event and passed to loop for triggering a new arrival. List of order entities which constitute demand entities are created together with demand entities. This loop has a delay which represents inter arrival time. *Demand_Arrival* events can be multiplied according to arrival needs. *Demand_Arrival* event also triggers *Demand_to_Order_Transition* event up to dock availability condition. Besides any demand arrival events can be thought as sub-components. They can be removed from the model or new demand arrival events can be added to the model easily.

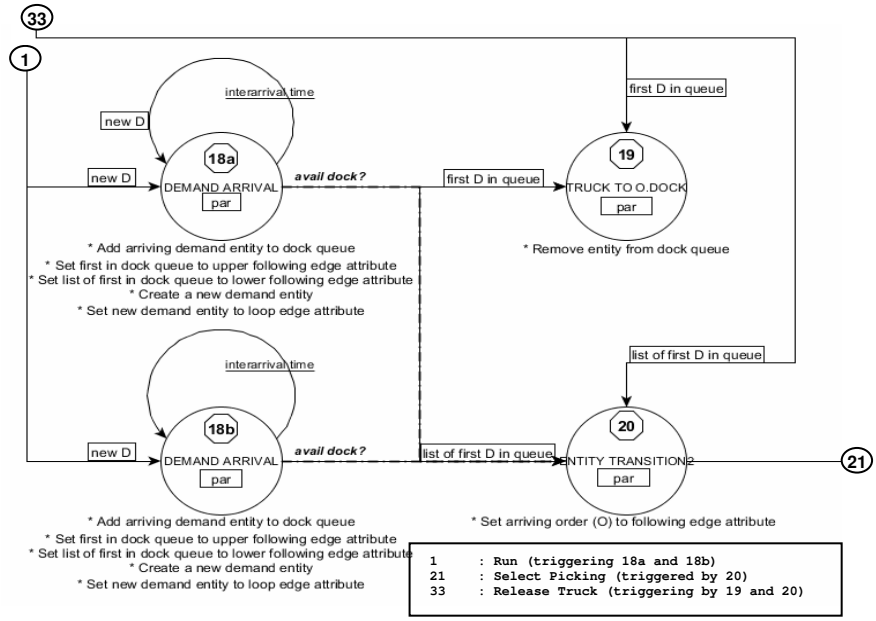

Figure 4-10: Demand Receiving Component / Conceptual Model

Truck_to_Shipping_Dock is represented by the node number 19 in the EG model. Each demand is shipped with a separate shipping truck.

Demand_to_Order_Transition is represented by the node number 20 in the EG model. Demands are a set of orders and each order in a demand order list should be treated one by one. Multi-passing between *Demand_Arrival* event and *Demand_to_Order_Transition* event provide the triggering of a number of orders in return of an arriving demand.

Demand satisfaction sub-process is grouped into crossdocking, picking and no

picking components. Here, the picking type is determined and if possible it is satisfied. Offloading area is primarily checked for an available item to satisfy demand by crossdocking. If crossdocking is not possible, then the storage area is checked for picking. In case, order can not be satisfied by both functions, no picking occurs. For this model, selecting a picking type and no picking are assumed to be a mid-component under demand satisfaction.

4.3.2 Selecting a Picking Type and No Picking

Picking type is selected to satisfy demand. If demand can not be satisfied, then no picking occurs. This is shown in Figure 4-11.

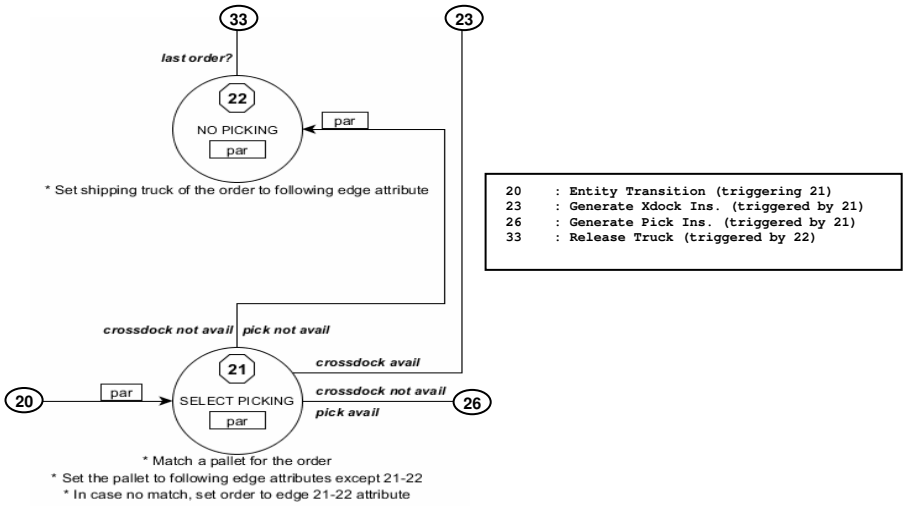

Figure 4-11: Selecting a Picking Type Component / Conceptual Model

Select_Picking is represented by the node number 21 in the EG model. An order can be satisfied directly from offloading area (crossdocking) and if there is no available pallet in offloading area, it can be picked from storage area or in case there is no convenient pallet to satisfy the order, order is rejected. Type of picking is selected here. According to the result of this check, this event triggers one of three events; *Generate_Crossdocking_Instructions*, *Generate_Picking_Instructions* or *No_Picking*. Order entity is passed as the parameter.

No_Picking is represented by the node number 22 in the EG model. No Picking event triggers a *Release_Shipping_Truck* event in case all other transactions of relevant truck already completed.

4.3.3 Crossdocking Component

Crossdocking component is shown in Figure 4-12 and includes generate crossdocking instructions, start crossdocking and end crossdocking events.

Generate_Crossdocking_Instructions is represented by the node number 23 in the EG model. *Generate_Crossdocking_Instructions* event triggers a *Start_Crossdocking* event up to crossdocking forklift and shipping area availability conditions with a delay of assigned forklift arrival time. It passes its order entity parameter to following event. Items are carried by forklifts one by one. Pallet entities waiting for crossdocking are kept in crossdocking queue.

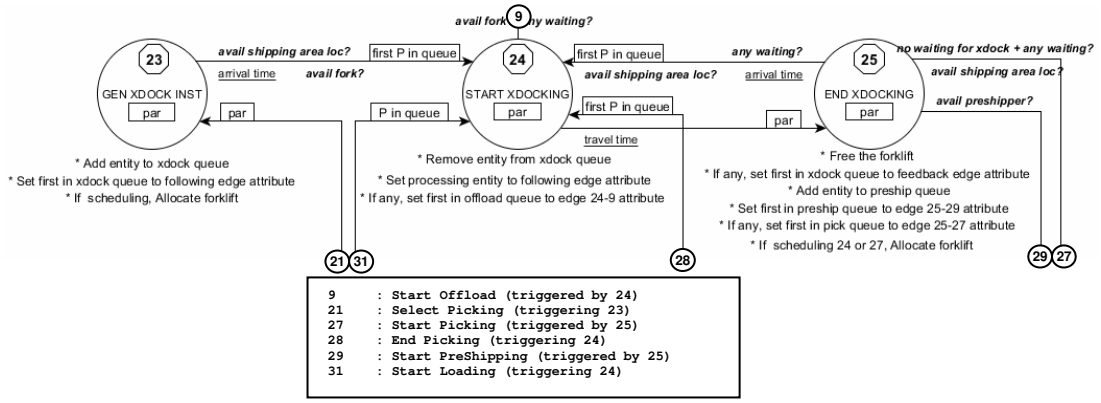

Figure 4-12: Crossdocking Component / Conceptual Model

Start_Crossdocking is represented by the node number 24 in the EG model. *Start_Crossdocking* event triggers an *End_Crossdocking* event with a delay of the transportation time between offloading area location and allocated shipping area location. It passes its pallet entity parameter to *End_Crossdocking* event. Since starting to crossdocking will free one of an offloading area location,

Start_Crossdocking event also triggers a *Start_Offload* event up to offloader availability and offloading queue conditions.

End_Crossdocking is represented by the node number 25 in the EG model. This event triggers a *Start_PreShipping* event up to preshipper availability condition. It passes its pallet entity parameter to *Start_PreShipping* event. *End_Crossdocking* event also triggers a *Start_Crossdocking* or *Start_Picking* events up to queue conditions and the first entities in the queues are passed to this event.

4.3.4 Picking Component

Picking component is shown in Figure 4-13 and includes generate picking instructions, start picking and end picking events.

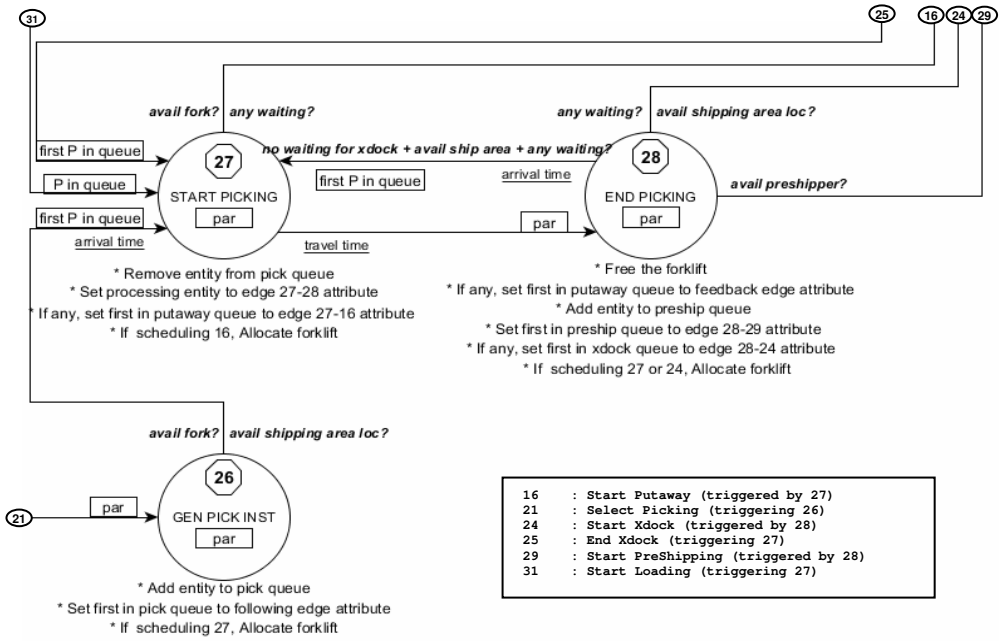

Figure 4-13: Picking Component / Conceptual Model

Generate_Picking_Instructions is represented by the node number 26 in the EG model. *Generate_Picking_Instructions* event triggers a *Start_Picking* event up to picking forklift and shipping area availability conditions with a delay of assigned

forklift arrival time. It passes its order entity parameter to following event. Items are carried by forklifts one by one. Entities waiting for picking are kept in picking queue.

