İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

PATTERNS IN SYSTEM MODELING THEORY

M.Sc. Thesis by
Ömer Gökçek
(507031118)

Date of submission : 24 December 2007

Date of defence examination: 29 January 2008

Supervisor (Chairman): Doç.Dr. Mehmet Mutlu YENISEY (İTÜ.)

Members of the Examining Committee **Doç.Dr. Tufan Vehbi KOÇ (İTÜ.)**

Prof.Dr. Demet BAYRAKTAR (İTÜ.)

FEBRUARY 2008

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

MODELLEME TEORİSİNDE MODÜLERLİK

MASTER TEZİ

Ömer GÖKÇEK

Tezin Enstitüye Verildiği Tarih : 24 Aralık 2007

Tez Danışmanı : Doç.Dr. Mehmet Mutlu YENISEY (İTÜ.)

Diğer Jüri Üyeleri **Doç.Dr. Tufan Vehbi KOÇ (İTÜ.)**

Prof.Dr. Demet BAYRAKTAR (İTÜ.)

ŞUBAT 2008

PREFACE

This report is of value for those who are interested in architectures and methodologies used in Process modeling.

It is a great pleasure for me to thank the many people who in different ways have supported my studies and contributed to the process of writing this thesis. First I would like to thank my coach Asist. Prof. Mehmet Mutlu Yenisey for his inspiration in starting to the project. The subject was completely state of science, and I am grate full for the guidance. Finally, I want to thank my family and friends for their unconditional support.

February 2008 Ömer GÖKÇEK

INDEX

PREFACE ii

INDEX iii

LIST OF ABBREVIATIONS vi

LIST OF FIGURES vii

LIST OF TABLES ix

ÖZET x

ABSTRACT xi

1. INTRODUCTION 1
1.1. Problem description 1
1.2. Research objective 3
1.3. Research plan and structure of the report 3

2. SIMULATION MODELING, COLOR PETRI NETS & UML 4
2.1. Simulation Modeling 4
2.1.1. The Nature of Simulation 4
2.1.2. System, Model and Simulation 5
2.1.3. Discrete Event Simulation 6
2.1.4. Time- Advance Mechanisms 6
2.1.5. Components and Organization of a Discrete Event Simulation Model 7
2.2. Petri Nets 7
2.2.1. Application of Petri Nets: 8
2.2.2. Description of Petri Nets 9
2.2.3. Properties of Petri Nets 12
2.2.4. Colored Petri Nets 13
2.3. Unified Modeling Language 15
2.3.1. The Meta-model 16
2.3.2. The Notation 16
2.3.3. Class Diagrams 16
2.3.4. Inheritance 17
2.3.5. Aggregation / Association 18
2.3.6. Interfaces 18

3. CONTEXT **20**
3.1. Steps for Creating a Decision Study 20
3.2. Evaluation 20

4. RESEARCH **22**
4.1. Selected Cases 22
4.2. Solving the cases 23
4.3. Pattern Search 23
4.4. Without Hierarchy 23
4.4.1. With Hierarchy (additive) 23
4.4.2. With Hierarchy (Multiplicative) 23

5. SHIP TRANSPORTATION CASE **24**
5.1. Case Description 24
5.2. Decision Description 25
5.3. The Black Box representation 25
5.4. UML Model 27
5.5. Petri Net Patterns 29
5.5.1. Tug Activity Pattern 29
5.5.2. Storm Pattern 31
5.5.3. Halt Pattern 33
5.5.4. Switch Pattern 33
5.5.5. Interrupt Pattern 34
5.6. Model Creation 35
5.6.1. Adding structure without hierarchy 35
5.6.2. Adding structure with hierarch 37

6. SHIP LOADING CASE **39**
6.1. Case Description 39
6.2. Decision Description 39
6.3. Black Box Representation 39
6.4. UML model 41
6.5. Petri Net patterns 41
6.5.1. Source Increase Pattern 41
6.5.2. Source Decrease Pattern 43
6.6. Model Creation 45
6.6.1. Adding structure without hierarch 45
6.6.2. Adding structure with hierarch 45
6.6.3. Adding structure with hierarch (Multiplicative) 46

7. ELEVATOR CASE **47**
7.1. Case Description 47
7.2. Decision Description 48

7.3. Black Box Representation 49
7.4. UML model 50
7.5. Petri Net Patterns 50
7.5.1. Call Pattern 50
7.5.2. Ascending/Descending Transportation Pattern 52
7.5.3. Occupy Pattern 53
7.5.4. Switch Pattern 53
7.6. Model Creation 54
7.6.1. Adding structure with hierarch 54

8. INVENTOR SYSTEM CASE 57
8.1. Case Description 57
8.2. Decision Description 58
8.3. The Black Box representation 59
8.4. UML Model 59
8.5. Petri Net Patterns 59
8.5.1. Delivery Pattern 59
8.5.2. Inventory Control Pattern 60
8.5.3. Inventory Set Pattern 61
8.5.4. Inventory Log Pattern 62
8.6. Model Creation 63
8.6.1. Adding structure without Hierarch 63
8.6.2. Adding Structure with Hierarch 63

9. EVALUATION 65

10. CONCLUSION AND RECOMENDATION 68
10.1.Impact of Pattern Use 68
10.2.Use of Connectors 69
10.3.Recommendations 69

REFEFENCES 70

CURRICULUM VIATE 73

LIST OF ABBREVIATIONS

CPN	: Color Petri Nets
UML	: Unified Modelng Language
FIFO	: First in First Out
FMS	: Flexiable Manufacturing Systems

LIST OF FIGURES

Page No

Figure 2.1: Basic Petri Net 10
Figure 2.2: Transition 12
Figure 2.3 : Firing 12
Figure 2.4:Meta Figure 17
Figure 2.5: Class Diagram 17
Figure 2.6: Inheritance 18
Figure 5.1: Black box Representation 27
Figure 5.2: Ship Transportation Figure 28
Figure 5.3: Tug Activity 29
Figure 5.4: Storm 32
Figure 5.5: Halt 33
Figure 5.6: Switch 33
Figure 5.7: Interrupt 34
Figure 5.8: Figure without Hierarchy 36
Figure 5.9: Combined Pattern of Tug and Storm 37
Figure 5.10: Top hierarch Figure 38
Figure 6.1: Ship Loading Blackbox 40
Figure 6.2: Ship Loading Figure 41
Figure 6.3: Source Increase basic pattern 42
Figure 6.4: Source Increase complex pattern 42
Figure 6.5: Source Decrease complex pattern 44
Figure 6.6: Ship Loading Figure without Hierarch 45
Figure 6.7: Combined pattern of Source increase and decrease 45
Figure 6.8: Top Hierarch Figure for Ship Loading 46
Figure 6.9: Multiplicative Figure of ship loading 46
Figure 7.1: Black box of Elevatation 49
Figure 7.2: Elevator Figure 50
Figure 7.3: Call 51

Page No

Figure 7.4: Trasport up/ Trasport down 52
Figure 7.5: Occupy 53
Figure 7.6: Switch 53
Figure 7.7: Elevation logic 55
Figure 7.8: Top Elevator Figure 56
Figure 8.1: Black box of Inventory system 59
Figure 8.2: Inventory System Figure 59
Figure 8.3: Delivery Pattern 60
Figure 8.4: Inventory Control 61
Figure 8.5: Inventory Set 62
Figure 8.6: Inventory Log 62
Figure 8.7: Inventory Figure without Hierarch 63
Figure 8.8: Combined patterns of inventory control and inventory set 64
Figure 8.9: Combined pattern of Inventory order and delivery 64
Figure 8.10: Inventory Figure with hierarch 64
Figure 9.1: Combination of Inventory, ship trasport and ship loading cases 66

LIST OF TABLES

Page No

Table 5.1: Loading Times 24

Table 8.1: Inventor Policy 58

MODELLEME TEORİSİNDE MODÜLERLİK

ÖZET

Günümüzde, İşletmeler sürekli değişen dış ortam ile karşı karşıyadırlar. Uygun yönetim için daha iyi iş akışı ve değişime uyum sağlayacak işletme modelleri gereklidir. İş akışı modellemenin hedefi, iş akışı sisteminini anlamak, performans değerlendirmesini üstünleştirmek, karar vermeyi desteklemek ve sürekli iyileştirme için değiştirme cesareti vermektir. Bu hedefe ulaşmak için, etkili enformasyon teknolojisi gerkelidir. Bu nedenle, İşletme içersinde bilgiye ulaşımı destekleyecek etkili enformasyon teknolojisi gereklidir.

Organizasyon yönetimine destek amacı ile model entegrasyonuna destek için bu proje referans mimarisi ve modellemesi çercevesinde pattern modellerin geliştirilmesi üzerine odaklanmıştır. Dahası, Referans mimarisini belirleyen ve Üretim proseslere rehberlik eden pattern modelleri, ilgisi olamayan komponentler arasındaki ilişkiyi kurup prosesin ana elemanlarını tanımlar. Bu çalışmamda içeriği sağlamak için elektronik ticaret konusunda uluslararası camia tarafından Kabul gören Renkli Petri ağları ve UML dili şeçilmiştir.

Birbirinden farklı optimizasyon durumları incelenmiş ve her duruma ait olan prosesler, daha temel alt proseslere bölünmüştür, Her bir temel alt proses patern olarak adlandırılır.

Paternler, Colour Petri net paternleri gibi bir çok farklı kombinasyonda simulasyon modeli tasarımı yapılandırılabilecek tutarlı araçları ve kurallara ulaşmak için incelenir. Patternler systemin çalışma mekanizmasını ve yapılandırma şartlarını anlamaya yarar. Bulunan değerler özellikle sıra sistemleri ve depo sistemleri olmak üzere sistemin ana modeli üzerinde daha sonra genellenebilir. Böylelikle, Modelin geri dönüşebilirliği ve devam edilebilirliği sağlanabilir.

PATTERNS IN SYSTEM MODELING THEORY

SUMMARY

Enterprises today are faced with a rapidly changing environment. There is a need for better process management and increased integration, and enterprise modeling is necessary for proper management. The goal of process modeling is to improve understanding of what happens in the process system, enhance performance evaluation of some parts of the process, support decision-making activities and encourage changes for continuous improvement. Therefore, efficient information technology is required to support the availability of information in the enterprise.

In an effort to achieve the integration of models aimed at supporting various facets of organizational management, this project focused on the development of pattern modeling in the context of reference architectures and methodologies. Furthermore the Pattern models, that underlie various reference architectures and the guidelines on modeling manufacturing processes, specify the main elements of an process and provides a relation between its interrelated components. To provide the context that this work was performed in, Colored Petri Nets and UML which were selected because they result from and are promoted by the major international coalition in the area of e-commerce.

Different Optimization cases have been examined, and the process of each case has been divided into several smaller sub processes which are called patterns.

Patterns are examined for coherent tools which include basic generalized colored Petri net patterns that can be configured as any combination in system model simulation design. The patterns are used to understand the mechanisms and conditions for system configuration. Findings might be generalized to on any system model, especially queuing systems and inventory systems .Thus, the recyclability and the sustainability of the modem can be acquired.

1. INTRODUCTION

1.1. Problem description

The importance of simulation for decision making on production systems increases. The success of many enterprises depends on the use of advanced decision making techniques. In a dynamic environment, the constraints are always changing, so exceptions or deviations from plans occur almost regularly (E. Liu, A. Kumar, & Aalst, 2007) It is necessary to have an effective and efficient method. A conceptual model is a necessary deliverable in a simulation enabled decision study. Creation of the conceptual model is a time consuming mental process which is labor intensive activity (Arons & Boer, 2001)

Colored Petri Nets (CPN), which is a graphical language, is extensively used for modeling and analysis of distributed systems with elements of concurrency. It is one of the most practical and efficient languages to create and evaluate a conceptual model for simulation base researches. The challenges in using colored Petri nets (CPN) for workflow management is interfacing the Petri nets to the real world in a general way, and in a way that is easily changeable. (Kristoffersen & Boudko, 2006) In the complex cases the modeling itself requires a large effort, more so if the resulting model can only be used once .(Zulch&Fisher,2001) Alignment of the systems, however, implies extensive configuration and customization efforts in the implementation process and may lead to significant implementation costs that exceed the price of software licenses by factor five to ten. (Davenport, 2000)

Current systems do not provide any knowledge about previously applied process instance changes when a new ad-hoc deviation becomes necessary; users have to define each ad-hoc change from scratch. This does not only lead to high efforts and lower user acceptance, but also to greater variability of change definitions and more noise. (Aalst, Guenther, J. Recker, & M. Reichert, 2006)

There is a need for a clearly structured configuration procedure. (J. Recker, J. Mendling, Aalst, & M. Rosemann, 2006) and the challenge that we undertake in

this report is to investigate the use of patterns in the early phases of the simulation study.

But it is a possible solution direction to split the decision study into several smaller phases, the deliverables of which can be reused in other decision studies. (Zulch, Jonsson, & Fisher, 2002) In cases such as flexible manufacturing systems, it is then possible to configure a variety of general operation simulation models on the basis of reusable smaller operation patterns .

There are several reasons for using Colored Petri Nets and CPN Tools (I. Vanderfeesten, Aalst, & Reijers, 2005):

- First, the use of Petri nets provides correctness analysis of the model by means of, for instance, a state space analysis.
- It is very easy to make small adaptations to the models and then compare the outcomes.
- Moreover, the integration of process and data is essential to this view on a workflow process.
- CPN Tools provides the possibility to integrate them in one model. The data is then captured in the data structures of the model, i.e. in the colors or types, and the process in the PetriNet itself.
- Finally, CPN Tools contains a simulation environment, in which one can go step-by-step through the model, following or defining oneself a sequence of firing.

In the context of the ten step method, the focus is on the three first steps which conclude with conceptual system model. For this model we make use of a set of empirically gathered patterns in Colored Petri nets. The main focus of the CPN patterns is to aid better simulation modeling and analysis for queuing problems and work flows (Aalst & Mulyar, 2005) It allows modeler to design sub operations which are independent from the uniformity of the general operation model.

The main aim of this project is searching for coherent tools which include basic generalized colored Petri net patterns that can be configured as any combination in system model simulation design. The work will focus on small examples to understand the mechanisms and conditions for system configuration. Findings can then be generalized to on any system model, especially queuing systems and inventory systems which are focused in this study.

1.2. Research objective

As mentioned in the problem description, main objective of this study is exploring a way to combine specific system modules as patterns which will collaborate in any system model structure. The study will draw on existing CPN patterns (Mulyar & van der Aalst, 2005) and on the conceptual system model used in the course Simulation of Operational Processes (Pels & Goossenaerts, 2005)

1.3. Research plan and structure of the report

First the main concept of the simulation modeling, the high level Petri nets and CPN needs to be understood. To do this several sources and methodologies have been examined, simulation problems, work flow logic have been reviewed. The book of Law&Kelton (2000) is one of the main resources in this study. Since the aim of the study is to acquire the patterns, similar studies on that subject should be overviewed.

Secondly the scope and the context of the project have to be defined in order to become a properly focused effort on specific problem types. This is done in chapter 2 where the methodology for the simulation concept is recalled. Concepts such as problem formulation, problem objectives, level of details and design study are mentioned in this section.

Thirdly the criteria for comparing the available methodology with and without the use of patterns for pattern gathering is described and illustrated. Those criteria are related to the examination of the methodology in the cost aspect, time aspect, team and organization aspect, insight aspect and the misguidance aspect (Goossenaerts & Pels (2005)) for details on these aspects. Conceptual models in Petri nets and UML will be gathered and the possible variation for basic elemental patterns such as additive and multipartite will be described.

The final step is to gather the pattern information by solving selected problems that are basically about operational processes. Each problem will be explained briefly and next, the problem description, its black box representation, and the conceptual model will be defined. The pattern acquisition will be done in the way that has been described in the methodology section. The different patterns that have been gathered by different ways will be compared and examined in detail.

A Chapter with conclusions and recommendations for further work concludes the report.

2. SIMULATION MODELING, COLOR PETRI NETS & UML

2.1. Simulation Modeling

2.1.1. The Nature of Simulation

Most real world systems are too complex to allow realistic models to be evaluated analytically, and these models must be studied by means of simulation. In a simulation, computer is used to evaluate a model numerically, and data are gathered in order to estimate the desired true characteristics of the model.

Application areas for simulation are numerous and diverse. Below is a list of some particular kinds of problems for which simulation has been found to be a useful and powerful tool. (Law & Kelton, 2000)

- Designing and analyzing manufacturing systems
- Evaluating military weapons systems or their logistics requirements
- Determining hardware and software requirements for a computer system.
- Designing and operating transportation systems such as call airports, freeways, ports, and subways.
- Evaluating designs for service organizations such as call centers, fast food restaurants, hospitals, and post offices.
- Reengineering of business processes
- Determining ordering policies for an inventory system
- Analyzing financial or economic systems.

Simulation is one of the most widely used operations-research and management science techniques, if not the most widely used. There are several surveys related to the use of operations – research techniques. According to Law& Kelton (2000)

simulation was consistently ranked as one of the three most important "operation-research techniques". The other two were "math programming" and "statistics".

There have been however several impediments to even wider acceptance and usefulness of simulation. First, models used to study large scale systems tend to be very complex and writing computer programs to execute them can be an arduous task indeed. This task has been made much easier in recent years by the development of excellent software products that automatically provide many of the features needed to program a simulation model. (Law & Kelton, 2000)

A second problem with simulation of complex systems is that a large amount of computer time is sometimes required. However, this difficulty is becoming much less severe as computers become faster and cheaper. (Law & Kelton, 2000)

Finally, there appears to be an unfortunate impression that simulation is just an exercise in computer programming, albeit a complicated one. Consequently, man simulation studies have been composed of heuristic model building, coding and a single run of the program to obtain "the answer." (Law & Kelton, 2000)

2.1.2. System, Model and Simulation

Law and Kelton (2000) define a system to be a collection of entities, e.g., people or machines that act and interact together towards the accomplishment of some logical end. In practice, what is meant by the system depends on the objectives of a particular study. The collection of entities that comprise a system for one study might be only a subset of the overall system for another.

