ANALYZING AND IMPROVING AIR CARGO TERMINAL OPERATIONS

LIU BIN

(B. Eng. THU)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2004

Acknowledgement

I would like to express my profound gratitude to my supervisors, Dr. Lee Chulung and Prof Huang Huei Chuen, for their invaluable guidance and support for my study. Without them this thesis can never be finished.

My sincere thanks are conveyed to National University of Singapore for offering me a Research Scholarship and the Department of Industrial and Systems Engineering for usage of its facilities, without any of which it would be impossible for me to complete the work reported in this thesis.

I extend my gratitude to all of my friends who have made my life in Singapore an experience I will never forget. Thanks Mok Tsuey Wei, Ivy, Leong Chun How, Xu Zhiyong, Huang Peng, Bao Jie, Liu Rujing, Yang Guiyu, Lin Wei, Zeng Yifeng, Liang Zhe, Peng Ji, Hu Qingpei, and Sun Hainan who have contributed in one way or another towards the fulfillment of this thesis.

Finally, thank my family for their support, understanding, and encouragement throughout the course of my study and research.

List of Abbreviation

AS/RS: Automated Storage/Retrieval Systems

PN: Petri Net

CPN: Coloured Petri Net

TCPN: Timed Coloured Petri Net

SPN: Stochastic Petri Net

GSPN: Generalized Stochastic Petri Net

CTOPN: Colored Timed Object-oriented Petri Net

SCPN: Stochastic Colored Petri net

S/R machine: Storage/Retrieval machine

ULD: Unit Load Devices

MHS: Material Handling System

FG: Floor Goods

OA: Officer Assistant

I/O: Input/Output

FMS: Flexible Manufacturing System

AGV: Automated Guided Vehicle

EO: Equipment Operator

ETV: Elevator Transfer Vehicle

BV: Bridge Vehicle

Table of Contents

Summary

This thesis investigates two research problems regarding air cargo terminal operations. Firstly, the performance of cargo retrieval operations in an import terminal is analyzed for the objective of reducing the time of retrieval operations. As air cargo volume is fast increasing and airports need to compete with each other, air cargo terminals which handle cargos for carriers face challenges to streamline their operations. For cargo storage, terminals employ diverse storage media, such as Automated Storage/Retrieval Systems (AS/RS), storage racks, carousels, and so forth and are also equipped with various material handling equipments, such as forklifts and tractors. Cargos must go through many different processes, such as being transported by forklifts and being retrieved from AS/RS. Due to the complex nature of cargo handling operations and stochastic customer requests and processing times, which make formulation of analytical model difficult, a Petri Net (PN) model is developed to describe the cargo retrieval process, from which a corresponding discrete event simulation model is developed. In this model, we represent steps of processes, e.g., forklifts moving cargos from the workstation to a truck dock, as transitions and each request of retrieving cargo as a token. Because terminal operations constitute a large and complex system, we employ Coloured PN (CPN) to describe the system in a compact and manageable way. To investigate the performance of retrieval operations by means of cargo processing time, we construct the Timed CPN model. Actual terminal data are employed to validate the proposed model. Since the results show flow times from the model

approximates servicing times in the real operations, the model is capable of giving an insight into the behavior of current terminal operations and evaluating system performance for proposed changes in the operation design and control. A simple application of comparing the system performance based on different numbers of forklifts is also provided.

Secondly, we propose a new storage policy for AS/RS in import terminals, which reduces the expected Storage/Retrieval (S/R) machine travel time. S/R machines travel in aisles of AS/RS and serve storage and retrieval orders, thus AS/RS throughput can be improved by reducing the S/R machine travel time. Most researchers study AS/RS based on inventory control policies. However, the storage and retrieval of air cargos are different: storage requests depend on flight schedules and retrieval requests depend on forwarders, i.e., this system is not operating based on inventory control policies for AS/RS in literature. A discrete time Markov chain model is proposed, which computes the exact travel time by capturing the dynamic feature of storage opening status. An approximation model is developed which gives static estimation on the travel time with less computational effort. Finally, a simulation model is employed which follows the same rules like the Markov model but can solve real industrial-size problems. The conclusion is the generally approved class-based storage policy does not fit the proposed problem in this thesis due to the stochastic storage duration. The reservation storage policy provides the minimum expected travel time compared with other storage policies.

List of Figures

List of Tables

1. Introduction

In recent years, air cargos are no longer confined to rapid shipment or to fulfill emergent demands. As air cargo volume is fast increasing and airports need to compete with each other, air cargo terminals which handle cargos for carriers face challenges to streamline their operations. For cargo storage, terminals employ diverse storage media, such as Automated Storage/Retrieval Systems (AS/RS), storage racks, carousel, and so forth and are also equipped with various material handling equipments, such as forklifts and tractors. Cargos must go through many different processes, such as transported by forklifts and retrieved from AS/RS. In this thesis, two research problems with the objective of improving cargo terminal operations are presented. In Section 1.1, a brief introduction to an import terminal, especially the retrieval operations is provided. For the problem discussed in Chapter 4, Section 1.2 gives an extended introduction on AS/RS in import terminals.

1.1. Introduction to an import terminal and its operations

In this section, a brief description for cargo import operations is provided. For a typical airport, it normally has separated passenger terminals and cargo terminals. For cargo terminals, they can be classified according to their functions as: import terminals and export terminals. Import terminals take charge of inbound cargos. Unit Load Devices (ULD) are towed to import terminals by tractors after flights arrive. Then after breakdown, cargos are stored in terminals temporarily until forwarders come to take

them away. On the contrary, export terminals take charge of outbound cargos. Forwarders bring their cargos to export terminals, and then before the corresponding flights take off, the cargos are stored in terminals temporarily. In this thesis, only import terminals are concerned, thus, export terminals will not be discussed any further. The following description of import terminal operations is based on the observation in one of the busiest airports in the world.

In the observed terminal, most cargos are stored in the AS/RS, Material Handling System (MHS) and Floor Goods (FG) area. Cargos stored in these three locations weight about 80 percent among all cargos stored in the terminal. There are also other types of cargos stored in the terminal, such as perishable cargos, vulnerable cargos, valuable cargos and dangerous cargos. But since these cargos only weight small percentage among all cargos, they are stored in special places, e.g., valuable cargos are stored in the storeroom by special staffs, e.g., cargo officers, and their retrieval processes are too complicated, the retrieval operations for them are not included in the proposed model. In this thesis, we consider only general cargos, which include cargos stored in the AS/RS, MHS and FG area only. According to their storage locations, they are named as ASRS cargo, MHS cargo and FG cargo, respectively. ASRS cargos are stored in the AS/RS, MHS cargos are stored in the MHS and FG cargos are stored in the FG area.

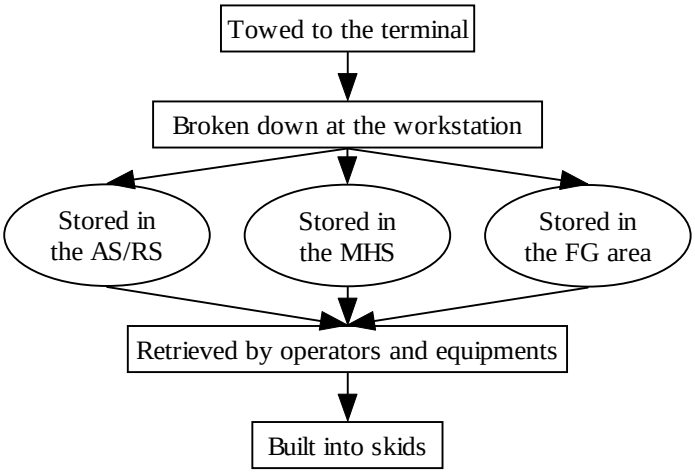

Figure 1.1 General cargo flow in the import terminal

The general cargo flow in the observed import terminal is shown in Figure 1.1. After inbound cargos have been towed by tractors to the terminal, import checkers break down the cargos at the workstation. Then the cargos are stored in different storage media according to their conditions: cargos remain on a big pallet or in a container are stored in the MHS, cargos on a wooden pallet are put into storage racks in the FG area, and loose cargos are put into standard-size bins and stored in the AS/RS. After a forwarder arrives in the terminal, an Officer Assistant (OA) receives retrieval requests from the forwarder and dispatches retrieval jobs to operators. Transporting cargos from the MHS to a truck dock is performed by mechanical rollers and forklifts, and transporting cargos from the FG area and AS/RS to a truck dock is performed only by forklifts. All the equipments, such as rollers and forklifts, are controlled by their operators, but for the reason of conciseness, equipments are used to represent both the equipments and the corresponding operators. After cargos are retrieved, the forwarder builds them into skids and loads the skids onto his truck. After collecting all his cargos, the forwarder leaves

the terminal. The model presented in this thesis is developed for the retrieval part of import process, i.e., from forwarders come and request cargos till they finish collecting cargos. Thus the replenishment process (from cargos arrive in the terminal till they are stored in different storage locations) is not considered thereafter. The development of Petri Net (PN) simulations for the retrieval process will be discussed in Chapter 3.

1.2. Introduction to automated storage/retrieval systems in import terminals

AS/RS are widely used in different warehouses around the world. A typical AS/RS is an automated warehousing system that comprises one or multiple parallel aisles with storage openings on both sides of the aisles. A Storage/Retrieval (S/R) machine in AS/RS serves one or several aisles to execute storage and retrieval orders. S/R machines can travel horizontally and vertically simultaneously. Thus, the travel time between any storage opening and the Input/Output (I/O) point of AS/RS is the maximum of horizontal and vertical travel times, which is commonly called Tchebychev metric. In air cargo terminals, AS/RS are also a key storage medium for cargo handling operations. Air cargo handling has an important requirement for all relevant operations: swiftness. Thus, all the terminal operations, including AS/RS storage and retrieval operations must be performed in the most efficient manner. In this thesis, we focus on analyzing operation characteristics of cargo terminal AS/RS with the objective of improving their throughput.

Most researchers study AS/RS based on inventory control policies, e.g., continuous review, order quantity, reorder point inventory policy. The famous “ABC” curve and EOQ model are applied to develop analytical and simulation models on the purpose of optimizing AS/RS, e.g., minimizing the expected S/R machine travel time. However, the storage and retrieval of air cargos can not be controlled by the above mentioned inventory control policies. Different from other common warehouses, executions of cargo terminal operations are triggered by both flight schedules and requests from forwarders. Thus, storage requests depend on flight schedules and retrieval requests depend on forwarders. According to the unique trait of cargo terminal AS/RS, a new storage policy, which can reduce the expected S/R machine travel time is proposed. Details will be provided in Chapter 4.

1.3. Outline of thesis

The remainder of this thesis is organized as follows. Chapter 2 gives the literature survey on both PN modeling and AS/RS performance analysis and optimization. Section 2.1 gives a short introduction about PN; applications of PN modeling related to the system presented in this thesis, including research results in air industry and complex manufacturing systems are also provided. Section 2.2 summarizes the research results on AS/RS performance analysis and optimization, especially on improving system throuthput by minimizing the expected S/R machine travel time.

Chapter 3 introduces the development of Coloured PN (CPN) simulations for the cargo retrieval prosess. Section 3.1 describes the PN model for retrieval process, including processes for retrieving ASRS, MHS and FG cargos and building skids. Section 3.2 develops a simulation model based on the PN model. It has three main parts: initialization, main controller and output, and several function-based modules are also included. The proposed model is validated in Section 3.3 by comparing results from the model with actual data from the terminal. A simple application of the developed model to compare the system performance based on different numbers of forklifts is provided in Section 3.4.

Chapter 4 focuses on the system performance improvement of AS/RS in import terminals. Section 4.1 gives details of the reservation storage policy. Analytical models are developed to compute the expected S/R machine travel time. A simulation model is developed in Section 4.2, which aims to solve real industrial-size problems.

Chapter 5 concludes the whole thesis and lists potential research in the future.

2. Literature survey

Both PN modeling and AS/RS performance optimization have rich research results in literature. Research reports in these two areas are reviewed in Section 2.1 and Section 2.2, respectively.

2.1. Literature survey on Petri Net modeling

In this section, literature on PN modeling is provided; for more information about PN, readers are referred to Jensen (1997). PN have a well-developed formalism and semantics that can model systems with interacting concurrent components, e.g., forwarders, forklifts, and mechanized machines commonly operating in parallel. In any net, there are two basic elements: nodes and links. PN have two types of nodes: places and transitions. Places are used to represent different kinds of resources, e.g., storage space, or states of processes, e.g., cargos waiting for retrieval. Transitions are used to indicate actions or operations, e.g., forklifts transporting cargos from the workstation to a truck dock. PN employ directed arcs to connect from places to transitions or vice versa. The dynamic feature of PN is achieved by tokens, which can represent cargo retrieval requests. An arbitrary distribution of tokens on the places is called a marking. Each marking corresponds to one state of modeled systems. The execution of PN is regulated by the number and distribution of tokens and changes the system state. By equipping each token with an attached token colour, standard PN are extended to CPN, which can

represent a large and complex system in a much more compact and manageable way. In our model, each type of cargos that requires distinct retrieval operations is assigned one colour. To investigate the performance of terminal operations by means of cargo processing time, we construct the Timed CPN (TCPN) model, i.e., allowing each token to carry a time stamp. In TCPN, transition rates can be either deterministic or random with a specified distribution. For example, when retrieving cargos stored in racks, the searching time can be constant or a random variable following a pre-defined distribution.

In 1962, Carl A. Petri created a net-like mathematical tool for the purpose of investigating communication with automata; which is later named as Petri Nets. For the last 40 years, PN have been applied in computer science, manufacturing system and also other areas. Since a lot of research works have been done by different researchers in many different areas, in this section we only focus on the papers related to the problem studied in this thesis.

Kanoun et al. (1996) selected a new architecture with the highest level of availability for a subset of an air traffic control system. The detailed model of the architecture was constructed by Generalized Stochastic PN (GSPN). Fota et al. (1999) evaluated dependability of an air traffic control computing system by using Stochastic PN (SPN). They modeled the system as a set of modules interconnected through coupling mechanisms.

Al-Jaar and Desrochers (1990) used GSPN to model and analyze complex manufacturing systems. The model was proven to be live and bounded and was applied for several manufacturing case studies. Wang and Wu (1998) developed a new paradigm, called Colored Timed Object-oriented PN (CTOPN). The definition of CTOPN was introduced, the procedure of developing a CTOPN model was illustrated and applications of the CTOPN approach in scheduling and controlling an AMS were presented. Moore and Gupta (1999) modeled and analyzed kanban-controlled (just-in-time) production systems under realistic assumptions by using Stochastic Colored Petri net (SCPN). The model conducted both qualitative and quantitative analysis. Different control policies are considered and simple simulation results are provided.

Menon et al. (1989) developed a CPN based approach for modeling a Flexible Manufacturing System (FMS), which could be used for simulation and control of the system. Hatono et al. (1989) applied SPN with hierarchical structure to construct a FMS model under uncertainty. And they used the proposed FMS simulation system to obtain an efficient schedule with on-line real-time basis. Srihari et al. (1990) focused on PN use in the modeling and analysis of a working FMS. The systematic procedure used in the modeling and analysis of the FMS through PN use was described. Kochikar and Narendran (1990) modeled a FMS as a CPN. They checked the correctness of model representation and investigated qualitative properties of the FMS. Borusan (1993)

analyzed FMS by using CPN, which can encompass the hierarchical structure of manufacturing system organization, different material and data flows, concurrency and asynchrony of process execution, buffer sizes, production routes, transport, assembly and machining operation times, and so on. Shiizuka and Suzuki (1994) modeled Automated Guided Vehicle (AGV) networks in FMS by CPN. AGV models were constructed as parts and the parts could be combined. Ziegler and Szczerbicka (1998) modeled a FMS by using GSPN; the performance and performability of the FMS were evaluated carefully.

Lin and Wang (1995) described the behaviors of AS/RS by using SPN; performance of different control policies of such systems was evaluated. Zimmermann and Freiheit (1998) developed TimeNETMS, a software tool integrating manufacturing system modeling and performance analysis with an easy-to-use graphical user interface by using SPN. Zuberek (2000) performed hierarchical analysis of manufacturing systems in a top-down manner by using PN; a general model was used to capture the main effects of components interconnections and more detailed models of components provided performance characteristics of the entire system.

Some research works mentioned above used low-level PN, such as SPN and GSPN. These PN can generate analytical models, which can provide the exact values of system performance measures. But two drawbacks exist. One is that transitions in these models must follow exponential distributions, which requires strict assumptions on the studied

system. The other is these models need generate a full state space for the investigated system, which can arouse the state explosion problem. These two weak points may prevent such PN models to be applied to solve real industrial-size problems.

CPN belongs to high-level PN, which has at least three advantages compared with low-level PN. Firstly, CPN employ token colours, which allow them to construct large systems in a compact and manageable way, while low-level PN have to represent similar parts of a system by disjoint subnets with an almost same structure. Secondly, CPN generate simulation models, which require fewer assumptions than analytical models, e.g., transitions can be defined to follow any probabilistic distributions, not limited to exponential distribution only. Thirdly, since simulations are used, there is no state explosion problem any more.

2.2. Literature survey on automated storage/retrieval system performance analysis and optimization

As one of the major material handling systems widely applied in modern manufacturing, AS/RS have been studied by a lot of researchers for many years. In this section, a brief review on AS/RS performance analysis and optimization is given.

Hausman et al. (1976) identified three aspects of AS/RS research issues: Pallet Assignment problem, the assignment of multiple items to the same pallet; Storage

Assignment problem, the assignment of pallet loads to storage locations; and Interleaving problem, rules for sequencing storage and retrieval requests. They focused on the Storage Assignment problem and compared the operating performance of three storage assignment rules: random assignment, full turnover-based assignment and class-based turnover assignment. They developed both the discrete model and continuous representation based on the "ABC" curve and basic EOQ model. Their results showed turnover-based rules could reduce expected travel time for S/R machine. Two assumptions were made: "square" in time, which means the time required to travel from the front of AS/RS to the back equals to the time required to travel from the bottom to the top; and turnover frequency of each item is known and constant through time. Graves et al. (1977) studied the Interleaving problem on the base of Hausman et al. (1976). The results showed travel time reduction by combining the turnover-based storage rule and interleaving. Schwarz et al. (1978) carried simulations which validated the conclusion of Graves et al. (1977). Kim and Seidmann (1990) derived more general expressions for the expected travel time of S/R machine, which extended the result of Hausman et al. (1976); they also assumed "square" in time. Thonemann and Brandeau (1998) applied Hausman et al. (1976) into a stochastic environment; the computational results showed the turnover-based storage policy could reduce much travel time compared with the random storage policy in stochastic environments.

Goetschalckx and Ratliff (1990) addressed duration-of-stay-based shared storage policies for a unit load warehouse. An optimal policy with respect to travel time and

storage space was developed for systems with balanced input and output and two heuristic policies were developed for more complex systems. Kulturel et al. (1999) used simulations to compare the two proposed shared storage assignment policies in Hausman et al. (1976) and Goetschalckx and Ratliff (1990). The AS/RS was assumed to operate under a continuous review, order quantity, reorder point inventory policy and their conclusion was that the turnover-based policy, in general, outperformed the duration-of-stay-based policy.

Bozer and White (1984) derived an analytical expression for expected S/R machine travel times under randomized storage conditions for both single and dual command cycle modes. Different storage rack shapes, various dwell point strategies, and alternative I/O point locations were examined. Based on their travel time model, Bozer and White (1990) developed an algorithm to minimize the number of storage aisles with throughput and storage space constraints for the purpose of designing systems with lower construction cost. They assumed continuous dual command cycles. Hwang and Lee (1990) and Chang et al. (1995) extended Bozer and White (1984) by considering the acceleration and deceleration of S/R machine. Pan and Wang (1996) proposed a framework for the dual command cycle travel time model under an n-class-based storage policy. Wen et al. (2001) proposed travel time models for the class-based storage assignment and full-turnover-based storage assignment which considered various travel speeds with known acceleration and deceleration rates.

Tompkins and White (1984) introduced a static analysis to estimate the utilization of S/R machine with known number of requests. This analysis relied on a guess of the ratio of single cycles to dual cycles to be performed. Ashayeri et al. (2002) presented a geometry-based analytical model to compute cycle times for zone-based storage layouts. They assumed the fraction of single command cycles was known. These two performance models both required assuming a proportion of dual versus single command cycles.

Some researchers recognized the state dependant nature of AS/RS service process. Lee (1997) presented a stochastic analysis for performance evaluation of a unit-load AS/RS by using a single-server queuing model with two queues. Cycle times were assumed to follow exponential distributions. Bozer and Cho (1998) developed the throughput performance model, which provided a significant implementation advantage due to its computational efficiency. Eldemir et al. (2003) suggested that the throughput performance model, Bozer and Cho (1998) was a dominating alternative for AS/RS conceptualizing applications over Lee (1997) since it could provide equivalent accuracy with less computational effort.