Start_Picking is represented by the node number 27 in the EG model. *Start_Picking* event triggers an *End_Picking* event with a delay of the transportation time between storage area location and allocated shipping area location. It passes its pallet entity parameter to *End_Picking* event. Since starting to picking will free one of storage area locations, *Start_Picking* event also triggers a *Start_Putaway* event up to putaway forklift availability and putaway queue conditions.

End_Picking is represented by the node number 28 in the EG model.This event triggers a *Start_PreShipping* event up to preshipper availability condition. It passes its pallet entity parameter to *Start_PreShipping* event. *End_Picking* event also triggers a *Start_Picking* or *Start_Crossdocking* events up to queue conditions and the first entities in the queues are passed to this event.

Shipping sub-process is grouped into pre-shipping and loading components.

4.3.5 Pre-shipping Component

Pre-shipping component is shown in Figure 4-14 and includes start pre-shipping and end pre-shipping events.

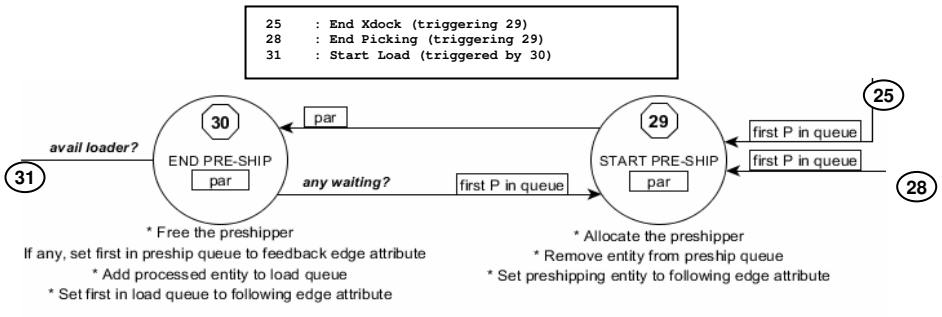

Figure 4-14: Pre-shipping Component / Conceptual Model

Start_PreShipping is represented by the node number 29 in the EG model. Pre-

Shipping process represents some activities before loading process. This event triggers an *End_PreShipping* event with a delay of process time and passes its pallet entity to *End_PreShipping* event.

End_PreShipping is represented by the node number 30 in the EG model. *End_Pre-Shipping* event triggers *Start_Loading* event. Besides, a *Start_PreShipping* event is triggered up to pre-shipping queue condition. The first entity in the pre-shipping queue is passed to *Start_PreShipping* event.

4.3.6 Loading Component

Loading component is shown in Figure 4-15 and includes start loading, end loading and release truck events.

Start_Load is represented by the node number 31 in the EG model. All pallets are loaded from shipping area to truck. This event triggers an *End_Load* event by passing its parameter. Inter event time is computed according with the distance between shipping location cell and dock. *Start_Load* event also triggers a *Start_Crossdocking* or *Start_Picking* event up to their queue conditions since a location in the shipping area gets free.

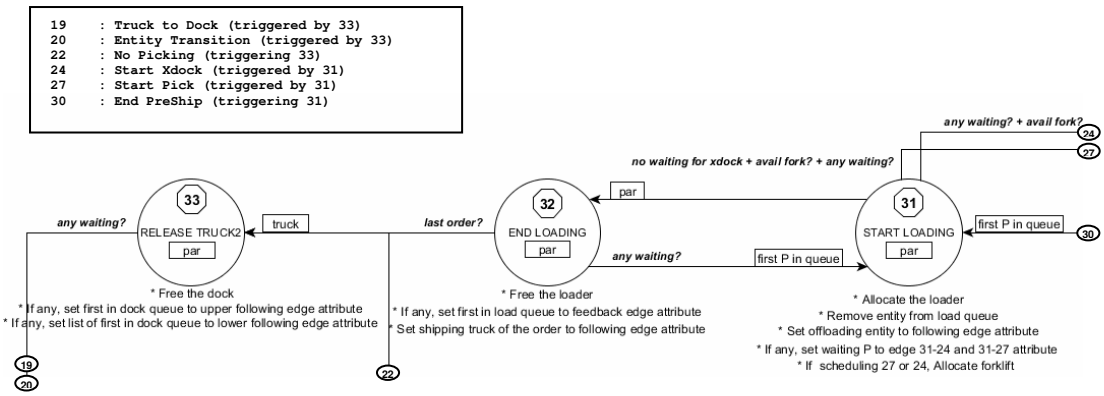

Figure 4-15: Loading Component / Conceptual Model

End_Load is represented by the node number 32 in the EG model. *End_Load*

event triggers a *Release_Shipping_Truck* event in case loaded pallet is the last pallet in the truck.

Release_Shipping_Truck is represented by the node number 33 in the EG model. Since a dock will be available with the release of a truck entity, a *Truck_to_Shipping_Dock* and a *Demand_to_Order_Transition* events are triggered up to dock queue condition. The first entity in the dock queue is passed to following events.

4.4 General Events

This section covers events required by simulation logic. These events are *Run*, *Ping* and *Terminate*. *Run* event is the sparking plug for the model. It triggers arrivals of incoming materials and demands and thus simulation starts. Furthermore, initial values are set and first arriving entities are created at the execution of *Run* event. *Ping* event is associated with animation and triggered by *Run* event. It triggers itself after a deterministic delay. In each *Ping* event, the positions of moving items are upgraded and thereby regular move of items is shown on animation. *Terminate* event is used to stop simulation. It is not shown in the model. *Run* and *Ping* events are shown in Figure 4-16.

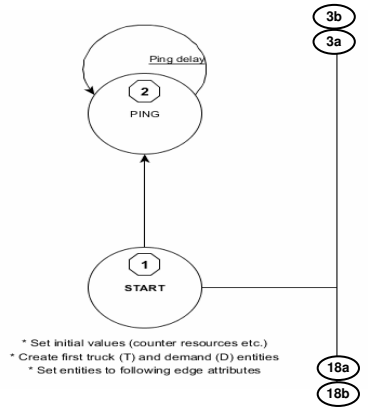

Figure 4-16: General Events / Conceptual Model

4.5 Implementation

To develop our analysis tool, we have implemented the conceptual model defined above as a Visual CSharp project. The project refers two main libraries; SharpSim and WareLib which were introduced in previous chapters. SharpSim provides the required simulation environment for the implementation of the conceptual model, while WareLib is used to simulate a generic warehouse required for the simulation. The analysis tool offers a user-friendly interface for easy handling of the program.

Main properties of the model are a *Simulation* type simulation object provided by SharpSim and a *Warehouse* type warehouse object provided by WareLib. Mainly these two objects interact with each other on implementation form of the project.

Simulation model is constituted by instantiating *Simulation, Event* and *Edge* objects as explained in Chapter 2. Warehouse environment is defined by Excel inputs. Sample layout is presented in Figure 4-17. The meanings of the abbreviations used in layout spreadsheet are as follows. SL: Storage Location, M: Main Aisle, C: Cross Aisle, CC: Critical Cells, O: Offloading Area Location, S: Shipping Area Location, ID: Inbound Dock and OD: Outbound Dock. These abbreviations are required by WareLib for recognition.

There are some other spreadsheets required for dedicating locations for item types, setting traffic direction along the aisles and setting initial configuration of pallets inside the warehouse. Truck load information is set through implementation form.

The interface of the project has been designed to handle the model easily. The interface is mainly composed of three forms. Main form (GWS: Generic Warehouse Simulation), warehouse and simulation forms. The interface is shown in Figure 4-18.