The systems can be categorized to be of two types, discrete and continuous. A discrete system is one for which the state variables change instantaneously at separated points in time. A bank is an example of a discrete system, variables change only when a customer arrives or when a customer finishes being served and departed. A continuous system is one for which the state variables change continuously with respect to time. An airplane moving through the air is an example of a continuous system. State variables such as position and velocity can change continuously with respect to time. (Law & Kelton, 2000)

A mathematical model should be studied by means of simulation in most situations, due to the sheer complexity of the system of interest and of the models necessary to

represent them in a valid way. For this purpose it is useful to classify simulation models along three different dimensions. (Law & Kelton, 2000)

Static vs. Dynamic Simulation Models: A static simulation model is a representation of a system at a particular time or one that may be used to represent a system in which time simply plays no role; on the other hand a dynamic simulation model represents a system a s it evolves over time such as a conveyor system in the factory. (Law & Kelton, 2000)

Deterministic vs. Stochastic simulation Models: If a simulation model does not contain any probabilistic components it is called deterministic; and the output is determined once the set of input quantities and relationships in the model have been specified. Many items however must be modeled as having at least some random input components, and these give rise to stochastic simulation models. (For example of the ship randomness in ship transport case.) Stochastic simulation models produce output that is itself random and must therefore be treated as only an estimate of the true characteristics of the model; this is one of the main disadvantages of simulation. (Law & Kelton, 2000)

2.1.3. Discrete Event Simulation

Discrete event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. These points in time are the ones at which an event occur, where an event is defined as an instantaneous occurrence that may change the state of the system. Although discrete event simulation could conceptually be done by hand calculations, the amount of data that must be stored and manipulated for most real world systems dictates that discrete event simulations be done on a digital computer. (Law & Kelton, 2000)

2.1.4. Time- Advance Mechanisms

Because of the dynamic nature of discrete event simulation models, track of the current value of simulated time as the simulation proceeds should be kept, and a mechanism is needed to advance simulated time from one value. The unit of time for the simulation clock is never stated explicitly and it is assumed to be in the same units as the input parameter. Also, there is generally no relationship between simulated time and the time needed to run a simulation on the computer. The major approach for advancing the simulation clock is next event time advance approach. In

this approach the simulation clock is initialized to zero and the times of occurrence of future event are determined. The same approach has been used in the current study. (Law & Kelton, 2000)

2.1.5. Components and Organization of a Discrete Event Simulation Model

Although simulation has been applied to a great diversity of real world systems, discrete-event simulation Figures all share a number of common components and there is logical organization for there components that promotes the programming debugging, and future changing of a simulation Figure's computer program. In particular, the following components will be found in most discrete event simulation Figures using the next-event time- advance approach programmed in a general-purpose language: (Law & Kelton, 2000)

- System State: The collection of state variable necessary to describe the system at a particular time.
- Simulation Clock: A variable giving the current value of simulated time.
- Event List: A list containing the next time when each type of event will occur.
- Statistical Counters: variables used for storing statistical information about system performance.
- Initialization Routine: A subprogram to initialize the simulation model at time 0.
- Timing Routine: A subprogram determines the next event from the event list and then advances the simulation clock to the time when that event is to occur.
- Event routine: A subprogram that updates the system state when a particular type of event occurs.
- Library routines: A set of subprograms used to generate random observations from probability distributions that were determined as a part of the simulation model.

2.2. Petri Nets

Petri nets are graphical and mathematical tools which provide a standardized setting for modeling, formal analysis, and design of discrete event systems. (Zurawski & Zhou, 1994)

The main advantages of using Petri net models is that the same model can be used for logical structure of discrete event simulators and controllers, as well as the study of behavioral properties and performance valuation.

Petri nets were named after Carl A. Petri who created in 1962 a net-like mathematical tool for the study of communication with automata and were used as net-like mathematical tools for communication study in 1962. Since then, many extensions and variations have been developed. In the current study these extensions and variations will be referred as colored Petri net which can be the interface between the problem situation and the methods of analysis. (Wong, Parkin, & Coy, 2007)

2.2.1. Application of Petri Nets:

- Graphical Tool: Petri nets provide a powerful communication medium between the user, typically requirements engineer, and the customer. Complex requirements specifications, instead of using ambiguous textual descriptions or mathematical notations difficult to understand by the customer, can be represented graphically using Petri nets. (Zurawski & Zhou, 1994)
- Mathematical tool: a Petri net model can be described by a set of linear algebraic equations, or other mathematical models reflecting the behavior of the system. It allow for the performance evaluation of the modeled systems. (Zurawski & Zhou, 1994)
- Modeling and analysis of communication protocols: SDL, Lotos, Estelle based protocol specifications into Petri net has been transformed for performance and reliability analysis. (Zurawski & Zhou, 1994)
- Modeling sequence controllers: Programmable Logic Controllers are commonly used for the sequence control in automated systems. Petri net modeling reduced the development time compared with the traditional approach. (Zurawski & Zhou, 1994)
- Analysis of manufacturing systems: Petri nets have been used extensively to model and analyze manufacturing systems. In this area, Petri nets were used to represent simple production lines with buffers, machine shops, automotive production systems, flexible manufacturing systems, automated assembly lines, resource-sharing systems, and recently just-in-time and KANBAN manufacturing systems. (Adamou, 1993).
- The performance of production systems, involving simple production lines, job shops, robotic assembly cells, flexible manufacturing systems, etc., can be

analyzed with Petri Nets. (Zurawski & Zhou, 1994). The discrete-event simulation can be driven from the model, sometimes using complex algorithmic strategies representing real-time scheduling and control policies of production systems.

- Petri nets with time extensions, combined with heuristic search techniques, were used to model and study scheduling problems involving manufacturing systems as well as robotic systems (Zurawski & Zhou, 1994)
- Software development: Petri nets have been extensively used in software development. The work in this area focused on modeling and analysis of software systems using Petri nets (Zurawski & Zhou, 1994).
- Communication networks: Work was conducted on Fiber Optics Local Area Networks such as Expressnet, Fastnet, D-Net, U-Net. Token Ring (Zurawski & Zhou, 1994).
- Petri nets have recently become widely accepted as a description method for biological pathways by researchers in computer science as well as those in biochemistry (Matsuno & Miyano, 2006)

2.2.2. Description of Petri Nets

A Petri net may be recognized as a particular type of bipartite directed graph populated by three types of objects. These objects are places, transitions, and directed arcs connecting places to transitions and transitions to places. Pictorially, places are illustrated by circles and transitions as bars or boxes. A place is an input place to a transition if there exists a directed arc linking this place to the transition. A place is an output place of a transition if there exists a directed arc linking the transition to the place. (Zurawski & Zhou, 1994)

In its simplest form, a Petri net may be represented by a transition together with its input and output places. This basic net may be used to correspond to various aspects of the modeled systems. For example, input (output) places may represent reconditions (post conditions), the transition an event. Input places may stand for the availability of resources, the transition their utilization, output places the release of the resources. (Zurawski & Zhou, 1994)

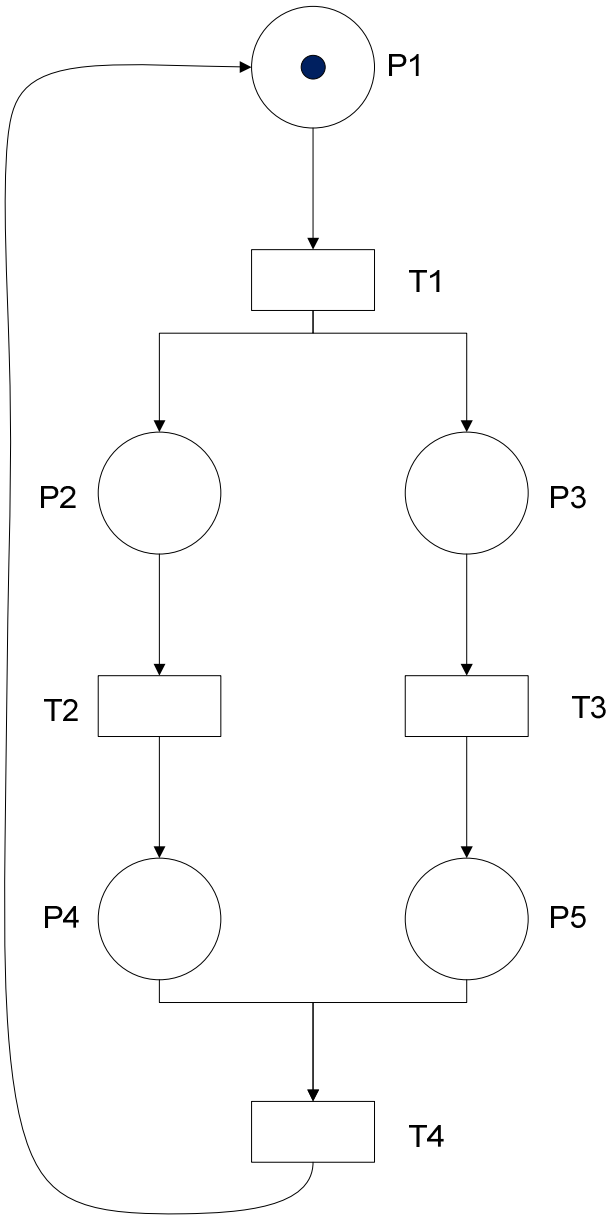

Figure 2.1: Basic Petri Net

The Petri net in Figure 2.1 consists of five places, represented by circles, four transitions, depicted by boxes, and directed arcs connecting places to transitions and transitions to places. In this net, place p1 is an input place of transition T1. Places P2, and P3 are output places of transition T1. (Zurawski & Zhou, 1994)

In order to study dynamic behavior of the modeled system, in terms of its states and their changes, each place may potentially hold either none or a positive number of tokens, pictured by small solid dots, as shown in Figure. (Zurawski & Zhou, 1994)

The presence or absence of a token in a place can indicate whether a condition associated with this place is true or false, for instance. For a place representing the

availability of resources, the number of tokens in this place indicates the number of available resources. At any given time instance, the distribution of tokens on places, called Petri net marking, defines the current state of the modeled system. (Zurawski & Zhou, 1994)

A marking of a Petri net with m places is represented by an (m x 1) vector M, elements of which denoted as M (p), are nonnegative integers representing the number of tokens in the corresponding places. (Zurawski & Zhou, 1994)

A Petri net containing tokens is called a marked Petri net. For example, in the Petri net model shown in Fig. 1, $M = (1,0,0,0,0)^T$.

Formally, a Petri net can be defined as follows:

$PN = (P,T,I,O,M_o)$; where

$P = \{p_1,p_2,..p_m\}$ is a finite set of places,

$T = \{ t_1 , t_2 \ldots,t_n\}$is a finite set of transitions $P \cup T \neq \emptyset$, and $P \cap T = \emptyset$,

I : (P x T) $\rightarrow$N is an input function that defines directed arcs from places to transitions, where N is a set of nonnegative integers,

O: (P x T) $\rightarrow$N is an output function which defines directed arcs from transitions to places, and

M_0 : P $\rightarrow$N is the initial marking.

If I(p, t) = k (O (p , t) = k). Then there exist k directed (parallel) arcs connecting place p to transition t. (transition t to place p). If I (p . t) = 0 (O (p : / ,) = U), then there exist no directed arcs connecting p to t (t to p) .

One can study dynamic behavior of the modeled system by changing distribution of tokens on places, which may reflect the rate of events or execution of operations. The following rules are used to govern the flow of tokens. (Zurawski & Zhou, 1994)

Enabling Rule: A transition t is said to be enabled if each input place p of t contains at least the number of tokens equal to the weight of the directed arc connecting p to t.

Firing Rule: An enabled transition f may or may not fire depending on the additional interpretation, and a firing of an enabled transition t removes from each input place p the number of tokens equal to the weight of the directed arc connecting p to t. It also deposits in each output place p the number of tokens equal to the weight of the directed arc connecting t to p.

The enabling and firing rules are illustrated in Figure 2.2 & Model 2.3.

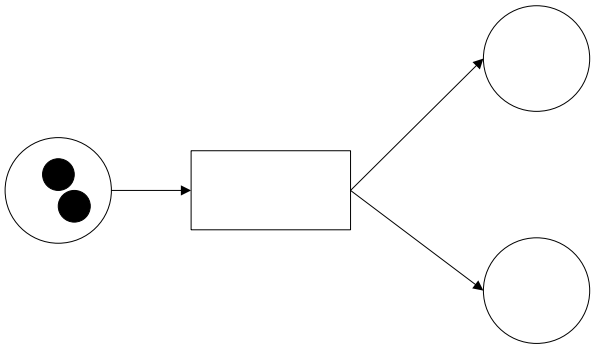

Figure 2.2: Transition

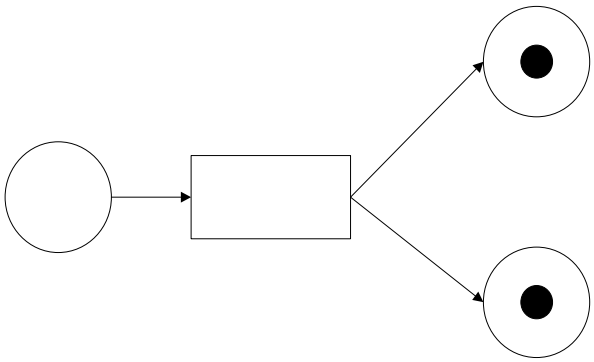

Figure 2.3 : Firing

A Petri net is said to be pure or self-loop free if no place is an input place to and output place of the same transition. A Petri net that contains self-loops can always be converted to a pure Petri net.

2.2.3. Properties of Petri Nets

Petri nets as mathematical tools possess a number of properties. These properties, when interpreted in the context of the modeled system, allow the system designer to identify the presence or absence of the application domain specific functional properties of the system under design.

Reachability :An important issue in designing distributed systems is whether a system can reach a specific state, or exhibit a particular functional behavior. In general, the question is whether the system modeled with Petri nets exhibits all desirable properties, as specified in the requirements specification, and no undesirable ones. (Zurawski & Zhou, 1994)

Boundedness and Safeness: Places are frequently used to represent information storage areas in communication and computer systems, product and tool storage areas in manufacturing systems, etc. It is important to be able to determine whether

proposed control strategies prevent from the overflows of these storage areas. The information storage areas can hold, without corruption, only a restricted number of pieces of data. The Petri net property which helps to identify in the modeled system the existence of overflows is the concept of boundedness. (Zurawski & Zhou, 1994)

Conservativeness: In real systems, the number of resources in use is typically restricted by the financial as well as other constraints. If tokens are used to represent resources, the number of which in a system is typically tixed, then the number of tokens in a Petri net model of this system should remain unchanged irrespective of the marking the net takes on. This follows from the fact that resources are neither created nor destroyed, unless there is a provision for this to happen. For instance, a broken tool may be removed from the manufacturing cell, thus reducing the number of tools available by one. (Zurawski & Zhou, 1994)

Liveness: The concept of liveness is closely related to the deadlock situation, which has been studied extensively in the context of operating systems. Four conditions must hold for a deadlock to occur (Zurawski & Zhou, 1994). These four conditions are:

Mutual exclusion: a resource is either available or allocated to a process which has an exclusive access to this resource.

Hold and wait: a process is allowed to hold a resource(s) while requesting more resources.

No preemption: a resource(s) allocated to a process cannot be removed from the process, until it is released by the process itself.

Circular wait: two or more processes are arranged in a chain in which each process waits for resources held by the process\s next in the chain.

Reversibility and Home State: An important issue in the operation of real systems, such as manufacturing systems, process control systems, etc., is the ability of these systems for an error recovery. These systems are required to return from the failure states to the preceding correct states. (Zurawski & Zhou, 1994)

2.2.4. Colored Petri Nets

Colored Petri Nets extend the classical Petri Nets with colors (to model data), time (to model durations), and hierarchy (to structure large models). Like in classical Petri Nets, CPNs use three basic concepts: transition, place, and token. (N. Mulyar & Aalst, 2005).