Other aspects of AS/RS optimization were also studied. Eynan and Rosenblatt (1994) developed procedures for dividing rectangular warehouses into classes. They assumed single command cycle only and full utilization of storage space. Chang and Egbelu (1997) presented a mathematical programming model for deriving the optimal dwell

point which minimized the expected S/R machine response time. In their investigated system, one S/R machine served several neighboring aisles. Van den Berg (2002) presented analytic expressions of optimal dwell point position to minimize the expected travel time for randomized and class-based storage policies. The problem was modeled as a facility location problem with rectilinear distances and solved. Lee and Schaefer (1997) presented sequencing methods of storage and retrieval requests under static and dynamic approaches, which reduced S/R machine travel time for randomized and dedicated storage policies. Mahajan et al. (1998) developed a nearest-neighbour retrieval sequencing heuristic, which aimed to improve throughput of miniload AS/RS. The system carried dual command cycles only.

Simulations were applied for the objective of performance evaluation and improvement of AS/RS. Elsayed and Unal (1989) proposed four heuristics to batch orders into tours instead of carrying orders one by one. Simulations were used to evaluate these four heuristics on the basis of minimizing total travel time. Randhawa et al. (1991) developed a simulation model to compare system performance of “square” in time AS/RS under different layout configurations and storage/retrieval rules. Randhawa and Shroff (1995) extended the former paper. Designs of both “square” in time and not “square” in time AS/RS were evaluated. Van den Berg and Gademann (2000) presented a simulation study which examined a wide variety of control policies, including a new proposed continuous storage policy. They evaluated the trade-off between travel times and storage space requirements.

It can be seen from the above listed papers, to improve AS/RS throughput by minimizing the expected S/R machine travel time, at least three aspects on AS/RS operating policies have been explored a lot. Firstly, storage policy, three main storage policies have been paid much attention. Randomized storage policy is inefficient in terms of minimizing the expected travel time; turnover-based storage policy is optimal; class-based storage policy is near optimal and practicable. Most studied systems were operating based on inventory control policies and some were assumed to be “square” in time. However, not much research of the impact on space requirement by applying different storage policies has been considered. Due to the unique trait of cargo storage and retrieval, AS/RS in terminals are not operating based on inventory control policies like common AS/RS. In this thesis, we propose a new storage policy for them, and a different result of the expected travel time is obtained. At the same time, the space requirement is also provided. Secondly, dwell point policy. Thirdly, interleaving policy. Satisfactory research results have been achieved in these two aspects; we do not consider them in this thesis, but they may be worth research in the future.

The existing research achievements related to this thesis are stated above. In the next chapter, CPN is applied to describe and evaluate the cargo retrieval operations, and in Chapter 4, the new storage policy with the objective of improving AS/RS throughput is proposed.

3. Developing coloured Petri Net simulations for cargo retrieval operations

In section 1.1, cargo handling operations in the air cargo terminal are introduced. On account of the keen competition among different cargo terminals, the terminal is eager to improve the effectiveness of its operations. This chapter is focused on bettering cargo retrieval operations. Thus performance analysis is necessary. Through such a study the time that can be saved after the system is improved can be predicted.

Measurement, analytical models and simulation models are three alternatives that are often applied to analyze the performance of a system. Measuring the performance of a system can provide exact answers regarding the system performance. But in the cargo terminal, proposed schemes can not be implemented immediately, thus measurement is not applicable for analysing the performance of cargo terminal operations. Analytical models, such as queuing models, may give exact results on system performance. However, for industrial systems, like a cargo terminal, the exact results can hardly be accurate, because many unrealistic assumptions have to be made when creating the analytical models. Thus, simulaton-based performance analyses are used for the cargo tyerminal. In this chapter, a PN model is developed for the retrieval operations, from which a corresponding discrete event simulation model is developed. Actual terminal data are employed to validate the proposed model. A simple application of comparing the system performance based on different number of forklifts is also provided.

3.1. Modeling cargo retrieval operations with Petri Nets

A CPN based simulation model is identified as the performance evaluation tool for cargo retrieval operations, thus this section presents the PN model for cargo retrieval operations in the observed import terminal, which is the base of the discrete event simulaton model. In this terminal, different types of cargos are handled and go through different retrieval processes. In the model, steps of processes, e.g., forklifts moving cargos from the workstation to a truck dock, are represented as transitions and each request of retrieving cargo as a token. In the model, resources are represented by means of a single place; the place of “Resources” holds all the tokens that indicate the availability of resources, and different types of resources are represented by different token colours. In TCPN, transition rates can be either deterministic or random with a specified distribution. All transition rates are defined according to the actual terminal situation; data are obtained from equipment specifications and time study results. The whole PN model for the cargo retrieval process is isolated into parts; each part is explained individually with a corresponding PN diagram. In all the figures in Section 3.1, ellipses represent places and rectangles represent transitions. To make the diagrams easy to read, we use full mnemonic names for all the places and transitions. Resources necessary for the firing of transitions and resources released from transitions are specified on arcs connecting the resource place with transitions. The resource types are used to represent one unit of such resources. From the resource place, a token is taken

for the firing of a transition if it is available and will be returned to the place after the transition releases it, which makes the resource available for another transition. The connecting points for separated diagrams of the whole model are represented by dotted ellipses or circles with numbers in them. The connecting points have the same number are the same point. The “or” means the token goes to one direction of the two alternatives, which direction it goes to depends on specific conditions. The “and” means one token for each direction. The “end” means the token is absorbed. To make the diagrams clear and emphasize the process flow, transition rates are not tagged.

3.1.1. Petri Net model for retrieving ASRS cargos

Th retrieval process for ASRS cargos follows these steps: firstly, an OA sends a retrieval order for ASRS cargos to the AS/RS control system; secondly, the S/R machine in the AS/RS retrieves the bin that contains the requested cargos; thirdly, after a bin is put at the I/O point of the AS/RS, a forklift picks it up and transports it to a pre-assigned truck dock; fourthly, after a bin is put on a truck dock, the corresponding forwarder unloads his cargos from the bin; fifthly, if any other forwarder requests cargos in the same bin, the bin will be transferred to the next forwarder by a forklift; at last, when a bin does not hold cargos required by any forwarder, it is returned to the AS/RS.

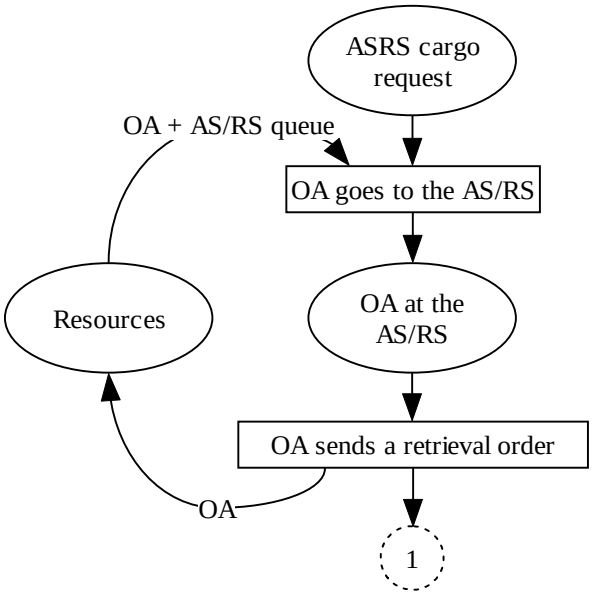

Figure 3.1 PN model for OA requesting a bin

The first step of retrieving ASRS cargos is “OA requesting a bin”, modeled as shown in Figure 3.1. If there are requests for ASRS cargos on a Pick List, an OA will provide the specific storage locations for these cargos to the AS/RS control system. Since the AS/RS in the observed terminal can store only 10 storage/retrieval orders at one time, the “AS/RS queue” is considered a resource with the maximum capacity of 10 in this model. Upon sending an order to the control system, the “OA” is released, while the completion of a storage/retrieval operation of the AS/RS will remove the order from the “AS/RS queue”.

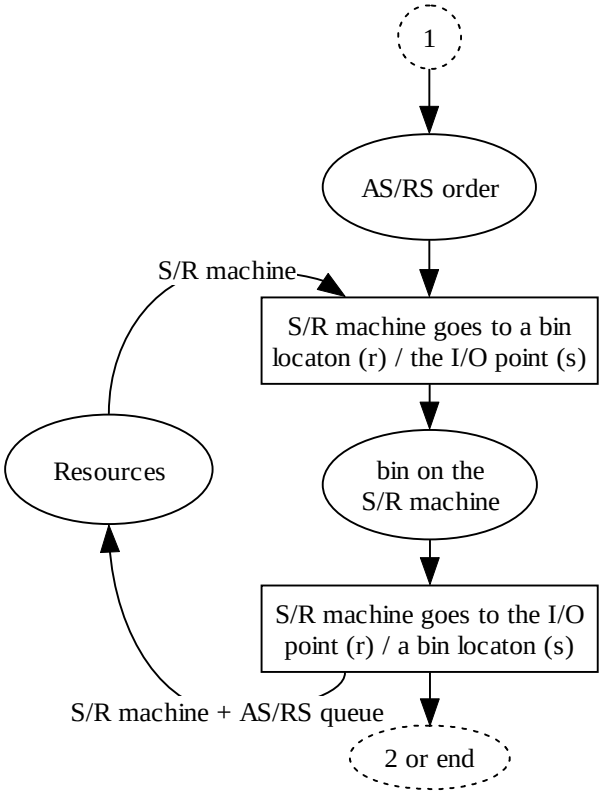

Figure 3.2 PN model for the AS/RS operations

Figure 3.2 illustrates the PN model for the operations of the AS/RS. The S/R machine stores/retrieves cargos to/from storage locations in the AS/RS to fulfill storage/retrieval orders. For each ASRS cargo, a bin number is recorded for the system to recognize the specific location of the cargo. For retrieval orders, the S/R machine first travels to the specific bin location and then retrieves the bin to the I/O point of the AS/RS; for storage orders, the S/R machine travels to the I/O point of the AS/RS to pick up a bin, then travels to the pre-defined location and puts the bin into the location. Then a space in the AS/RS queue will be released to be available for future orders from OA. For retrieval operations, the connecting point is “2”; for storage operations, it is “end”.

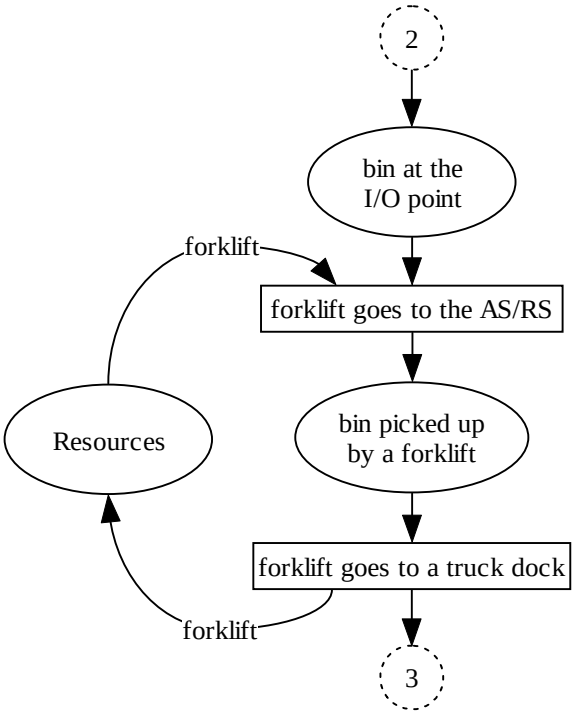

Figure 3.3 PN model for forklifts transporting a bin

“Forklifts transporting a bin” is modeled as shown in Figure 3.3. After a bin is retrieved at the I/O point of the AS/RS, a forklift picks it up and transports it to a truck dock. Since a forklift requires an Equipment Operator (EO) for this operation, thus each forklift is considered a token to represent the availability of a forklift and its EO. Throughout this thesis, the resource forklifts means both the forklifts and their EOs.

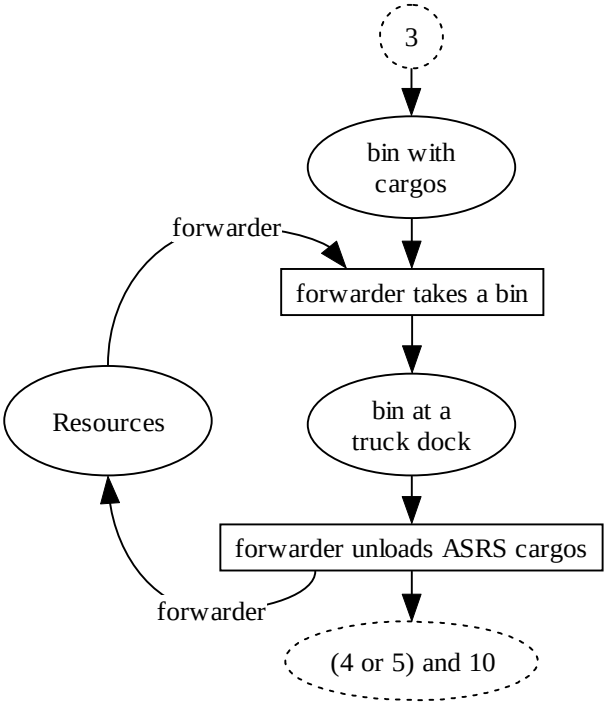

Figure 3.4 PN model for forwarders unloading ASRS cargos

“Forwarders unloading ASRS cargos” is modeled as shown in Figure 3.4. When a bin is put on a truck dock, the corresponding forwarder unloads his cargos from the bin. In this process, “forwarder” is considered the type of resource to fire the transition. Likewise, MHS cargos also need unloading; however, FG cargos do not need unloading. If the bin is requested by other forwarders, the connecting point is “4”; otherwise, it is “5”. Meanwhile, a token goes to the “building skids” process.

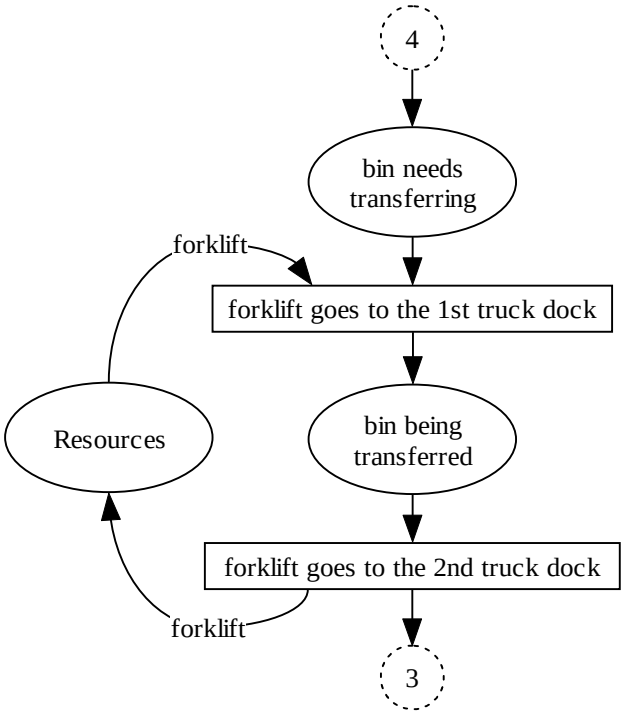

Figure 3.5 PN model for forklifts transferring a bin

Figure 3.5 illustrates the PN model for transferring a bin from a truck dock to another. After a forwarder unloads his cargos from a bin, if there is any other forwarder requesting cargos in the same bin, the bin will be transferred to the next forwarder. “Forklift” is the type of resource to fire the transition in this process. Then the process goes to “forwarder unloading ASRS cargos” again. When no forwarder requests cargos in the bin, it will be returned to the AS/RS.

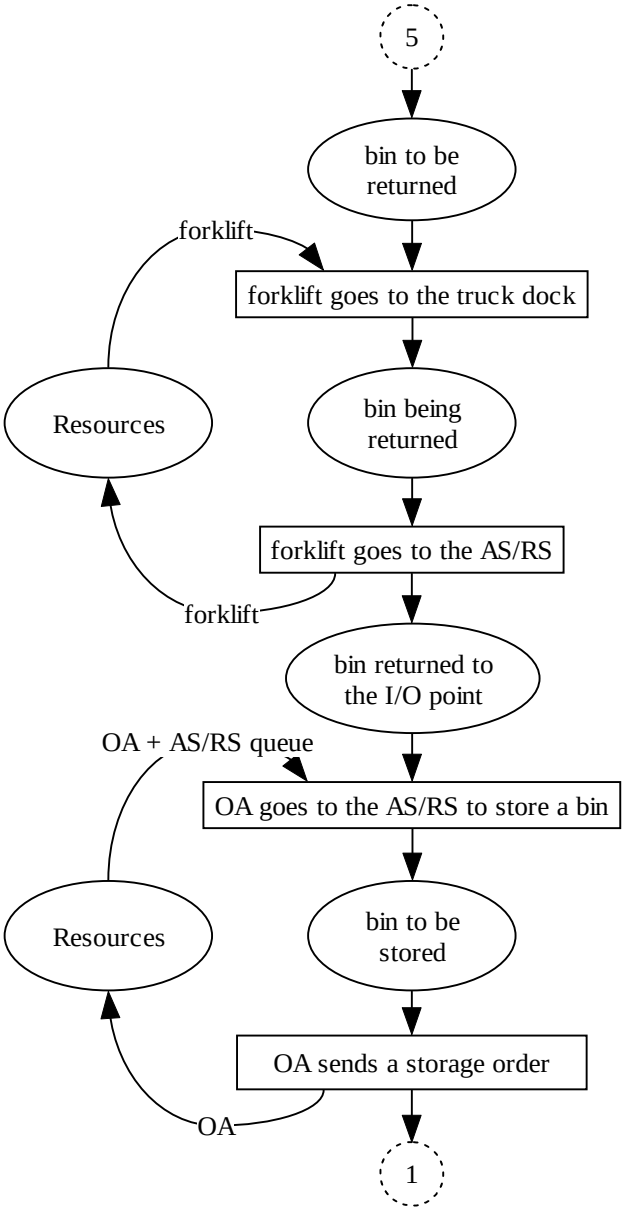

Figure 3.6 PN model for returning a bin to the AS/RS

Figure 3.6 illustrates the PN model for how a bin is returned to the AS/RS. Firstly, a forklift picks up the bin to be returned at the truck dock where the bin is and transports it to the I/O point of the AS/RS. Secondly, an OA sends a storage order to the AS/RS control system and determines the specific storage location for the bin. This triggers a storage operation of the AS/RS which has been discussed in “PN model for the AS/RS

operations” already. The two ellipses “Resources” in this figure refer to the same unique place “Resources”. The same drawing technique is employed in Figure 3.9 and Figure 3.11. The above is the complete process for retrieving AS/RS cargos.

3.1.2. Petri Net model for retrieving MHS cargos

The retrieval process for MHS cargos is similar to that for AS/RS cargos, but some differences exist. The retrieval process for MHS cargos follows these steps: firstly, an OA sends a retrieval order for MHS cargos; secondly, the Elevator Transfer Vehicle (ETV) retrieves the container or pallet which contains the requested cargos from its storage cell to the I/O point of the MHS; next, a mechanical roller of the workstation transports the container or pallet to a deck and a forklift transports the container or pallet from the deck to a pre-assigned truck dock; then, the corresponding forwarder unloads the cargos from the container or pallet; at last, the container or pallet is returned by a forklift, a roller and the ETV to a storage cell in the MHS. The detailed explanation is followed.

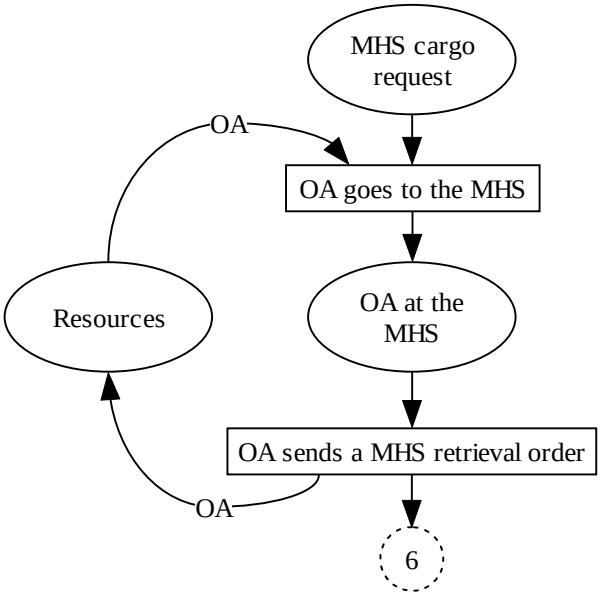

Figure 3.7 PN model for OA requesting MHS cargos

Figure 3.7 shows the PN model for “OA requesting MHS cargos”: similar to AS/RS cargos, MHS cargos also have to be requested by an OA. The processing time of sending a retrieval order for MHS cargos is longer than sending a retrieval order for ASRS cargos; and since the queue capacity for storing MHS orders is set to be always sufficient, there is no resource limit like “AS/RS queue”.