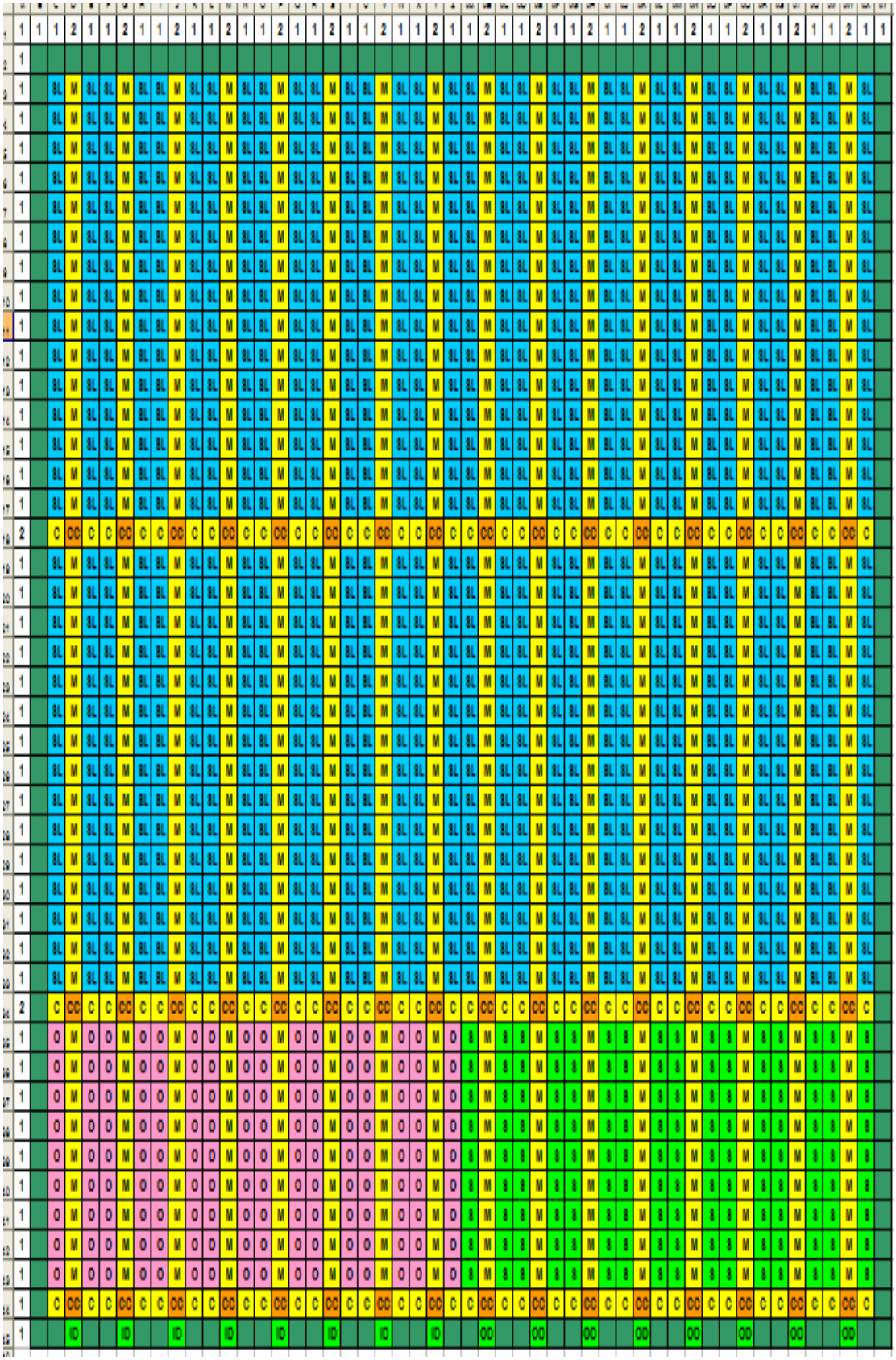

Figure 4-17: Layout - Spreadsheet

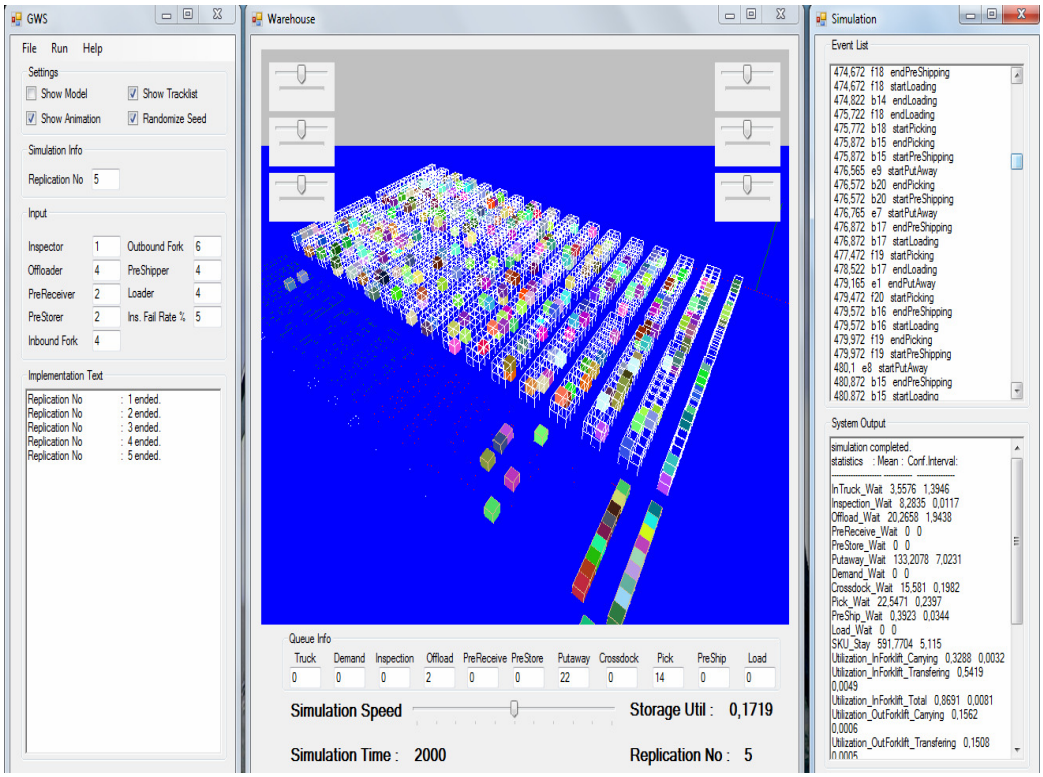

Figure 4-18: Interface

Main form is the first form displayed when the tool is functioned. It has three strip menus. Strip menu File is used to deploy the tool with simulation input and warehouse input through Excel files (See Figure 4-17). Strip menu Run has items run and exit. Last strip menu is Help. The checkboxes on the main form are used to enable/disable properties of the tool which are related with animating warehouse objects, writing executed events to text control and randomizing seed. Furthermore, the number of handling resources such as number of forklifts can be set through this form.

When the animation option is enabled on the main form, warehouse form is displayed. The warehouse form hosts display control. Display control provides a 3D animation of the warehouse model. Camera position, camera view and animation speed can be changed with the track bars placed on the form. Furthermore, warehouse utilization and handling resources' queue information are displayed

momentarily with the form.

Simulation form has two text controls. The first one displays executed event lists in case relevant checkbox is enabled on the main form. Second control is used to display simulation output values.

The tool provides some statistics such as resource utilizations (e.g. inbound forklift utilization), queue information (average truck wait) and number of transactions (e.g. number of putaways). In addition, since each entity is tracked during the simulation, average time spent by passing-by entities between any event couple (e.g. average warehouse stay for pallets) can be obtained from the model.

4.6 Verification

As stated in Chapter 2, verification is to make sure that the implementation represents the conceptual model. The same methods explained in Chapter 2 have been applied here for the verification of our model. First, the implementation has been controlled line by line. Next, state change methods associated to each event (e.g. start offloading, start putaway) has been observed whether they produce expected results. Later, the implementation of each component (e.g. putaway, picking) of the conceptual model has been checked. Finally, the whole implementation of the conceptual model has been considered and input – output relations have been verified. In addition, entity and resource behaviors have been observed. In verifying the model, animation gave us a valuable insight as one of the most powerful and important verification tools.

4.7 Validation

The validation of the model has been conducted with a sensitivity analysis and

extreme condition tests by observing model responses to input changes and extreme input values. For the extreme condition tests, each handling resource's availability (e.g. offloaders, putaway forklifts, picking forklifts, preshippers, loaders) has been set to 0 respectively. Each time, the model has been blocked up to shortage of this resource. For the sensitivity analysis, model inputs such as layout, number of available resources, arrival and demand patterns have been changed and model responses have been observed. In these tests, model always behaved as expected.

Finally, some inferences made in the literature have been tested with our model. In Gu et al. (2007), it is indicated that material handling systems can be used in a more effective way with dedicated storage policy, since fast-moving items can be placed close to docks. This inference has been tested with our model and the model has produced lower forklift utilization with dedicated storage policy comparing to random storage policy. In Bartholdi (2004), it is stated that the center doors of an I-shaped warehouse are the most convenient doors in terms of forklift utilization for crossdock warehouses. Our model has produced same inference by generating lower forklift utilization for central docks.

4.8 About The Limitations of The Analysis Tool

The analysis tool is developed for unit-load warehouses operating with single item putaway and picking policies. Some amendments in the tool are necessary when the granularity of items in warehouse is decreased such as packages and/or individual products in pallets, and when the policies for putaway and picking operations are changed.

For less than unit-load transactions, first of all, a Package entity should be introduced to the warehouse library. Besides, the Warehouse object should be

upgraded to track each package entity and to provide required package level information when requested. Obviously, less than unit-load transactions will require some changes on the conceptual model either. These changes will mainly be about picking function.

For multi-item putaway and picking policies, the putaway and picking functions of the conceptual model should be altered. When multi-item putaway and picking policies are adopted, pickers do not travel just between two points but travel among multi points. In this case, we face a Travelling Salesman Problem (TSP). This problem can be solved with some heuristics at each transaction or some routing policies can be applied directly. The warehouse library should be upgraded to respond these additional needs either.

5 EXPERIMENTATION

5.1 Introduction

In this chapter, we present our experiments which we designed to investigate the effects of dock positioning on forklift utilizations. The model presented in the previous chapters is used for this reason in the experiments. We created four different warehouse designs, namely I-Shape, U-Shape, L-Shape and O-Shape and pointed out the best performing designs under the same input conditions.

5.2 Problem Definition

We created four different warehouse layout designs, namely I-Shape, U-Shape, L-Shape and O-Shape for these experiments. Each warehouse design has a number of candidate dock positions. By using these candidate dock positions, we created a set of configurations for each warehouse designs in which two cases are considered; crossdocking_not_adopted (CNA) and crossdocking_adopted (CA) cases. CNA case includes receiving, putaway, picking and shipping functions of a warehouse whereas CA case adopts crossdocking additional to these functions. CA case is represented with the model explained in chapter 5. CNA case is obtained with the removal of crossdocking component and all connecting edges from the model. In CNA case, putaway transactions are executed by inbound forklifts and picking transactions by outbound forklifts. In CA case, crossdocking transactions are executed by outbound forklifts either.

5.3 Assumptions

Only unit-load transactions are executed in the warehouse. Unit-load refers a pallet load. In every putaway and picking tour, only one pallet is carried by the

forklifts. Single deep storage system is used. Arriving items are placed in the warehouse randomly. To satisfy an order, the oldest item is picked from the storage area, Putaway and picking transactions are done separately. In every tour, one of these functions can be executed by a forklift. Interarrival times of the arriving trucks and demands are exponentially distributed.

5.4 Inputs

Inputs of the model are warehouse designs, truck arrival patterns, truck loads (pallets), demand arrival patterns, demand lists (orders) and number of resources.

Warehouse designs are prepared as spreadsheets as required by the WareLib. Each warehouse design has a set of candidate dock positions. The designs and the candidate dock positions are given in Figure 5-1.