CPNs have an intuitive, graphical representation which is appealing to human beings. A CPN model consists of a set of modules (pages) which each contain a network of places, transitions and arcs. The modules act together through a set of well-defined interfaces, in a similar way as known from many modern programming languages. The graphical demonstration makes it easy to see the basic structure of a complex CPN model, i.e., understand how the individual processes work together. (Li, Chapa, Marin, & Cruz, 2004)

CPNs also have a formal, mathematical representation with a well-defined structure and semantics. This representation is the foundation for the definition of the diverse behavioral properties and the analysis methods. Without the mathematical representation it would have been absolutely impossible to develop a sound and powerful CPN language. On the other hand, for the convenient use of CPNs and their tools, it suffices to have an intuitive understanding of the syntax and semantics. This is similar to programming languages which are effectively applied by users who are not familiar with the formal, mathematical definitions of the languages. (Jensen, 2004)

CPN models can be prepared with or without unambiguous reference to time. Untimed CPN models are typically used to validate the functional/logical correctness of a system, while timed CPN models are used to calculate the performance of the system. There are many other languages which can be used to examine the functional/logical correctness of a system or the performance of it. Still, it is rather rarely to find modeling languages that are well-suited for both kinds of analysis. (Gehlot & Hayrapetyan, 2007)

CPNs can be simulated interactively or automatically. In an interactive simulation the user is in control. It is possible to see the effects of the individual steps directly on the graphical representation of the CP-net. This means that the user can investigate the different states and choose between the enabled transitions. An interactive simulation is similar to single-step debugging. It provides a way to "walk through" a CPN model, investigating different scenarios and checking whether the model works as expected. This is in contrast to many off-the-shelf simulation packages which often act as black boxes, where the user can define inputs and inspect the results, but otherwise have very little possibility to understand and validate the models on which the simulations build. It is our experience that the insight and detailed knowledge of a system, which the users gain during the development and validation of a simulation model, is often as important as the results

that the users get from the actual simulation runs. (Jensen, Special section on the practical use of high-level Petri nets, 2004)

Automatic simulations are similar to program executions. At present the purpose is to be able to execute the CPN models as fast and resourceful as possible, without detailed human interface and inspection. But the user still needs to interpret the simulation results. For this purpose it is often suitable to use animated, graphical representations providing an abstract, application-specific view of the current state and activities in the system. (Liu, 2005)

CPNs also present more formal verification methods, well-known as state space analysis and invariant analysis. In this way it is likely to prove, in the mathematical sense of the word, that a system has a certain set of behavioral properties. However, manufacturing systems are often so complex that it is impossible or at least very costly to make a full proof of system correctness. For this reason, the formal verification methods should be seen as a complement to the more informal validation by means of simulation. The use of formal verification is often restricted to the most important subsystems or the most important aspects of a complex system. (Jensen, Special section on coloured Petri nets, 2004)

2.3. Unified Modeling Language

The UML is considered to be the de facto standard for object-oriented modeling.

UML Models supports designer by letting work at a higher level of abstraction. A model may do this by hiding or masking details, bringing out the big picture, or by focusing on different aspects of the prototype. In UML, the user can zoom out from a detailed view of an application to the environment where it executes, visualizing connections to other applications or, zoomed even further, to other sites. Alternatively, user can focus on different aspects of the application, such as the business process that it automates, or a business rules view. (Kobryn, 2007)

Unified Modeling Language helps the user specify, visualize, and document business models of systems, including their structure and design, in a way that meets all of these requirements. Using any one of the large number of UML-based tools on the market, future application's requirements and design a solution that meets them can be analyzed, representing the results using UML 2.0's thirteen standard diagram types. (Kobryn, 2007)

UML is a natural fit for object-oriented languages and environments such as C++, Java, and the recent C#, but it can be used to model non Object oriented applications

as well in, for example, Fortran, VB, or COBOL. UML Profiles help to model Transactional, Real-time, and Fault-Tolerant systems in a natural way. (Kobryn, 2007)

In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML. (Martin, 2003) The UML classes will be described briefly in the following section.

2.3.1. The Meta-model

UML is unique in that it has a standard data representation. This representation is called the metamodel. The metamodel is a description of UMLin UML. It describes the objects, attributes, and relationships necessary to represent the concepts of UML within a software application. (Martin, 2003)

This provides CASE manufacturers with a standard and unambiguous way to represent UML models. Hopefully it will allow for easy transport of UML models between tools. It may also make it easier to write ancillary tools for browsing, summarizing, and modifying UML models. (Martin, 2003)

2.3.2. The Notation

The UML notation is rich and full bodied. It is comprised of two major subdivisions. There is a notation for modeling the static elements of a design such as classes, attributes, and relationships. There is also a notation for modeling the dynamic elements of a design such as objects, messages, and finite state machines.

In this study we have used some of the aspects of the static modeling notation. Static models are presented in diagrams called: Class Diagrams. (Martin, 2003)

2.3.3. Class Diagrams

The purpose of a class diagram is to depict the classes within a model. In an object oriented application, classes have attributes (member variables), operations (member functions) and relationships with other classes. The fundamental element of the class diagram is an icon the represents a class.

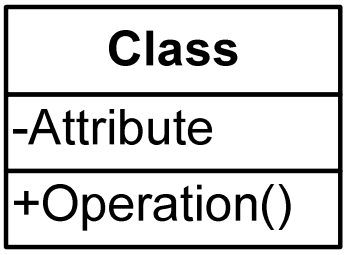

Figure 2.4:Meta Figure

A class icon which is shown in Figure 2.1 is simply a rectangle divided into three compartments . The topmost compartment contains the name of the class. The middle compartment contains a list of attributes (member variables), and the bottom compartment contains a list of operations (member functions). In many diagrams, the bottom two compartments are omitted. Even when they are present, they typically do not show every attribute and operations. The goal is to show only those attributes and operations that are useful for the particular diagram which can be seen in Figure 2.2. (Martin, 2003)

This ability to abbreviate an icon is one of the hallmarks of UML. Each diagram has a particular purpose. That purpose may be to highlight on particular part of the system, or it may be to illuminate the system in general. The class icons in such diagrams are abbreviated as necessary. There is typically never a need to show every attribute and operation of a class on any diagram. (Martin, 2003)

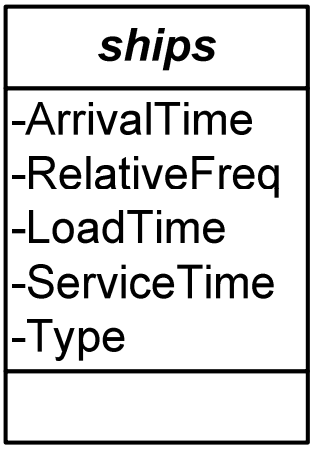

Figure 2.5: Class Diagram

2.3.4. Inheritance

The inheritance relationship in UML is depicted by a peculiar triangular arrowhead. This arrowhead, that looks rather like a slice of pizza, points to the base class. One or

more lines proceed from the base of the arrowhead connecting it to the derived classes. (Martin, 2003)

UML 2.3 shows the form of the inheritance relationship. In this diagram It can be seen that Circle and Square both derive from Shape . Note that the name of class Shape is shown in italics. This indicates that Shape is an abstract class. Note also that the operations, Draw() and Erase() are also shown in italics. This indicates that they are pure virtual. (Martin, 2003)

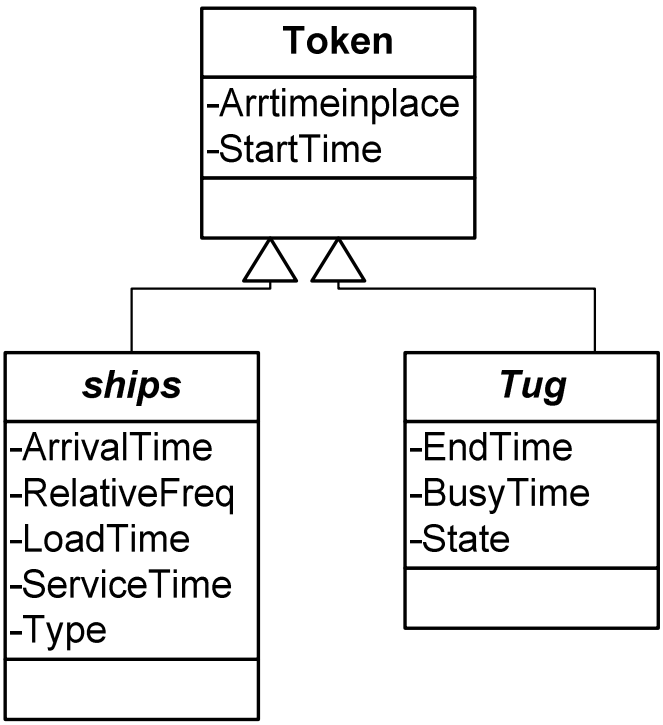

Figure 2.6: Inheritance

2.3.5. Aggregation / Association

The weak form of aggregation is denoted with an open diamond. This relationship denotes that the aggregate class which is the class with the white diamond touching it) is in some way the "whole", and the other class in the relationship is somehow "part" of that whole. (Martin, 2003)

2.3.6. Interfaces

Interfaces are such classes that have nothing but pure virtual functions. In Java such entities are not classes at all; they are a special language element called an interface. UML has followed the Java example and has created some special syntactic elements for such entities. (Martin, 2003)

3. CONTEXT

3.1. Steps for Creating a Decision Study

According to Goossenaerts & Pels (2005) a simulation based decision study consists of 10 steps. In this study, reuse is considered in the first 3 steps, with a focus on reuse of conceptual models. Those steps are as follows:

- **Step 1**.The Project Definition: The problem of the object system and the research questions of a decision study are formulated. In this study the problems are divided into basic structural units, and redefined in using the pattern of a consistent triplet of decision objects (objectives, performance indicators and action means).
- **Step 2**. Black Box & Assumptions: The system is generally represented as a box with indication of input, environmental and output parameters.
- **Step 3.** Conceptual Model: in this step the modeling of the object system (inside the black box) proceeds by using computer independent modeling techniques such as UML and Petri nets.

3.2. Evaluation

Criteria for evaluating patterns and their effect on the decision study will help a better comparison. The criteria for comparing methods with or without the use of patterns are (Goossenaerts & Pels, 2005):

- The cost aspect: Simulation enables to test every aspect of a proposed change or addition without committing resources to their acquisition. This is critical because once the hard decisions have been made, the bricks have been laid or the material-handling systems have been installed, changes and corrections can be extremely expensive. (Goossenaerts & Pels, 2005)
- The time aspect: Simulations are precisely repeatable; you can explore new policies, operating procedures, or methods without the expense and disruption of experimenting with the real system. . (Goossenaerts & Pels, 2005)

- The team and organization aspect: This aspect is focused on training the team, building consensus, appropriate use of the simulation, level of necessary communication and knowledge.
- The insight aspect: The necessity of simulation, plan visualization, ability to change, lack of exact results are the main points of this aspect. The functionality of the simulation models is investigated.
- The misguidance aspect: The art of simulation involves assessing what level of detail is required to support the project's goals. It's tough to do this right, though, because often you can't tell whether the detail is needed until you've developed it; and once the work is done, it's hard to justify removing it if it's unimportant. [Pels&Goossenaerts,2005] Problems with the data, level of detail are the main misguidance in simulation modeling.

4. RESEARCH

4.1. Selected Cases

In the study the cases that have been selected are based on the real life simulation problems which are explained in Simulation modeling and Analysis of Law & Kelton. The main reason for selecting these cases is that the processes mentioned in the book are mainly operation processes that include queuing situations in dynamic systems. The following criteria have been looked for the selection of the problems:

- The problem has to have complex operations which can be decomposed into simple operations.
- The resulting simple operations can be defined as general sub (or aspect) functions which can be used in other cases.

The following cases are selected according to these criteria:

1) Ship Transportation Case ((Law & Kelton, 2000), p.188, q2.23]: This case is mainly based on the modeling of an activity of a tug which carries ships between harbor and berths. The complexity of the case gives us the opportunity of creating many patterns.

2) Ship Loading Case (Law & Kelton (2000), p186, q2.19]: In this case the loading activity of ships in a harbor has been modeled. The cases are mostly related to the ship transportation case. It includes the multiplicative pattern.

3) Elevator Case[Law & Kelton (2000), p199, q2.35]: Elevator case is selected because it includes both inventory aspects and queuing aspects

4) Inventory Case: This case is selected because of the data structure and inventory control specializations. It also gives us the opportunity to directly use the patterns which are created in the previous works of van der Aalst(2005) (Law & Kelton (2000), p60].

4.2. Solving the cases

For the solution of the selected case the first three steps of simulation are applied to each problem. Problem description, Black box, UML model, and finally Petri net model is acquired.

4.3. Pattern Search

For determining the patterns the solution of the problem is redefined in a new way. The problem itself is divided in to basic components. Each component requires a definition, Petri net and the coding of the component. Then the step by step evolution of the system is made on Petri nets and the compatibility of the patterns is shown in the following ways:

4.4. Without Hierarchy

The case of acquiring the Petri nets via adding patterns without any hierarchy is the first option to create a model. This technique has been used in Patterns in Colored Petri Nets (Mulyar& Aalst) In this option the inputs and the outputs of the basic patterns are assemble together to create more complex pattern which can be used in the system.

4.4.1. With Hierarchy (additive)

The Second technique to acquire the Petri net is using building blocks method to create a complex model. The main difference between the previous option and this option is a single pattern is shown as an action which has the same input and output in the main Petri net instead of an addition Petri net. The attributes and the instances are reset according to that system.

4.4.2. With Hierarchy (Multiplicative)

The Third option is used in special cases of operation processes which have repeated actions of patterns. It is also shown in the hierarchy, but the function itself represents the loop actions. Therefore this view shows much more clear view of the model. The only flaw of the view is that the reversibility of the model is more difficult than the additive Hierarchical model.

5. SHIP TRANSPORTATION CASE

5.1. Case Description

A port in Africa loads tankers with crude oil for over water shipment and the port have facilities for loading as many as three tankers simultaneously. The tankers which arrive at the port every 11+- 7 hours are of three different types. (All times given as a+- ranges in this problem are distributed uniformly over the range.) The relative frequency of the various types and their loading time requirements are defined on table 5.1:

Table 5.1: Loading Times

Type	Relative Frequency	Loading Time, hours
1	0.25	18+- 2
2	0.25	24+- 4
3	0.50	36+- 4

There is one tug at the port. Tanker of all types requires the services of a tug to move from the harbor into a berth and later to move out of a berth into the harbor. When the tug is available, any berthing or deberthing activity takes 1 hour. It takes the tug 0.25 hour to travel from the harbor to the berths, or vice versa, when not pulling a tanker. When the tug finishes a berthing activity, it will deberth the first tanker in the deberthing queue if this queue is not empty. If the deberthing queue is empty but the harbor queue is not the tug will travel to the harbor and begin berthing the first tanker in the harbor queue (if both queues are empty, the tug will remain idle at the berths.) When the tug finishes a deberthing activity it will berth the first tanker in the harbor queue if this queue is not empty and a berth is available. Other wise the tug will travel to the berths and if the deberthing queue is not empty, will begin deberthing the first tanker in the queue. If the deberthing queue is empty the tug will remain idle at the berths.

The situation is further complicated by the fact that the area experiences frequent storms that last 4+-2 hours. The time between the end of one storm and the onset of the next is an exponential random variable with mean 48 hours. The tug will not start a new activity when a storm is in progress but will always finish an activity already in progress. (The berths will operate during a storm). If the tug is traveling from the berths to the harbor without a tanker when a storm begins, it will turn around and head for the berths. Run this simulation model for a 1 year period (8760 hours) and estimate:

a. The expected proportion of time the tug is idle, is traveling without a tanker, and is engaged in either a berthing or deberthing activity.
b. The expected proportion of time each berth is unoccupied, is occupied but not loading, and is loading.
c. The expected time average number of tankers in the deberthing queue and in the harbor queue.
d. The expected average in port residence time of each type of tanker.

5.2. Decision Description

The decision process can be simplified in 3 elements so that each element can be modeled separately:

Ships: There are ships coming to harbor and they will need to be taken in to the berth. After the berthing activity they need to be taken to the harbor again and sail to the open sea.

Tug Activity: There is one tug which carries all types of tankers from harbor into a berth and out of the berth into the harbor. There is certain logical rules apply to the tugs actions while carrying the tankers according to the situation on the berth queue and harbor queue.

Storm: Periodically storms occur. The storms affect the activity of the tug.

5.3. The Black Box representation

As it has been described in the section 3.1 The black box assumption is visualized for this case. The black box of the problem is shown in figure 5.1.
The input variables are listed as follows:

- Environmental Variables
    - Relative Frequency
    - Loading Times
    - Storm duration
    - Storm inter arrival time
    - Arrival Time
- Control Variables
    - Queuing policy
- Output Variables
    - The expected proportion of time of Tug
    - The expected proportion of time each berth
    - The expected time average number of tankers
    - The expected average in port residence time

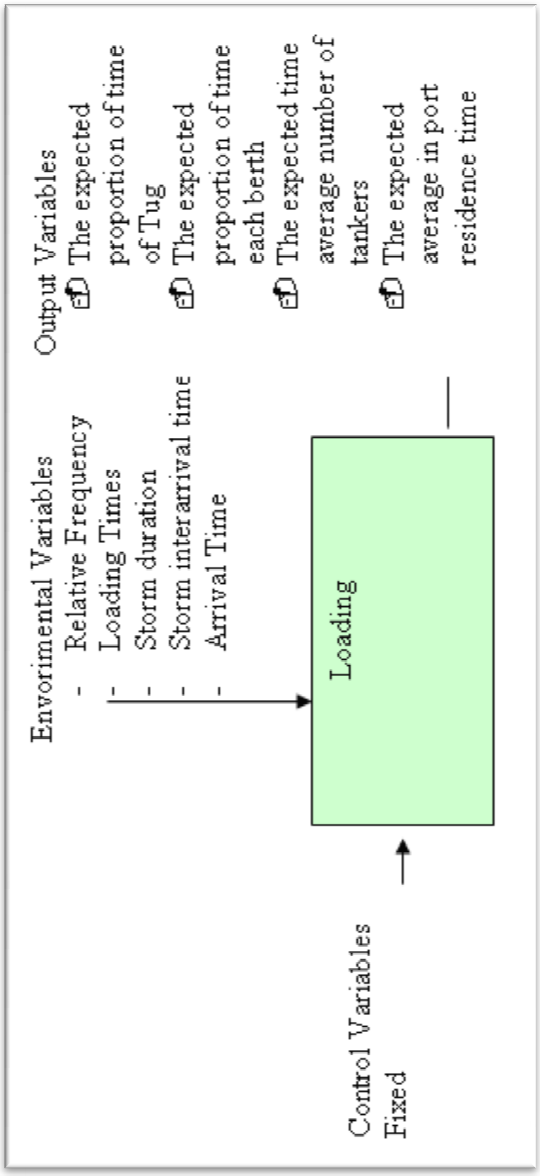

Figure 5.1: Black box Representation

5.4. UML Model

UML model of the problem is shown in Figure 5.1.