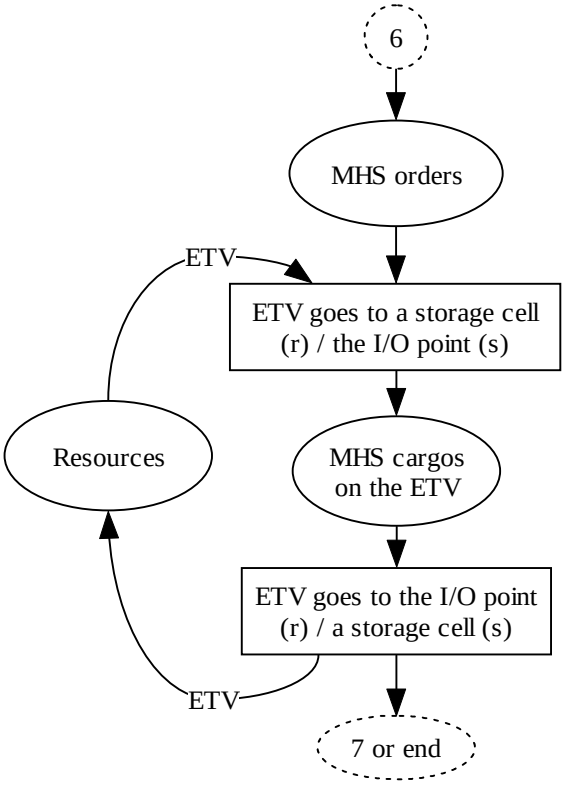

Figure 3.8 PN model for the MHS operations

As modeled and shown in Figure 3.8, the operations of the MHS are also similar to those of the AS/RS. The ETV in the MHS is equivalent to the S/R machine in the AS/RS. But the layout and storage allocation of the MHS are different from those of the AS/RS. The MHS has three levels: the ground level is mainly used as buffers; the middle level is used as the main storage space for MHS cargos which are on pallets or in containers; the top level is reserved for recycled empty containers or pallets only. Thus, when the ETV is performing a retrieval operation, it first goes to the middle level and then brings the requested container or pallet to the I/O point on the ground level; when it is performing a storage operation for an empty container or pallet, it first picks up the container or pallet at the I/O point, and then goes to the top level, at last it unloads the container or pallet to

an idling storage cell. For retrieval operations, the connecting point is “7”; for storage operations, it is “end”. The empty containers or pallets stored on the top level will be transported to an export terminal by the Bridge Vehicle (BV) to ensure enough free storage cells for future coming empty containers or pallets. This operation is beyond the model developed in this chapter; in the model, we simply assume empty containers or pallets can be removed from the top level by the BV in time.

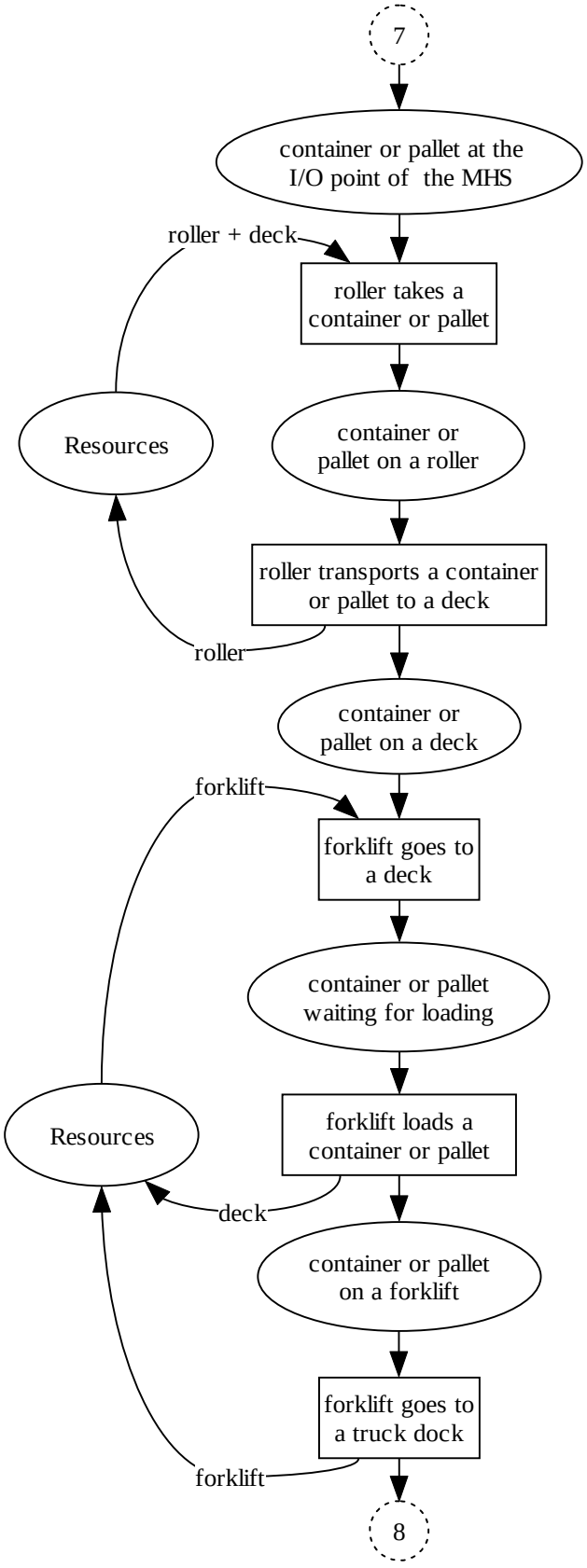

Figure 3.9 PN model for transporting MHS cargos to a truck dock

Figure 3.9 shows the PN model for “transporting MHS cargos to a truck dock”. MHS cargos are in containers or on pallets when being transported. There is a workstation next to the MHS. The workstation has totally 24 decks. After a container or pallet has been retrieved by the ETV to the I/O point of the MHS, a mechanical roller connecting the I/O point and decks is used to transport it from the I/O point onto a deck. Thus, the other type of resource needed to fire the transition is decks. After a container or pallet is put on a deck, the roller is available for the next transport immediately, but the deck can only be released after a forklift loading the container or pallet and removing it from the deck. Then the forklift is used to transport the container or pallet to a truck dock.

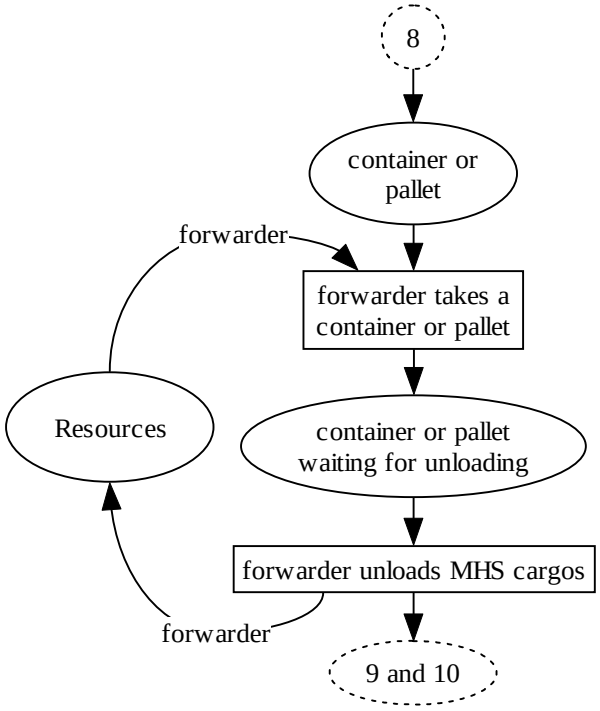

Figure 3.10 PN model for forwarders unloading MHS cargos

The process of “forwarders unloading MHS cargos” modeled as shown in Figure 3.10 resembles “forwarders unloading ASRS cargos” in Section 3.1.1. The unloading time depends on the quantity of cargos: more cargos take more time. At the end of this process, one token goes to “returning a container or pallet to the MHS” process; one token goes to “building skids” process.

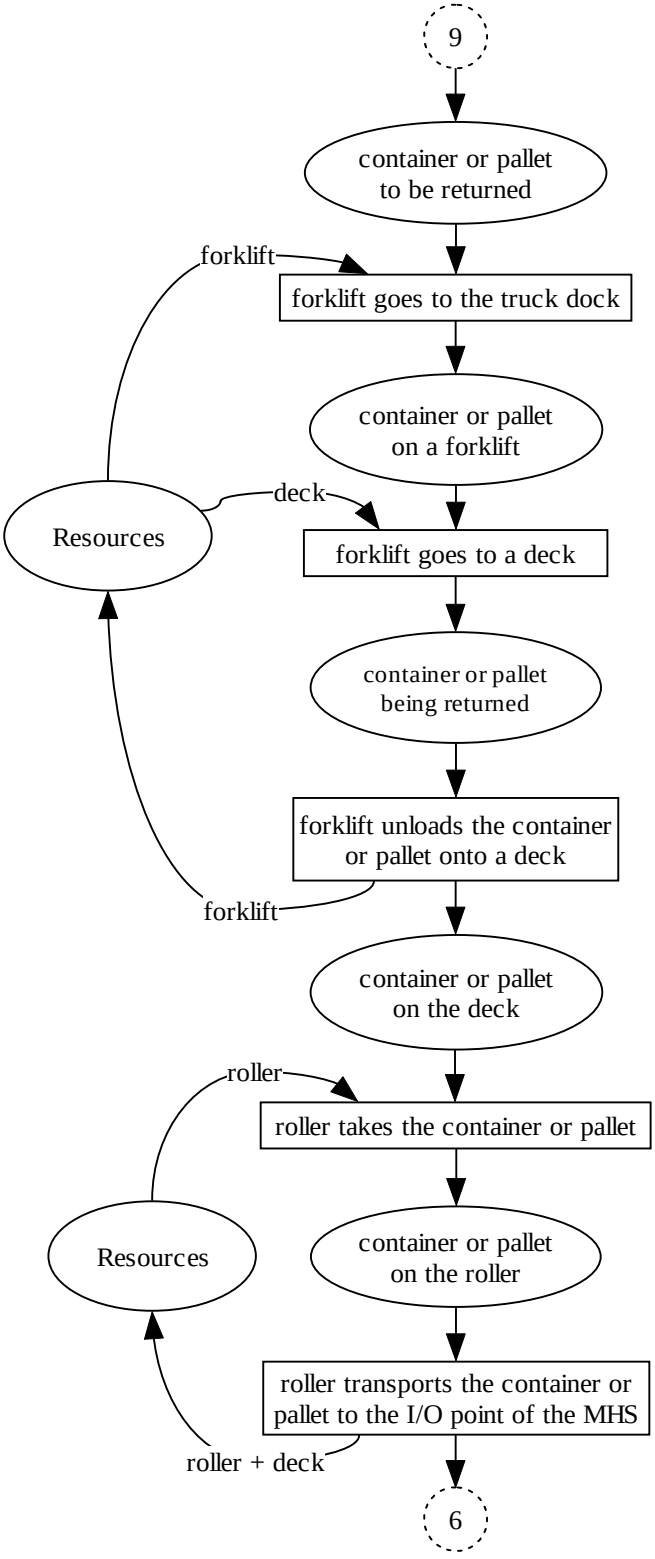

Figure 3.11 PN model for returning a container or pallet to the MHS

Since there is a rule in the observed terminal: after breakdown, the cargos remaining in

a container or on a pallet must belong to one forwarder only. Thus, transferring MHS cargos from a truck dock to another like AS/RS cargos does not happen. After a forwarder unloads his cargos from a container or pallet, the container or pallet is returned to the MHS. This process is modeled as shown in Figure 3.11. As illustrated in the figure, “returning a container or pallet to the MHS” is like the reverse process of “transporting MHS cargos to a truck dock”. Forklift is the type of resource used to transport the container or pallet from a truck dock to a deck while a mechanical roller of the workstation is used to transport the container or pallet from the deck to the I/O point of the MHS. Only when a deck is available, the forklift can start returning a container or pallet; and only when the roller removes the container or pallet from the deck, the deck is available for use again. After a container or pallet is returned to the I/O point, it is stored in a proper location by the ETV, which has been introduced in “the MHS operations”. The above description in Section 3.1.2 illustrates the PN model for MHS cargo retrieval process; the similarity of retrieval processes for ASRS cargos and MHS cargos is pointed out and the difference is compared.

3.1.3. Petri Net model for retrieving FG cargos

Compared with the retrieval processes for ASRS and MHS cargos, the retrieval process for FG cargos is much simpler. The whole FG cargo retrieval process is modeled as shown in Figure 3.12.

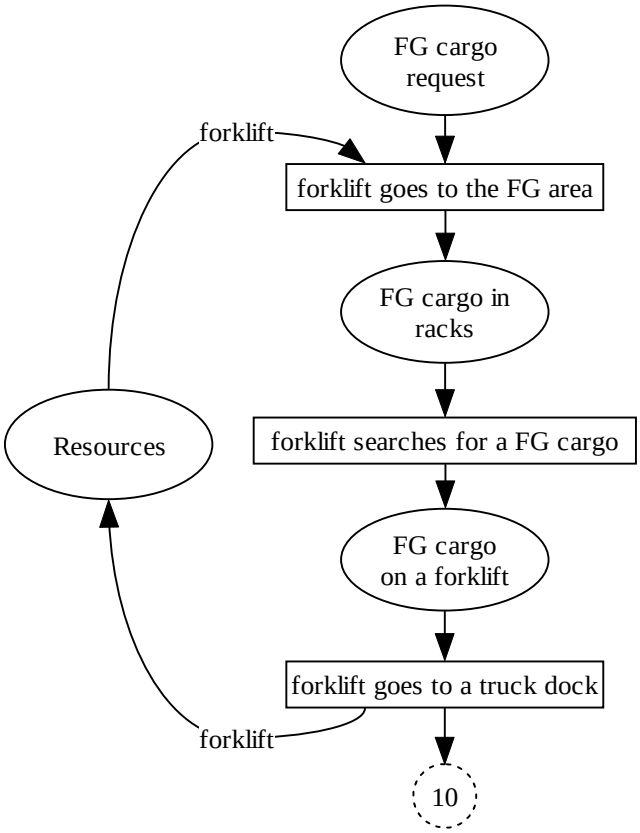

Figure 3.12 PN model for the FG cargo retrieval process

The only type of resource needed for the FG cargo retrieval process is forklifts. When there is a retrieval request for FG cargos, an available forklift goes to the FG area. Different from ASRS and MHS cargos, there is no record for specific locations of FG cargos. Thus, the searching operation is required. There are tens of racks in the FG area, and FG cargos are stored randomly, which means the requested FG cargo may be stored in any rack in this area. Therefore the searching time depends on how many racks have been searched when the forklift EO finally finds the requested cargo. Then after a forklift transports the FG cargo to a truck dock, it is released to be available for firing another transition. The FG cargo retrieval process is relatively short compared with the AS/RS and MHS cargo retrieval processes; the searching time varies a lot and

sometimes takes a large weight in the whole processing time of retrieving FG cargos.

3.1.4. Petri Net model for building skids

After cargos have been retrieved and transported to a truck dock, no matter whether they are AS/RS cargos, MHS cargos or FG cargos, they need to be built into skids before they can be loaded onto a truck. The building skids process is modeled as shown in the below figure.

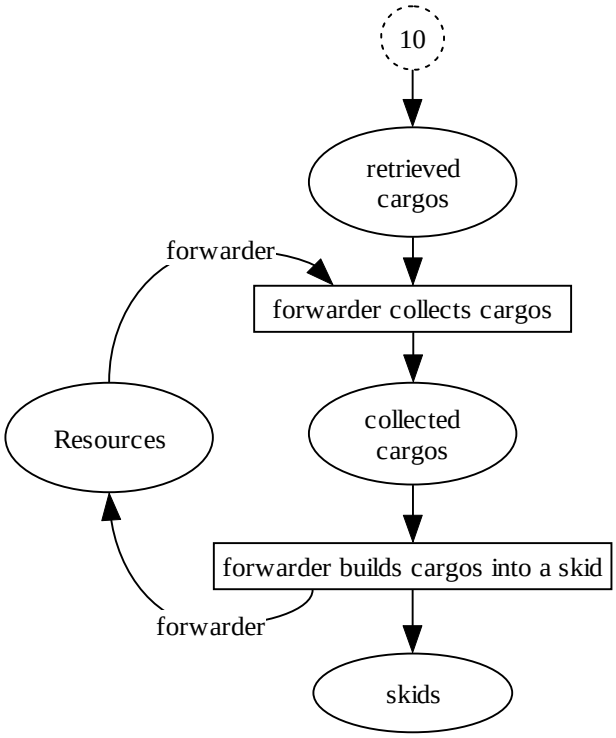

Figure 3.13 PN model for building skids

Forwarder is the only type of resource needed to complete this process. Building skids is a necessary process that can not be ignored, because when the quantity of cargos is large, especially if the cargos also vary in size and shape, the building skids process takes a

long time. Upon completion of the building skids process, the whole cargo retrieval process is regarded as ended in the model described in this chapter. The above description is about the PN model for the whole cargo retrieval process, including retrieving AS/RS, MHS and FG cargos and building skids.

3.2. Simulation model

Based on the PN model illustrated in Section 3.1, the corresponding simulation model is developed. As a simulation model, it is very close to the actual system, except for the following assumptions:

1. Equipments and operators of the same type have the same performance.
2. Accidents, like absence of operators and lost of cargos are not considered.
3. All the resources serve on a First Come First Served basis. In reality, this rule may be ruined by individual operators occasionally.

The simulation model contains initialization, main controller and output three main parts. The initialization part has two steps: the first step, reading data from the input file (Retrieval List); the second step, initializing all variables. The Retrieval List provides the following information: storage location, which storage area the requested cargo is being stored in; truck dock number, which truck dock is the requested cargo's destination; and request time, when the retrieval request is sent by the corresponding

forwarder. After reading all data from the input file, an initial future event list is generated. For the second step, among all variables, resource capacities are the most important ones; they are defined according to the actual terminal situation. The main controller part is the one to control the simulations after the initialization. It is based on the PN model described in the previous section, managing and updating the future event list. When all the events in the list have been executed, before the end of simulations, the output part generates simulation results. Two kinds of performance measures are included in the output file: flow time, the time consumed for retrieval processes, indicating the service level for forwarders; resource utilization, providing the status of resource usage.

To complete the whole simulation model, several modules are developed and plugged in. Since the AS/RS in the actual terminal has its own rules for operations, an AS/RS module is developed to fulfill its functions. The AS/RS is operating based on dual command cycles, i.e., in each cycle, it can perform one storage operation and one retrieval operation sequentially if both storage and retrieval orders exist. The adopted dwell point policy is after completion of a storage/retrieval operation, if no order in the AS/RS queue, the S/R machine will stay at the location where it finishes the last operation until a new order comes. The AS/RS layout is typical: one aisle with storage openings on its both sides and the I/O point is at the left-lower corner. The bins containing cargos are stored based on randomized storage policy. The AS/RS module serves all the functions as described above. Similarly, the MHS in the actual terminal

also has its own operating rules. It also operates based on dual command cycles and it has the same dwell point policy as the AS/RS. The containers and pallets are stored in it based on randomized storage policy, except that it reserves a certain level for empty containers or pallets recycle only. An MHS module is developed to serve these functions. Since the travel time of forklifts can not be simply computed by a single distribution, a layout module is developed. Firstly, this module provides the location list recording horizontal ordinates and vertical ordinates of all the locations where forklifts may reside. Secondly, the travel time model is developed, taking travel speed, acceleration/deceleration and turning movement into consideration. The parameters are different when forklifts carry different types of loads.

Before testing the simulation model, the model structure is summarized as follows. Due to the limitaton of measurements and analytical models, the technique of simulation is used for performance analysis of the cargo terminal. The CPN based simulation model constructs a virtual system, which is very close to the actual cargo terminal. Firstly, the retrieval process is represented by a CPN model, which is the base of the discrete event simulation model. Secondly, all the related elements are defined, such as resources and the terminal layout. Thirdly, parameters are set according to equipment specifications and data collected in the terminal, e.g., forklift speed, AS/RS dimensions, operation duration distributions and so on. For each operation, there is a certain corresponding duration rate; it is a constant or follows a specific distribution. Fourly, before the startup of simulations, the input must be preprepared. The retrieval

list for model validation is obtained from the actual terminal and the retrieval list for other performance experiments are generaetd according to terminal historical data. Thus the inputs and distributions needed to run the simulation model include: operation durations, terminal layout and AS/RS, MHS and FG area dimensions, equipment performance, actual resource configuration and retrieval list. At last, output is designed to log the progress of simulations and provide statistics of performance measures. Once start simulations, they first read data from the input file and generate an initial future event list. During the simulations, the future event list is updated continually. When there is no more future event, output is generated and simulations are ended. The developed simulation model is thus coded in C++ and can be implemented in PC environments. For the simulation experiments in the following two sections, the computer running time is short.