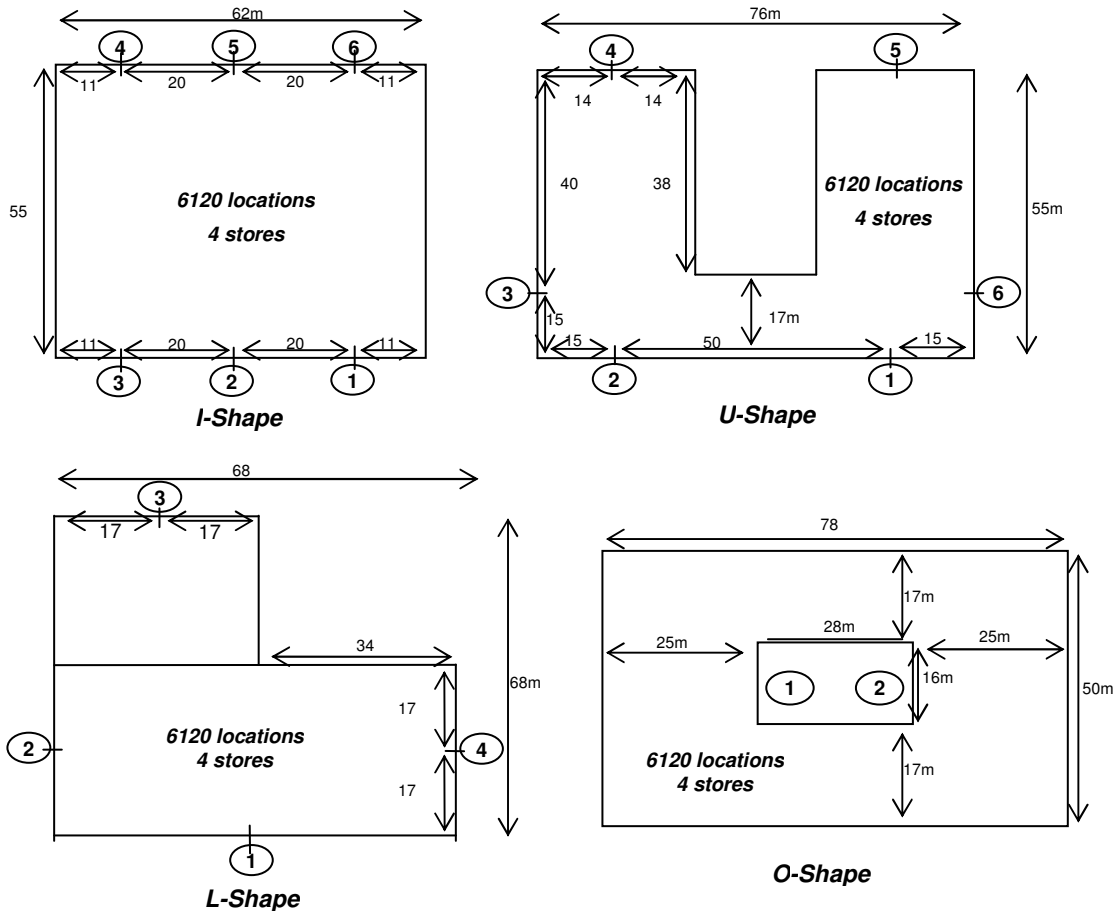

Figure 5-1: Warehouse Designs

There are 10 different trucks carrying 20 different types of items. Totally, there are 200 types of items processed. On the other hand, there are 10 different demands and each demand has a list involving 20 items. The inter arrival times of trucks and demands are exponentially distributed. Time unit is minute. (See Table 5-1 and Table 5-2)

Truck No	Load	The Parameter of The Distribution
1	A1 to A20	600
2	B1 to B20	600
3	C1 to C20	600
4	D1 to D20	600
5	E1 to E20	600
6	F1 to F20	600
7	G1 to G20	600
8	H1 to H20	600
9	I1 to I10	600
10	J1 to J20	600

Table 5-1: Truck Information

Demand No	List	The Parameter of The Distribution
1	A1 to A20	600
2	B1 to B20	600
3	C1 to C20	600
4	D1 to D20	600
5	E1 to E20	600
6	F1 to F20	600
7	G1 to G20	600
8	H1 to H20	600
9	I1 to I10	600
10	J1 to J20	600

Table 5-2: Demand Information

Each design has the same number of storage locations (6120). Putaway function is executed by inbound forklifts and there are 4 inbound forklifts. Picking and crosdocking functions are executed by outbound forklifts and there are 6 outbound

forklifts. Arriving trucks are inspected at the dock and some are rejected. Inspection fail rate is 5%. There are some other resources which are material handling resources such as inspectors, offloaders, pre-receivers, pre-shippers and location resources such as offloading area capacity. These resources are given values not to create any bottleneck and not to affect the value of performance criteria.

5.5 Outputs

Observed outputs of the model are Inbound Forklift Utilization (IFU), Outbound Forklift Utilization (OFU), Number of Staisfied Orders, Number of Putaways, Number of Crossdocks, Number of Picks, Average Truck Wait and Average Demand Wait. Interms of IFU and OFU; a forklift is accepted to be busy when it is assigned to a new job and idle when it completes the transaction,

5.6 Scenarios

Dock configurations are presented in Table 5-3, Table 5-4 and Table 5-5. These scenarios should be read with Figure 5-1. As mentioned before, each warehouse has some candidate dock positions and we have created a set of configurations with them. Table 5-3 shows the configurations having one receiving and one shipping dock. I-Shape has 8, U-Shape has 15 and L-Shape has 7 configurations in this table. For example, in I-Shape's 1st configuration, receiving dock is positioned at dock_no_1 and shipping dock is positioned at dock_no_2 (See Figure 5-1). Table 5-4 includes the configurations having one receiving, two shipping docks and Table 5-5 includes the configurations having two receiving, one shipping docks.

For O-Shape design, there is only one configuration, since there are just two candidate dock positions and they are symmetrical. All dock configurations are experimented with two cases as mentioned above. As a result, totally 130 scenarios

are created for experimentation.

Design	*Dock Type*	*Dock Configurations*														
		1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
I Shape	Receiving	1	1	1	1	1	2	2	2	-	-	-	-	-	-	-
	Shipping	2	3	4	5	6	1	4	5	-	-	-	-	-	-	-
U Shape	Receiving	1	1	1	1	1	3	3	3	3	3	4	4	4	4	4
	Shipping	2	3	4	5	6	1	2	4	5	6	1	2	3	5	6
L Shape	Receiving	1	1	2	2	3	3	3	-	-	-	-	-	-	-	-
	Shipping	2	3	1	3	1	2	4	-	-	-	-	-	-	-	-

Table 5-3: One to One Configurations (C11)

Design	*Dock Type*	*Dock Configurations*						
		1	*2*	*3*	*4*	*5*	*6*	*7*
I Shape	Receiving	1	1	1	1	2	2	2
	Shipping	2-3	3-4	4-5	5-6	1-3	3-4	4-5
U Shape	Receiving	1	1	1	3	3	3	-
	Shipping	2-3	3-4	5-6	1-2	1-6	5-6	-
L Shape	Receiving	1	2	3	3	-	-	-
	Shipping	2-3	1-4	1-2	1-4	-	-	-

Table 5-4: One to Two Configurations (C12)

Design	*Dock Type*	*Dock Configurations*						
		1	*2*	*3*	*4*	*5*	*6*	*7*
I Shape	Receiving	2-3	3-4	4-5	5-6	1-3	3-4	4-5
	Shipping	1	1	1	1	2	2	2
U Shape	Receiving	2-3	3-4	5-6	1-2	1-6	5-6	-
	Shipping	1	1	1	3	3	3	-
L Shape	Receiving	2-3	1-4	1-2	1-4	-	-	-
	Shipping	1	2	3	3	-	-	-

Table 5-5: Two to One Configurations (C21)

5.7 Simulation Settings

Time unit is minute. The simulation run length is 500000 simulation time. This equals almost to a year. Warm-up period is computed as 50000 by using Welch

method. Number of replications is set to 10 considering inbound and outbound forklift utilizations. (See Figure 5-2 and Figure 5-3)

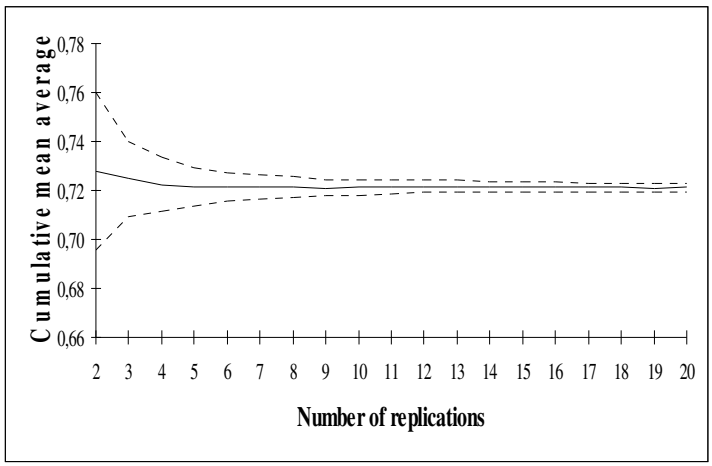

Figure 5-2: IFU / Determining Number of Replications

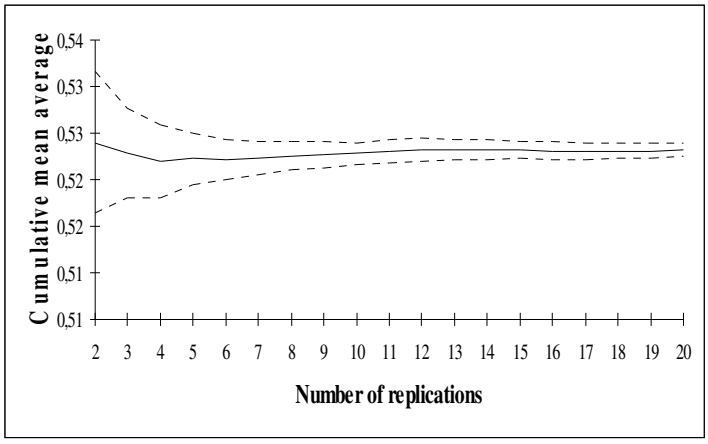

Figure 5-3: OFU / Determining Number of Replications

5.8 Results

Since we aim to make some general inferences, in all our experiments, our results are presented in a relative manner, not in an absolute manner.