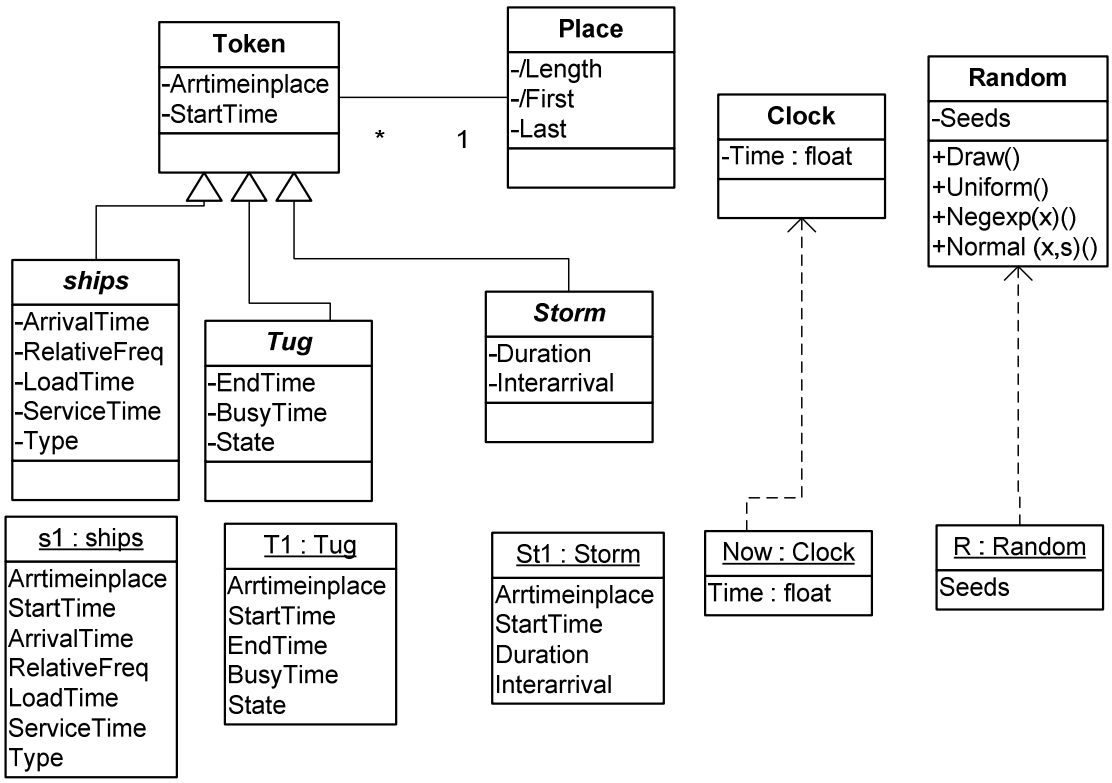

Figure 5.2: Ship Transportation Figure

The following rules have been considered for creating the UML models:

Tug Rules:

- When the tug finishes a berthing activity, it will deberth the first tanker in the deberthing queue if this queue is not empty.
- If the deberthing queue is empty but the harbor queue is not the tug will travel to the harbor and begin berthing the first tanker in the harbor queue (if both queues are empty, the tug will remain idle at the berths.)
- When the tug finishes a deberthing activity it will berth the first tanker in the harbor queue if this queue is not empty and a berth is available. Other wise the tug will travel to the berths and if the deberthing queue is not empty, will begin deberthing the first tanker in the queue.
- If the deberthing queue is empty the tug will remain idle at the berths.

Storm Rules:

- The tug will not start a new activity when a storm is in progress but will always finish an activity already in progress.

- If the tug is traveling from the berths to the harbor without a tanker when a storm begins, it will turn around and head for the berths.

5.5. Petri Net Patterns

5.5.1. Tug Activity Pattern

This pattern can be described as the logical phase of the tug movement. As mentioned in the problem description tanker of all types requires the services of a tug to move from the harbor into a berth and later to move out of a berth into the harbor. When the tug is available, any berthing or deberthing activity takes 1 hour. It takes the tug 0.25 hour to travel from the harbor to the berths, or vice versa, when not pulling a tanker. When the tug finishes a berthing activity, it will deberth the first tanker in the deberthing queue if this queue is not empty. If the deberthing queue is empty but the harbor queue is not the tug will travel to the harbor and begin berthing the first tanker in the harbor queue (if both queues are empty, the tug will remain idle at the berths.) When the tug finishes a deberthing activity it will berth the first tanker in the harbor queue if this queue is not empty and a berth is available. Other wise the tug will travel to the berths and if the deberthing queue is not empty, will begin deberthing the first tanker in the queue. If the deberthing queue is empty the tug will remain idle at the berths.

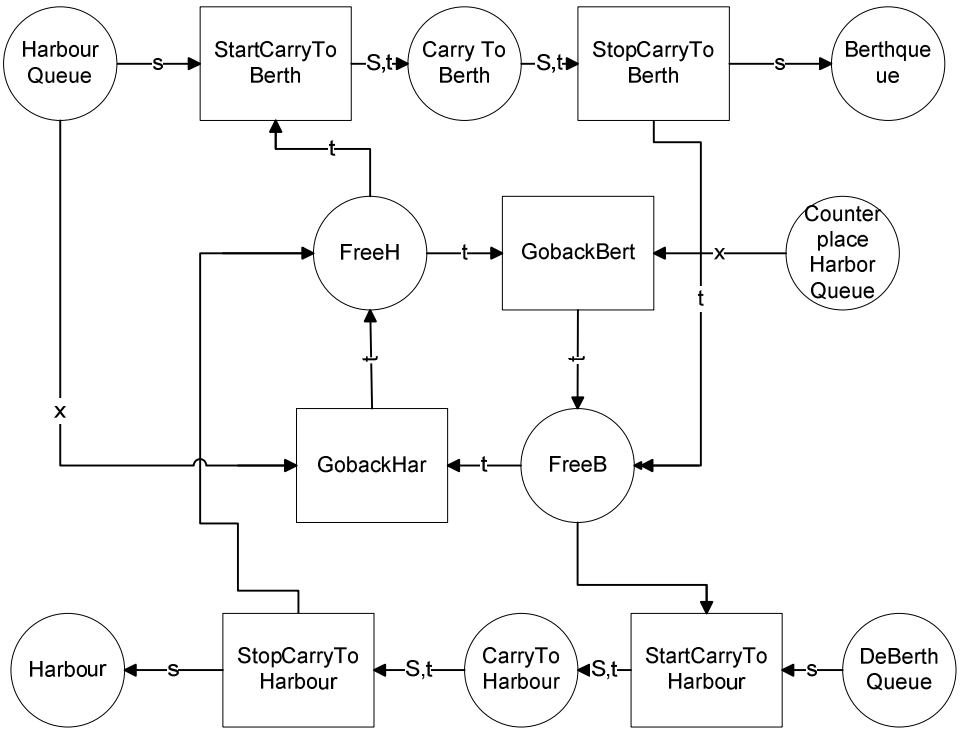

Figure 5.3: Tug Activity

The Petri net scheme which is shown in Figure 5.1 can be described as the actions of the Tug itself. The upper row and the lower row are mainly the transportation of the ship by the tug between Harbor and the Berth area.

Connectors:

Inputs Harbor Queue: S1

Deberthing queue: S1

Outputs

Berth queue: S1

Harbor: S1.

In the tug activity pattern model all the places are connectors. These connectors will be used for connecting the other patterns.

Codes for Tug Activity pattern

Start t1.place = FreeH

S1.place = HarborQueue

StartCarrytoBerth

Pre

S = Harborqueue.first

Post

CarrytoBerth <- (s,t)

t.endtime:= now.time + 60 (1 hour)

StopCarrytoBerth

Pre Now.Time = T.EndTime

Post deBerthQueue <- s

StartCarrytoHarbor

Pre

S = DeberthQueue.First

Post

CarrytoHarbor <- (s,t)

T.Endtime:= Now.Time + 60 (1 hour)

StopCarrytoBerth

Pre Now.Time = T.EndTime

Post harbor <- s

GobackBerth

Pre

T= FreeH

X= CounterPlaceHarborQueue

Post

FreeB<- t

GobackHar

Pre

T= FreeB

z= HarborQueue

Post

FreeH<- t

5.5.2. Storm Pattern

This pattern is expressing the effect of the storm. Storm affects the tug which is free at the harbor (Place shown as "FreeH") or Berth (FreeB) and the tug which carries a ship to harbor (Carry to Harbor). Frequent storms that last 4+-2 hours occur. The time between the end of one storm and the onset of the next is an exponential random variable with mean 48 hours. The tug will not start a new activity when a storm is in progress but will always finish an activity already in progress. (The berths will operate during a storm). If the tug is traveling from the berths to the harbor without a tanker when a storm begins, it will turn around and head for the berths. The pattern is showed in the figure 5.2.

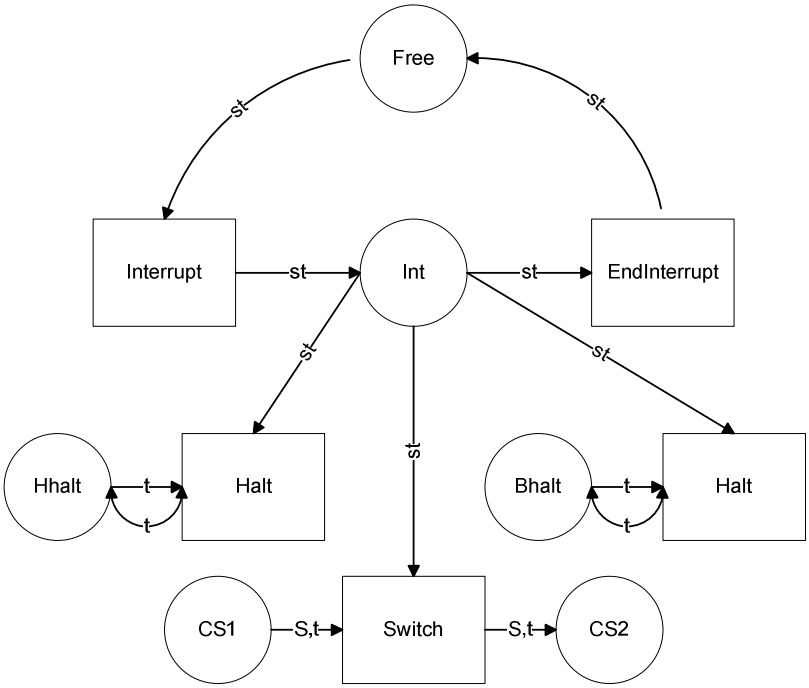

Figure 5.4: Storm

This Petri net pattern consists of 3 basic patterns which has general purpose such as Interrupt, Switch and Halt. These patterns may also be used in other models because of their purpose generality. If the Storm pattern is divided into general patterns the result will be the following.

Connectors:

Storm(Harbor)= harbor with FreeH = Hhalt

FreeB = Bhalt

Carrytoharbor = CS1

Carrytoberth = CS2

In Hhalt: T1

Bhalt: T1

CS1: S1,

Int: Storm.

Out Hhalt: T1

Bhalt: T1

CS2: ST1

Codes for the storm patter will be build up from the following patterns.

5.5.3. Halt Pattern

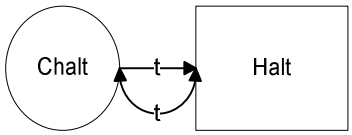

Figure 5.5: Halt

Main purpose of this pattern is to take out one or more tokens from the system for a temporary period to manipulate system flow in condition such that a disturbance affects the systems. In the Storm pattern this pattern is used to halt the processes of the tug when it's in the Free State. The presentation is shown in Figure 5.3.

Connectors:

In Chalt: T

Out Chalt: T

Halt(storm) = storm with CS1 = Chalt CS2 = Chalt

Code for Hatl Pattern:

Pre

T = State

Post

State<- t

Nowtime:= Nowtime+Halt.time

5.5.4. Switch Pattern

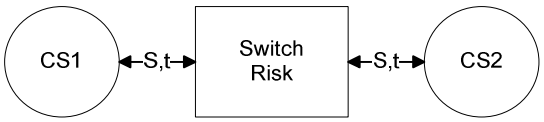

Figure 5.6: Switch

This pattern can be described as a simple switch for the tokens from one place to another in a time of disturbance. The pattern can be vary and can be evaluate further such that defining the instances of the places as a risk taker or wanted/ unwanted situation. After that, the pattern will work allover the system as an auto switch which will manipulate the system in necessary states. The pattern is shown in Figure 5.4.

Connectors:

In CS1: ST

CS2: ST

Out CS1: ST

CS2: ST

Switch(Harbor) = Harbor with CarrytoHarbor = CS1

CarrytoBerth = CS2

Codes for switch pattern

SwitchRisk

Pre

(S,t) = State1

Post

State2<- (s,t)

Now.time:= Now.time+service.time

5.5.5. Interrupt Pattern

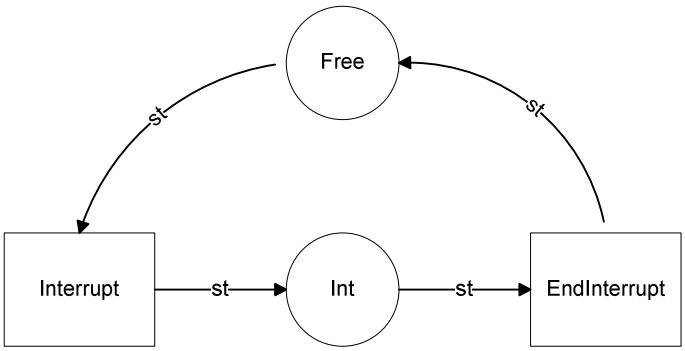

Figure 5.7: Interrupt

The pattern will work parallel to the designed system. It will create impulse for necessary states which can affect the Halt and switch patterns. The pattern can be represented as Interrupt(harbor) = harbor. Pattern is shown in Figure 5.5.

Codes for input pattern

Start

St.place= FreeSt

Interrupt

Pre: st.arrTime= now.time

Post storm<- st

State.Tug= pasive

St.arrtime: = now.time+Exp(48)

End.time:= now.time+ Unif(2.6)

EndInterrupt

Pre

Now.time=End.time

State.Tug= active

5.6. Model Creation

5.6.1. Adding structure without hierarchy

In this Structure switch pattern, interrupt pattern, halt pattern and the tug activity pattern are added together without hierarchy. The Structure system allows creating the system model without any pattern connection priority. The disadvantage of that structure is the chaotic side of viewing all system in once. The model is complex therefore it is hard to read and inefficient. Connectors are difficult to trace.

In the figure 5.6 complete detail of the model which includes the ship movement, tug movement, storm action, can be seen.

This view is not considered to be practical for reuse, and modification.