3.3. Model validation

In order to validate the proposed model, results from the model are compared with actual servicing times collected from the observed terminal. In the validation, flow time is used as the performance measure for cargo retrieval operations. The flow time is measured from the moment a forwarder requests cargos till the moment he collects all his requested cargos and builds them into skids. We determine transition times of the PN model based on equipment specifications and time study results. For the forklift travel time, speed, acceleration/deceleration and turning are taken into consideration; in

addition, a forklift with different types of cargos travels in different speeds. The “building skids” operation that palletizes cargos for loading onto trucks is an important element in the total flow time, because it takes quite a long time when the quantity of requested cargos is large. From the observation, the “building skids” time depends on whether the cargos are uniform cargos or complex cargos. Uniform cargos are in the same standard size, thus they cost less time for building skids; on the contrary, complex cargos vary one from another, which results in more time to build skids. A time study is conducted for building skids, as well as searching cargo, loading and unloading times for forklifts. We collect the data of arrival and departure times of forwarders along with the number of airway bills and the number of cargo pieces. The data obtained from the cargo terminal contains 13 retrieval cases, among which the biggest one is retrieving 573 pieces of cargos. Since the test is carried case by case, there are only 13 samples. Thus the validation is based on simple comparison, not statistical analysis. Since the arrival and departure times of forwarders are recorded in minutes, the actual servicing times are provided in minutes. To validate with the collected servicing times, we configure the resource capacity in the model to be the same with the terminal where data are collected.

Table 3.1 Comparison between actual servicing times and flow times from the proposed model

Case No.	No. of airway bills	Total No. of cargo pieces	Actual servicing time (min)	Flow time from the model (min)	Difference (min)
1	2	24	6	5	-1
2	1	4	1	1	0
3	7	128	25	23	-2
4	10	573	91	90	-1
5	1	14	5	6	+1
6	3	36	6	6	0
7	2	34	5	6	+1
8	5	378	68	61	-7
9	1	149	24	19	-5
10	1	1	2	2	0
11	3	244	44	39	-5
12	1	73	12	12	0
13	2	13	6	6	0
Total	39	1671	295	276	-19

The results are shown in Table 3.1. Most flow times from the proposed model are approximate to the corresponding actual servicing times, thus the developed model

provides a good estimate for the terminal performance. Since the model does not consider any time delay caused by interruptions between operations, missing cargos or equipment breakdown, flow times tend to be underestimated in most cases.

3.4. Comparing performance of retrieval operations based on different numbers of forklifts

To show the functionality of the model presented in this chapter, a simple application is illustrated in this section. The system performance based on different numbers of forklifts is compared by the developed model. As one of the key resources in terminals, forklifts function as the main transport vehicle, thus changing the number of forklifts may affect the system performance a lot.

According to historical data of retrieval requests in the actual terminal, the Retrieval List is generated to be used as input. The Retrieval List triggers the execution of simulations. All resource capacities expect the number of forklifts and all parameters are kept unchanged and the same as the actual terminal situation. The number of forklifts is the only factor that changes the system performance. According to the literature, for similar systems, two kinds of performance measures are of great interests. Firstly, flow time, or turnaround time, different from throughput, which is the theoretical value of best performance systems can reach, flow time indicates the actual system performace in real operations. The flow time in our model is very important, since it reflects the service

level for forwarders, the customer of air cargo terminals. Secondly, resource utilization, which no doubt provides the invaluable information about use of resources, has been measured in almost all similar systems. This kind of measures is very important to evaluate the efficiency of terminal operations. Thus, two measures are defined: Retrieval Cycle Time, from the time when a retrieval request is sent by a forwarder till the time when the requested cargo is built into a skid; Forklift Utilization, total service time of all forklifts divided by their total working hours. Actually, Forklift Utilization does not only show the utilization of this equipment, but also the manpower, forklift EO.

From the experiment results for the purpose of test, it is identified that the stead-state performance measures can be obtained when the number of retrieval requests for input is no less than 3,000. For the reason of convenience, each replication employs a Retrieval List of one week's retrieval requests as input, which contains around 4,000 retrieval requests; to provide enough retrieval lists for input, a quarter's Retrieval List is generated, thus the simulation results for each configuration are based on 12 replications. For each replication, all the configurations have the same Retrieval List as input, and employ the same random number generator with the same seed. Simulation results are provided in the following table:

Table 3.2 Simulation results for different numbers of forklifts

Number of Forklifts	Retrieval Cycle Time (sec)		Forklift Utilization	
	Mean	Standard Deviation	Mean	Standard Deviation
1	1203.49	170.63	64.61%	3.72%
2	278.06	7.94	39.33%	1.07%
3	259.28	7.24	26.27%	0.69%
4	256.83	6.68	19.87%	0.52%

The average of Retrieval Cycle Time of all the requests in one replication is recorded as Average Retrieval Cycle Time. Then the mean of Average Retrieval Cycle Time of 12 replications is listed in the table. And the mean of Forklift Utilization of 12 replications is also listed. Take both Retrieval Cycle Time and Forklift Utilization into consideration, Thress-forklift configuration is better than the other three configurations, because it provides the nearly best performance and has the reasonable forklift utilization. Two-forklift configuration is another good choice; it can finish retrieval requests in reasonable time and with the higher utilization. Number of forklifts need not exceed 3, since more forklifts than 3 can not further improve the system performance much but at the same time decrease the utilization. One forklift is not enough; it can not complete retrieval requests in time.

Workload in cargo terminals is determined by fixed flight schedules and stochastic forwarder requests, which results in workload unbalance among different time in a day.

It is observed in the actual terminal that the difference between forklift utilization in peak hour (around midnight) and non-peak hour is obvious. The following figure shows the forklift utilization in different time of a week:

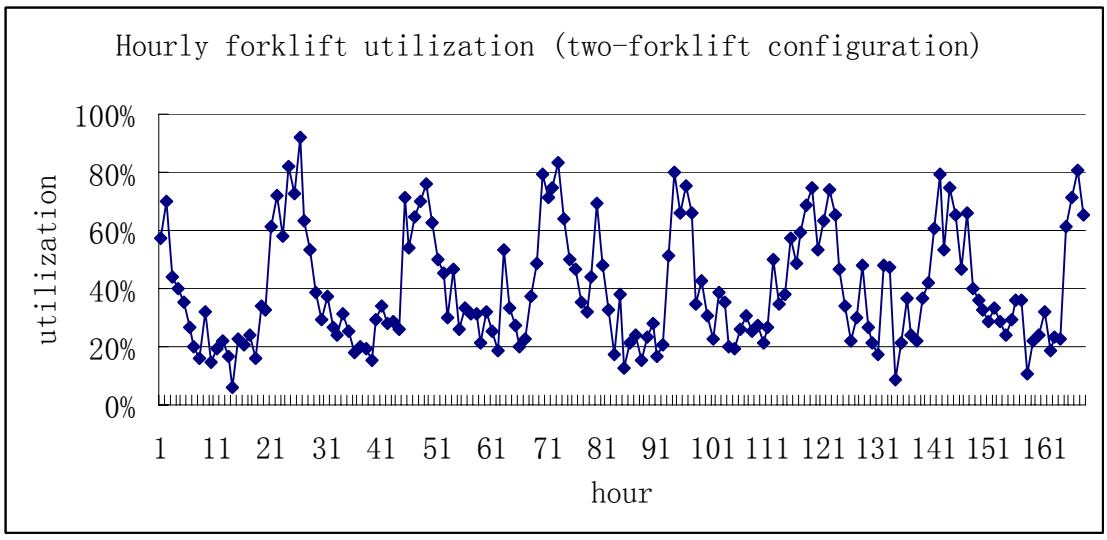

Figure 3.14 Hourly forklift utilization for two-forklift configuration

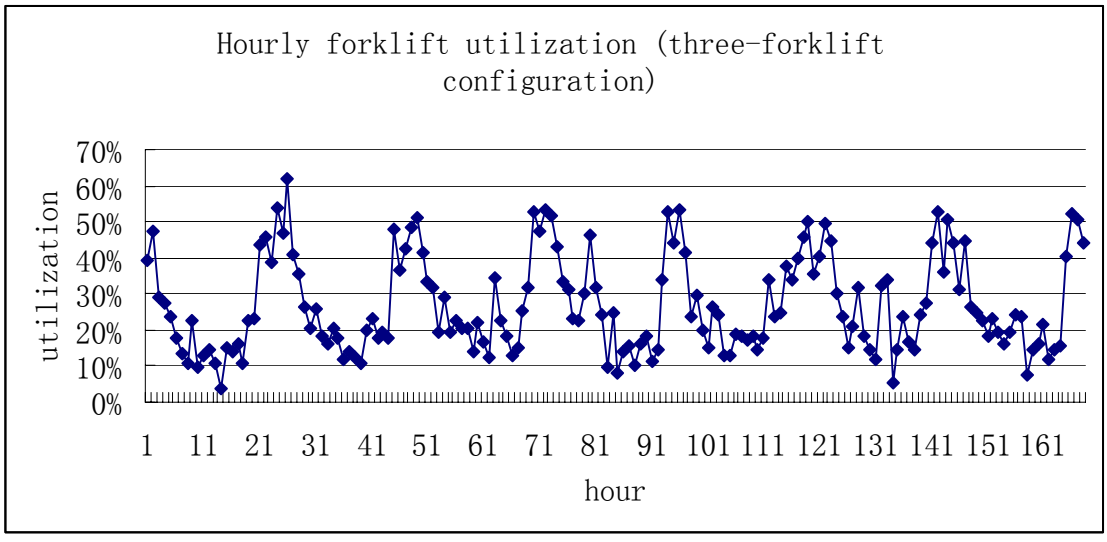

Figure 3.15 Hourly forklift utilization for three-forklift configuration

In one week, there are 168 hours. The forklift utilization in each hour is recorded. As the results shown in the above figure, whatever 2 or 3 forklifts configuration, the Forklift Utilization is high during peak hour (from 8 PM to 5 AM) and low during non-peak hour. Moreover, the Forklift Utilization of 2 forklifts configuration during non-peak hour is normally no more than 50%, which implys during non-peak hour, 2 forklifts configuration has already reached the nearly minimum Retrieval Cycle Time. 3 forklifts configuration can give little improvement on the Retrieval Cycle Time during non-peak hour; resource is wasted. We also analyze the input of retrieval requests, it is found the workload, i.e. number of retrieval requests per hour, in non-peak hour is only about half of that in peak hour.

Thus, one adjustment can be applied to the 3 forklifts configuration, i.e., keeping 3 forklifts during peak hour but reducing the number of forklifts to 2 only during non-peak hour. Simulation result of this adjusted configuration is provided in the following table.

Table 3.3 Simulation results for the adjusted configuration

Retrieval Cycle Time (sec)		Forklift Utilization	
Mean	Standard Deviation	Mean	Standard Deviation
262.09	7.45	33.14%	0.89%

From the result provided in Table 3.3, conclusion can be drawn that the adjusted

configuration can be the best configuration. Compared with the 3 forklifts configuration, the adjusted configuration saves 21% resource with only 1% increasement in the Average Retrieval Cycle Time; at the same time, the Forklift Utilization is improved by 26%.

4. Developing the reservation storage policy for automated storage/retrieval systems in import terminals

This chapter describes and analyzes the characteristics of AS/RS in import terminals in order to improve their throughput. A new storage policy is proposed to reduce the expected S/R machine travel time. Both analytical and simulation models are developed to compute the travel time. The Markov model computes the exact travel time, thus the least travel time can be identified. Another proposed analytical model approximates the travel time to save computational effort, the result from which has the same trend as that from the Markov model. The simulation model is based on the Markov model, which provides a very good estimation on the travel time and needs only a little computational effort thus can be applied to solve real industrial-size problems.

4.1. Analytical models for the reservation storage policy

In import terminals, after flights arrive and cargos are broken down, loose cargos request to be stored in AS/RS. When forwarders come to terminals and send retrieval requests, these cargos will be retrieved. Thus the storage duration of AS/RS cargos is defined as the time interval from the moment when they are stored to the moment when they are retrieved. This can be regarded as a general definition for AS/RS cargo storage duration. Some researchers assume this value is constant and known for each

item, while we consider the stochastic storage duration. Because forwarders come to collect cargos at their own convenient time which means the storage duration of each item can not be predicted as a constant. However, based on the historical data from cargo terminals, we still can treat the storage duration of terminal AS/RS cargos as a variable following a known probabilistic distribution. Another important variable affecting the AS/RS operation performance is the cargo volume. Many factors influence the arrived cargo volume in each unit time, e.g., flight type. We can also derive the probabilistic distribution for the cargo arrival rate from historical data. To simplify the problem in this chapter and make it tractable, hour is used as the time unit, i.e., the number of storage requests in one hour is defined as cargo arrival rate and the storage duration is measured in hours and rounded to be integers. For example, 2 cargos arrive in one hour; one will stay for 3 hours and the other will stay for 5 hours.

To minimize the expected travel time for S/R machines, one popular approach in literature is sorting items according to their turnover rates, which is the reciprocal of storage durations, and assigning items with higher turnover rates, i.e., shorter storage durations to storage openings with shorter travel times to the I/O point, which is called turnover-based storage policy in literature. If group items into classes to make each class contain items with similar turnover rates and assign classes of higher turnover rate items to storage openings with shorter travel times to the I/O point, it is the class-based storage policy. For the class-based storage policy, within each class, items are stored randomly. Research results in literature show applying the class-based

storage policy with a few classes could reduce travel time to be approximate to the minimum travel time provided by the turnover-based storage policy. Although AS/RS cargos do not have known and constant storage durations, the same principle, improving the utilization of storage openings with shorter one-way travel times to the I/O point, is followed. Since there are more than one hundred forwarders totally, it is impractical to treat each forwarder as one class. It is found that some forwarders own large volume of cargos, thus they often come to terminals to collect cargos while the other forwarders have only a few cargos and seldom come to collect cargos. Therefore, all the forwarders are divided into two groups: the forwarders whose cargos have shorter expected storage durations belong to “Big Group” (BG) and the other forwarders whose cargos have longer expected storage durations belong to “Small Group” (SG). These two groups are regarded as independent. According to the class-based storage policy in literature, the storage space requirement of BG and SG cargos should be computed. However, due to the stochastic nature of the storage duration for AS/RS cargos, it is difficult to estimate the storage space requirement since it is apparently not the expected storage space requirement. One way is to reserve enough storage openings to meet the maximum possible storage space requirement, but it is obviously uneconomical and even impossible in most actual systems. In this chapter, our objective is to identify the number of reserved storage openings for BG cargos, which reduces the expected S/R machine travel time most. The general assumptions for the proposed models are listed below:

1. Each bin contains only one unit of cargo.

2. All bins and storage openings are in the same size, thus all storage openings are candidates for storing any bin.

3. The AS/RS studied here has only one aisle, which is served by one S/R machine.

4. The I/O point of the AS/RS is located at the lower left-hand corner.

5. The S/R machine forking time is constant and not included in the travel time.

6. Interleaving is ignored. The AS/RS is operating in single command cycles only. For a storage order, the S/R machine picks up a bin at the I/O point, travels to the pre-assigned storage opening, puts the bin into that opening and goes back to the I/O point; for a retrieval order, the S/R machine travels from the I/O point to the storage opening keeping the requested cargo, picks up the bin and takes it to the I/O point. Both storage and retrieval orders initiate and end at the I/O point. Under this operating condition, the travel times of storage and retrieval operations for each cargo are exactly the same. The travel time for both storage and retrieval operations for one cargo can be computed by simply doubling the round-trip travel time for either the storage or the retrieval operation for that cargo and the round-trip travel time is exactly as twice as the one-way travel time. In this chapter, we take the one-way travel time for storage operations to represent the travel time, i.e., only the travel time for the S/R machine traveling from the I/O point to a pre-assigned opening is counted.

7. The system layout is known. The S/R machine travel speed is known and constant.

8. The travel time satisfies Tchebychev metric.

9. The storage duration and cargo arrival rate follow known distributions. Probabilistic density functions are derived from historical data.

10. All new cargos arrive together at the beginning of each hour. Thus they can be stored in any sequence, no priority for BG or SG cargos. Since the time spent on the operations is very short compared with the storage duration, it has no impact on the storage duration. And at the end of each hour, all leaving cargos are removed in a very short time.

To illustrate the new proposed reservation storage policy clearly, a simplified example is given. For each hour, two BG cargos and two SG cargos arrive, which means the cargo arrival rates for two groups in this example are restricted to be constant. For each cargo, whatever it belongs to BG or SG, it may stay in the system for one or two hours, which means the storage duration for cargos of two groups has only two possibilities: either one hour or two hours. Thus, the cargo arrival rates and the possible storage durations are same for both BG and SG cargos. But BG cargos have the shorter storage duration with higher probability than SG cargos, i.e., BG cargos may stay for one hour with the probability 0.9 and two hours with the probability 0.1 while SG cargos may stay for one hour with the probability 0.6 and two hours with the probability 0.4. Then BG cargos have shorter expected storage duration than SG cargos.

It is easy to know the space required to store cargos in this example is 8. Thus for each hour, the status of storage openings can be represented by an 8-digit number. The 8-digit number is written in this manner: the first digit represents the storage opening with the shortest travel time to the I/O point, the second digit represents the opening with the second shortest travel time to the I/O point, and so forth, the last digit represents the farthest opening to the I/O point. And we use different numbers to represent different status of each storage opening: “1” represents one new arrived BG cargo stored in that storage opening, “2” represents one new arrived SG cargo stored in that storage opening, “0” represents that storage opening is empty and “3” represents one cargo arrived in the last hour stored in that storage opening. Since the maximum storage duration is only two, the cargos arrived one hour ago will leave the system in the next hour for sure. Thus, old BG and SG cargos are not discriminated. For example, 12210033: at that moment, two BG cargos are stored in the first and fourth storage openings, two SG cargos are stored in the second and third storage openings, two cargos arrived one hour before remain in the seventh and eighth storage openings and the fifth and sixth storage openings are empty. This is the state representing the status of all storage openings in that hour. In different hours, the states are different. For the above illustrated state, in the next hour, the first four openings will change to “0” or “3”, depending on whether these cargos will leave or stay; the two “3” will be removed to be “0”; but “0” may be occupied again by new arrived cargos immediately; the two “0” may remain “0” or occupied by BG or SB cargos. Thus a discrete time Markov

chain model is employed to generate the full state space.

Two storage rules exist. Nearest rule: store cargos to the openings with the shortest travel time to the I/O point if possible and not against any other rule. This rule is from the closest opening storage policy, which is often treated as the equivalent to the randomized storage policy in literature. In this chapter, the nearest rule is always valid for cargo storage unless further illumination is given. Some states are eliminated due to this rule, e.g., 01122000. Reservation rule: a certain number of openings closest to the I/O point can be reserved for BG cargos only. For example, if the first two openings are reserved, states like 12123003 are eliminated. The state transition probability can be obtained since the probabilities for different storage durations of different cargos are already known. By solving the state matrix, the limiting probability for each state is obtained. Without affecting the essence of this problem, the one-way travel time function of each opening to the I/O point in the simplified example is defined as:

$$y_i = i \tag{4.1}$$

i index of storage openings
y_i travel time from the ith opening to the I/O point, in seconds

Thus, the expected one-way travel time for one storage operation is computed as shown in the following table:

Table 4.1 Expected travel times for the simplified example

Number of Reserved Openings	Expected Travel Time (s)
0	3.074
1	3.006
2	**2.938**
3	3.326
4	3.820

In this example, the number of BG cargos in the system at any time is no more than four. Thus the expected travel times are computed for the number of reserved openings from 0 to 4. When reserving 0 opening, it is actually the closest opening storage policy. When reserving 4 openings, it resembles the class-based storage policy in literature; one enclosed storage zone for one class of cargos, no shared storage openings allowed. It can be seen from the results shown in Table 4.1 that reserving 2 openings for BG cargos is optimal in terms of minimizing the expected S/R machine travel time in this example. When reserving 2 openings, if there is any BG cargo can not be stored in the reserved area, it will be stored outside the reserved area to be mixed with SG cargos. From this example, it is recognized that due to the stochastic nature of storage duration, simply applying the class-based storage policy may be suboptimal, or even worse than the closest opening storage policy.

But one drawback is found in the above applied Markov model. Even for small-size

problems, like the simplified example, the state space is still very large. Thus it is unrealistic to apply the Markov model to real industrial-size problems. Then an approximation model is proposed, which can save much computational effort. The approximation model unlike the Markov model does not consider the specific storage location for each cargo. It only considers the inventory level (amount of cargos) and computes travel times based on the following assumption. It is assumed that at any time all the cargos in the system are stored in the possible closest openings while not against the reservation rule, which means all the cargos are compact to be close to the I/O point as much as possible. Unlike the Markov model, which captures the dynamic feature of the system storage status, the approximation model only considers the static state of the system storage status, assuming cargos are stored in the ideal way. Thus it is not difficult to imagine that results from the approximation model underestimate the expected travel time. Details of the approximation model are explained as follows. We first consider the travel time in one hour.