5.8.1 One to One Configurations (C11)

In these configurations, there is one receiving and one shipping dock.

5.8.1.1 I-Shape

No bottleneck detected in C11 configurations of I-Shape design. In all configurations of CNA and CA cases, numbers of satisfied orders are close to each other and high, average queue waiting times are low and reasonable. Consequently, all the utilization values are realistic. Therefore, an accurate evaluation can be made over utilizations.

The receiving dock is positioned at dock_no_1 in configurations 1 to 5 (Group A) and at dock_no_2 in configurations 6 to 8 (Group B). When the receiving dock is positioned at a more centralized position, average travel distance reduces, thus inbound forklifts are utilized more effectively. Consequently, average number of pallets waiting in the offloading area for putaway is decreased. This reduces the number of crossdocks. The rate of number of crossdocks to number of satisfied orders for Group A and Group B is shown in Table 5-6.

Configuration	***Rate***
Group A	11.5%
Group B	7.5%

Table 5-6: Rate of # Crossdocks to # Satisfied Orders / C11 / I-Shape

Group B produce lower inbound forklift utilization (IFU) in both cases comparing to Group A, since dock_no_2 is closer to center. IFU changes are presented in Table 5-7.

Case	***Group A to Group B % Change***
CA	- 11%
CNA	- 16%

Table 5-7: IFU Changes up to Configurations / C11 / I-Shape

When crossdocking is not adopted, more putaway transactions should be done and

this increases IFU as shown in Table 5-8. The reason of the difference between two groups is the number of crossdocks as shown in Table 5-6.

Configuration	***Cross. to No Cross. % Change***
Group A	+ 14.5%
Group B	+ 8%

Table 5-8: IFU Change up to Cases / C11 / I-Shape

The shipping dock is positioned at dock_no_2 in configurations 1, 4 to 8 (Group A) and at dock_no_1 in other configurations (Group B). In CNA case, there is only one important factor that forms outbound forklift utilization (OFU). This is the position of shipping dock. OFU is 14.5% lower in Group A in CNA cases comparing to Group B.

In CA case, there are three important factors that form OFU. These are the position of shipping dock, the position of receiving dock (which affects number of crossdocks) and the relative positions of receiving and shipping docks (which affects crossdock distance). For this case, relative changes of OFU are presented in Table 5-9. OFU tends to get lower in Group A (configurations 1, 4, 8) where shipping dock is centralized. Additionally, number of crossdocks and crossdock distance also affect utilizations. With the impacts of these factors, configuration 1 produces lower utilization than others since it has shorter crossdock distance and higher number of crossdocks. Configuration 8 has the second lowest OFU value.

In CA case, configurations 1, 6 and 7 where crossdock distance is shorter than average picking distance result in lower OFU values comparing to CNA cases. The values are presented in Table 5-10. Since average crossdocking rate is about %10 (See Table 5-6), it is reasonable that these values are low.

Configurations	*To*							
from	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	---	+ 21%	+ 33%	+ 13%	+ 23%	+ 18%	+ 26%	+ 7%
2	- 18%	---	+ 10%	- 7%	+ 2%	- 3%	+ 4%	- 12%
3	- 25%	- 9%	---	- 15%	- 8%	- 12%	- 5%	- 20%
4	- 11%	+ 7%	+ 18%	---	+ 9%	+ 4%	+ 12%	- 6%
5	- 19%	- 2%	+ 8%	- 8%	---	- 4%	+ 2%	- 14%
6	- 15%	+ 3%	+ 13%	- 4%	+ 5%	---	+ 7%	- 10%
7	- 21%	- 4%	+ 6%	- 11%	- 2%	- 7%	---	- 16%
8	- 6%	+ 14%	+ 25%	+ 6%	+ 16%	+ 11%	+ 18%	---

Table 5-9: OFU Change / CA Case / I-Shape

In CA case, configurations 1, 6 and 7 where crossdock distance is shorter than average picking distance result in lower OFU values comparing to CNA cases. The values are presented in Table 5-10. Since average crossdocking rate is about %10 (See Table 5-6), it is reasonable that these values are low.

Configuration	***No Cross. To Cross. % Change***
1	- 4%
6	- 3%
7	- 3%

Table 5-10: OFU Change / C11 / I-Shape

In Figure 5-4, configuration 4 and 8 are compared. In this case, receiving dock is positioned at a more centralized position, therefore IFU gets lower. Since inbound forklifts are utilized more effectively, average number of pallets waiting in the offloading area for putaway is decreased. This reduces the number of crossdocks.

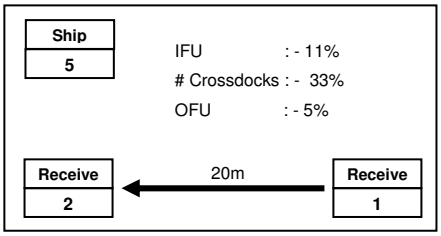

Figure 5-4: Conf 4 to 8 / C11 / I-Shape

On the other hand, moving receiving dock from position 1 to 2 decreases crossdock distance. Conclusively, OFU decrease either.

5.8.1.2 U-Shape

Some bottlenecks depending on resource inadequacy detected for C11 configurations of U-Shape design. These are 3rd, 4th, 8th, 9th and 14th configurations for CA cases, and 3rd, 4th, 8th, 9th and 11th to 15th configurations for CNA cases. In all of these configurations, one of the docks or both is positioned at dock_no_4 or dock_no_5 where average travel distances of forklifts are relatively longer. Average queue waiting times show that inbound and/or outbound forklift inadequacies depending on long travel distances cause the bottlenecks for these configurations. Consequently, we conclude that these configurations perform badly without any further analysis and they can not satisfy system needs depending on the reasons explained above.

We continue our analysis with the configurations resulted in no bottleneck. These are presented in Table 5-11. An accurate evaluation can be made over utilizations produced by these configurations.

Case	***Configurations***
CA	1,2,5,6,7,11,12,13,15
CNA	1,2,5,6,7,10

Table 5-11: Valid Configurations / C11 / U-Shape

We can categorize six dock positions of U-Shape into three groups since some positions are symmetrical. Let call Dock_no_1 and dock_no_2 as Dock_A, dock_no_3 and dock_no_6 as Dock_B and dock_no_4 and dock_no_5 as Dock_C.

The receiving dock is positioned at Dock_A in configurations 1 to 5 (Group A), at Dock_B in configurations 6 to10 (Group B) and at Dock_C in configurations 11 to

15 (Group C). The rates of number of crossdocks to number of satisfied orders for Group A, Group B and Group C are shown in Table 5-12. The reason of the difference among values is the position of the receiving dock. More centralized positions result in less number of crossdocks.

Configuration	***Rate***
Group A	12%
Group B	13%
Group C	19%

Table 5-12: Rate of # Crossdocks to # Satisfied Orders / C11 / U-Shape

Since Dock_A is closer to center, Group A produce lower IFUs in both cases comparing to Group B and C. Second good performer is Group B. IFU changes are presented in Table 5-13. Group C is out of considerations in CNA case since they result in bottleneck.

Case	***Group A to Group B % Change***	***Group A to Group C % Change***	***Group B to Group C % Change***
CA	+ 3%	+ 13%	+ 9%
CNA	+ 3.5%	-	-

Table 5-13: IFU Change up to Configurations / C11 / U-Shape

In CNA case, more putaway transactions should be done and this increases IFU as shown in Table 5-14. Group C is out of considerations in CNA case, therefore no comparison can be made.

Configuration	***Cross. to No Cross. % Change***
Group A	+ 15%
Group B	+ 16%

Table 5-14: IFU Change up to Cases / C11 / U-Shape

The shipping dock is positioned at Dock_A in configurations 1, 6, 7, 11 and 12 (Group A), at Dock_B in configurations 2, 5, 10, 13 and 15 (Group B), at Dock_C in

configurations 3, 4, 8, 9 and 10 (Group C). Again we continue our analysis with the configurations resulted in no bottleneck. For CNA cases, Group A produces %5 lower OFU comparing to Group B. Group C is out of consideration in CNA cases, because, the increase in IFU depending on absence of crossdocking function cause a bottleneck in these configurations.

In CA case, relative changes of OFU are presented in Table 5-15. OFU tends to get lower in configurations (Group A) where shipping dock is centralized. Additionally, number of crossdocks and crossdock distance also affect utilizations. With the effects of these factors, configuration 7 produces lower utilization than others since it has shorter crossdock distance and high number of crossdocks.

Configurations	***To***								
from	**1**	**2**	**5**	**6**	**7**	**11**	**12**	**13**	**15**
1	---	+ 11%	+ 1%	+ 8%	- 3%	+ 17%	+ 3%	+ 4%	+ 19%
2	- 10%	---	- 9%	- 3%	- 13%	+ 5%	- 8%	- 7%	+ 7%
5	- 1%	+ 10%	---	+ 7%	- 4%	+ 16%	+ 2%	+ 3%	+ 18%
6	- 7%	+ 3%	- 6%	---	- 10%	+ 9%	- 4%	- 3%	+ 10%
7	+ 3%	+ 15%	+ 4%	+ 11%	---	+ 21%	+ 6%	+ 7%	+ 22%
11	- 15%	- 5%	- 14%	- 8%	- 17%	---	- 12%	- 11%	+ 1%
12	- 3%	+ 8%	- 2%	+ 5%	- 6%	+ 14%	---	+ 1%	+ 15%
13	- 4%	+ 7%	- 3%	+ 3%	- 7%	+ 13%	- 1%	---	+ 14%
15	- 16%	- 6%	- 15%	- 9%	- 18%	- 1%	- 13%	- 12%	---

Table 5-15: OFU Change / CA Case / C11 / U-Shape

In CA cases, configurations 5 and 7 produce respectively %5 and %3.5 lower OFUs comparing to CNA cases. Note that average crossdocking rate is low comparing to number of satisfied orders, therefore it is reasonable that these values are also low.

5.8.1.3 L-Shape

Some bottlenecks depending on resource inadequacy detected for C11

configurations of L-Shape design. These are 7th configuration for CA cases, and 5th to 7th configurations for CNA cases. In all of these configurations, one of the docks or both is positioned at dock_no_3 or/and dock_no_4 where average travel distance of forklifts are relatively longer. Average queue waiting times show that inbound and/or outbound forklift inadequacies depending on long travel distances cause the bottlenecks for these configurations. We conclude that these configurations perform badly without any further analysis and they can not satisfy system needs depending on the reasons explained above.