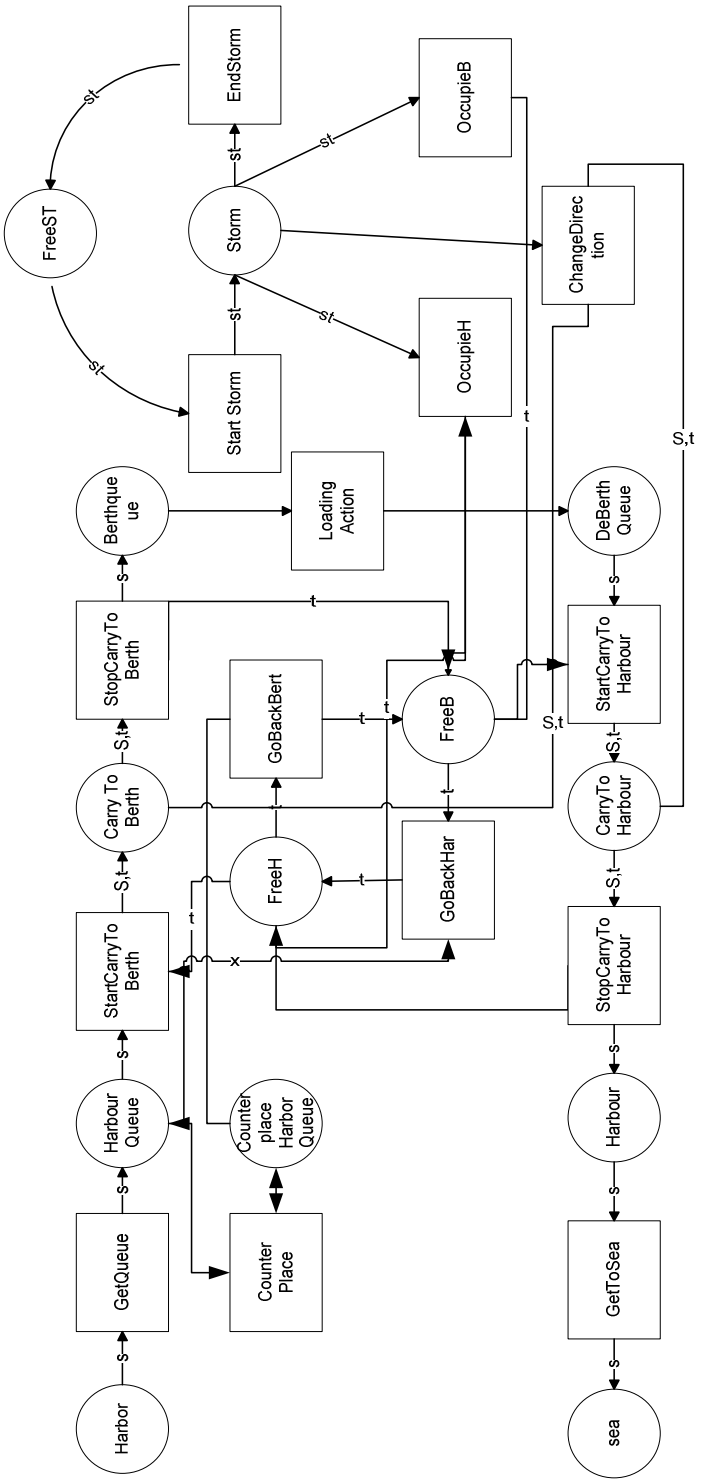

Figure 5.8: Figure without Hierarchy

5.6.2. Adding structure with hierarch

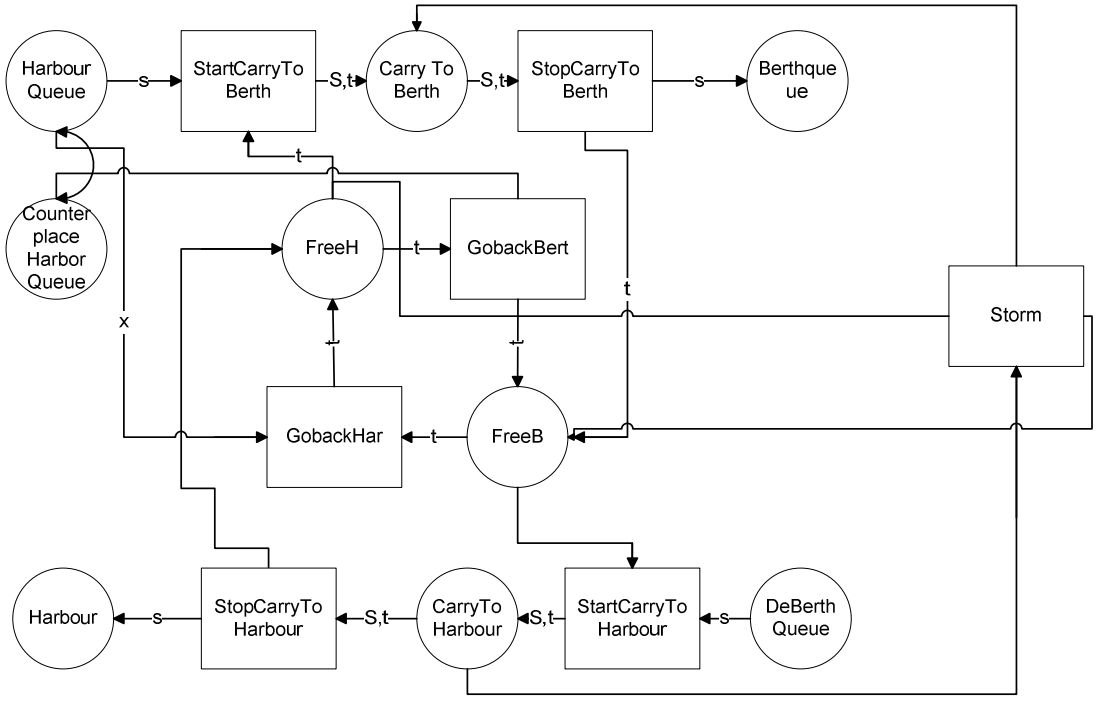

Figure 5.9: Combined Pattern of Tug and Storm

The structure is mainly based on adding one ore more patterns together and creating an upper pattern which can be added with other patterns and creates a hierarchy. In this example the storm pattern which is constructed from halt, interrupt and switch patterns is showed as a transition which is connected to the storm patterns input and output in the pattern of tug activity pattern. The patterns connection priority is necessary. The hierarch process is is shown step by step in in Figure 5.7 as first hierarch model, and model 5.8 as the second hierarch model.

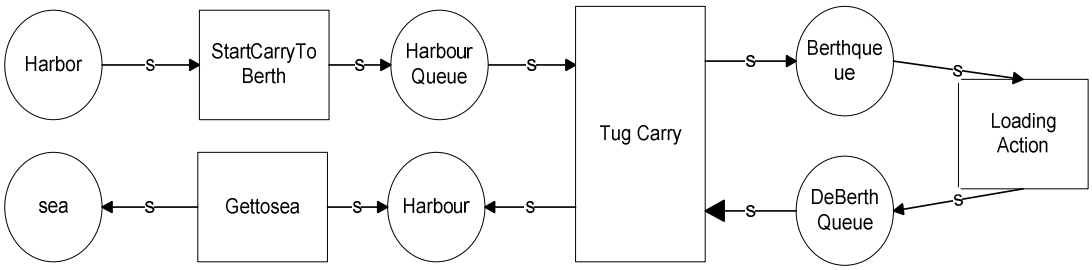

Figure 5.10: Top hierarch Figure

The pattern created by the unification of two patterns is shown as one transition in Figure 5.10. The whole system can be showed in a simple scheme since the connector structure.

6. SHIP LOADING CASE

6.1. Case Description

Ships arrive at a harbor with inter arrival times that IID exponential random variable with a mean of 1.25 days. The harbor has a dock with two berth and two cranes for unloading the ships; ship arriving when both berths are occupied join a FIFO queue. The time for one crane to unload ships distributed uniformly between 0.5 and 1.5 days. If only one ship is in the harbor, both cranes unload the ship and the (remaining) unloading time is cut in half. When two ships are in the harbor, one crane works for each ship. If both cranes are unloading one ship when a second ship arrives, one of the cranes immediately begins serving the second ship and the remaining service time of the first ship is doubled.

Assuming that no ships are in the harbor at time 0, run the simulation for 90 days and compute the minimum, maximum, and average time that ships are in the harbor (which includes their time in berth). Also estimate the expected utilization of each berth and the cranes. Use stream 1 for the inter arrival times and stream 2 for the unloading times.

6.2. Decision Description

There are two Cranes that should unload the ships which come to harbor. The cranes can work separately or together according to the rules:

- If there are no other ships in the queue the cranes will work together on a single ship.
- If the Cranes are working together and an other ship comes to harbor, the cranes will stop working together and one of them start to unload the new ship.

6.3. Black Box Representation

From the objective it is clear that Minimum, maximum, and average time that ships are in the harbor and the expected utilizations of each berth and the cranes has to be

determined. This is the output variables, which have to be delivered by the observer. In order to deliver this output variable the observer has to record some items within the simulated system.

The control variable is fixed. The dock allocation strategy and the queue

Discipline has been given in the problem. The environmental variables are the interarrival time, service time. Figure 6.1 presents the representation.

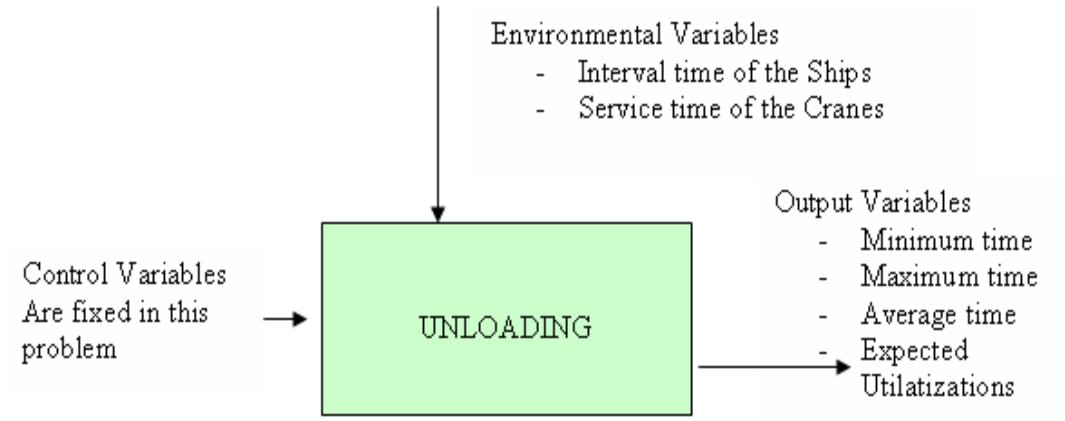

Figure 6.1: Ship Loading Blackbox

The input variables are:

- Environmental variables:
    - Interval time of the Ships
    - Service time of the Cranes
- Control variables:
    - Are fixed

The output variables are:

- Minimum time
- Maximum time
- Average time
- Expected Utilizations

6.4. UML model

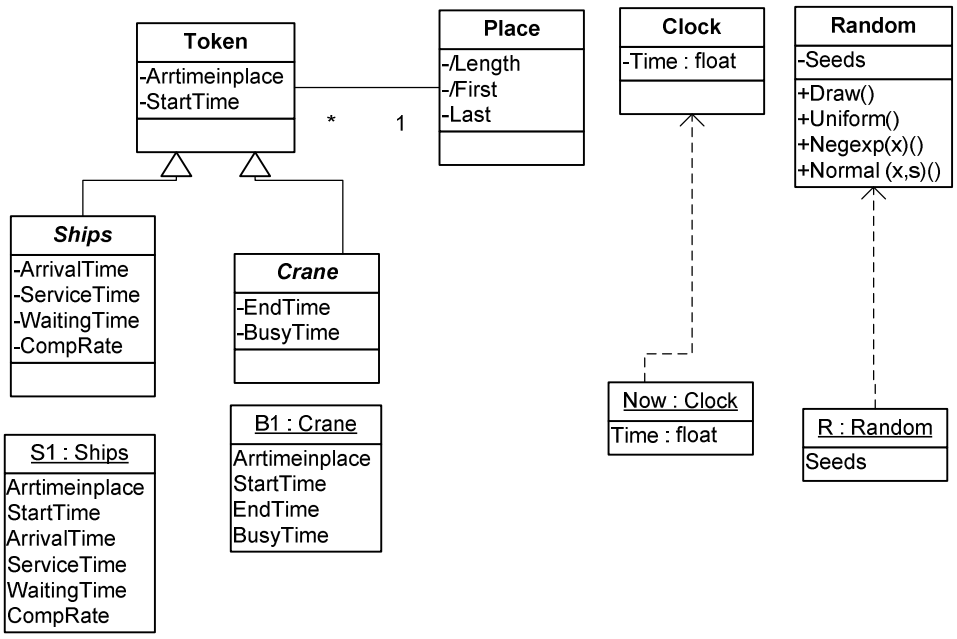

Figure 6.2: Ship Loading Figure

The classes and the inheritance is shown in Figure 6.1. The following rules are based on the UML meta model.

Crane rules:

- If only one ship is in the harbor, both cranes unload the ship and the (remaining) unloading time is cut in half.
- When two ships are in the harbor, one crane works for each ship.
- If both cranes are unloading one ship when a second ship arrives, one of the cranes immediately begins serving the second ship and the remaining service time of the first ship is doubled.

6.5. Petri Net patterns

6.5.1. Source Increase Pattern

Described in the question “If only one ship is in the harbor, both cranes unload the ship and the (remaining) unloading time is cut in half.” The action of using the sources on one Customer/Ship, or in other words increase the sources that are used on one customer may be patterned in the figure 6.1.

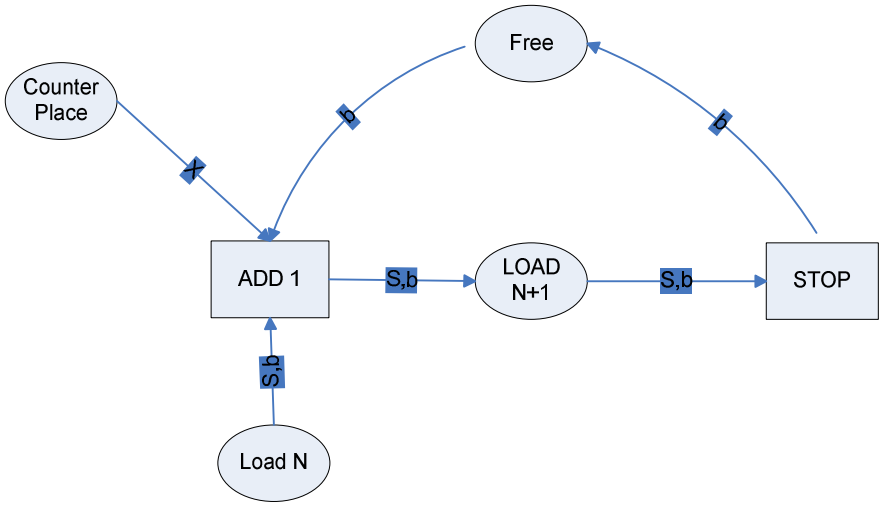

Figure 6.3: Source Increase basic pattern

The pattern can be evaluated further as a hierarchy by adding add and stop transition together. Since the number of the working sources is an instance of the token, place that describes the loading of ship with N sources (LoadN) and N+1 sources (LoadN+1) can be showed in the same place. The evaluation is as model 6.2.

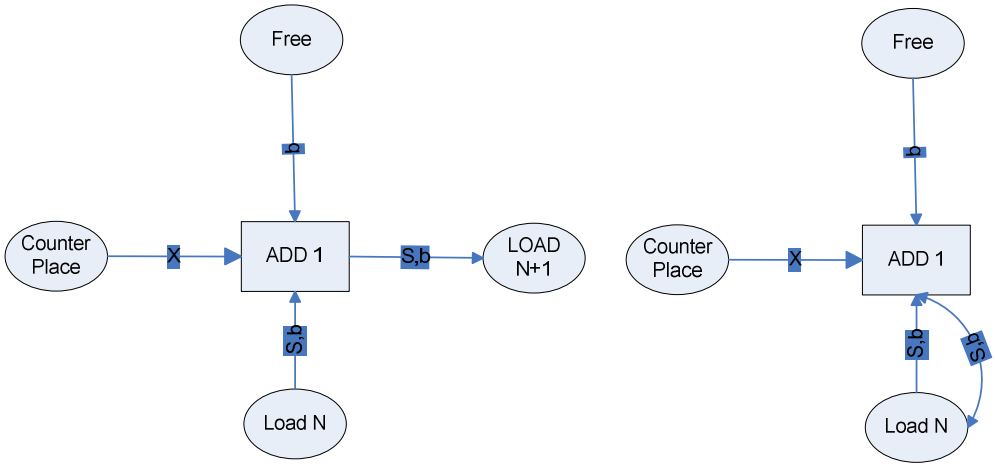

Figure 6.4: Source Increase complex pattern

Codes:

Start

s.Place = LoadN

∧ b1.Place = Free;

ADD1

Pre x= CounterPlace

Post LoadN+1 <- S,b

s.XCompRate: = s.CompRate

s.BusySource: = s.busySource+1

^^s.CompRate := s.busysource* (NowTime - NewarrTime)/ Uniform(0.5, 1.5) + s.XCompRate

If (s.CompRate <1) Then S.ServiceTime:= S.ServiceTime + (NowTime - NewarrTime)

Else S.ServiceTime:= (1- s.XCompRate)*Uniform(0.5, 1.5) + S.serviceTime

Connectors:

In CounterPlace: X

LoadN: SB1

Free: B1

Out LoadN+1: SB1

ADD1(Loading) = Loading with Loading = LoadN

Loading = LoadN+1

Free = Free

6.5.2. Source Decrease Pattern

Described in the question “If both cranes are unloading one ship when a second ship arrives, one of the cranes immediately begins serving the second ship and the remaining service time of the first ship is doubled.” The action of using the switching the sources from one customer to two customer, or in other words decrease the sources that are used on one customer may be patterned in the figure 6.3.

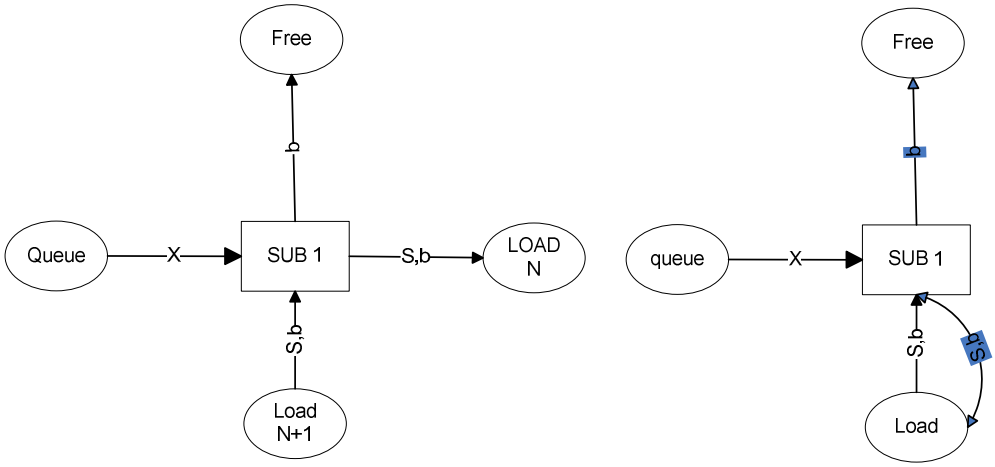

Figure 6.5: Source Decrease complex pattern

Code:

Start

s.Place = LoadN+1

∧ b.Place = LoadN+1;

Sub1

Pre x= Queue

Post LoadN <- S,b

s.XCompRate: = s.CompRate

s.BusySource: = s.BusySource-1

^^s.CompRate := s.busysource* (NowTime - NewarrTime)/ Uniform(0.5, 1.5) + s.XCompRate

If (s.CompRate <1) Then S.ServiceTime:= S.ServiceTime + (NowTime - NewarrTime)

Else S.ServiceTime:= (1- s.XCompRate)*Uniform(0.5, 1.5) + S.serviceTime

Connectors:

In Queue: X

LoadN+1: SB1

Out Free: B1

LoadN+1: SB1

ADD1(Loading) = Loading with Loading = LoadN

Loading = LoadN+1

Free = Free

6.6. Model Creation

6.6.1. Adding structure without hierarch

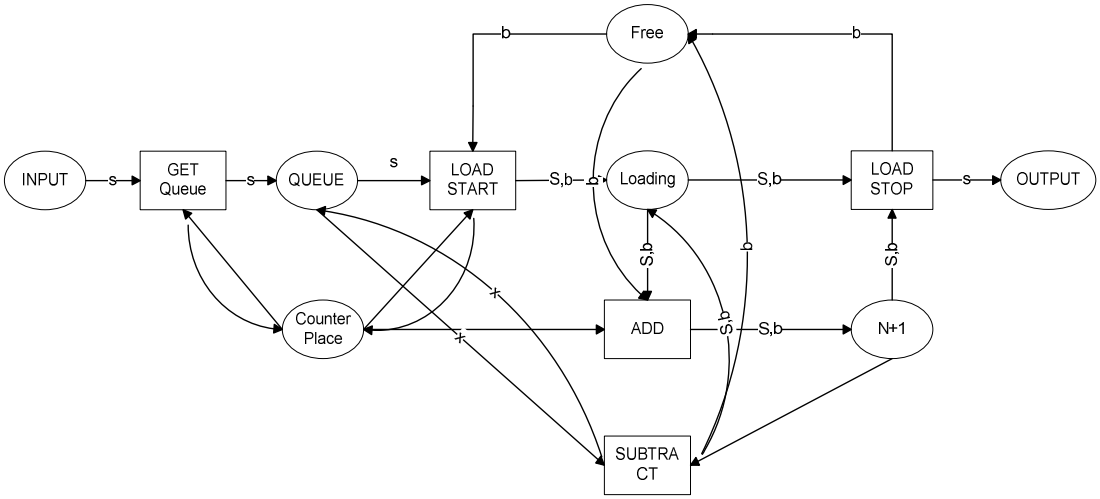

Figure 6.6: Ship Loading Figure without Hierarch

In this structure all the transitions can be seen as individually in the figure 6.4. Due to the simplicity of the model this type of structure may be possible to model.