Notation and Assumption

N total number of openings in the system, large enough

y_i one-way travel time from the ith opening to the I/O point
satisfy $y_i \le y_{i+1}$ for $1 \le i \le N$ -1

s number of reserved storage openings for BG cargos

s_B number of storage openings required by BG cargos, >0

s_S number of storage openings required by SG cargos, >0

v_B number of new arrived BG cargos, constant, >0

v_S number of new arrived SG cargos, constant, >0

z total one-way travel time for storage operations in one hour

z can be computed by

$$\text{if } s_B \le s \quad z = \left(\frac{1}{s_B} \times \sum_{i=1}^{s_B} y_i\right) \times v_B + \left(\frac{1}{s_S} \times \sum_{i=s+1}^{s+s_S} y_i\right) \times v_S \tag{4.2}$$

$$\text{else } s_B > s \quad z = \left(\frac{1}{s} \times \sum_{i=1}^{s} y_i\right) \times \left(\frac{s}{s_B} \times v_B\right) + \left(\frac{1}{s_B + s_S - s} \times \sum_{i=s+1}^{s_B+s_S} y_i\right) \times \left(v_S + \frac{s_B - s}{s_B} \times v_B\right) \tag{4.3}$$

When the reserved space is enough for storing all BG cargos, (4.2) is used to compute the total one-way travel time for storing new arrived BG and SG cargos in that hour. Else when the reserved space is not enough, the travel time can be obtained from (4.3). Parts in the formulas like $\frac{1}{b-a+1}\sum_{i=a}^{b} y_i$ compute the average (expected) one-way travel time to the I/O point for openings with the indexes from *a* to *b*. When the reserved space is not enough, new arrived BG cargos are expected to be stored in the reserved area with the ratio $\frac{s}{s_B}$ and the others are stored with SG cargos. The total number of storage openings is temporarily assumed to be large enough for any applied storage policy. Later, the feasibility of each storage policy can be examined by the number of storage openings in the actual systems. To apply the approximation model easily, cargo arrival rates are assumed to be constant.

Since the storage duration is considered as a stochastic variable, the required storage space is definitely a stochastic variable determined by it. Once the probabilistic density function of the storage duration is given, the probabilistic density function of the required storage space can be found. Then the expected travel time can be computed as follows:

Notation:

T_B maximum storage duration of BG cargos, integer hours

T_S maximum storage duration of SG cargos, integer hours

$P(s_B)$ probability of s_B

$P(s_S)$ probability of s_S

z total one-way travel time for storage operations in one hour

Z expected one-way travel time for storage operations in one hour

$$Z = \sum_{s_B=v_B}^{v_B \times T_B} \sum_{s_S=v_S}^{v_S \times T_S} \left(z \times P(s_B) \times P(s_S) \right) \tag{4.4}$$

In every hour, v_B Big cargos arrive and they stay in the system at least for 1 hour and at most for T_B. Thus, the number of required storage openings for Big cargos is from v_B to $v_B \times T_B$. Similarly, the number of required storage openings for Small cargos is from v_S to $v_S \times T_S$. With known v_B, v_S, T_B, T_S and the probabilistic density functions for storage durations of two group cargos, $P(s_B)$ and $P(s_S)$ are obtained for each s_B and s_S. For each number of reserved openings, Z can be computed as the weighed average of z using (4.4). Since the cargo arrival rates are constant, expected one-way travel time for one storage operation can be easily obtained from the expected total one-way travel time in one time unit. To compare the approximation model with the Markov model, the results for the simplified example from the approximation model are generated.

Table 4.2 Compare the approximation model with the Markov model

Number of Reserved Openings	Exact Travel Time (s)	Approximated Travel Time (s)
0	3.074	3.000
1	3.006	2.923
2	2.938	2.846
3	3.326	3.254
4	3.820	3.750

Table 4.2 shows the comparison of travel time between results from the Markov and the approximation models. It is shown results from the approximation model are very close to exact values from the Markov model. And it is also verified that results from the approximation model tend to be a little underestimated.

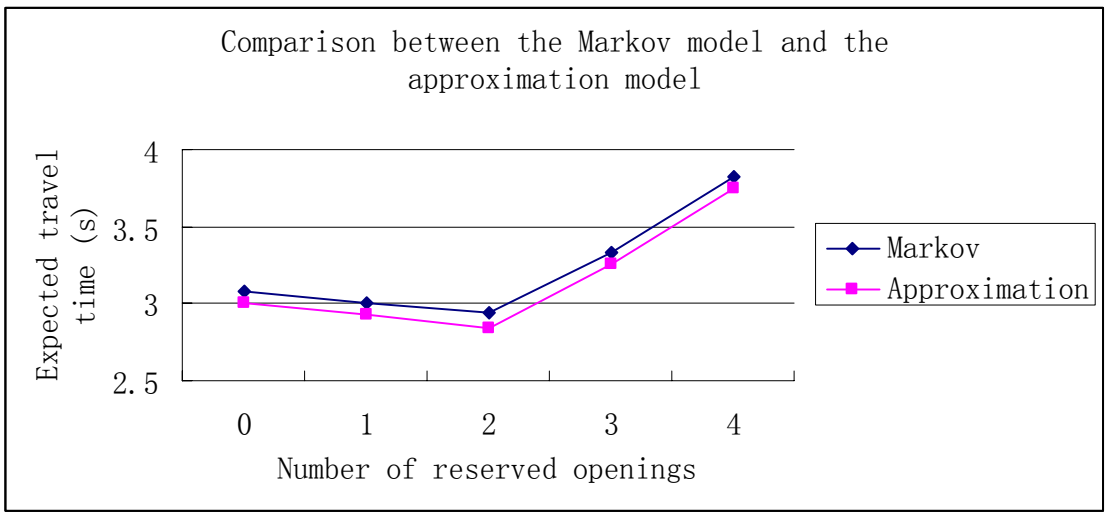

Figure 4.1 Comparison between the Markov model and the approximation model

Figure 4.1 shows the result from the approximation model has the same trend as that

from the Markov model. The curves are convex, which is reasonable. There should be a lowest point, but it is difficult to prove the convexity. Thus this problem is left for future research. Since the approximation model need less computational effort, it can solve larger size problems. As an analytical model, the approximation model can provide exact results and due to its approximation, it can solve larger size problems than the Markov model can, thus it is one of the alternative solutions for the proposed problem. However when both storage duration and cargo arrival rate are considered as stochastic variables, it is extremely difficult to get the probabilistic density function of required storage space for two group cargos. Thus simulation is applied, which relaxes the assumptions needed by the Markov and the approximation models.

4.2. Simulation study

In this section, simulation is used to compare different storage policies. The simulation model is based on the Markov model, following the same storage rules and also capturing the dynamic feature of system storage status, which means considering the specific storage location of each cargo. The simulation model is actually an approximation calculation tool for the Markov model, which employs sampling and avoids enumerating all states. The advantage of the simulation model is elimination of the state explosion problem in the Markov model. Appropriate running periods and number of replications are tested to assure accurate results. To validate the simulation model, the results for the simplifed example are generated from the simulation model.

Table 4.3 Compare the simulation model with the Markov model

Number of Reserved Openings	Exact Travel Time (s)	Travel Time from Simulations (s)	Difference
0	3.074	3.073	-0.033%
1	3.006	3.003	-0.100%
2	2.938	2.942	0.136%
3	3.326	3.326	0.000%
4	3.820	3.819	-0.026%

It is shown in Table 4.3, the simulation model provides a very good estimation for the exact values from the Markov model, at the same time it needs almost negligible computational effort compared with the Markov model and the approximation model. Thus we apply the simulation model to compare different storage policies based on the actual data from the observed terminal. The input data include: cargo arrival rate, storage duration and one-way travel time for each opening to the I/O point. The probabilistic density functions of arrival rates and storage durations for two group cargos are derived from the data; the maximum cargo arrival rate is 30 cargos per hour and the maximum storage duration is 24 hours. From the AS/RS specification, the layout and S/R machine speed are known, thus the one-way travel time from each opening to the I/O point is obtained. With all the given information, travel times based on different numbers of reserved storage openings can be obtained from the simulation model. Part

of the simulation results are shown in Tables 4.4 and 4.5.

Table 4.4 Simulation results for different numbers of reserved openings

Number of Reserved Openings	One-way Travel Time (sec)		Storage Space Requirement		Maximum Storage Space Requirement
	Mean	Standard Deviation	Mean	Standard Deviation	
0	4.7010	0.0164	65.33	0.43	127
1	4.6896	0.0164	65.37	0.43	127
2	4.6776	0.0162	65.43	0.41	127
3	4.6699	0.0164	65.48	0.42	127
4	4.6617	0.0159	65.54	0.42	127
5	4.6567	0.0157	65.62	0.42	127
6	4.6520	0.0155	65.69	0.42	127
7	4.6493	0.0153	65.77	0.40	127
8	**4.6474**	**0.0157**	**65.86**	**0.40**	**127**
9	4.6477	0.0155	65.97	0.41	127
10	4.6494	0.0158	66.10	0.40	127
11	4.6521	0.0159	66.23	0.41	127
12	4.6569	0.0161	66.39	0.42	127

It is shown in Table 4.4 that reserving 8 storage openings produces the least expected

S/R machine one-way travel time for storage operations. The third column in the table is the average storage space requirement. In each hour, the farthest opening to the I/O point which is not empty determines the storage space requirement in that hour. The average of storage space requirements in all time units increases as number of reserved openings increases. Maximum storage space requirement is the index of the farthest storage opening that has ever been used in at least one hour, which determines the feasibility of a proposed storage policy. It is found that less than a certain number; the increment of reserved openings is not large enough to raise the maximum storage space requirement. Since the actual AS/RS in the observed terminal has totally 188 storage openings, the reservation storage policy with 8 reserved openings is applicable.

Since the maximum storage space requirement can be identified by the simulation model, it is found the maximum storage space requirement for BG is 76 openings, which means the probability of number of storage openings required by BG exceeding 76 is less than 1.5×10^{-5}. Thus, the results of travel time based on a 2-class-based storage policy can be obtained. Part of the simulation results for number of reserved openings no less than 76, which grants the feasibility of the 2-class-based storage policy are provided.

Table 4.5 Simulation results for the 2-class-based storage policy

Number of Reserved Openings	One-way Travel Time (sec)		Storage Space Requirement		Maximum Storage Space Requirement
	Mean	Standard Deviation	Mean	Standard Deviation	
76	**7.9349**	**0.0317**	**117.26**	**0.32**	**165**
77	7.9763	0.0320	118.26	0.32	166
78	8.0197	0.0325	119.26	0.32	167
79	8.0648	0.0325	120.26	0.32	168
80	8.1120	0.0326	121.26	0.32	169

Reserving 76 openings is regarded as the representative of the 2-class-based storage policy. The results in Table 4.5 show the 2-class-based storage policy has the expected travel time 71% worse than the reservation storage policy and it is even 69% worse than the closest opening storage policy (reserving 0 storage opening). Moreover, its space requirement is also higher than the other two policies.

At last, we compute the expected travel time of the current operating storage policy, uniform storage policy, which stores cargos randomly in any empty opening. Unlike the closest storage policy, which actually can reduce a lot of travel time for the proposed problem, the uniform storage policy is the real randomized storage policy, i.e., any opening in the system has the equal chance to be used. Then the expected travel time of

the uniform storage policy can be simply computed as the average of one-way travel times of all storage openings in the AS/RS to the I/O point. One-way travel times of all the 188 storage openings are know, then the result is easily obtained as 12.65 seconds, which is nearly three times as the minimum one from the reservation storage policy. The expected travel times and storage space requirements for different storage policies by using the actual data are summarized in the following table.

Table 4.6 Comparison among different storage policies

Storage policies	Expected travel time (s)		Storage space requirement	
	Value	Percentage	Value	Percentage
Closest opening	4.70099	101.15%	**127**	100.00%
Reservation	**4.64741**	100.00%	**127**	100.00%
Class-based	7.9349	170.74%	165	129.92%
Uniform	12.65	272.19%	188	148.03%

From the results shown in Table 4.6, it is not difficult to recognize that the reservation storage policy is the best storage policy among all the four policies for the proposed problem, which produces the minimum expected travel time without increasing the storage space requirement. In the provided results, the closest opening storage policy shows nearly the same performance as the reservation storage policy; it is due to the actual data. When data change, which means the cargo arrival rate and storage duration follow other probabilistic density functions, the advantage of reservation storage policy

can be larger.

In this chapter, three models are proposed to compute the expected travel time for different storage policies. The Markov model gives accurate answers for limited problems, the approximation model provides exact results for larger size problems and the simulation model estimates the travel time for industrial size problems with high accuracy.

5. Conclusions and future research

This chapter concludes the whole thesis and identifies related future research.

5.1. Conclusions

In this thesis, the objective is to improve the performance of cargo terminal operations. Two problems are studied. Firstly, a CPN model for the cargo retrieval operations in an import terminal is developed. PN is employed to model the retrieval operations, and a corresponding simulation model is developed. The model is validated by using actual service times from the observed terminal. Since the results show flow times from the model are approximate to servicing times in real operations, this model is capable of giving an insight into the behavior of the current terminal operations and evaluating system performance for proposed changes in the operation design and control. Furthermore, the developed model is modularized, and thus, modules can be added and modified without difficulty. This model simulates air cargo terminal operations effectively and efficiently; simulation results are generated for a simple application.

Secondly, the storage operation of AS/RS in import terminals is studied. Unlike common AS/RS which are operating based on inventory control policies, storage and retrieval of terminal ASRS cargos are determined by flight schedules and forwarder requests, which can not be controlled by inventory policies for AS/RS in literature. The

uniform, closest opening, 2-class-based and reservation storage policies are examined in order to reduce the expected S/R machine travel time. In literature, the closest opening storage policy is treated as randomized storage policy and the class-based storage policy can save much travel time. However, for the problem proposed in this thesis, due to the stochastic nature of storage duration, a different result is found. The 2-class-based storage policy does not provide minimum travel time but the closest opening storage policy can save much travel time. The new proposed reservation storage policy is applicable and provides the least expected S/R machine travel time among all the examined storage policies. Although this policy is initially designed for particular AS/RS in import terminals, it can be used in other automated warehousing systems which have the similar characteristics.

5.2. Future research

In Section 3.4, the developed model is applied to compare the system performance based on different numbers of forklifts. It is simple; the best configuration is determined by balancing the average cycle time and forklift utilization. If the cost estimation is obtained, future research can aim to optimize the number of forklifts in terms of minimizing the cost. The performance of air cargo terminal operations is affected by a lot of different factors, such as workload distribution, manpower schedule, terminal layout and equipment capacity. Investigating the system performance affected by other changes, such as layout reconfiguration, manpower reassignment or changes on

workload or equipments other than forklifts is of interest. In this thesis, the performance of terminal operations is analyzed by means of simulations; a CPN model is developed. Although it is very difficult to build any analytical model to substitute for the simulation model completely, building specific analytical models for parts of the system is helpful for the system evaluation. More performance measures, e.g., request waiting time, can provide additional information on evaluation for the system operating status. Performance measures like waiting time can be used to identify the system bottleneck. In this thesis, retrieval flow times from the model are optimistic compared with servicing times from the actual terminal, because some noises are ignored in the current model, such as manpower work rate fluctuation, process delays caused by retrieving special cargos and interactions between the retrieval process and other terminal processes. If the information or data for the above mentioned noises can be obtained and these noises can be considered in the model, it can give a better estimation for the terminal operations.

For the problem of reducing expected S/R machine travel time to improve AS/RS throughput, the reservation storage policy based on 2-class has been developed. Extending this policy to be applied to multi-class problems can broaden its range of application. The difficulty is the storage rules for 2-class systems may not be applied to multi-class systems directly. Thus, the storage rules and analytical models have to change. Sensitivity analysis on the number of reserved openings is worth investigating. Whether the results of travel time are sensitive to the distributions of cargo arrival rate

and storage duration, and one-way travel time from each opening to the I/O point or not is of great research interest. Any other analytical model which can provide approximate results and need only reasonable computational effort for real industrial-size problems deserves research too. The results on the reservation storage policy proposed in this thesis can be regarded as a preliminary exploration; it is critical to conduct a more detailed comparison between it and other storage policies in literature. Furthermore relaxation of some assumptions in this thesis may make the model more realistic and still manageable. For example, to simplify the problem, only systems operating based on single command cycles are investigated. Studying systems operating based on dual command cycles while integrating proper dwell point and interleaving policies can further reduce the expected travel time.

Reference

Al-Jaar, R.Y. and A.A. Desrochers. Performance Evaluation of Automated Manufacturing Systems Using Generalized Stochastic Petri Nets, IEEE Transactions on Robotics and Automation, 6(6), pp. 621-639. 1990.

Ashayeri, J., R.M. Heuts, M.W.T. Valkenburg, H.C. Veraart and M.R. Wilhelm. A geometrical approach to computing expected cycle times for zonebased storage layouts in AS/RS, International Journal of Production Research, 40(17), pp. 4467-4483. 2002.

Borusan, A. Colored Petri net based modeling of FMS. In Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'93), October 1993, Le Touquet, France, pp. 54-59.

Bozer, Y.A. and M. Cho. Throughput performance of automated storage/retrieval systems under stochastic demand. Technical Report, The University of Michigan, Ann Arbor. 1998.

Bozer, Y.A. and J.A. White. Design and performance models for end-of-aisle order picking systems, Management Science, 36(7), pp. 852-866. 1990.

Bozer, Y.A. and J.A. White. Travel-time models for automated storage/retrieval systems, IIE Transactions, 16(4), pp. 329-338. 1984.

Chang, D.T., U.P. Wen and J.T. Lin. The impact of acceleration/deceleration on travel-time models for automated storage/retrieval systems, IIE Transactions, 27(1), pp. 108-111. 1995.

Chang, S.H. and P.J. Egbelu. Relative pre-positioning of storage/retrieval machines in

automated storage/retrieval system to minimize expected system response time, IIE Transactions, 29(4), pp. 313-322. 1997.

Eldemir, F., R.J. Graves and C.J. Malmborg. A comparison of alternative conceptualizing tools for automated storage and retrieval systems, International Journal of Production Research, 41(18), pp. 4517-4539. 2003.

Elsayed, E.A. and O.I. Unal. Order batching algorithms and travel-time estimation for automated storage/retrieval systems, International Journal of Production Research, 27(7), pp. 1097-1114. 1989.

Eynan, A. and M.J. Rosenblatt. Establishing zones in single-command class-based rectangular AS/RS, IIE Transactions, 26(1), pp. 38-46. 1994.

Fota, N., M. Kaaniche and K. Kanoun. Dependability evaluation of an air traffic control computing system, Performance Evaluation, 35(3-4), pp. 253-273. 1999.

Goetschalckx, M. and H.D. Ratliff. Shared storage policies based on the duration stay of unit loads, Management Science, 36(9), pp. 1120-1132. 1990.

Graves, S.C., W.H. Hausman and L.B. Schwarz. Storage-retrieval interleaving in automatic warehousing systems, Management Science, 23(9), pp. 935-945. 1977.

Hatono, I., N. Katoh, Y. Keiichi and H. Tamura. Modeling of FMS under Uncertainty Using Stochastic Petri Nets. In Proc. of the Third International Workshop On Petri Nets and Performance Models, 1989, Kyoto, Japan, pp. 122-129.

Hausman, W.H., L.B. Schwarz and S.C. Graves. Optimal storage assignment in automatic warehousing systems, Management Science, 22(6), pp. 629-638. 1976.

Hwang, H. and S.B. Lee. Travel-time models considering the operating characteristics

of the storage and retrieval machine, International Journal of Production Research, 28(10), pp. 1779-1789. 1990.

Jensen, K. Coloured PN: Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 2nd corrected printing. 1997.

Jensen, K. Coloured PN: Basic Concepts, Analysis Methods and Practical Use. Vol. 2, Analysis Methods. Monographs in Theoretical Computer Science, Springer-Verlag, 2nd corrected printing. 1997.

Jensen, K. Coloured PN: Basic Concepts, Analysis Methods and Practical Use. Vol. 3, Practical Use. Monographs in Theoretical Computer Science, Springer-Verlag. 1997.

Kanoun, K., M. Borrel, T. Morteveille and A. Peytavin. Modeling the dependability of CAUTRA, a subset of the French Air Traffic Control system. In Proc. 26th Annual Int. Symp. on Fault-Tolerant Computing, June 1996, Sendai, Japan, pp. 106-115.

Kim, J. and A. Seidmann. A framework for the exact evaluation of expected cycle times in automated storage systems with full-turnover item allocation and random service requests, Computers and Industrial Engineering, 18(4), pp. 601-612. 1990.

Kochikar, V.P. and T.T. Narendran. Coloured Petri Net Modelling and Analysis of Flexible Manufacturing Systems, Advances in Mechanical Engineering. In Proc. of the 7th ISME Conference, 1990, New Delhi, India, pp. 336-345.

Kulturel, S., N.E. Ozemirel, C. Sepil and Z. Bozkurt. Experimental investigation of shared storage assignment policies in automated storage/retrieval systems, IIE Transactions, 31(8), pp. 739-749. 1999.

Lee, H.F. Performance analysis for automated storage and retrieval systems, IIE Transactions, 29(1, pp. 15-28. 1997.

Lee, H.F. and S.K. Schaefer. Sequencing methods for automated storage and retrieval systems with dedicated storage, Computers and Industrial Engineering, 32(2), pp. 351-362. 1997.

Lin, S.C. and H.P.B. Wang. Modeling an automated storage and retrieval system using Petri nets, International Journal of Production Research, 33(1), pp. 237-260. 1995.

Mahajan, S., B.V. Rao and B.A. Peters. A retrieval sequencing heuristic for miniload end-of-aisle automated storage/retrieval systems, International Journal of Production Research, 36(6), pp. 1715-1731. 1998.

Menon, S.R., P.M. Ferreira and S.G. Kapoor. A Colored Petri Net System for Simulation and Control of FMSs, Advances in Manufacturing Systems Engineering. In Proc. of the Winter Annual Meeting of the ASME, 1989, San Francisco, USA, pp. 1-16.