We continue our analysis with the configurations resulted in no bottleneck. These are presented in Table 5-16. An accurate evaluation can be made over utilizations produced by these configurations.

Case	***Configurations***
CA	1,2,3,4,5,6
CNA	1,2,3,4

Table 5-16: Valid Configurations / C11 / L-Shape

The receiving dock is positioned at dock_no_1 in configurations 1, 2 (Group A), at dock_no_2 in configurations 3, 4 (Group B) and at dock_no_3 in configurations 5, 6 and 7 (Group C). The rates of number of crossdocks to number of satisfied orders are presented in Table 5-17. Again, centralized receiving positions reduce the number of crossdocks.

Configuration	***Rate***
Group A	8%
Group B	9%
Group C	15%

Table 5-17: Rate of # Crossdocks to # Satisfied Orders / C11 / L-Shape

Since Dock_A is closer to center, Group A produce lower IFUs in both cases comparing to Group B and C. Second good performer is Group B. IFU changes are

presented in Table 5-18. Group C is out of considerations in CNA case.

Case	*Group A to Group B % Change*	*Group A to Group C % Change*	*Group B to Group C % Change*
CA	+ 3%	+ 18%	+ 14%
CNA	+ 5%	-	-

Table 5-18: IFU Change up to Configurations / C11 / L-Shape

When crossdocking is not used more putaway transactions should be done and this increases IFU as shown in Table 5-19. Group C is out of considerations in CNA case, therefore no comparison can be made.

Configuration	*Cross. to No Cross. % Change*
Group A	+ 10%
Group B	+ 10%

Table 5-19: IFU Change up to Cases / C11 / L-Shape

The shipping dock is positioned at dock_no_1 in configurations 3, 5 (Group A), at dock_no_2 in configurations 1, 6 (Group B), at dock_no_3 in configurations 2, 4 (Group C) and at dock_no_4 in configurations 7 (Group D). Again we continue our analysis with the configurations resulted in no bottleneck. OFU changes for CNA cases are presented in Table 5-20. This table is valid only for configurations 1, 2, 3 and 4.

Comparison of Configuration Groups	*% Change*
Conf 3 to Conf 1	+ 4%
Conf 3 to Conf 2,4	+ 25%
Conf 1 to Conf 2,4	+ 22%

Table 5-20: OFU Change / CNA Case / C11 / L-Shape

In CA case, relative changes of OFUs are presented in Table 5-21. OFU tends to get lower in Group A (configurations 3, 5) where shipping dock is centralized.

Additionally, number of crossdocks and crossdock distance also affect utilizations.

Configurations	*to*					
from	**1**	**2**	**3**	**4**	**5**	**6**
1	---	+ 26%	- 3%	+ 22%	+ 7%	+ 6%
2	- 21%	---	- 23%	- 3%	- 15%	-16%
3	+ 3%	+ 29%	---	+ 25%	+ 10%	+ 9%
4	- 18%	+ 3%	- 20%	---	- 12%	-13%
5	- 7%	+ 17%	- 9%	+ 14%	---	- 1%
6	- 6%	+ 17%	- 8%	+ 15%	+ 1%	---

Table 5-21: OFU Change / CA Case / C11 / L-Shape

5.8.1.4 O-Shape

Lastly, we made an experiment with O-Shape. Since both docks are positioned centrally and crossdock distance is shorter than any other configurations, O-Shape performed better than other configurations. IFU and OFU changes comparing to best performing configurations are presented in Table 5-33, Table 5-34, Table 5-35 and Table 5-36.

5.8.1.5 Overall Evaluation of C11

The configurations producing lowest utilization values are presented in Table 5-22.

Designs	**IFU**	**OFU**	
		CNA	**CA**
I-Shape	Conf. 6-8	Conf. 1,4,8	Conf. 1
U-Shape	Conf. 1,2,5	Conf. 1,6,7	Conf. 7
L-Shape	Con 1,2	Conf. 3	Conf. 3

Table 5-22: Overall Evaluation of C11

5.8.2 One to Two Configurations (C12)

In these configurations, there are one receiving dock and two shipping docks. The outbound forklifts are using the middle point between two shipping docks as the

main station in these configurations.

5.8.2.1 I-Shape

No bottleneck detected in C12 configurations of I-Shape design. In all configurations of CNA and CA cases, numbers of satisfied orders are close to each other and high, average queue waiting times are low and reasonable. Consequently, all the utilization values are realistic. Therefore, an accurate evaluation can be made over utilizations.

The receiving dock is positioned at dock_no_1 in configurations 1 to 4 (Group A) and at dock_no_2 in configurations 5 to 7 (Group B). Same dock positions produce same IFUs. Therefore, inferences made about IFU in C11 configurations are valid also for C12 configurations.

In configurations 1, 3, 4, and 7, one of the shipping docks is positioned at the middle dock positions, literally dock_no_2 and dock_no_5. The shipping docks of other three configurations are positioned on the corner dock positions. In CNA cases, configurations 1, 3, 4, and 7 produce %12 and %13.5 lower OFU comparing to configuration 5 and configurations 2, 6, since former ones have one of the shipping docks in the central position. The reason of the difference between configuration 5 and 2/6 is the relative distance of main station to shipping docks. Note that main station of forklifts is placed at the middle point of two shipping docks.

In CA case, relative changes of OFU are presented in Table 5-23. The position of shipping dock, number of crossdocks and crossdock distance affects the outbound forklift utilization.

In configurations 1 and 5, OFUs are respectively 2% and 4% lower comparing to CNA cases as shown in Table 5-24. Note that average crossdocking rate is about

10% (See Table 5-6), therefore it is reasonable that these values are low.

Configurations	*to*						
from	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	---	+ 25%	+ 11%	+ 6%	+ 11%	+ 18%	+ 5%
2	- 20%	---	- 11%	- 15%	- 11%	- 6%	- 16%
3	- 10%	+ 13%	---	- 5%	0%	+ 6%	- 6%
4	- 6%	+ 17%	+ 5%	---	+ 5%	+ 11%	- 1%
5	- 10%	+ 13%	0%	- 4%	---	+ 6%	- 6%
6	- 15%	+ 6%	- 6%	- 10%	- 6%	---	- 11%
7	- 5%	+ 19%	+ 6%	+ 1%	+ 6%	+ 13%	---

Table 5-23: OFU Change / CA Case / C12 / I-Shape

In configurations 1 and 5, OFUs are respectively 2% and 4% lower comparing to CNA cases as shown in Table 5-24. Note that average crossdocking rate is about 10% (See Table 5-6), therefore it is reasonable that these values are low.

Configuration	***No Cross. To Cross. % Change***
1	- 2%
5	- 4%

Table 5-24: OFU Change up to Cases/ C12/ I-Shape

The position of receiving dock affects the number of crossdocks and crossdock distance and these two affect OFU. The change of values for two configurations, 3 and 7, of I-Shape is presented in Figure 5-5.

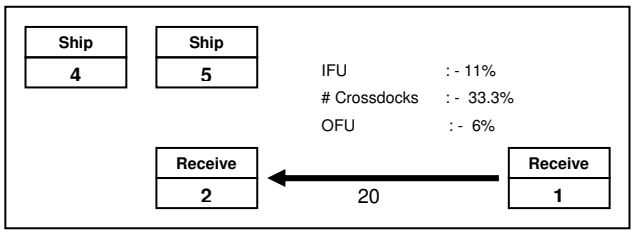

Figure 5-5: Conf 3 to 7 / C12 / I-Shape

5.8.2.2 U-Shape

No bottleneck detected in C12 configurations of U-Shape design. In all

configurations of CNA and CA cases, numbers of satisfied orders are close to each other and high, average queue waiting times are low and reasonable. Consequently, all the utilization values are realistic. Therefore, an accurate evaluation can be made over utilizations.

We can categorize six dock positions of U-Shape into three groups since some positions are symmetrical. Let call Dock_no_1 and dock_no_2 as Dock_A, dock_no_3 and dock_no_6 as Dock_B and dock_no_4 and dock_no_5 as Dock_C.

The receiving dock is positioned at dock_no_1 in configurations 1 to 3, at dock_no_3 in configurations 4 to 6. Same dock positions produce same inbound forklift utilizations. Therefore, inferences made about IFU in C11 configurations are valid also for C12 configurations.

In CNA cases, the position of receiving dock does not affect OFU, therefore there are mainly three different configurations. First is configuration 4 (Group A) which occupies two docks from Dock_A. Second is configurations 1 and 5 (Group B) which occupies one dock from Dock_A, and one from Dock_B. Third one is configurations 2, 3 and 6 (Group C) which occupies one dock from Dock_B, one dock from Dock_C. Group A produces %9 and %20 lower OFUs comparing to Group B and Group C. Group B produces %12 lower utilization comparing to Group C. Utilization changes are presented in Table 5-25.

Comparison of Configuration Groups	***% Change***
Group A to Group B	- 9%
Group A to Group C	- 20%
Group B to Group C	- 12%

Table 5-25: OFU Change / CNA Case / C12 / U-Shape

In CA case, relative changes of OFU are presented in Table 5-26. The position of shipping dock, number of crossdocks and crossdock distance affects the OFU for the same reasons explained above.

Configurations	***to***					
from	**1**	**2**	**3**	**4**	**5**	**6**
1	---	+ 18%	+ 9%	- 9%	+ 4%	+19%
2	- 15%	---	- 8%	- 23%	- 12%	+0.5%
3	- 8%	+ 8%	---	- 16%	- 4%	+ 9%
4	+ 10%	+ 29%	+ 20%	---	+ 14%	+ 30%
5	- 3%	+ 13%	+ 5%	- 13%	---	+ 14%
6	- 16%	- 0.5%	- 8%	- 23%	- 12%	---

Table 5-26: OFU Change / CA Case / C12 / U-Shape

The change of values for two configurations, 3 and 6, of U-Shape is presented in Figure 5-6.