6.6.2. Adding structure with hierarch

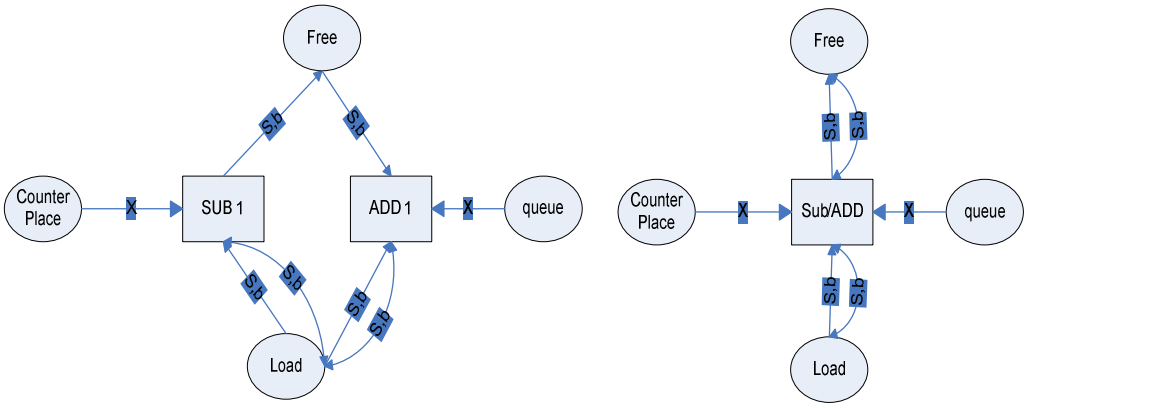

Figure 6.7: Combined pattern of Source increase and decrease

The first step of the hierarchy is to use the source decrease pattern and source increase pattern as a sub net of the transition called Sub/Add. (Model 6.5) Another pattern is also included to the system which is called the Counter place pattern. (Mulyar & Aalst, 2005) It is necessary for the input of the Sub/Add pattern and the main purpose of this pattern is to have a token in the counter place when there is no token in the queue place. Since the input and the output of the patterns are the similar, the structural effect of the Sub/Add pattern is as model 6.6:

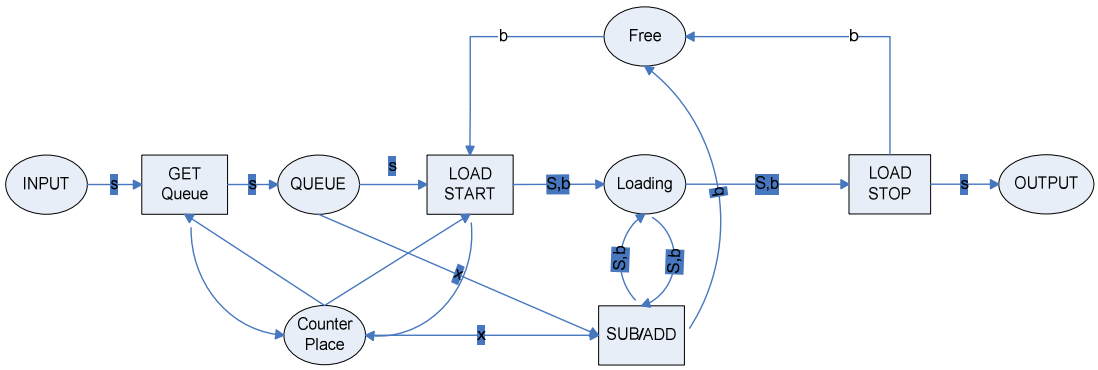

Figure 6.8: Top Hierarch Figure for Ship Loading

6.6.3. Adding structure with hierarch (Multiplicative)

In this structure The Sub/Add pattern mentioned above is not combined with the model via adding but integrating with Load Stop transition on the system. After the integration the model is as model 6.7:

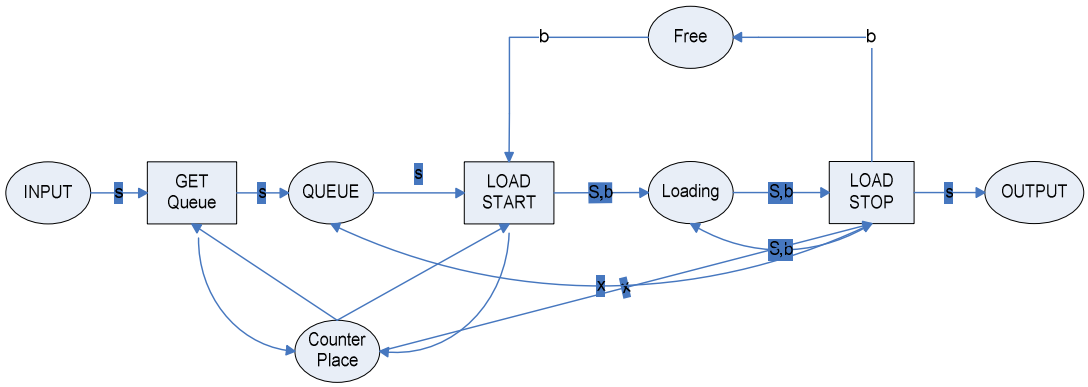

Figure 6.9: Multiplicative Figure of ship loading

7. ELEVATOR CASE

Elevator traffic calculation is a critical issue in the elevator system projects. For that purpose the given situation shall be analyzed, i.e. it may turn out as necessary to elaborate the utilization concept as a first step. Since the expected traffic in the building is complex, a simulation shall give more information whether the calculated group will meet the requirements in any traffic situation. It will then be defined which area of the building shall be utilized by calculated elevators. (Lohri, 2000)

The entire flow of traffic must perhaps be handled by more than one elevator group (e.g. hospital: division into bed and passenger transport, high building: division into several zones). Further the need for special service or goods elevators must be clarified and if these might be required for passenger transport as well, and also whether the traffic should rather be handled by escalators, etc.. In this projecting stage we recommend to represent the capacity concept graphically. (Lohri, Traffic Simulation, 2000)

All large elevator manufacturers are in the position to perform traffic calculations. Normally the results do not differ much - under equal marginal conditions. Deviating variants are presented rather for other reasons, e.g. because a bidder cannot offer a particular configuration (suitable machine for the demanded speed not available, or no suitable control for the size of group). The quality in the capacity of elevators from different manufacturers will not show up before the control can prove its effect. Respective investigations must be performed by means of a simulation. (Lohri, Traffic Simulation, 2000)

7.1. Case Description

A five- story office building is served by a single elevator. People arrive to the ground floor (floor 1) with independent exponential interarrival times having mean 1 minute. A person will go to each of the upper floors with probability 0.25. It takes the elevator 15 seconds to travel one floor. Assume, however, that there is no loading or unloading time for the elevator at a particular floor. A person will stay on a particular floor for an amount of time that is distributed uniformly between 15 and 120 minutes. When a person leaves floor i (where i=2,3,4,5), he or she will go to floor 1 with probability 0.7, and will go to each of the other three floors with

probability 0.1. The elevator can carry six people, and starts on floor 1. If there is not room to get all people waiting at a particular floor on the arriving elevator, the excess remain in queue. A person coming down to floor 1 departs from the building immediately. The following control logic also applies to the elevator:

- When the elevator is going up, it will continue in that direction if a current passenger wants to go to a higher floor or if a person on a higher floor wants to get on the elevator.
- When the elevator is going down it will continue in that direction if it has at least one passenger or if there is a waiting passenger at a lower floor.
- If the elevator is at floor i (where i= 2,3,4) and going up (down), then it will not immediately pick up a person who wants to go down (up) at that floor.
- When the elevator is idle, its home base is floor 1
- The elevator decides at each floor when floor it will go to next. It will not change directions between floors.

7.2. Decision Description

Elevator: There is an elevator which carries people between five floors. People enter to the system from the first floor and leave the elevator system to the first floor.

Waiting time: Each passenger has his personal waiting time. This is the time that a passenger spends waiting for an elevator measured from the instant when the passenger has arrived (and possibly registers a landing call) until the instant the passenger enters the car. The totality of the personal waiting time during a simulation can be evaluated according to several statistical criteria. (Lohri, Traffic Simulation, 2000)

Some examples: mean waiting time or the waiting time limit for 90 % of all passengers or the maximum waiting time. The evaluation for the mentioned criteria can be selected, e.g. according to the following sub-criteria:

- for all passengers in the simulation
- for all passengers on a selected stop or a selected range of stops
- for all passengers which are traveling in a selected direction
- for all passengers within a selected time period

• Transit time: This is the time that a passenger spends traveling in a elevator car, measured from the instant that the passenger boards the car until the instant that the passenger alights at the destination floor. (Lohri, 2000)

Time to destination (journey time): Each passenger has his personal time to destination. The time of destination results in an addition of the waiting time with the transit time. The evaluation of the time to destination can be carried out analogously how remarked by waiting time. (Lohri, Traffic Simulation, 2000)

• Waiting queue: Persons who are waiting for an elevator serving them.

There are three criteria that effect the elevators movement:

- The people inside the elevator
- The people waiting for the elevator in the floors (Call button)
- The direction of movement of the elevator

7.3. Black Box Representation

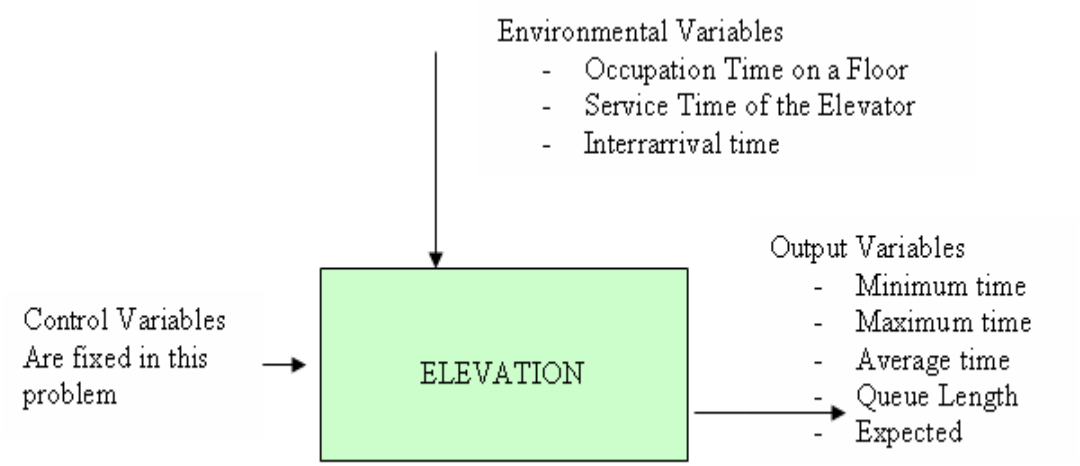

Figure 7.1: Black box of Elevatation

The black box representation on figure 7.1, describes the control variables, environmental variables, and output variables.

7.4. UML model

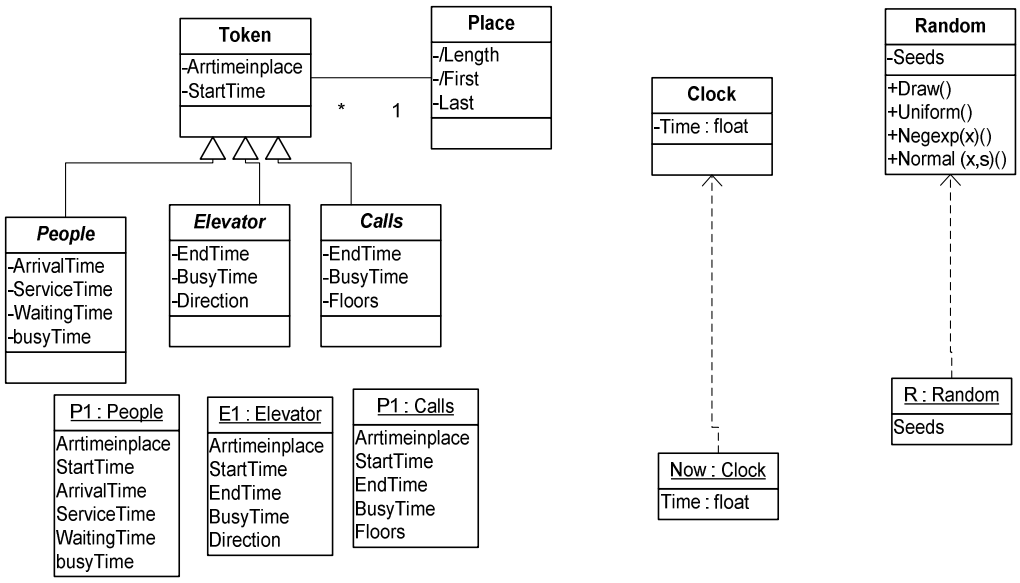

Figure 7.2: Elevator Figure

UML meta model is presented in Figure 7.1. The rules are based on the UML meta model datas.

- When the elevator is going up, it will continue in that direction if a current passenger wants to go to a higher floor or if a person on a higher floor wants to get on the elevator.
- When the elevator is going down it will continue in that direction if it has at least one passenger or if there is a waiting passenger at a lower floor.
- If the elevator is at floor i (where i= 2,3,4) and going up (down), then it will not immediately pick up a person who wants to go down (up) at that floor.
- When the elevator is idle, its home base is floor 1
- The elevator decides at each floor when floor it will go to next. It will not change directions between floors.

7.5. Petri Net Patterns

7.5.1. Call Pattern

In the system there are two main factors that effect the direction of the elevator: The people in the elevator and the people who call the elevator. In this pattern the relation

between the people waiting in one floor for the elevator and the call button has been described. This pattern is based on the Data Distributor pattern of Aalst(2005). This pattern modifies the attributes of the entity “Calls” and shown in Figure 7.1.

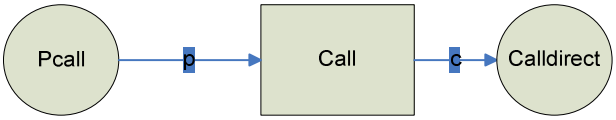

Figure 7.3: Call

Code:

Start

p=Pcall

Call

Pre c=Pcall.first

Post

Callup<- c

Fi

Connectors:

In Pcall: p

Out Calldirect:c

Call (elevator) = Elevator with Callqueuup = CallDirect

Callqueuedown = CallDirect

UpFloorNqueue = Pcall

DownFloorNqueue = Pcall

7.5.2. Ascending/Descending Transportation Pattern

In this pattern the first two motives of the elevator that has been described in the first two logic paragraphs of the problem. This pattern enables the elevator to move in the same direction (Up/Down) in the possible state. The pattern also includes the sub pattern of Capacity bounding pattern in patterns in colored Petri nets (Aalst, 2005). The logic consists on comparing the people getting in to the elevator and elevator call list, then selecting the nearest floor in the same direction. Paterns are presented in Figure 7.2

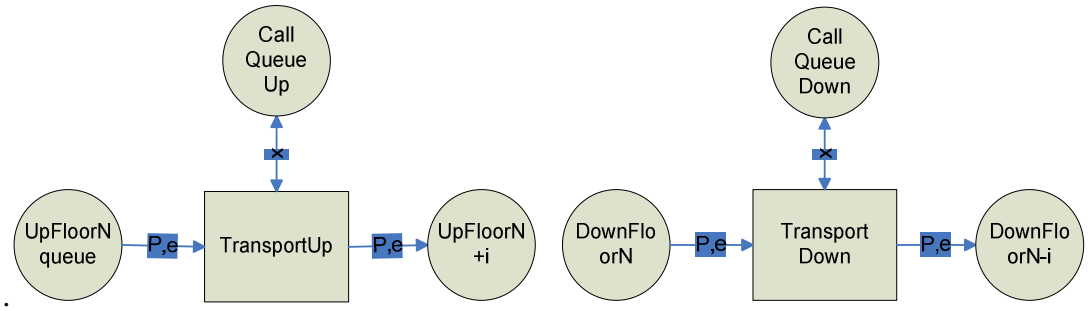

Figure 7.4: Trasport up/ Trasport down

Code:

TransportUp

Pre n+I= First(UpfloorNqueue.first, CallqueueUp)

Post

UpFloorN+i<-P,e

The coding of the Descending transport pattern is similar to the ascending transport pattern.