Moore, K.E. and S.M. Gupta. Stochastic colored Petri net (SCPN) models of traditional and flexible kanban systems, International Journal of Production Research, 37(9), pp. 2135-2158. 1999.

Pan, C.H. and C.H. Wang. A framework for the dual command cycle travel time model in automated warehousing system, International Journal of Production Research, 34(8), pp. 2099-2117. 1996.

Randhawa, S.U., E.D. Mcdowell and W.T. Wang. Evaluation of schduling rules for single- and dual- dock automated storage/retrieval system, Computers and Industrial Engineering, 20(4), pp. 401-410. 1991.

Randhawa, S.U. and R. Shroff. Simulation-based design evaluation of unit load automated storage/retrieval systems, Computers and Industrial Engineering, 28(1), pp. 71-79. 1995.

Schwarz, L.B., S.C. Graves and W.H. Hausman. Scheduling policies for automatic warehousing systems: simulation results, AIIE Transactions, 10(3), pp. 260-270. 1978.

Shiizuka, H. and M. Suzuki. Modeling of AGV networks in flexible manufacturing systems, Computers and Industrial Engineering, 27(1-4), pp. 81-86. 1994.

Srihari, K., J.A. Cecil and C.R. Emerson. An Application of Petri Nets in FMS Modeling, Analysis, and Control. In Proc. of Manufacturing International'90 Conference, 1990, Atlanta, USA, pp. 107-111.

Thonemann U.W. and M.L. Brandeau. Optimal storage assignment policies for automated storage and retrieval systems with stochastic demands, Management Science, 44(1), pp. 142-148. 1998.

Tompkins, J.A. and J.A. White. Facilities Planning. John Wiley, New York. 1984.

Van den Berg, J.P. Analytic expressions for the optimal dwell point in an automated storage/retrieval system, International Journal of Production Economics, 76(1), pp. 13-25. 2002.

Van den Berg, J.P. and A.J.R.M. Gademann. Simulation study of an automated storage/retrieval system, International Journal of Production Research, 38(6), pp. 1339-1356. 2000.

Wang, L.C. and S.Y. Wu. Modeling with colored timed object-oriented Petri nets for automated manufacturing systems, Computers and Industrial Engineering, 34(2), pp.

463-480. 1998.

Wen, U.P., D.T. Chang and S.P. Chen. The impact of acceleration/deceleration on travel-time models in class-based automated S/R systems, IIE Transactions, 33(1), pp. 599-608. 2001.

Ziegler, P. and H. Szczerbicka. Performance and performability evaluation of FMS using GSPN. In Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'98), October 1998, San Diego, USA, pp. 511-516.

Zimmermann, A. and J. Freiheit. TimeNETMS - An Integrated Modeling and Performance Evaluation Tool for Manufacturing Systems. In Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'98), October 1998, San Diego, USA, pp. 535-540.

Zuberek, W.M. Hierarchical analysis of manufacturing systems using Petri nets. In Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'2000), October 2000, Nashville, TN, pp. 3021-3026.

Appendices

A. Cargo retrieval simulation program code

The retrieval simulation program consists of the following code files:

SimRetrieval.cpp: main program, reads data from the input file, controls the simulation process and generates the output file.

Phase.h: defines the structure of steps in the cargo retrieval process.

Procedure.h: defines the structure of retrieval processes.

ImportProcess.h: defines the top structure of cargo retrieval PN model, which comprises retrieval processes.

PhaseResource.h: defines the structure of resources needed for firing transitions or released when transitions fired.

PhaseResourceSet.h: defines the structure of resource list needed for firing transitions or released when transitions fired.

Resource.h: defines the structure of resources.

ResourcePool.h: defines the structure of resource list; the function for calculating resource utilization is included.

RetrievalOrder.h: defines the structure of retrieval requests.

RetrievalList.h: defines the structure of retrieval list.

Event.h: defines the structure of events other than AS/RS and MHS operations.

EventList.h: defines the structure of future event list.

ASRSEvent.h: defines the structure of AS/RS operation events.

ASRSEventSet.h: implements the functions of AS/RS; both the operating mechanism and parameters are defined.

MHSEvent.h: defines the structure of MHS operation events.

MHSEventSet.h: implements the functions of MHS; both the operating mechanism and parameters are defined.

Location.h: defines the structure of locations where forklifts may reside.

Layout.h: has the location list and defines the forklift travel time model.

Stdafx.h: provides include files for the program.

SimRetrieval.cpp

```
// SimRetrieval.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "RetrievalOrder.h"
#include "RetrievalList.h"
#include "Event.h"
#include "EventList.h"

int _tmain(int argc, _TCHAR* argv[])
{
	//Input and Output Files
	ifstream inRetrievalList ("inRetrievalList.xls");
	ofstream outMeasure ("outMeasure.xls");
	ofstream outResponse ("outResponse.xls");
	ofstream out ("result.txt");

	//Retrieval Process
	PhaseResourceSet asrs_a_1 ("OA", true);
	asrs_a_1.addPhaseResource("ASRSQueue");
	PhaseResourceSet asrs_a_2 ("OA", false);
	Procedure asrs_a ("asrs_a", "request", asrs_a_1, '0');
	asrs_a.addPhase("key in", asrs_a_2, 'c', 10);
	ImportProcess importProcess (asrs_a);
```

```
PhaseResourceSet asrs_b_1 ("forklift", true);
PhaseResourceSet asrs_b_4 ("forklift", false);
Procedure asrs_b ("asrs_b", "--> ASRS", asrs_b_1, "ASRS");
asrs_b.addPhase("load", 'c', 10);
asrs_b.addPhase("--> truck dock", "truck dock");
asrs_b.addPhase("unload", asrs_b_4, 'c', 10);
importProcess.addProcedure(asrs_b);

PhaseResourceSet asrs_c_1 ("agent", true);
PhaseResourceSet asrs_c_2 ("agent", false);
Procedure asrs_c ("asrs_c", "take cargo", asrs_c_1, '0');
asrs_c.addPhase("put cargo aside", asrs_c_2, 'q', 2.26);
importProcess.addProcedure(asrs_c);

PhaseResourceSet bin_1 ("forklift", true);
PhaseResourceSet bin_4 ("forklift", false);
PhaseResourceSet bin_5 ("OA", true);
bin_5.addPhaseResource("ASRSQueue");
PhaseResourceSet bin_6 ("OA", false);
Procedure bin ("bin", "--> truck dock", bin_1, "truck dock");
bin.addPhase("load", 'c', 10);
bin.addPhase("--> ASRS", "ASRS");
bin.addPhase("unload", bin_4, 'c', 10);
bin.addPhase("return", bin_5, '0');
bin.addPhase("key in", bin_6, 'c', 10);
importProcess.addProcedure(bin);

PhaseResourceSet mhs_a_1 ("OA", true);
PhaseResourceSet mhs_a_2 ("OA", false);
Procedure mhs_a ("mhs_a", "request", mhs_a_1, '0');
mhs_a.addPhase("submitted", mhs_a_2, 'c', 20);
importProcess.addProcedure(mhs_a);

PhaseResourceSet mhs_b_1 ("WSroller", true);
mhs_b_1.addPhaseResource("deck");
PhaseResourceSet mhs_b_2 ("WSroller", false);
PhaseResourceSet mhs_b_3 ("forklift", true);
PhaseResourceSet mhs_b_4 ("deck", false);
PhaseResourceSet mhs_b_6 ("forklift", false);
Procedure mhs_b ("mhs_b", "take", mhs_b_1, '0');
mhs_b.addPhase("transfer", mhs_b_2, 'w');
mhs_b.addPhase("--> deck", mhs_b_3, "deck");
mhs_b.addPhase("load", mhs_b_4, 'c', 10);
mhs_b.addPhase("--> truck dock", "truck dock");
mhs_b.addPhase("unload", mhs_b_6, 'c', 10);
```

```
importProcess.addProcedure(mhs_b);

PhaseResourceSet mhs_c_1 ("agent", true);
PhaseResourceSet mhs_c_2 ("agent", false);
Procedure mhs_c ("mhs_c", "take cargo", mhs_c_1, '0');
mhs_c.addPhase("put cargo aside", mhs_c_2, 'q', 2.26);
importProcess.addProcedure(mhs_c);

PhaseResourceSet container_1 ("forklift", true);
PhaseResourceSet container_3 ("deck", true);
PhaseResourceSet container_4 ("forklift", false);
PhaseResourceSet container_5 ("WSroller", true);
PhaseResourceSet container_6 ("WSroller", false);
container_6.addPhaseResource("deck");
Procedure container ("container", "--> truck dock", container_1, "truck dock");
container.addPhase("load", 'c', 10);
container.addPhase("--> deck", container_3, "deck");
container.addPhase("unload", container_4, 'c', 10);
container.addPhase("take container", container_5, '0');
container.addPhase("transfer", container_6, 'w');
importProcess.addProcedure(container);

PhaseResourceSet fg_1 ("forklift", true);
PhaseResourceSet fg_5 ("forklift", false);
Procedure fg ("fg", "--> FG area", fg_1, "FG area");
fg.addPhase("search", 'u', 0, 135);
fg.addPhase("load", 'c', 10);
fg.addPhase("--> truck dock", "truck dock");
fg.addPhase("unload", fg_5, 'c', 10);
importProcess.addProcedure(fg);

PhaseResourceSet bs_1 ("agent", true);
PhaseResourceSet bs_2 ("agent", false);
Procedure bs ("bs", "collect cargo", bs_1, '0');
bs.addPhase("build skid", bs_2, 'b');
importProcess.addProcedure(bs);

//Layout
Layout layout;
layout.initialize();

//Resource Pool Initialization
ResourcePool resourcePool ("OA");
resourcePool.addResource("forklift", "forklift_1");
resourcePool.addResource("forklift", "forklift_2");
```

```
resourcePool.addResource("forklift", "forklift_3");
resourcePool.addResource("WSroller", "roller_A");
resourcePool.addResource("WSroller", "roller_B");
resourcePool.addResource("WSroller", "roller_C");
resourcePool.initializeResources();
resourcePool.initializeResourceLocations();

//Varibles Initialization
boost::timer totalTime;
double startTime = 9999999;
double endTime = 0;
int timer = 3600;
double piecesY1, piecesY2, piecesM;
double mainStartTime;
double asrsStartTime;
double mhsStartTime;
double duration;
string destination;
string load;
int deckIndex;
list <Event>::iterator eventIter;
PhaseResourceSet phaseResourceSet;
list <PhaseResource>::iterator phaseResourceIter;
list <Resource>::iterator resourceIter;
list <Resource>::iterator deckIter;
list <Location>::iterator originIter;
list <Location>::iterator targetIter;
list <ASRSEvent>::iterator asrsEventIter;
list <MHSEvent>::iterator mhsEventIter;
list <RetrievalOrder>::iterator retrievalOrderIter;
boost::hellekalek1995 generatorAP1(17);
boost::hellekalek1995 generatorAP2(18);
boost::hellekalek1995 generatorFP1(19);
boost::hellekalek1995 generatorFP2(20);
boost::hellekalek1995 generatorMP1(21);
boost::hellekalek1995 generatorMP2(22);
boost::hellekalek1995 generatorBL(23);
boost::hellekalek1995 generatorBLB(23);
boost::hellekalek1995 generatorBR(24);
boost::hellekalek1995 generatorBRB(24);
boost::hellekalek1995 generatorMR(25);
boost::hellekalek1995 generatorMRB(26);
boost::hellekalek1995 generatorUBT(27);
boost::hellekalek1995 generatorST(28);
boost::gamma_distribution<boost::hellekalek1995> genASRSpieces1(generatorAP1, 0.504);
```

```
boost::gamma_distribution<boost::hellekalek1995> genASRSpieces2(generatorAP2, 1.28);
boost::gamma_distribution<boost::hellekalek1995> genFGpieces1(generatorFP1, 0.448);
boost::gamma_distribution<boost::hellekalek1995> genFGpieces2(generatorFP2, 1.66);
boost::gamma_distribution<boost::hellekalek1995> genMHSpieces1(generatorMP1, 0.253);
boost::gamma_distribution<boost::hellekalek1995> genMHSpieces2(generatorMP2, 0.992);
boost::uniform_smallint<boost::hellekalek1995> bin_level(generatorBL, 1, 7);
boost::uniform_smallint<boost::hellekalek1995> bin_level_back(generatorBLB, 1, 7);
boost::uniform_smallint<boost::hellekalek1995> bin_row(generatorBR, 1, 14);
boost::uniform_smallint<boost::hellekalek1995> bin_row_back(generatorBRB, 1, 14);
boost::uniform_smallint<boost::hellekalek1995> mhs_row(generatorMR, 1, 15);
boost::uniform_smallint<boost::hellekalek1995> mhs_row_back(generatorMRB, 1, 15);
boost::triangle_distribution<boost::hellekalek1995> genUnitBuildingTime(generatorUBT, 2.26, 2.56, 6.55);
boost::uniform_smallint<boost::hellekalek1995> genSearchingTime(generatorST, 0, 135);

//Read Retrieval List
string storageLocation;
string target;
int requestTime;
string requestTimeString;
RetrievalList retrievalList;
while(inRetrievalList >> storageLocation)
{
    inRetrievalList >> target;
    inRetrievalList >> requestTimeString;
    requestTime = lexical_cast<int>(requestTimeString) * 60;
    if(storageLocation == "ASRS")
    {
        piecesY1 = genASRSpieces1();
        piecesY2 = genASRSpieces2();
        piecesM = 10;
    }
    else if(storageLocation.substr(0,2) == "FG")
    {
        piecesY1 = genFGpieces1();
        piecesY2 = genFGpieces2();
        piecesM = 9;
    }
    else
    {
        piecesY1 = genMHSpieces1();
        piecesY2 = genMHSpieces2();
        piecesM = 42;
    }
    retrievalList.addRetrievalOrder(storageLocation, target, requestTime, piecesY1 / (piecesY1 + piecesY2)
* piecesM + 1);
```

```
	}

	//Events Initialization
	EventList eventList;
	eventList.initialize(retrievalList, importProcess);
	ASRSEventSet asrsEventSet;
	asrsEventSet.initialize();
	MHSEventSet mhsEventSet;
	mhsEventSet.initialize();

	//Main Controller
	while(eventList._eventList.size() != 0 || asrsEventSet._asrsEventSet.size() != 0 ||
mhsEventSet._mhsEventSet.size() != 0)
	{
//		out << eventList.display() << endl;
		eventIter = eventList._eventList.begin();
		eventList._upComingEventIter = eventIter;
		//Identify the next event
		while(eventIter->_availableTime <= eventList._upComingEventIter->_startTime && eventIter !=
eventList._eventList.end())
		{
			eventIter->_startTime = eventIter->_availableTime;
			phaseResourceSet = eventIter->_phaseSet.front()._phaseResourceSet;
			if(phaseResourceSet._sign == true)
			{
				phaseResourceIter = phaseResourceSet._phaseResourceSet.begin();
				while(phaseResourceIter != phaseResourceSet._phaseResourceSet.end())
				{
					if(phaseResourceIter->_type == "agent")
						resourceIter = resourcePool.searchResourceByName("A" +
retrievalList.searchRetrievalOrderById(eventIter->_requestOrderId)->_target);
					else if(phaseResourceIter->_type == "WSroller")
					{
						deckIter =
resourcePool.searchResourceByRetrievalOrderId(eventIter->_requestOrderId, "deck");
						if(deckIter == resourcePool._resourcePool.end())
							deckIter = resourcePool.searchResourceByType("deck");
						resourceIter = resourcePool.searchResourceByName("roller_" +
deckIter->_name.substr(2,1));
					}
					else
						resourceIter = resourcePool.searchResourceByType(phaseResourceIter->_type);
					if(resourceIter->_availableTime > eventIter->_startTime)
						eventIter->_startTime = resourceIter->_availableTime;
					phaseResourceIter++;
```

```
                    }
                }
                if(eventIter->_startTime < eventList._upComingEventIter->_startTime)
                    eventList._upComingEventIter = eventIter;
                eventIter++;
            }
            if(eventList._eventList.size() != 0)
                mainStartTime = eventList._upComingEventIter->_startTime;
            else
                mainStartTime = 9999999;
            if(asrsEventSet._asrsEventSet.size() != 0)
                asrsStartTime = max(asrsEventSet._availableTime,
asrsEventSet._asrsEventSet.front()._startTime);
            else
                asrsStartTime = 9999999;
            if(mhsEventSet._mhsEventSet.size() != 0)
                mhsStartTime = max(mhsEventSet._availableTime,
mhsEventSet._mhsEventSet.front()._startTime);
            else
                mhsStartTime = 9999999;
            if(startTime > mainStartTime)
                startTime = mainStartTime;
            //Trigger a event other than AS/RS or MHS operations
            if(mainStartTime <= asrsStartTime && mainStartTime <= mhsStartTime)
            {
//                  out << mainStartTime << endl;
//                  out << eventList._upComingEventIter->_phaseSet.front()._name << endl; ////////////
/*                  if(mainStartTime > timer)
                    {
                        outResponse << mainStartTime << '\t' <<
resourcePool.calculateForkliftUtilization(mainStartTime - startTime) << endl;
                        timer = ((int)(mainStartTime / 3600) + 1) * 3600;
                        resourcePool.resetForklifts();
                        startTime = mainStartTime;
                    } */
                    if((int)(mainStartTime + 0.5) % 86400 > 72000 &&
resourcePool.searchResourceByName("forklift_3")->_availableTime == 99999999)
                        resourcePool.activateForklifts();
                    destination = eventList._upComingEventIter->_phaseSet.front()._destination;
                    if(destination == "truck dock")
                        destination = "TD" +
retrievalList.searchRetrievalOrderById(eventList._upComingEventIter->_requestOrderId)->_target;
                    else if(destination == "deck")
                    {
                        deckIter =
```

```
resourcePool.searchResourceByRetrievalOrderId(eventList._upComingEventIter->_requestOrderId, "deck");
                        if(deckIter == resourcePool._resourcePool.end())
                                deckIter = resourcePool.searchResourceByType("deck");
                        destination = deckIter->_name;
                }
                duration = 0;
                //Calculate the transition duration
                if(eventList._upComingEventIter->_phaseSet.front()._durationType == 't')
                {
                        resourceIter =
resourcePool.searchResourceByRetrievalOrderId(eventList._upComingEventIter->_requestOrderId, "forklift");
                        if(resourceIter == resourcePool._resourcePool.end())
                                resourceIter = resourcePool.searchResourceByType("forklift");
                        originIter = layout.searchLocationByName(resourceIter->_location);
                        targetIter = layout.searchLocationByName(destination);
//                      out << "travel from " + originIter->_name + " to " + targetIter->_name << endl;
                        if(originIter->_name != targetIter->_name)
                        {
                                load = "empty";
                                if((eventList._upComingEventIter->_type == "asrs_b" &&
eventList._upComingEventIter->_phaseSet.front()._destination == "truck dock") ||
(eventList._upComingEventIter->_type == "bin" &&
eventList._upComingEventIter->_phaseSet.front()._destination == "ASRS") ||
(eventList._upComingEventIter->_type == "asrs_pass_b"))
                                        load = "bin";
                                else if((eventList._upComingEventIter->_type == "mhs_b" &&
eventList._upComingEventIter->_phaseSet.front()._destination == "truck dock") ||
(eventList._upComingEventIter->_type == "container" &&
eventList._upComingEventIter->_phaseSet.front()._destination == "deck"))
                                        load = "MHS";
                                else if(eventList._upComingEventIter->_type == "fg" &&
eventList._upComingEventIter->_phaseSet.front()._destination == "truck dock")
                                        load = "FG";
                                duration = layout.calculateTravelTime(originIter, targetIter, load);
                                resourceIter->_location = destination;
                        }
                }
                else if(eventList._upComingEventIter->_phaseSet.front()._durationType == 'q')
                        duration =
retrievalList.searchRetrievalOrderById(eventList._upComingEventIter->_requestOrderId)->_pieces *
eventList._upComingEventIter->_phaseSet.front()._parameter_1;
                else if(eventList._upComingEventIter->_phaseSet.front()._durationType == 'w')
                {
                        deckIter =
resourcePool.searchResourceByRetrievalOrderId(eventList._upComingEventIter->_requestOrderId, "deck");
```

```
                    deckIndex = lexical_cast<int>(deckIter->_name.at(4));
                    duration = ((10 - deckIndex) / 2 * 2 + 1) * 12;
                }
                else if(eventList._upComingEventIter->_phaseSet.front()._durationType == 'c')
                    duration = eventList._upComingEventIter->_phaseSet.front()._parameter_1;
                else if(eventList._upComingEventIter->_phaseSet.front()._durationType == 'u')
                    duration = genSearchingTime();
                else if(eventList._upComingEventIter->_phaseSet.front()._durationType == 'b')
                    duration = genUnitBuildingTime() *
retrievalList.searchRetrievalOrderById(eventList._upComingEventIter->_requestOrderId)->_pieces;
//                  out << "duration:" << duration << endl; //////////
                    //Update the resource pool

        if(eventList._upComingEventIter->_phaseSet.front()._phaseResourceSet._phaseResourceSet.size() != 0)
                    {
                        phaseResourceIter =
eventList._upComingEventIter->_phaseSet.front()._phaseResourceSet._phaseResourceSet.begin();
                        while(phaseResourceIter !=
eventList._upComingEventIter->_phaseSet.front()._phaseResourceSet._phaseResourceSet.end())
                        {
                            if(phaseResourceIter->_type == "agent" &&
eventList._upComingEventIter->_phaseSet.front()._phaseResourceSet._sign)
                                resourcePool.searchResourceByName("A" +
retrievalList.searchRetrievalOrderById(eventList._upComingEventIter->_requestOrderId)->_target)->update(event
List._upComingEventIter->_requestOrderId, eventList._upComingEventIter->_startTime);
                            else if(phaseResourceIter->_type == "WSroller" &&
eventList._upComingEventIter->_phaseSet.front()._phaseResourceSet._sign)
                            {
                                deckIter =
resourcePool.searchResourceByRetrievalOrderId(eventList._upComingEventIter->_requestOrderId, "deck");
                                if(deckIter == resourcePool._resourcePool.end())
                                    deckIter = resourcePool.searchResourceByType("deck");
                                resourcePool.searchResourceByName("roller_" +
deckIter->_name.substr(2,1))->update(eventList._upComingEventIter->_requestOrderId,
eventList._upComingEventIter->_startTime);
                            }
                            else if(phaseResourceIter->_type == "forklift"
&& !eventList._upComingEventIter->_phaseSet.front()._phaseResourceSet._sign)
                            {
                                resourceIter =
resourcePool.searchResourceByRetrievalOrderId(eventList._upComingEventIter->_requestOrderId,
phaseResourceIter->_type);
                                resourceIter->update(eventList._upComingEventIter->_startTime + duration);
                                if(resourceIter->_name != "forklift_1" && resourceIter->_name != "forklift_2"
&& (int)(eventList._upComingEventIter->_startTime + duration + 0.5) % 86400 > 18000 &&
```

```
(int)(eventList._upComingEventIter->_startTime + duration + 0.5) % 86400 < 72000)
                                        resourceIter->_availableTime = 99999999;
                                }
                                else

        resourcePool.updateResource(eventList._upComingEventIter->_phaseSet.front()._phaseResourceSet._sign,
phaseResourceIter->_type, eventList._upComingEventIter->_requestOrderId,
eventList._upComingEventIter->_startTime, duration);
                                        phaseResourceIter++;
                                }
//                              out << resourcePool.display() << endl;
                        }
                        //Generate the next event
                        eventList._upComingEventIter->_phaseSet.pop_front();
                        if(eventList._upComingEventIter->_phaseSet.size() != 0)
                        {
                                Event event (eventList._upComingEventIter->_type,
eventList._upComingEventIter->_startTime + duration, eventList._upComingEventIter->_requestOrderId);
                                event._phaseSet = eventList._upComingEventIter->_phaseSet;
                                eventList.addEvent(event);
                        }
                        else
                        {
                                if(eventList._upComingEventIter->_type == "asrs_a")
                                {
                                        asrsEventSet.addASRSEvent('r', "B" + lexical_cast<string>(bin_level() * 100 +
bin_row()), eventList._upComingEventIter->_requestOrderId, eventList._upComingEventIter->_startTime +
duration);
//                                      out << asrsEventSet.display() << endl;
                                }
                                else if(eventList._upComingEventIter->_type == "mhs_a")
                                {
                                        mhsEventSet.addMHSEvent('r', "mhs" + lexical_cast<string>(mhs_row() + 200),
eventList._upComingEventIter->_requestOrderId, eventList._upComingEventIter->_startTime + duration);
//                                      out << mhsEventSet.display() << endl;
                                }
                                else if(eventList._upComingEventIter->_type == "asrs_b")
                                        eventList.addEvent("asrs_c", eventList._upComingEventIter->_startTime + duration,
importProcess, eventList._upComingEventIter->_requestOrderId);
                                else if(eventList._upComingEventIter->_type == "mhs_b")
                                        eventList.addEvent("mhs_c", eventList._upComingEventIter->_startTime + duration,
importProcess, eventList._upComingEventIter->_requestOrderId);
                                else if(eventList._upComingEventIter->_type == "asrs_c")
                                {
                                        eventList.addEvent("bs", eventList._upComingEventIter->_startTime + duration,
```

```
importProcess, eventList._upComingEventIter->_requestOrderId);
                    eventList.addEvent("bin", eventList._upComingEventIter->_startTime + duration,
importProcess, eventList._upComingEventIter->_requestOrderId);
                }
                else if(eventList._upComingEventIter->_type == "mhs_c")
                {
                    eventList.addEvent("bs", eventList._upComingEventIter->_startTime + duration,
importProcess, eventList._upComingEventIter->_requestOrderId);
                    eventList.addEvent("container", eventList._upComingEventIter->_startTime +
duration, importProcess, eventList._upComingEventIter->_requestOrderId);
                }
                else if(eventList._upComingEventIter->_type == "bin")
                {
                    asrsEventSet.addASRSEvent('s', "B" + lexical_cast<string>(bin_level_back() * 100 +
bin_row_back()), eventList._upComingEventIter->_requestOrderId, eventList._upComingEventIter->_startTime +
duration);
//                  out << asrsEventSet.display() << endl;
                }
                else if(eventList._upComingEventIter->_type == "container")
                {
                    mhsEventSet.addMHSEvent('s', "mhs" + lexical_cast<string>(mhs_row_back() + 300),
eventList._upComingEventIter->_requestOrderId, eventList._upComingEventIter->_startTime + duration);
//                  out << mhsEventSet.display() << endl;
                }
                else if(eventList._upComingEventIter->_type == "fg")
                    eventList.addEvent("bs", eventList._upComingEventIter->_startTime + duration,
importProcess, eventList._upComingEventIter->_requestOrderId);
                else if(eventList._upComingEventIter->_type == "bs")
                {
                    retrievalOrderIter =
retrievalList.searchRetrievalOrderById(eventList._upComingEventIter->_requestOrderId);
                    cout << "Order " << retrievalOrderIter->_id << " fulfilled." << '\t' <<
totalTime.elapsed() << endl;
                    outMeasure << retrievalOrderIter->_id << '\t' << retrievalOrderIter->_storageLocation
<< '\t' << eventList._upComingEventIter->_startTime + duration - retrievalOrderIter->_requestTime << endl;
                    if(eventList._upComingEventIter->_startTime + duration > endTime)
                        endTime = eventList._upComingEventIter->_startTime + duration;
                }
            }
            eventList._eventList.erase(eventList._upComingEventIter);
        }
        //Trigger an AS/RS operation
        else if(asrsStartTime <= mhsStartTime)
        {
            asrsEventIter = asrsEventSet.searchASRSEvent();
```

```
//                    out << asrsStartTime << " " << asrsEventIter->_type << "_" + asrsEventIter->_binNo << endl;
////////////
                    duration = asrsEventSet.calculateDuration(asrsEventIter);
//                    out << "duration:" << duration << endl;
                    if(asrsEventIter->_type == 'r')
                    {
                        eventList.addEvent("asrs_b", asrsStartTime + duration, importProcess,
asrsEventIter->_retrievalOrderId);
//                        out << eventList.display() << endl;
                    }
                    resourcePool.searchResourceByRetrievalOrderId(asrsEventIter->_retrievalOrderId,
"ASRSQueue")->update(asrsStartTime + duration);
                    asrsEventSet.update(asrsEventIter, duration);
//                    out << asrsEventSet.display() << endl;
                    if(asrsStartTime + duration > endTime)
                        endTime = asrsStartTime + duration;
                }
                //Trigger a MHS operation
                else
                {
                    mhsEventIter = mhsEventSet.searchMHSEvent();
//                    out << mhsStartTime << " " << mhsEventIter->_type << "_" + mhsEventIter->_mhsNo << endl;
//////////////
                    duration = mhsEventSet.calculateDuration(mhsEventIter);
//                    out << "duration:" << duration << endl;
                    if(mhsEventIter->_type == 'r')
                    {
                        eventList.addEvent("mhs_b", mhsStartTime + duration, importProcess,
mhsEventIter->_retrievalOrderId);
//                        out << eventList.display() << endl;
                    }
                    mhsEventSet.update(mhsEventIter, duration);
//                    out << mhsEventSet.display() << endl;
                    if(mhsStartTime + duration > endTime)
                        endTime = mhsStartTime + duration;
                }
        }
//      outMeasure << resourcePool.calculateIndividualUtilization(endTime - startTime) << endl;
        outMeasure << endl << "forklift" << '\t' << resourcePool.calculateForkliftUtilization(endTime - startTime) * 3
/ 2.375 << endl;
//      outMeasure << totalTime.elapsed() << endl;
//      outMeasure << resourcePool.calculateForkliftUtilization(endTime - startTime) << endl;
        cout << totalTime.elapsed() << endl;
        return 0;
}
```