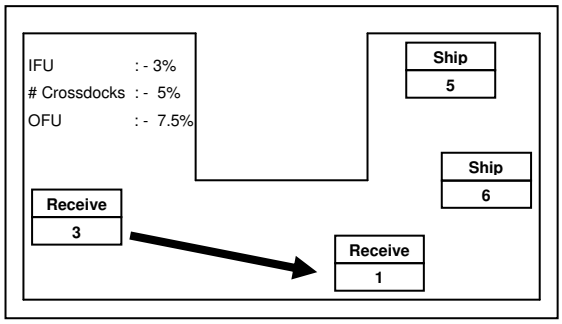

Figure 5-6: Conf 6 to 3 / One to Two Conf. / U-Shape

5.8.2.3 L-Shape

Some bottlenecks depending on resource inadequacy detected for C12 of L-Shape design. These are configurations 3, 4 for CNA cases. In these configurations, receiving dock is positioned at dock_no_3 where average travel distance is relatively longer. Average truck queue waiting times show that inbound forklift inadequacies depending on long travel distances cause the bottlenecks for these configurations. We conclude that these configurations perform badly without any further analysis

and they can not satisfy system needs depending on the reasons explained above.

We continue our analysis with the configurations resulted in no bottleneck.. These are 1, 2, 3 and 4 for CA case and 1, 2 for CNA case.

The receiving dock is positioned at dock_no_1 in configuration 1, at dock_no_2 in configuration 2, and at dock_no_3 in configurations 3, 4. Same dock positions produce same IFUs. Therefore, inferences made about IFU in C11 configurations are valid also for C12 configurations.

In CNA cases, configuration 1 produces %2.5 lower OFU comparing to configuration 2. In CA case, relative changes of OFU are presented in Table 5-27. The position of shipping dock, number of crossdocks and crossdock distance affects the outbound forklift utilization.

Configurations	***to***			
from	**1**	**2**	**3**	**4**
1	---	- 1%	- 9%	+ 7%
2	+ 1%	---	- 8%	+ 8%
3	+ 10%	+ 9%	---	+ 18%
4	- 7%	- 8%	- 15%	---

Table 5-27: OFU Change / CA Case / C12 / L-Shape

The position of receiving dock affects the number of crossdocks and crossdock distance and these two affect the OFU. The changes of values with the change of receiving dock are presented in Figure 5-7.

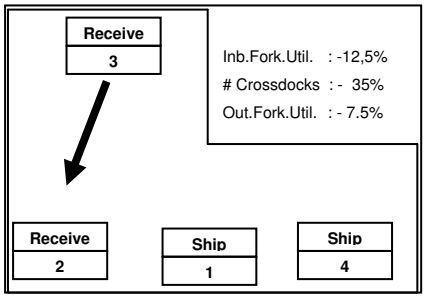

Figure 5-7: Conf 4 to 2/ C12 / L-Shape

5.8.2.4 Overall Evaluation of C12

The configurations producing lowest utilization values in terms of OFU are presented in Table 5-28. Same inferences made about IFU in C11 are valid here.

Designs	CNA	CA
I-Shape	Conf. 1,3,4,7	Conf. 1
U-Shape	Conf. 4	Conf. 4
L-Shape	Conf. 1	Conf. 3

Table 5-28: Overall Evaluation of C12

5.8.3 Two to One Configurations (C21)

In these configurations, there are two receiving docks and one shipping dock where inbound forklifts are using the middle point between two receiving docks as the main station. For these configurations, we present the comparisons of OFU values for CA case in Table 5-29, Table 5-30 and Table 5-31.

Other evaluations are parallel with earlier inferences made in C11 and C12 configurations.

- **I-Shape**

Configurations	***to***						
from	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	---	+ 11%	+ 11%	+ 7%	- 16%	- 9%	- 7%
2	- 10%	---	0	- 3%	- 24%	- 18%	- 16%
3	- 9%	0	---	- 3%	- 24%	- 18%	- 16%
4	- 7%	+ 3%	+ 3%	---	- 22%	- 15%	- 13%
5	+ 20%	+ 32%	+ 32%	+ 28%	---	+ 9%	+ 11%
6	+ 10%	+ 22%	+ 22%	+ 18%	- 8%	---	+ 3%
7	+ 7%	+ 19%	+ 19%	+ 15%	- 10%	- 3%	---

Table 5-29 OFU Change / CA Case / C21 / I-Shape

- **U-Shape**

Configurations	*to*					
from	**1**	**2**	**3**	**4**	**5**	**6**
1	---	+ 12%	- 8%	0%	+ 8%	+ 13%
2	- 10%	---	- 17%	- 10%	- 3%	+ 2%
3	+ 8%	+ 21%	---	+ 8%	+ 17%	+ 23%
4	0%	+ 13%	- 7%	---	+ 9%	+ 14%
5	- 8%	+ 3%	- 15%	- 8%	---	+ 4%
6	- 12%	- 2%	- 18%	- 12%	- 4%	---

Table 5-30: OFU Change / CA Case / C21 / U-Shape

- **L-Shape**

Configurations	*to*			
from	**1**	**2**	**3**	**4**
1	---	0	+ 18%	+ 24%
2	0	---	+ 18%	+ 24%
3	- 15%	- 15%	---	+ 6%
4	- 20%	- 20%	- 5%	---

Table 5-31: OFU Change / CA Case / C21 / L-Shape

Configurations producing the lowest utilizations are presented in Table 5-32.

Designs	**Configurations**
I-Shape	Conf. 1
U-Shape	Conf. 3
L-Shape	Conf. 1,2

Table 5-32: Overall Evaluation of C21

5.9 Comparisons of Designs

In this section, we made an overall comparison of best performing configurations in terms of IFU and OFU for one to one configurations (See Table 5-33, Table 5-34, Table 5-35 and Table 5-36).

- **IFU**

	I-Shape Conf. 6-8	**U-Shape Conf. 1,2,5**	**L-Shape Conf. 1,2**	**O-Shape**
I-Shape Conf. 6-8	---	+ 14%	+ 3%	- 16%
U-Shape Conf. 1,2,5	- 12%	---	- 10%	- 27%
L-Shape Conf. 1,2	- 3%	+ 11%	---	- 19%
O-Shape	+ 20%	+ 36%	+ 23%	---

Table 5-33: IFU Change / CA Case / C11

	I-Shape Conf. 6-8	**U-Shape Conf. 1,2,5**	**L-Shape Conf. 1,2**	**O-Shape**
I-Shape Conf. 6-8	---	+ 21%	+ 3%	- 20%
U-Shape Conf. 1,2,5	- 16%	---	- 15%	- 34%
L-Shape Conf. 1,2	- 3%	+ 18%	---	- 22%
O-Shape	+ 26%	+ 52%	+ 29%	---

Table 5-34: IFU Change / CNA Case / C11

- **OFU**

	I-Shape Conf. 1	**U-Shape Conf. 7**	**L-Shape Conf. 3**	**O-Shape**
I-Shape Conf. 1	---	+ 20%	+ 8%	- 16%
U-Shape Conf. 7	- 17%	---	- 10%	- 30%
L-Shape Conf. 3	- 8%	+ 11%	---	- 23%
O-Shape	+ 20%	+ 44%	+ 29%	---

Table 5-35: OFU Change / CA Case / C11

	I-Shape Conf. 1,4,8	**U-Shape Conf. 1,6,7**	**L-Shape Conf. 3,5**	**O-Shape**
I-Shape Conf. 1,4,8	---	+ 20%	+ 6%	- 20%
U-Shape Conf. 1,6,7	- 16%	---	- 11%	- 33%
L-Shape Conf. 3,5	- 6%	+ 13%	---	- 24%
O-Shape	+ 25%	+ 48%	+ 32%	---

Table 5-36: OFU Change / CNA Case / C11

Since, each design has been experimented under the same conditions; a fair comparison among candidate designs and dock configurations can be made by considering forklift utilizations. In this context, the design and the dock configuration which satisfies the same number of orders with lower forklift utilization can be

accepted as a better option. Among all designs and dock configurations, O-Shape produced the lowest utilization in terms of IFU and OFU since the receiving and shipping docks are positioned at the center of the warehouse and the crossdock distance is quite short. Excluding O-Shape, I-Shape's configuration 1 in terms of OFU and configuration 8 in terms of IFU and OFU produce low utilizations.

5.10 Conclusions of Experimentation

As the result of our experiments, we showed how the changes of dock positions affect the utilization values in a warehouse. For this reason, we presented relative changes of utilizations in terms of IFU and OFU.

Depending on long travel distances according with the position of receiving and shipping docks, some configurations resulted in bottlenecks. These configurations are not included in our evaluation since the utilization values produced in these configurations are not realistic.

The results of our experiments revealed that central dock positions reduce forklift utilizations up to decreases in travel distances. In crossdocking_not_adopted (CNA) case, forklift utilization is a function of average travel distance and average travel distance is a function of warehouse design and dock positions. In crossdocking_adopted (CA) case however, crossdocking function has some additional impacts on utilizations. First of all, when crossdocking function is adopted, less putaway transactions are made, since crossdocking transactions are executed by outbound forklifts. Therefore, in CA case, IFU decreases up to reduction in the number of putaway transactions. On the other hand, the number of crossdocks are not always the same. The number of crossdocking transactions is also a function of receiving dock position. Because the position of the receiving dock affects the

efficiency of inbound forklifts, the efficiency of inbound forklifts affects the average number of pallets waiting for putaway (candidate pallets for crossdocking) and average number of candidate pallets for crossdocking affects the number of crossdocks.

On the other side, there are three factors affecting OFU in CA case. First of three is the position of shipping dock. Again, central positions produced lower utilizations. Other two, the number of crossdocks which is a function of the receiving dock position and crossdock distance which is determined according with the relative positions of receiving and shipping docks. The configuration in which the shipping dock is positioned more centrally, crossdock distance is short and the number of crossdocks is high produces lower OFU values.

As a result, we presented best performing configurations for each design in Table 5-22, Table 5-28 and Table 5-32. Furthermore, we made a comparison of warehouse designs for one to one configurations. The comparison of candidate designs and dock configurations by considering forklift utilizations is sensible since each design has been experimented under same conditions. The design and the dock configuration which produces lower forklift utilization with the same number of satisfied orders can be accepted as better options. Since it has its docks at the very center of the warehouse and its crossdock distance is quite short, O-Shape produce the lowest utilization in terms of IFU and OFU among all designs and dock configurations. Excluding O-Shape, I-Shape's configuration 1 in terms of OFU and configuration 8 in terms of IFU and OFU are some good performers.