Connector:

Ascending (elevator) = elevator with

UpfloorN = Upfloorqueue

ArrvUpN+1 = UpfloorN+1

7.5.3. Occupy Pattern

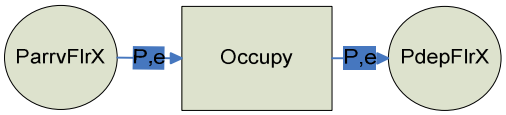

Figure 7.5: Occupy

The code of the Occupy transition is similar to the Halt pattern that has been mentioned in the ship transportation problem. The transition which is shown in Figure 7.3, similarly holds the token transfer for a period of time.

Connector:

Occupy (elevator) = elevator with ArrvUpN+1 = ParrFlrX

DepUp N+1 = PdepFlrX

7.5.4. Switch Pattern

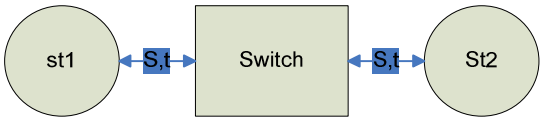

Figure 7.6: Switch

In this problem the switch pattern also used for determining the direction of the elevator. The impulse is triggered if there are no calls or people going in the same direction. The counter Place pattern (Mulyar & Aalst, 2005)is also necessary for this pattern for expressing the empty places for call list and floors. This can be also represented by aggregate objects. Switch pattern is presented in Figure 7.4.

Connector:

Switch(elevator) =elevator with UpfloorN = St1

DownFloorN = St2

Code:

Switch

If (Direction.elevator=up && up.Floors.calls=[])

Then Direction.elevator=down.

Else If (Direction.elevator=down && down.Floors.calls=[])

Then Direction.elevator=up

If(Floors.calls=[])

Then Floor.elevator=1

Else

Fi

7.6. Model Creation

7.6.1. Adding structure with hierarch

This structure in Figure 7.5 shows the relationship between the patterns. The switch pattern is the link between transportation patterns which will eventually change the direction of the elevator if there is no one waiting in that direction floors or in the elevator. The main aim of this pattern is to model the elevators actions which is represented as elevate in the figure 7.6.

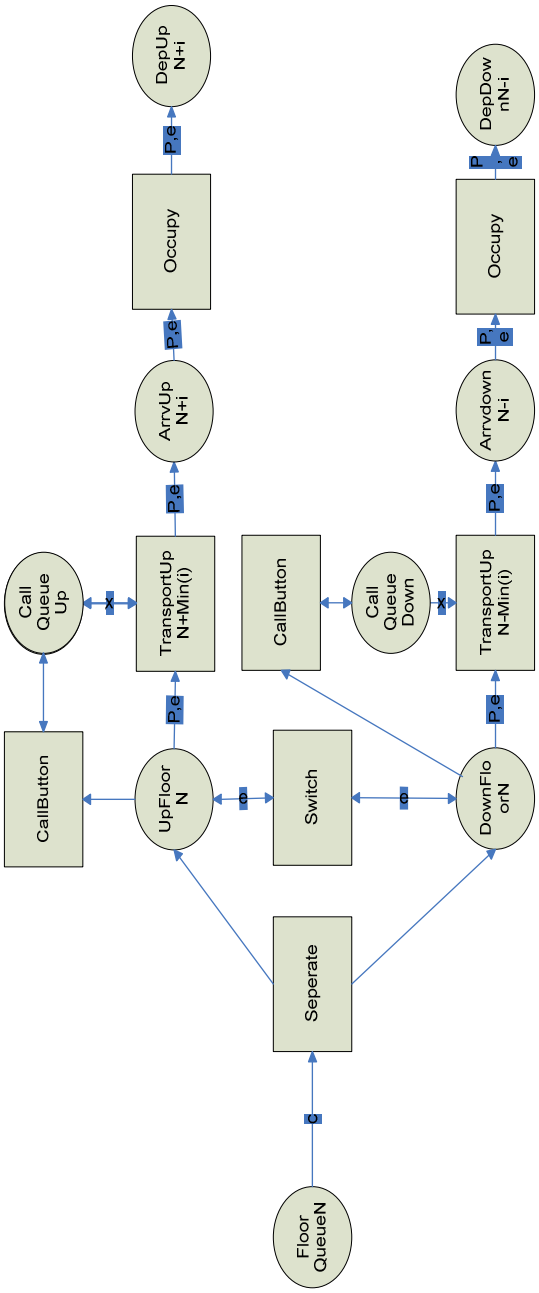

Figure 7.7: Elevation logic

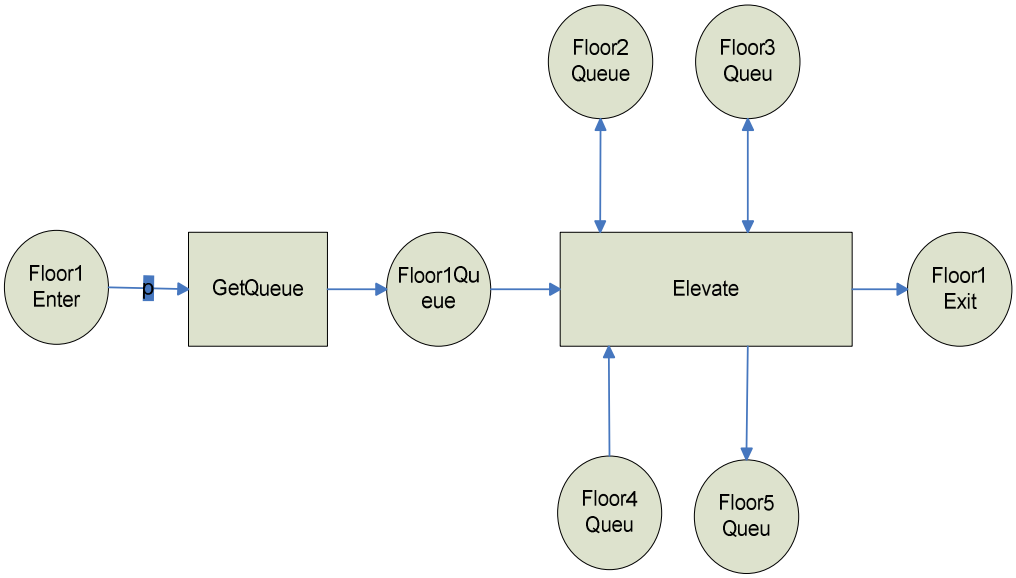

Figure 7.8: Top Elevator Figure

8. INVENTOR SYSTEM CASE

8.1. Case Description

A company that sells a single product would like to decide how many items it should have in inventory for each of the next n months (n is a fixed input parameter). The times between demands are IDD exponential random variables with a mean of 0.1 month. The sizes of the demands, D, are IID random variables (independent of when the demands occur), with

D= {1 w.p. 1/6,

2 w.p. 1/3

3 w.p. 1/3

4 w.p. 1/6}

At the beginning of each month, the company reviews the inventory level and decides how many items to order from its supplier. If the company orders Z items, it incurs a cost of K+iZ, where K=$32 is the setup cost and i=$3 is the incremental cost per item ordered. (If Z=0, no cost is incurred.) When an order is placed the time required for it to arrive (called the delivery lag or lead time) is a random variable that is distributed uniformly between 0.5 and 1 month.

The company uses a stationary (s,S) policy to decide how much to order, i.e.,

Z= { S- I if I<s

0 if I>s}

Where I is the inventory level at the beginning of the month.

When a demand occurs, it is satisfied immediately if the inventory level is at least as large as the demand. If the demand exceeds the inventory level, the excess of demand over supply is backlogged and satisfied by future deliveries. (In this case, the new inventory level is equal to the old inventory level minus the demand size, resulting in a negative inventory level.) When an order arrives, it is first used to eliminate as much of the backlog (if any) as possible; the remainder of the order (if any) is added to the inventory.

So far we have discussed only one type of cost incurred by the inventory system, the ordering cost. However, most real inventory systems also have two additional types of costs, holding and shortage costs, which we discuss after introducing some additional notation. Let I(t) be the inventory level at time t [note that I(t) could be positive, negative or zero]; let $I^+(t)=\max\{I(t),0\}$ be the number of items physically on hand in the inventory at time t [note that $I^+(t)>=0$]; and let $I^-(t)=\max\{-\{I(t),0\}$ be the backlog at time t [$I^-(t)>=0$ as well]. A possible realization of I(t), $I^+(t)$ and $I^-(t)$ is shown in fig 1.41. The time points at which I(t) decreases are the ones at which demands occur.

For out model, we shall assume that the company incurs a holding cost of h=$1 per item per month held in (positive) inventory. The holding cost includes such costs as warehouse rental, insurance taxes, and maintenance, as well as the opportunity cost of having capital tied up in inventory rather than invested elsewhere.

Similarly, suppose that the company incurs a backlog cost of π=$5 per item per month in backlog; this accounts for the cost of extra record keeping when a backlog exists, as well as loss of customers' goodwill.

Assume that the initial inventory level is I(0)=60 and that no order is outstanding. We simulate the inventory system for n = 120 months and use the average total cost per month (which is the sum of average ordering cost per month, the average holding cost per month, and average shortage cost per month) to compare the following nine inventory policies in table 8.1:

Table 8.1: Inventor Policy

s	20	20	20	20	40	40	40	60	60
S	40	60	80	100	60	80	100	80	100

8.2. Decision Description

A company which has monthly demands is having an inventory control and wants to know which inventory level limits are the most profitable.

- Inventory control system is based on the maximum and minimum inventory levels.

- Every demand affects the inventory directly.

- To increase the inventory level orders should be made to the supplier at the beginning of each month.

- Inventory level effects the cost such as holding cost and backlog cost.

8.3. The Black Box representation

The Black box representation for the inventory case is shown in figure 8.1.

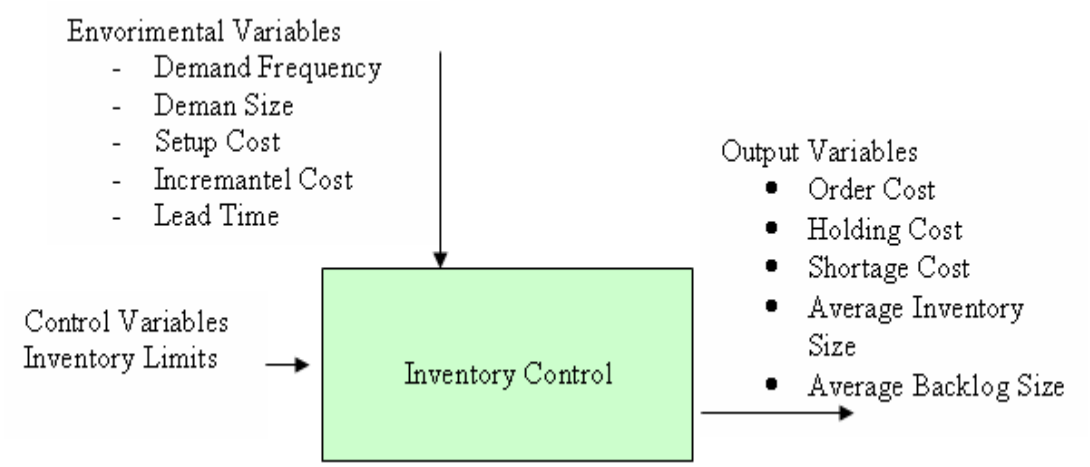

Figure 8.1: Black box of Inventory system

8.4. UML Model

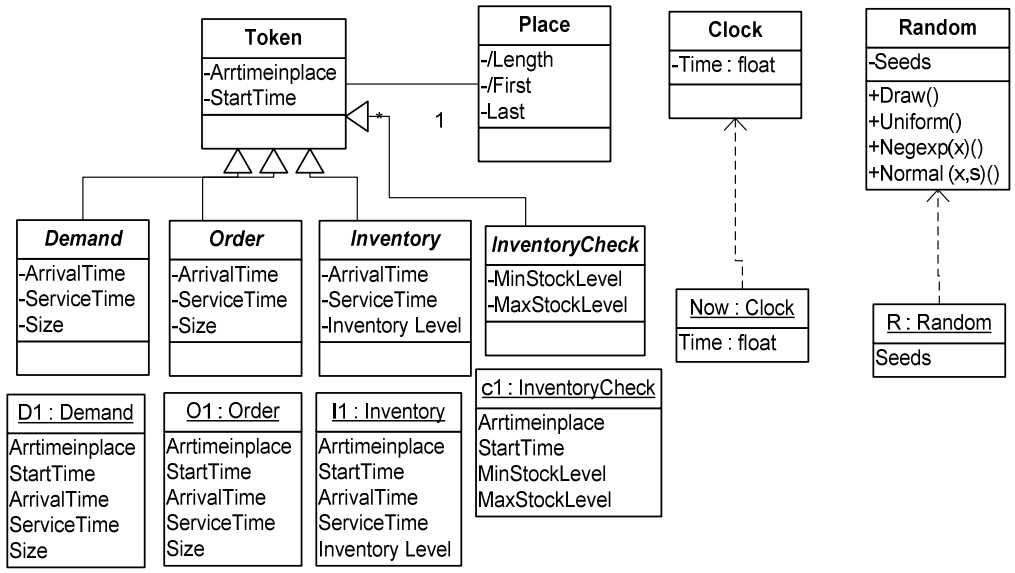

Figure 8.2: Inventory System Figure

8.5. Petri Net Patterns

8.5.1. Delivery Pattern

When a demand occurs, it is satisfied immediately if the inventory level is at least as large as the demand. If the demand exceeds the inventory level, the excess of

demand over supply is backlogged and satisfied by future deliveries. The input parameters are the current inventory level and the size of the demand. The demand will be serviced at the end of the transaction which is shown in Figure 8.1. This pattern is similar to the Inventory set pattern.

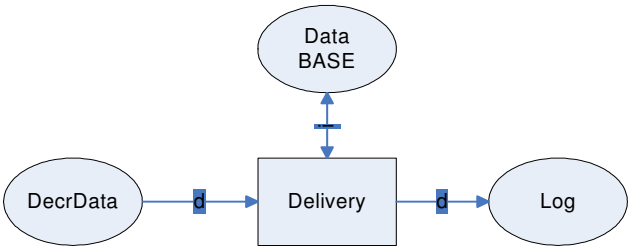

Figure 8.3: Delivery Pattern

Code:

Delivery

Post

Serviced<- D

i.inventorylevel := i.inventorylevel- d.size

Connector:

Delivery(Inventory)= Inventory with Demand = DecrData

InventoryDB = Database

8.5.2. Inventory Control Pattern

Description: "The company uses a stationary (s,S) policy to decide how much to order, i.e.,

Z= { S- I if I<s

0 if I>s}

where i is the inventory level at the beginning of the month." The input parameters are the level of the inventory and inventory control limits. The pattern decides on ordering process and the order size (o.size) according to the inventory level. The pattern which is shown in Figure 8.2, is based on Database management pattern of Aalst(2005).

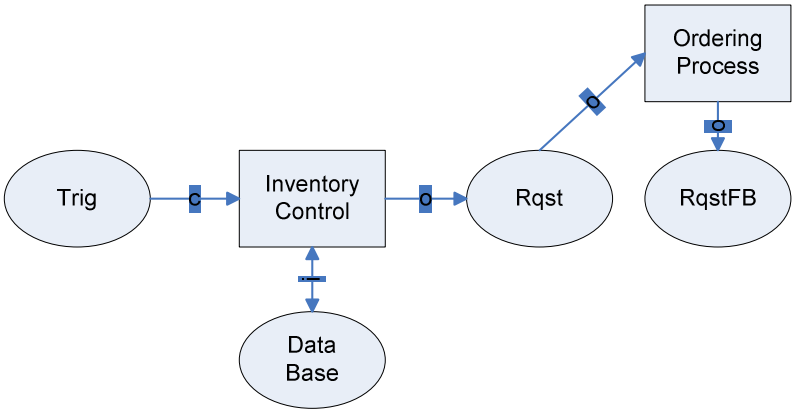

Figure 8.4: Inventory Control

Code:

InventoryControl

Post

Order<-o

If (i.level<c.mininventorylevel) Then o.size:= (c.maxinventorylevel-i.level)

Else o.size:= 0

FI

Control(Inventory) = Inventory with InventoryCheck = Trig

InventoryDB = Database

Order = Rqst

OrderRecieved = RqstFB

Inventory check place is an infinite source for the inventory control trigger. The interarrival times of periodical checks can be set in the system.

8.5.3. Inventory Set Pattern

This pattern is based on database management pattern of Aalst (2005). The purpose of the pattern is setting inventory level according to the order size. The value of inventory level (i.Level) is renewed at the end of the transition. The pattern is shown in Figure 8.3.

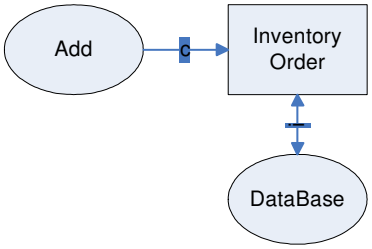

Figure 8.5: Inventory Set

Code:

InventorySet

Post inventoryDb<- i

i.level:=i.level+o.size

Connector:

Set(Inventory)= Inventory with OrderRecieved = Add

InventoryDB = Database

8.5.4. Inventory Log Pattern

This Pattern is based on Log management pattern of Aalst(2005). The purpose of the pattern is to log the inventory surplus or backlogs for future cost analysis. The inventory levels are recorded into two different places according to the sign of the inventory. The pattern is presented in Figure 8.4.