ASRSEvent.h

```
#pragma once

class ASRSEvent
{
public:

	ASRSEvent(void)
	{
	}

	~ASRSEvent(void)
	{
	}

	ASRSEvent(char ty, string bn, int roi, double st)
	{
		_type = ty;
		_binNo = bn;
		_retrievalOrderId = roi;
		_startTime = st;
	}

	char _type;
	string _binNo;
	int _retrievalOrderId;
	double _startTime;

};
```

ASRSEventSet.h

```
#pragma once

class ASRSEventSet
{
public:

	ASRSEventSet(void)
	{
	}
```

```
~ASRSEventSet(void)
{
}

void initialize()
{
	_machineLocation = "B104"; //Virtual "B104" means "I/O Point"
	_availableTime = 0;
	_busyPeriod = 0;
	_travelingSpeed = 2; //meters/second
	_hoistingSpeed = 0.33;
	_forkingSpeed = 0.6;
	_length = 2.5; //meters
	_width = 1.3;
	_height = 1.3;
	_unitTravelingTime = _width / _travelingSpeed; //0.65s
	_unitHoistingTime = _height / _hoistingSpeed; //3.9s
	_forkingTime = 4 * _length / _forkingSpeed; //16.7s
	_srTime = 20;
}

void addASRSEvent(char ty, string bn, int roi, double st)
{
	ASRSEvent ae (ty, bn, roi, st);
	_iter = _asrsEventSet.begin();
	bool sign = true;
	while(sign && _iter != _asrsEventSet.end())
	{
		if(st <= _iter->_startTime)
		{
			_asrsEventSet.insert(_iter, ae);
			sign = false;
		}
		_iter++;
	}
	if(sign)
		_asrsEventSet.push_back(ae);
}

list <ASRSEvent>::iterator searchASRSEventByType(char ty)
{
	_iter = _asrsEventSet.begin();
	while(_iter != _asrsEventSet.end() && _iter->_type != ty)
		_iter++;
	return _iter;
```

```
}

list <ASRSEvent>::iterator searchASRSEvent()
{
	if(_availableTime <= _asrsEventSet.front()._startTime)
		_iter = _asrsEventSet.begin();
	else if(_machineLocation == "B104")
	{
		_iter = searchASRSEventByType('s');
		if (_iter == _asrsEventSet.end() || _iter->_startTime > _availableTime)
			_iter = searchASRSEventByType('r');
	}
	else
	{
		_iter = searchASRSEventByType('r');
		if (_iter == _asrsEventSet.end() || _iter->_startTime > _availableTime)
			_iter = searchASRSEventByType('s');
	}
	return _iter;
}

double calculateDuration(list <ASRSEvent>::iterator it_ae)
{
	double d;
	double travelingTime;
	double hoistingTime;
	int machineLevel = lexical_cast<int>(_machineLocation.substr(1,1));
	int machineRow = lexical_cast<int>(_machineLocation.substr(2,2));
	int level = lexical_cast<int>(it_ae->_binNo.substr(1,1));
	int row = lexical_cast<int>(it_ae->_binNo.substr(2,2));
	if(it_ae->_type == 'r')
	{
		travelingTime = abs(machineRow - row) * _unitTravelingTime;
		hoistingTime = abs(machineLevel - level) * _unitHoistingTime;
		d = max(travelingTime, hoistingTime);
		travelingTime = abs(4 - row) * _unitTravelingTime;
		hoistingTime = abs(1 - level) * _unitHoistingTime;
		d += max(travelingTime, hoistingTime);
	}
	else
	{
		travelingTime = abs(machineRow - 4) * _unitTravelingTime;
		hoistingTime = abs(machineLevel - 1) * _unitHoistingTime;
		d = max(travelingTime, hoistingTime);
		travelingTime = abs(4 - row) * _unitTravelingTime;
```

```
                hoistingTime = abs(1 - level) * _unitHoistingTime;
                d += max(travelingTime, hoistingTime);
            }
            d += _forkingTime;
            return d;
        }

        void update(list <ASRSEvent>::iterator it_ae, double d)
        {
            _availableTime = max(_availableTime, it_ae->_startTime) + d;
            _busyPeriod += d;
            if (it_ae->_type == 's')
                _machineLocation = it_ae->_binNo;
            else
                _machineLocation = "B104";
            _asrsEventSet.erase(it_ae);
        }

        string display()
        {
            string s = "ASRS Events:\n";
            _iter = _asrsEventSet.begin();
            while (_iter != _asrsEventSet.end())
            {
                s += lexical_cast<string>(_iter->_type) + '\t' + _iter->_binNo + '\t' +
lexical_cast<string>(_iter->_startTime) + '\n';
                _iter++;
            }
            return s;
        }

        list <ASRSEvent> _asrsEventSet;
        list <ASRSEvent>::iterator _iter;
        string _machineLocation;
        double _availableTime;
        double _busyPeriod;
        double _travelingSpeed; //meters/second
        double _hoistingSpeed;
        double _forkingSpeed;
        double _length; //meters
        double _width;
        double _height;
        double _unitTravelingTime;
        double _unitHoistingTime;
        double _forkingTime;
```

```
        double _srTime;

};
```

Event.h

```
#pragma once

class Event
{
public:

        Event(void)
        {
        }

        ~Event(void)
        {
        }

        Event(string ty, double at, int roi = 0)
        {
                _id = _nextId++;
                _type = ty;
                _availableTime = at;
                _startTime = at;
                _requestOrderId = roi;
        }

        static int _nextId;
        int _id;
        string _type;
        list <Phase> _phaseSet;
        double _availableTime;
        double _startTime;
        int _requestOrderId;

};

int Event::_nextId = 1;
```

EventList.h

```
#pragma once

class EventList
{
public:

	EventList(void)
	{
	}

	~EventList(void)
	{
	}

	void addEvent(Event e)
	{
		_iter = _eventList.begin();
		while(e._availableTime > _iter->_availableTime && _iter != _eventList.end())
			_iter++;
		_eventList.insert(_iter, e);
	}

	void addEvent(Event e, ImportProcess ip)
	{
		list <Procedure>::iterator pIter;
		pIter = ip._importProcess.begin();
		while(pIter->_type != e._type)
			pIter++;
		e._phaseSet = pIter->_procedure;
		_iter = _eventList.begin();
		while(e._availableTime > _iter->_availableTime && _iter != _eventList.end())
			_iter++;
		_eventList.insert(_iter, e);
	}

	void addEvent(string ty, double ti, ImportProcess ip, int roi = 0)
	{
		Event e (ty, ti, roi);
		addEvent(e, ip);
	}

	void initialize(RetrievalList rl, ImportProcess ip)
	{
		list <RetrievalOrder>::iterator roIter = rl._retrievalList.begin();
		string ty;
```

```
            while(roIter != rl._retrievalList.end())
            {
                ty = "asrs_a";
                if(roIter->_storageLocation == "MHS")
                    ty = "mhs_a";
                else if(roIter->_storageLocation.substr(0,2) == "FG")
                    ty = "fg";
                Event e (ty, roIter->_requestTime, roIter->_id);
                addEvent(e, ip);
                roIter++;
            }
        }

        string display()
        {
            string s = "Future Event List:\n";
            _iter = _eventList.begin();
            while(_iter != _eventList.end())
            {
                if(_iter->_phaseSet.begin() != _iter->_phaseSet.end())
                    s += lexical_cast<string>(_iter->_id) + '\t' + _iter->_phaseSet.front()._name + '\t' +
lexical_cast<string>(_iter->_availableTime) + '\n';
                _iter++;
            }
            return s;
        }

        list <Event> _eventList;
        list <Event>::iterator _iter;
        list <Event>::iterator _upComingEventIter;

};
```

ImportProcess.h

```
#pragma once

class ImportProcess
{
public:

        ImportProcess(void)
        {
        }
```

```
        ~ImportProcess(void)
        {
        }

        ImportProcess(Procedure p)
        {
                _importProcess.push_back(p);
        }

        void addProcedure(Procedure p)
        {
                _importProcess.push_back(p);
        }

        list <Procedure> _importProcess;

};
```

Layout.h

```
#pragma once

class Layout
{
public:

        Layout(void)
        {
        }

        ~Layout(void)
        {
        }

        void addLocation(string n, double x, double y, string z = "0")
        {
                Location l (n, x, y, z);
                _layout.push_back(l);
        }

        list <Location>::iterator searchLocationByName(string n)
        {
                _iter = _layout.begin();
```

```
        while(_iter->_name != n)
            _iter++;
        return _iter;
    }

    void initialize()
    {
        addLocation("FG area", 7, 3);
        addLocation("FG1", 2.5, 11.8); //!!!
        addLocation("FG2", 7, 11.4); //!!!
        addLocation("ASRS", 5.2, 4.2);
        for(int i = 1; i < 4; i++)
            addLocation("TD10"+lexical_cast<string>(i), 6.7+1.1*i, 2.4);
        addLocation("TD104", 11.6, 2);
        for(int i = 5; i < 10; i++)
            addLocation("TD10"+lexical_cast<string>(i), 7.7+1.1*i, 2.4);
        for(int i = 10; i < 15; i++)
            addLocation("TD1"+lexical_cast<string>(i), 7.7+1.1*i, 2.4);
        for(int i = 1; i < 9; i++)
        {
            int j = 1 - (i % 2);
            addLocation("dkA_"+lexical_cast<string>(i), 8.6+2.4*j, 2.9+(i+1)/2*1.3, lexical_cast<string>(j));
            addLocation("dkB_"+lexical_cast<string>(i), 14.8+2.4*j, 2.9+(i+1)/2*1.3,
lexical_cast<string>(j+1));
            addLocation("dkC_"+lexical_cast<string>(i), 21+2.4*j, 2.9+(i+1)/2*1.3,
lexical_cast<string>(j+2));
        }
    }

    double calculateTravelTime(list <Location>::iterator it_o, list <Location>::iterator it_t, string load = "empty")
    {
        double dist;
        int turns = 1;
        double d;
        if(it_o->_zone == it_t->_zone)
            dist = (abs(it_o->_x_coordinate - it_t->_x_coordinate) + abs(it_o->_y_coordinate -
it_t->_y_coordinate)) * 3.5;
        else if(it_o->_zone != "0")
            dist = (it_o->_y_coordinate - 4.2 + abs(it_o->_x_coordinate - it_t->_x_coordinate) + abs(4.2 -
it_t->_y_coordinate)) * 3.5;
        else
            dist = (it_t->_y_coordinate - 4.2 + abs(it_o->_x_coordinate - it_t->_x_coordinate) +
abs(it_o->_y_coordinate - 4.2)) * 3.5;
        if(load == "empty")
        {
```

```
                    if(it_o->_zone == "3" && (it_t->_zone == "1" || it_t->_zone == "2" || it_t->_name.substr(0,3) ==
"dkA" || it_t->_name == "Cage"))
                        turns = 2;
                    if(it_o->_zone == "2" && (it_t->_zone == "1" || it_t->_zone == "3" || it_t->_name.substr(0,3) ==
"dkA" || it_t->_name == "Cage"))
                        turns = 2;
                    if(it_o->_zone == "1" && (it_t->_zone == "2" || it_t->_zone == "3" || (it_t->_name.substr(0,3) ==
"dkA" && it_t->_zone == "0") || it_t->_name == "Cage"))
                        turns = 2;
                    d = dist/3.7 + 1.98*turns + 4.47 + 0.5;
                }
                else if(load == "FG")
                    d = dist/3.7 + 1.98 + 4.47 + 0.5;
                else if(load == "bin" || load == "MHS")
                    d = dist/2.22 + 1.71 + 4.91 + 0.5;
                return d;
        }

        list <Location> _layout;
        list <Location>::iterator _iter;

};
```

Location.h

```
#pragma once

class Location
{
public:

        Location(void)
        {
        }

        ~Location(void)
        {
        }

        Location(string n, double x, double y, string z = "0")
        {
                _name = n;
                _x_coordinate = x;
                _y_coordinate = y;
```

```
            _zone = z;
        }

        string _name;
        double _x_coordinate; //1 = 3.5 meters
        double _y_coordinate;
        string _zone;

};
```

MHSEvent.h

```
#pragma once

class MHSEvent
{
public:

        MHSEvent(void)
        {
        }

        ~MHSEvent(void)
        {
        }

        MHSEvent(char ty, string mn, int roi, double st)
        {
            _type = ty;
            _mhsNo = mn;
            _retrievalOrderId = roi;
            _startTime = st;
        }

        char _type;
        string _mhsNo;
        int _retrievalOrderId;
        double _startTime;

};
```