6 CONCLUSION

6.1 Summary of the Thesis

This thesis is dedicated to develop a general analysis tool for unit-load warehouses operating with single putaway/picking policies to study some warehousing problems such as resource allocation, layout design and storage policies. We used discrete event simulation (DES) method and event scheduling (ES) approach in our study.

To reach our aim, we have determined some sub-objectives. These sub-objectives have turned out to be the phases of the thesis. First we developed a general purpose DES library which is written in Visual CSharp (C#) and created to implement Event Graph models. We built our warehouse model in this simulation environment. This made us gain the total control of the software to be developed at the first phase.

In the next step, we developed a warehouse library (WareLib). This library was created to maintain the basic warehouse objects and data structures required to simulate a warehouse. By using Excel input, WareLib provides flexibility in designing warehouses. Furthermore, 3D animation feature of WareLib gave us valuable insights throughout the thesis.

Later, we developed a general conceptual model for a unit-load warehouse operating with single putaway/picking policy. This model includes main warehouse functions, literally receiving, putaway, crossdocking, picking and shipping. By the implementation of the conceptual model with SharpSim and WareLib, we reached our aim in developing an analysis tool. In addition to this, we believe that the conceptual model can be used as a template by the modelers who intend to develop conceptual warehouse simulation models adopting ES approach.

Finally, we made some analysis with our model to show its usability. The experimentation was conducted to evaluate the effects of dock positioning on forklift utilization. We gained some insights based on relative comparisons among warehouse layouts and relevant dock configurations.

6.2 Contributions and Main Findings of the Thesis

The purpose of this study is to develop an analysis tool for unit-load warehouses operating with single picking/putaway policy. To achieve this objective, a set of sub-objectives are determined. First sub-objective was to develop a general purpose DES library. SharpSim has been developed for this reason. SharpSim adopts ES approach and offers a flexible and robust environment to implement any EG models. A comparison of SharpSim with MicroSaint Sharp 3.0 (a commercial simulation package which adopts another DES approach) over an M/M/1 model revealed that SharpSim is much faster in any case and especially for congested models. SharpSim is thought to be one of the main contributions of this study.

The WareLib which maintains the data structure to simulate a warehouse presents adequate capability for the analysis tool, but more importantly, it has a promising infrastructure for future development which may pave the way for developing more sophisticated warehouse analysis tools.

The EG model of the generic unit-load warehouse has been implemented to create the analysis tool as anticipated. However, we believe that the conceptual model offers more than being a base for the analysis tool. The conceptual model is a useful template for modelers who intend to develop a conceptual warehouse model adopting ES. Modelers can develop their adaptive models using this conceptual model. The component-based nature of the model supports its adaptability.

Finally, analysis tool offers valuable analysis capabilities. As mentioned earlier, it is designed to study warehousing problems such as layout design, storage policies and resource allocation. Warehouse design is a user input in the model and the design limitations are quite low. Furthermore, component-based conceptual model provides high flexibility in redesigning the model.

The tool has been used to evaluate the effects of dock positioning on forklift utilizations. As a result of our experiments, we showed how the changes of dock positions affect the forklift utilizations in a warehouse. Furthermore, we made a comparison of warehouse designs considering utilizations.

The results of our experiments revealed that central dock positions reduce forklift utilizations. In crossdocking_not_adopted (CNA) case, forklift utilization is a function of average travel distance which is determined by warehouse design and dock positions. In crossdocking_adopted (CA) case, less putaway transactions are made, since crossdocking transactions are executed by outbound forklifts. Therefore, in CA case, inbound forklift utilization (IFU) decreases up to reduction in the number of putaway transactions. In addition, the number of crossdocking transactions is also a function of receiving dock position. Since the position of the receiving dock affects the efficiency of inbound forklifts, the efficiency of inbound forklifts affects the average number of pallets waiting for putaway (candidate pallets for crossdocking). Likewise average number of candidate pallets for crossdocking affects the number of crossdocks.

On the other side, outbound forklift utilization (OFU) is affected by three factors in CA case. The position of shipping dock is the first factor. Again, central positions produce lower utilizations. The number of crossdocks which is a function of the receiving dock position and crossdock distance which is determined according with

the relative positions of receiving and shipping docks are the two other factors. OFU values are low in configurations in which the shipping dock is positioned centrally, crossdock distance is short and the number of crossdocks is high.

As a result, we presented best performing configurations for each design in the previous chapter. Furthermore, we made a comparison of warehouse designs for one to one configurations. Since it has central docks and short crossdock distance, O-Shape produces the lowest utilization in terms of IFU and OFU among all designs and dock configurations. Excluding O-Shape, I-Shape's configuration 1 in terms of OFU and configuration 8 in terms of IFU and OFU are other good performers.

6.3 Future Study

The model presented in chapter 5 is an analysis tool which can be used for further analysis to help answer warehouse problems on layout design, storage policies and resource allocation. As a future study, some general inferences can be made by conducting more experiments.

The capabilities of the analysis tool can be improved in many ways. New rack systems such as double-deep and drive-in rack systems and other storage policies such as class-based storage policy can be introduced to WareLib. Furthermore, some alternative conceptual models considering multi-picking or inter-leaving can be developed and implemented through same libraries.

An interface which will provide an easier handling of SharpSim can be developed. On the other hand, increasing number of models developed with SharpSim will contribute to popularity of this new DES library.

References

Abu Taieh, Evon, and El Sheikh, Asim. 2007. "Methodology and Approaches in Discrete Event Simulation". In *Simulation and Modeling: Current Technologies and Applications* edited by Dr. Asim El Sheikh, Dr. Evon Abu Taieh, Dr. Abid Thyab Al Ajeeli, Idea Group Inc.

Bafna, Kailash M. 1973. "A Simulator for Designing High-Rise Warehouse Systems." Paper presented at the annual meeting of the Winter Simulation Conference, San Francisco, California, January 17-19.

Bartholdi, John J., and Gue, Kevin R. 2000. "Reducing Labor Costs in an LTL Crossdocking Terminal." Operations Research 48(6):823-832.

Bartholdi, John J., and Gue, Kevin R. 2004. "The Best Shape for a Crossdock." Transportation Science 38(2):235-244.

Banks, Jerry. 1998. *Handbook of Simulation, Principles, Methodology, Advances, Applications, and Practice*. John Wiley & Sons, Inc.

Bruzzone, Agostino G., Mosca, Roberto, Spirito, Francesco, Coppa, Angela, and Simeoni, Simone. 2001. "Modelling & Simulation for Customer Satisfaction in Retail Warehouse Management." Paper presented at annual meeting of the EUROSIM Conference, Delft (NL), June 26-29.

Bruzzone. Agostino G., Curcio, Duilio, Mirabelli, Giovanni, Papoff, Enrico. 2009. "Warehouse Management: Inventory Control Policies Comparison." Paper presented at annual meeting of the MAS Conference,Tenerife, September 23-25.

Carson, John S. 1993. "Modeling and Simulation Worldviews." Paper presented at the annual meeting of the Winter Simulation Conference, Los Angeles, California, December 12-15.

Cassandras, Christos G., and Lafortune, Stephane. 1999. *Introduction to Discrete Event Systems*. Kluwer.

Ertek, Gürdal, İncel, Bilge, and Arslan, Mehmet C. 2007. "Impact of Cross Aisles in a Rectangular Warehouse: A Computational Study". In *Facility Logistics* edited by Maher Lahmar. Taylor & Francis Ltd.

Fishman, George S. 2001. *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer.

Frazelle, Edward. 2002. *World-Class Warehousing and Material Handling*. Mc Graw Hill, Inc.

Gagliardi, Jean P., Renaud, Jacques, and Ruiz, Angel. 2007. "A Simulation Model to Improve Warehouse Operations." Paper presented at the annual meeting of the Winter Simulation Conference, Washington, D.C., December 9-12.

Gu, Jinxiang X., Goetschalckx, Marc, and McGinnis, Leon F. 2007. "Research on warehouse operation: A comprehensive review." European Journal of Operational Research 177(1): 1–21.

Gu, Jinxiang X., Goetschalckx, Marc, and McGinnis, Leon F. 2010. "Research on warehouse design and performance evaluation: A comprehensive review." European Journal of Operational Research 203:539–549.

Hoyur, Gülizar. 2006. "A Simulation Model for Distribution Warehouses." MSc thesis, Boğaziçi University.

Law, Averill M., and Kelton, W.David. 2000. *Simulation Modeling and Analysis*. Mc Graw Hill, Inc.

Macro, Joseph G., and Salmi, Reino E. 2005. "A Simulation Tool to Determine Warehouse Efficiencies and Storage Allocations." Paper presented at the annual meeting of the Winter Simulation Conference, Orlando, Florida, December 4-7.

Microsoft. 2011. "XNA Game Studio 4.0." Accessed May 30. http://msdn.microsoft.com/en-us/library/bb200104.aspx

Paul, Ray J., and Balmer, David W. 1998. *Simulation Modeling*. Chartwell-Bratt Student Text Series.

Petersen, Charles G. 2002. "Considerations in order picking zone configuration." International Journal of Operations and Production Management 22(7):793–805.

Pidd, Micheal. 1998. *Computer Simulation in Management Science*. John Wiley & Sons, Inc.

Pohl, Letitia M., Meller, Russell D., and Gue, Kevin R. 2009. "An Analysis of Dual-Command Operations in Common Warehouse Designs." Transportation Research Part E 45:367-379.

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

Senko, J.M., and Suskind P.B. 1990. "Proper Planning and Simulation Play a Major Role in Proper Warehouse Design." Industrial Engineering 22(6): 34-37.

StarUML. 2011. "The Open Source UML/MDA Platform." Accessed June 30. http://staruml.sourceforge.net/en/

Taha, Hamdy A. 2007. *Operations Research: An Introduction*. Pearson Education, Inc.

Zhou, Ming, Kitti, Setavoraphan, and Chen, Zhimin. 2005. "Conceptual Simulation Modeling Of Warehousing Operations." Paper presented at the annual meeting of the Winter Simulation Conference, Orlando, Florida, December 4-7.