MHSEventSet.h

```
#pragma once

class MHSEventSet
{
public:

	MHSEventSet(void)
	{
	}

	~MHSEventSet(void)
	{
	}

	void initialize()
	{
		_etvLocation = "mhs107"; //Virtual I/O Point
		_availableTime = 0;
		_busyPeriod = 0;
	}

	void addMHSEvent(char ty, string mn, int roi, double st)
	{
		MHSEvent me (ty, mn, roi, st);
		_iter = _mhsEventSet.begin();
		bool sign = true;
		while(sign && _iter != _mhsEventSet.end())
		{
			if(st <= _iter->_startTime)
			{
				_mhsEventSet.insert(_iter, me);
				sign = false;
			}
			_iter++;
		}
		if(sign)
			_mhsEventSet.push_back(me);
	}

	list <MHSEvent>::iterator searchMHSEventByType(char ty)
	{
		_iter = _mhsEventSet.begin();
		while(_iter != _mhsEventSet.end() && _iter->_type != ty)
			_iter++;
		return _iter;
```

```
}

list <MHSEvent>::iterator searchMHSEvent()
{
    if(_availableTime <= _mhsEventSet.front()._startTime)
        _iter = _mhsEventSet.begin();
    else if(_etvLocation == "mhs107")
    {
        _iter = searchMHSEventByType('s');
        if (_iter == _mhsEventSet.end() || _iter->_startTime > _availableTime)
            _iter = searchMHSEventByType('r');
    }
    else
    {
        _iter = searchMHSEventByType('r');
        if (_iter == _mhsEventSet.end() || _iter->_startTime > _availableTime)
            _iter = searchMHSEventByType('s');
    }
    return _iter;
}

double calculateDuration(list <MHSEvent>::iterator it_me)
{
    double d;
    int row = lexical_cast<int>(it_me->_mhsNo.substr(4,2));
    int etvLevel = lexical_cast<int>(_etvLocation.substr(3,1));
    int etvRow = lexical_cast<int>(_etvLocation.substr(4,2));
    int hoistingTime = 0;
    if(it_me->_type == 'r')
    {
        if(etvLevel == 1)
            hoistingTime = _hoistingTimeBetweenLevel_1_2;
        if(etvLevel == 3)
            hoistingTime = _hoistingTimeBetweenLevel_2_3;
        d = max(hoistingTime, calculateTravelingTime(etvRow, row)) + _loadingTime +
max(_hoistingTimeBetweenLevel_1_2, calculateTravelingTime(row, 7)) + _unloadingTime;
    }
    else
    {
        hoistingTime = 0;
        if(etvLevel == 2)
            hoistingTime = _hoistingTimeBetweenLevel_1_2;
        if(etvLevel == 3)
            hoistingTime = _hoistingTimeBetweenLevel_1_3;
        d = max(hoistingTime, calculateTravelingTime(etvRow, 7)) + _loadingTime +
```

```
max(_hoistingTimeBetweenLevel_1_3, calculateTravelingTime(7, row)) + _unloadingTime;
            }
            return d;
        }

        int calculateTravelingTime(int sr, int er)
        {
            int d = 0;
            if(sr != er)
                d = 3 * abs(sr - er) + 14; //Empirical Function
            return d;
        }

        void update(list <MHSEvent>::iterator it_me, double d)
        {
            _availableTime = max(_availableTime, it_me->_startTime) + d;
            _busyPeriod += d;
            if (it_me->_type == 's')
                _etvLocation = it_me->_mhsNo;
            else
                _etvLocation = "mhs107";
            _mhsEventSet.erase(it_me);
        }

        string display()
        {
            string s = "MHS Events:\n";
            _iter = _mhsEventSet.begin();
            while (_iter != _mhsEventSet.end())
            {
                s += lexical_cast<string>(_iter->_type) + '\t' + _iter->_mhsNo + '\t' +
lexical_cast<string>(_iter->_startTime) + '\n';
                _iter++;
            }
            return s;
        }

        list <MHSEvent> _mhsEventSet;
        list <MHSEvent>::iterator _iter;
        string _etvLocation;
        double _availableTime;
        double _busyPeriod;
        const static int _hoistingTimeBetweenLevel_1_2 = 27; //Time Study (seconds)
        const static int _hoistingTimeBetweenLevel_1_3 = 32;
        const static int _hoistingTimeBetweenLevel_2_3 = 19;
```

```
        const static int _loadingTime = 18;
        const static int _unloadingTime = 14;

};
```

Phase.h

```
#pragma once

class Phase
{
public:

        Phase(void)
        {
        }

        ~Phase(void)
        {
        }

        Phase(string n, char dty = '0', double para_1 = 0, double para_2 = 0)
        {
                _name = n;
                _durationType = dty;
                _parameter_1 = para_1;
                _parameter_2 = para_2;
                _destination = "";
        }

        Phase(string n, string dest)
        {
                _name = n;
                _durationType = 't';
                _destination = dest;
        }

        Phase(string n, PhaseResourceSet prs, char dty = '0', double para_1 = 0, double para_2 = 0)
        {
                _name = n;
                _phaseResourceSet = prs;
                _durationType = dty;
                _parameter_1 = para_1;
                _parameter_2 = para_2;
```

```
            _destination = "";
        }

        Phase(string n, PhaseResourceSet prs, string dest)
        {
            _name = n;
            _phaseResourceSet = prs;
            _durationType = 't';
            _destination = dest;
        }

        string _name;
        PhaseResourceSet _phaseResourceSet;
        char _durationType;
        double _parameter_1;
        double _parameter_2;
        string _destination;

};
```

PhaseResource.h

```
#pragma once

class PhaseResource
{
public:

        PhaseResource(void)
        {
        }

        ~PhaseResource(void)
        {
        }

        PhaseResource(string ty)
        {
            _type = ty;
        }

        string _type;

};
```

PhaseResourceSet.h

```
#pragma once

class PhaseResourceSet
{
public:

	PhaseResourceSet(void)
	{
	}

	~PhaseResourceSet(void)
	{
	}

	PhaseResourceSet(string ty, bool si)
	{
		PhaseResource pr (ty);
		_phaseResourceSet.push_back(pr);
		_sign = si;
	}

	void addPhaseResource (string ty)
	{
		PhaseResource pr (ty);
		_phaseResourceSet.push_back(pr);
	}

	bool _sign;
	list <PhaseResource> _phaseResourceSet;

};
```

Procedure.h

```
#pragma once

class Procedure
{
public:
```

```
Procedure(void)
{
}

~Procedure(void)
{
}

Procedure(string ty, string n, PhaseResourceSet prs, char dty = '0', double para_1 = 0, double para_2 = 0)
{
	_type = ty;
	Phase ph (n, prs, dty, para_1, para_2);
	_procedure.push_back(ph);
}

Procedure(string ty, string n, PhaseResourceSet prs, string dest)
{
	_type = ty;
	Phase ph (n, prs, dest);
	_procedure.push_back(ph);
}

void addPhase(string n, char dty = '0', double para_1 = 0, double para_2 = 0)
{
	Phase ph (n, dty, para_1, para_2);
	_procedure.push_back(ph);
}

void addPhase(string n, string dest)
{
	Phase ph (n, dest);
	_procedure.push_back(ph);
}

void addPhase(string n, PhaseResourceSet prs, char dty = '0', double para_1 = 0, double para_2 = 0)
{
	Phase ph (n, prs, dty, para_1, para_2);
	_procedure.push_back(ph);
}

void addPhase(string n, PhaseResourceSet prs, string dest)
{
	Phase ph (n, prs, dest);
	_procedure.push_back(ph);
}
```

```
        string _type;
        list <Phase> _procedure;

};
```

Resource.h

```
#pragma once
#include "phaseresource.h"

class Resource :
        public PhaseResource
{
public:

        Resource(void)
        {
        }

        ~Resource(void)
        {
        }

        Resource(string ty, string n = "") : PhaseResource(ty)
        {
                _name = n;
                if(_name == "")
                        _name = ty;
                _availableTime = 0;
                _retrievalOrderId = 0;
                _startTime = 0;
                _busyPeriod = 0;
                _check = false;
                _location = "";
        }

        void update(int roi, double st)
        {
                _retrievalOrderId = roi;
                _startTime = st;
                _availableTime = 9999999;
        }
```

```
        void update(double at)
        {
                _retrievalOrderId = 0;
                _availableTime = at;
                _busyPeriod += _availableTime - _startTime ;
        }

        string _name;
        double _availableTime;
        int _retrievalOrderId;
        double _startTime;
        double _busyPeriod;
        bool _check;
        string _location;

};
```

ResourcePool.h

```
#pragma once

class ResourcePool
{
public:

        ResourcePool(void)
        {
        }

        ~ResourcePool(void)
        {
        }

        ResourcePool(string ty)
        {
                Resource r (ty);
                _resourcePool.push_back(r);
        }

        ResourcePool(string ty, string n)
        {
                Resource r (ty, n);
                _resourcePool.push_back(r);
        }
```

```
void addResource(string ty)
{
	Resource r (ty);
	_resourcePool.push_back(r);
}

void addResource(string ty, string n)
{
	Resource r (ty, n);
	_resourcePool.push_back(r);
}

string display()
{
	string s = "Resource in use:\n";
	_iter = _resourcePool.begin();
	while(_iter != _resourcePool.end())
	{
		if(_iter->_retrievalOrderId != 0)
			s += _iter->_name + '\t' + lexical_cast<string>(_iter->_retrievalOrderId) + '\n';
		_iter++;
	}
	return s;
}

void initializeResourceLocations()
{
	_iter = _resourcePool.begin();
	while(_iter != _resourcePool.end())
	{
		if(_iter->_type == "forklift")
			_iter->_location = "TD105";
		_iter++;
	}
}

void initializeResources()
{
	for(int i = 1; i < 11; i++)
		addResource("ASRSQueue", "aq_" + lexical_cast<string>(i));
	for(int i = 1; i < 9; i++)
	{
		addResource("deck", "dkA_" + lexical_cast<string>(i));
		addResource("deck", "dkB_" + lexical_cast<string>(i));
```

```
            addResource("deck", "dkC_" + lexical_cast<string>(i));
        }
        for(int i = 101; i < 111; i++)
            addResource("agent", "A" + lexical_cast<string>(i));
}

list <Resource>::iterator searchResourceByName(string n)
{
        _iter = _resourcePool.begin();
        while(_iter->_name != n)
            _iter++;
        return _iter;
}

list <Resource>::iterator searchResourceByType(string ty)
{
        _iter = _resourcePool.begin();
        list <Resource>::iterator earliestResourceIter;
        while(_iter->_type != ty)
            _iter++;
        earliestResourceIter = _iter;
        while (_iter != _resourcePool.end())
        {
            if(_iter->_type == ty && _iter->_availableTime < earliestResourceIter->_availableTime)
                earliestResourceIter = _iter;
            _iter++;
        }
        return earliestResourceIter;
}

list <Resource>::iterator searchResourceByRetrievalOrderId(int roi, string ty)
{
        _iter = _resourcePool.begin();
        while(_iter != _resourcePool.end() && (_iter->_type != ty || _iter->_retrievalOrderId != roi))
            _iter++;
        return _iter;
}

void updateResource(bool s, string ty, int roi, double st, double d)
{
        if (s)
            searchResourceByType(ty)->update(roi, st);
        else
            searchResourceByRetrievalOrderId(roi, ty)->update(st + d);
}
```

```
string displayUtilization(double tt)
{
	string s = "Resource Utilization:\n";
	_iter = _resourcePool.begin();
	while(_iter != _resourcePool.end())
	{
		if(_iter->_check != true)
		{
			double busyTime = _iter->_busyPeriod;
			int c = 1;
			list <Resource>::iterator it_r = _iter;
			it_r++;
			while(it_r != _resourcePool.end())
			{
				if(it_r->_type == _iter->_type)
				{
					busyTime += it_r->_busyPeriod;
					it_r->_check = true;
					c++;
				}
				it_r++;
			}
			s += _iter->_type + '\t' + lexical_cast<string>(busyTime/tt/c) + '\n';
		}
		_iter++;
	}
	return s;
}

double calculateForkliftUtilization(double tt)
{
	double busyTime = 0;
	int c = 0;
	_iter = _resourcePool.begin();
	while(_iter != _resourcePool.end())
	{
		if(_iter->_type    == "forklift" && _iter->_availableTime != 99999999)
		{
			busyTime += _iter->_busyPeriod;
			c++;
		}
		_iter++;
	}
	return busyTime / c / tt;
```

```
        }

        string calculateIndividualUtilization(double tt)
        {
                string s = "Individual Resource Utilization:\n";
                _iter = _resourcePool.begin();
                while(_iter != _resourcePool.end())
                {
                        if(_iter->_busyPeriod != 0)
                                s += _iter->_name + '\t' + lexical_cast<string>(_iter->_busyPeriod/tt) + '\n';
                        _iter++;
                }
                return s;
        }

        void resetForklifts()
        {
                _iter = _resourcePool.begin();
                while(_iter != _resourcePool.end())
                {
                        if(_iter->_type == "forklift")
                                _iter->_busyPeriod = 0;
                        _iter++;
                }
        }

        void activateForklifts()
        {
                _iter = _resourcePool.begin();
                while(_iter != _resourcePool.end())
                {
                        if(_iter->_availableTime == 99999999)
                                _iter->_availableTime = 0;
                        _iter++;
                }
        }

        list <Resource> _resourcePool;
        list <Resource>::iterator _iter;

};
```

RetrievalList.h

```
#pragma once

class RetrievalList
{
public:

	RetrievalList(void)
	{
	}

	~RetrievalList(void)
	{
	}

	RetrievalList(RetrievalOrder ro)
	{
		_retrievalList.push_back(ro);
	}

	RetrievalList(string sl, string tar, int rt, int p)
	{
		RetrievalOrder ro (sl, tar, rt, p);
		_retrievalList.push_back(ro);
	}

	void addRetrievalOrder(string sl, string tar, int rt, int p)
	{
		RetrievalOrder ro (sl, tar, rt, p);
		_retrievalList.push_back(ro);
	}

	list <RetrievalOrder>::iterator searchRetrievalOrderById(int roi)
	{
		_iter = _retrievalList.begin();
		while(_iter->_id != roi)
			_iter++;
		return _iter;
	}
	list <RetrievalOrder> _retrievalList;
	list <RetrievalOrder>::iterator _iter;

};
```

RetrievalOrder.h

```
#pragma once

class RetrievalOrder
{
public:

	RetrievalOrder(void)
	{
	}

	~RetrievalOrder(void)
	{
	}

	RetrievalOrder(string sl, string tar, int rt, int p)
	{
		_id = _nextId++;
		_storageLocation = sl;
		_target = tar;
		_requestTime = rt;
		_pieces = p;
	}

	static int _nextId;
	int _id;
	string _storageLocation;
	string _target;
	int _requestTime;
	int _pieces;

};

int RetrievalOrder::_nextId = 1;
```

Stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once
```

```
#include <fstream>
#include <tchar.h>
using namespace std;
#include <string>
#include <list>
#include <ctime>
#include <cmath>
#include <boost/timer.hpp>
#include <boost/random.hpp>
#include <boost/lexical_cast.hpp>
using boost::lexical_cast;

#include "PhaseResource.h"
#include "PhaseResourceSet.h"
#include "Phase.h"
#include "Procedure.h"
#include "ImportProcess.h"
#include "Resource.h"
#include "ResourcePool.h"
#include "ASRSEvent.h"
#include "ASRSEventSet.h"
#include "Location.h"
#include "Layout.h"
#include "MHSEvent.h"
#include "MHSEventSet.h"

// TODO: reference additional headers your program requires here
```

B. AS/RS simulation program code

Two files are included:

SimASRS.cpp: main program.

Stdafx.h: provides include files for the program.

SimASRS.cpp

```
// SimASRS.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
	boost::timer totalTime;
	ifstream in ("inDistance.xls");
	ofstream out ("result.xls");
	ofstream outPositions ("positions.xls");
	double vBp[] =
{0.321962438,0.413568417,0.510157148,0.586048294,0.658106554,0.71751629,0.768876964,0.823303948,0.853
583749,0.880030663,0.904177846,0.920659256,0.937140667,0.948639325,0.955921809,0.967803756,0.97585281
7,0.982752012,0.986968187,0.991950939,0.994250671,0.995783825,0.996933691,0.998083557,0.998466846,0.99
9233423,0.999233423,0.999616711,0.999616711,0.999616711,1}; //
	double vSp[] =
{0.321962438,0.410502108,0.498275201,0.583365274,0.66193944,0.717133001,0.770793407,0.816021464,0.852
433883,0.881180529,0.904561135,0.921042545,0.93637409,0.951322346,0.960904561,0.973553085,0.979685703
,0.985051744,0.989267919,0.993484094,0.995017248,0.996933691,0.99731698,0.998083557,0.998850134,0.9992
33423,0.999616711,0.999616711,1}; //
	double tBp[] =
{0,0.0064,0.134,0.44,0.6332,0.77,0.8144,0.8528,0.8792,0.8968,0.91,0.9232,0.936,0.948,0.9576,0.966,0.9708,0.976
,0.982,0.988,0.994,0.9955,0.997,0.9985,1}; //
	double tSp[] =
{0,0.018477458,0.109090909,0.270066519,0.388470067,0.485587583,0.562749446,0.629268293,0.674501109,0.7
03769401,0.725055432,0.76097561,0.789356984,0.802660754,0.829268293,0.862527716,0.885144124,0.9048780
49,0.918847007,0.932815965,0.946784922,0.960088692,0.973392461,0.986696231,1}; //
	double distance[189]; // (7 * 14 - 4) * 2 + 1
	int i = 1;
	while(in >> distance[i])
	{
		distance[i+1] = distance[i];
		i = i + 2;
	}
	int period;
	double randomNum;
	int vB;
	int vS;
	int binNo;
	double ratio_vB;
	double travelTimeSum = 0;
	int travelTimeNum = 0;
	int level;
	int warmUp = 20000;
```

```
for(int s = 0; s <= 99; s++) // Max InvB
{
	double travelTimeAll = 0;
	double levelAll = 0;
	boost::hellekalek1995 generatorVB(17);
	boost::hellekalek1995 generatorVS(18);
	boost::hellekalek1995 generatorTB(19);
	boost::hellekalek1995 generatorTS(20);
	boost::hellekalek1995 generatorBorS(21);
	boost::uniform_01<boost::hellekalek1995> gen_vB(generatorVB);
	boost::uniform_01<boost::hellekalek1995> gen_vS(generatorVS);
	boost::uniform_01<boost::hellekalek1995> gen_tB(generatorTB);
	boost::uniform_01<boost::hellekalek1995> gen_tS(generatorTS);
	boost::uniform_01<boost::hellekalek1995> gen_BorS(generatorBorS);
	int positions[189] = {0};
	int bin[189] = {0}; // Max Inv + 1
	for(period = 1; period <= 10 * warmUp; period++)
	{
		randomNum = gen_vB();
		vB = 0;
		while(randomNum >= vBp[vB])
			vB++;
		randomNum = gen_vS();
		vS = 0;
		while(randomNum >= vSp[vS])
			vS++;
		travelTimeNum += vB + vS;
		binNo = 1;
		while(vB > 0 && binNo <= s)
		{
			if(bin[binNo] == 0)
			{
				travelTimeSum += distance[binNo];
				randomNum = gen_tB();
				bin[binNo] = 1;
				while(randomNum >= tBp[bin[binNo]])
					bin[binNo]++;
				vB--;
			}
			binNo++;
		}
		binNo = s + 1;
		while(vB + vS > 0)
		{
			if(bin[binNo] == 0)
```

```
                {
                    travelTimeSum += distance[binNo];
                    ratio_vB = (double)vB / (vB + vS);
                    if(vB > 0 && gen_BorS() < ratio_vB)
                    {
                        randomNum = gen_tB();
                        bin[binNo] = 1;
                        while(randomNum >= tBp[bin[binNo]])
                            bin[binNo]++;
                        vB--;
                    }
                    else
                    {
                        randomNum = gen_tS();
                        bin[binNo] = 1;
                        while(randomNum >= tSp[bin[binNo]])
                            bin[binNo]++;
                        vS--;
                    }
                }
                binNo++;
            }
            level = 0;
            for(binNo = 1; binNo <= 188; binNo++) // Max Inv
                if(bin[binNo] != 0)
                {
                    bin[binNo]--;
                    level = binNo;
                    positions[binNo]++;
                }
            levelAll += level;
            if(period % warmUp == 0)
            {
                travelTimeAll += travelTimeSum / travelTimeNum;
                travelTimeSum = 0;
                travelTimeNum = 0;
            }
        }
        out << travelTimeAll / 10 << '\t';
        out << levelAll / (period - 1) << endl;
        for(binNo = 1; binNo <= 188; binNo++)
            outPositions << (double)positions[binNo] / (period - 1) << '\t';
        outPositions << endl;
        cout << totalTime.elapsed() << endl;
}
```

```
    return 0;
}
```

Stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once



#include <fstream>
#include <tchar.h>
using namespace std;
#include <string>
#include <list>
#include <ctime>
#include <cmath>
#include <boost/timer.hpp>
#include <boost/random.hpp>
#include <boost/lexical_cast.hpp>
using boost::lexical_cast;

// TODO: reference additional headers your program requires here
```