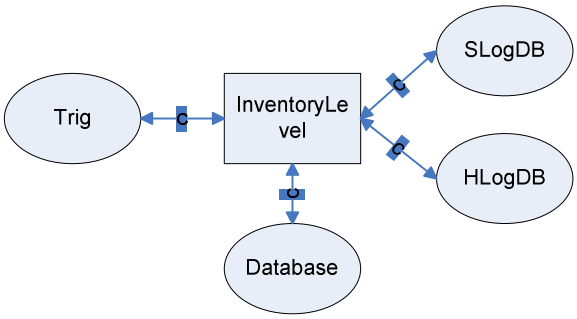

Figure 8.6: Inventory Log

Code:

InventoryLevel

Post IF i.level>0 slogDB<-c ELSE HlogDB<-c Fi

Connector:

Level(Inventory)= Inventory with InventoryCheck = trig

InventoryDB= Database

ShortageDb=sLogDb

HoldingDb= hLogDb

8.6. Model Creation

8.6.1. Adding structure without Hierarch

As it has been described in the section 4.4, patterns are united without any Hierarch. In this case, all the details of the model can be seen. It is the first phase of the model creation which is shown in Figure 8.5.

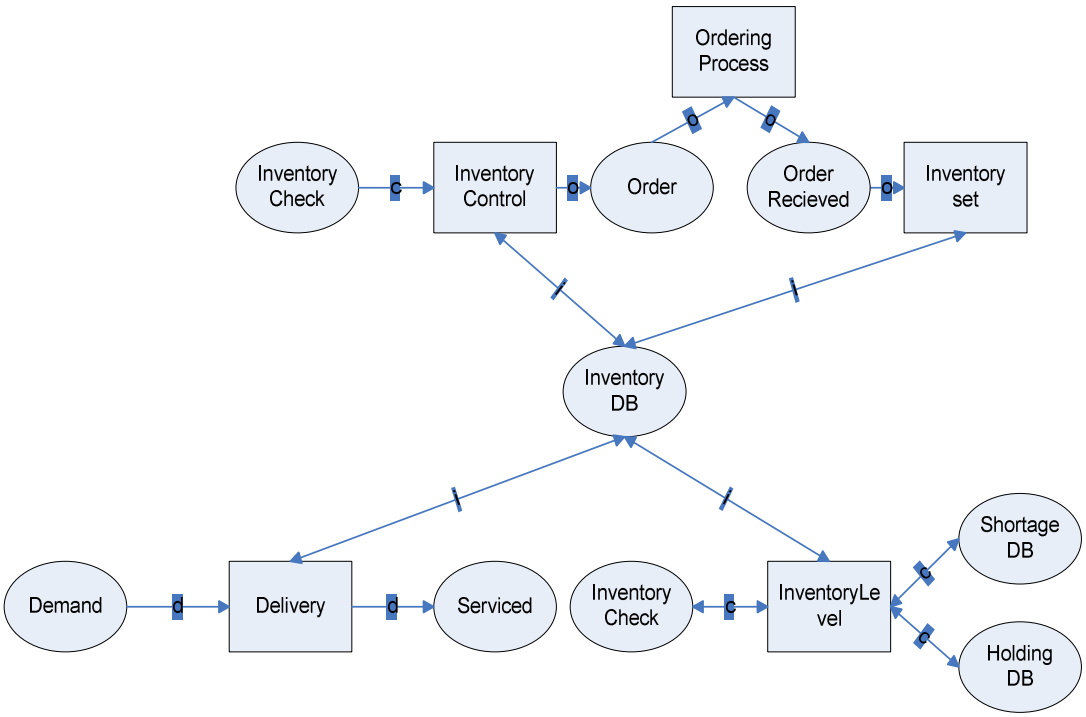

Figure 8.7: Inventory Figure without Hierarch

8.6.2. Adding Structure with Hierarch

If we evaluate the patterns in to major patterns we will acquire semi major patterns such as inventory order pattern which is the combination of inventory control pattern and inventory set pattern. The pattern is shown in Figure 8.6.

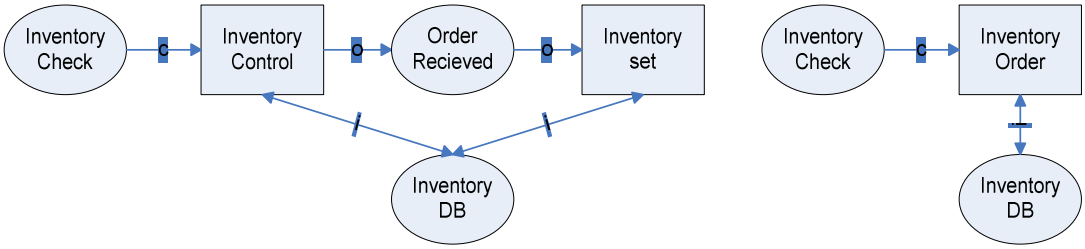

Figure 8.8: Combined patterns of inventory control and inventory set

The combination of inventory order pattern and Delivery pattern is resulted in the inventory management pattern which is managing all the inventory stock control and supplying of the orders.

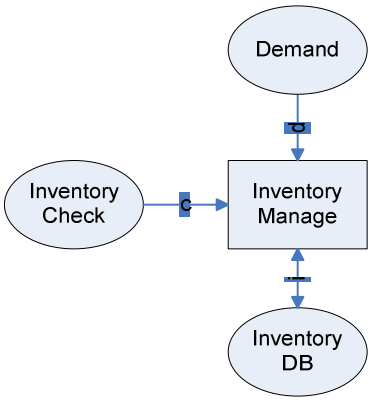

Figure 8.9: Combined pattern of Inventory order and delivery

The collection of the inventory management and Inventory log pattern gives us the solution of the problem which runs the system and calculates the cost of the inventory. The pattern is shown in Figure 8.7 & Figure 8.8.

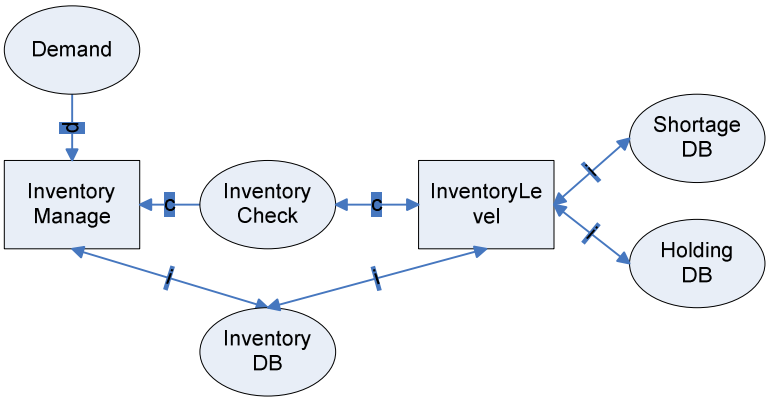

Figure 8.10: Inventory Figure with hierarch

9. EVALUATION

The main advantage of CPN patterns is the ability to create models easily without any compatibility issue. In this study certain cases has been examined. The problems are actually independent from each other, but after creating the individual models by patterns, connecting the independent cases won't be difficult.

If the actions of the examples are generalized it can be said that the ship transportation problem includes a ship loading transition. In the terms of hierarch it is possible to use the transition of Ship loading problem in this circumstances.

In the below figure, It has been showed a general demonstration of the combination of the models. The two models which have the main object as ship, has been combined without any difficulty or modification since the objects has the same descriptions.

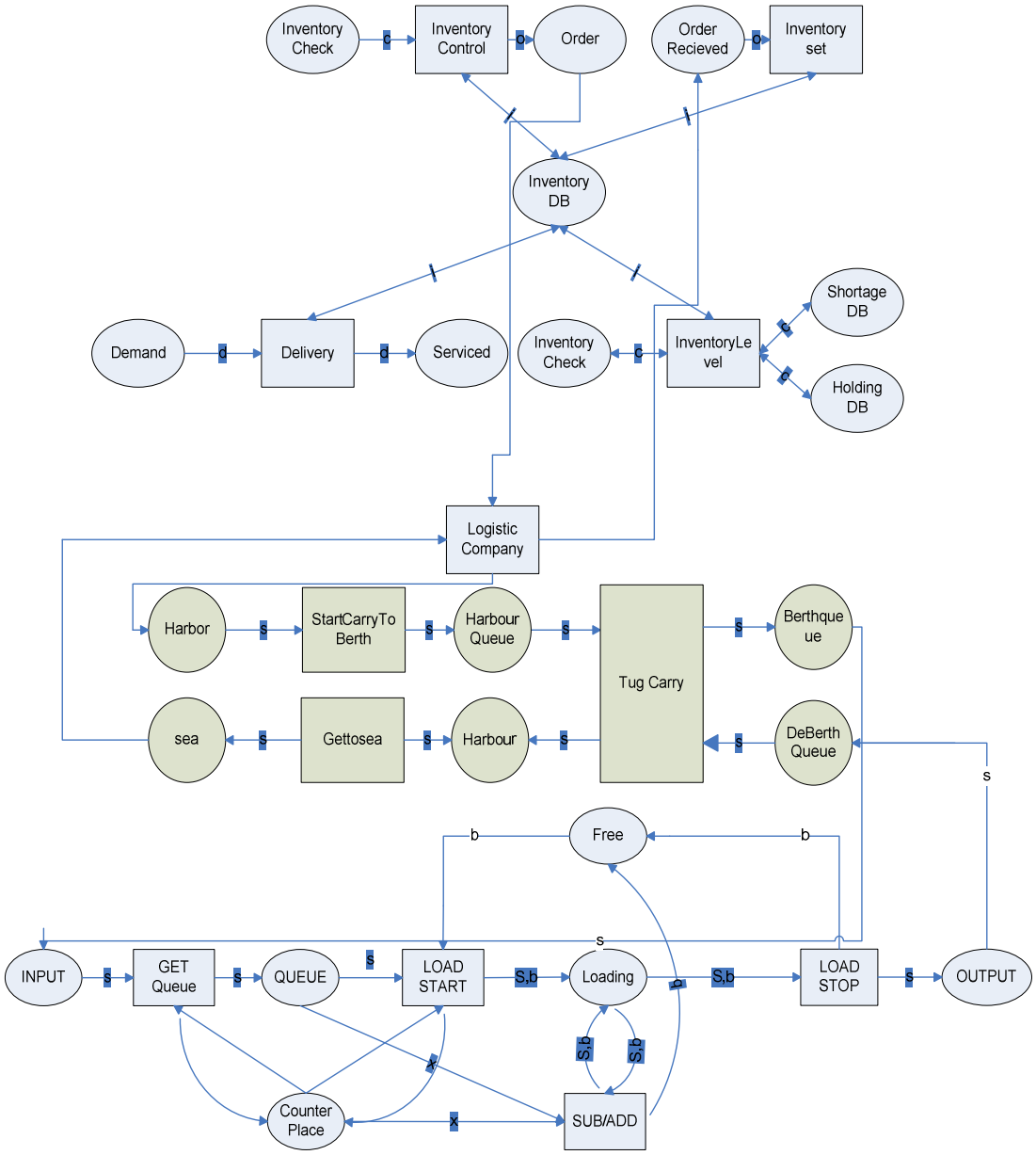

Figure 9.1: Combination of Inventory, ship trasport and ship loading cases

In the Evaluation of the cases, the vertical connectivity is seen between the cases as in Figure 9.1. The berthing activity transition in the ship transport case can be explained by the ship loading case. The two cases are different from each other and the token properties are different. But the UML allows us to combine two Figures without any problem; the last UML of the Figure will include all the properties of the tokens in the system.

The other vertical connection is the ordering process transition in the inventory case and the ship transportation case. The same logic may apply for the connection. The result is seen on the table.

If we think in the manufacturing concept, the modular system for creating simulation Figures can enable for re usage of the specific simulation Figures, opportunity to organize flexible manufacturing systems.

10. CONCLUSION AND RECOMENDATION

10.1. Impact of Pattern Use

In this work with Petri net patterns, certain queuing cases, inventory cases and their interconnectivity to found a complex Figure has been presented. The study shows the reusability of the partial Figures (patterns), and how to configure them to create a complex Figure. It also allows transferring of Figures from one decision study to another.

This is an improvement when comparing to the general and common situation where a Figure created conventionally for one system can not be used in another. (Klussmann & Caskey, 1998)The complexity and specificity of the Figure does not allow it to be reused in other systems.

Conventionally, Figureing the production line which is open to the dynamic changes for more efficiency and performance is costly. After the simulation the production line may change and a new simulation might be necessary for production management decision making. Patterns have the advantage of allowing flexible Figureing of the system that must be studied. Patterns that correspond to the changes in the production line can be added to earlier decision study deliverables. These deliverables must be kept in a repository. Pattern based configuration of the system Figure is compatible with the change methods in many operational processes: a new aspect or component is built into the production system, for instance a Flexible manufacturing systems (FMS). After the first Figure which is based upon the configuration of patterns, it is possible to Figure change every time the production line has changed.

From the cost/time viewpoint, and when no patterns are available yet, the investment in the first patterns and the Figure configuration will be larger than in conventional Figures. However, in the context where frequent changes are required, or where pattern libraries are available, the reuse of the Figure itself will be much more economic and feasible.

Regarding the cost/time aspect the gain of the pattern development and usage is in the reuse of the Figure in the other situations. Our main challenge here is how to

cope with the lack of negative examples. We do not have logs that show the forbidden (negative) behavior. (Medeiros, Weijters, & Aalst, 2005)

The other benefits of the pattern usage are vertical and horizontal connectivity among simulation based decision studies. Horizontal connectivity occurs where different manufacturing plants are Figureed separately and then simulated together. The Ship transport and the loading Figures illustrate the vertical connectivity of patterns and Figures.

Our main challenge here is how to cope with the lack of negative examples. We do not have logs that show the forbidden (negative) behavior. (Medeiros, Weijters, & Aalst, 2005)

10.2. Use of Connectors

Connectors are the main elements that can transform the attributes of the entities in one pattern to another. The only thing that has to be reset on the replication of the patterns is the logic of the connector conversation. The designer of the Figure doesn't need to know all the constraints but the entity attributes for managing the connectors between the patterns.

10.3. Recommendations

Previously, both ship and berth cases were examined. In addition to the previous studies, UML Figures & Patterns have been extended for basic patterns. The tools that have been used in the research are explained in various aspects.

The patterns and the connections that were described in this study give an idealized view of reuse of Petri net patterns. Different disciplines of processes can be studied in detail and the process Figureing and simulations can be improved. Especially new manufacturing techniques such as modular manufacturing systems, flexible manufacturing systems have the potential to the applications of patterns.

In this study the comparison is made most in the time aspect and insight aspect which is mentioned in the chapter 2. Further comparison should be made with the other aspects such as team and organizational aspect and cost aspect.

REFEFENCES

Aalst, W. v., & Mulyar, N., 2005. Patterns in colored Petri nets. *Phd Thesis*, University of Technology, Eindhoven.

Aalst, W. v., Guenther, C., Recker, J., & Reichert, M., 2006. Using Process Mining to Analyze and Improve Process Flexibility. *Proceedings of the BPMDS Workshop at the 18th International Conference on Advanced Information Systems Engineering , CAiSE'06* , s. 168-177. Namur University Press.

Adamou, A. M., 1993. Figureing and control of fexiablemanufacturing assembly systems using object oriented Petri Nets. *Workshop on Emerging Technology and Factory Automation* , 164.

Arons, H. d., & Boer, C. A., 2001. Sotrage and retrieval of discrete event simulation Figures. *Simulation Practice and Theory 8* .

Chan, F. T., & Chan, H., 2004. A comprehensive survey and future trend of simulation study on FMS scheduling. *Journal of Intelligent Manufacturing 15*

Davenport, T., 2000. Mission Critical: Realizing the Promise of Enterprise Systems. Boston: Harvard Business School Press.

Dr. Goossenaerts, J., & Dr.Pels, H., 2005. *Methodology for Simulation of Operational Processes.* Eindhoven: Cap. Eindhoven University of Technology, Department of technology management.

Gehlot, V., & Hayrapetyan, A., 2007. Systems Figureing and analysis using colored Petri Nets: a tutorial introduction and practical applications. *ACM Southeast Regional Conference* , s. 514 - 514. North Carolina: ACM New York, NY, USA .

Jensen, K., 2004. Special section on coloured Petri nets. International Journal on Software Tools for Technology Transfer , STTT , 95-97.

Jensen, K., 2004. Special section on the practical use of high-level Petri nets. *International Journal on Software Tools for Technology Transfer , STTT* , 369-371.

Klussmann, J., & Caskey, K., 1998. Increasing the usability of material flow simulation Figures through Figure data exchange. *Simulation practive and theory 6* .

Kristoffersen, T.& Boudko, S., 2006. Simplified Workflow Management with Metadata-Enhanced Petri Nets. *2006 IEEE International Conference on Systems, Man, and Cybernetics , Taipei, Taiwan* .

Law, A., & Kelton, W., 2000. Simulation Figureing and Analysis, 3d edition. McGraw-Hill.

Li, X., Chapa, S., Marin, J., & Cruz, J., 2004. An application of conditional colored Petri nets: active database system. *International Conference on Systems, Man and Cybernetics* , 4885 - 4890.

Liu, C.H., 2005. Figureing and Simulation of the MES-Controlled Wafer Fabrication Process. Jhongli City, Taiwan: National Central University.

Liu, E., Kumar, A. & Aalst, W. v., 2007. A Formal Figureing Approach for Supply Chain Event Management. *Decision Support Systems* , 761-778.

Lohri, R., 2000. Traffic Calculation. *Configuration Rules - Schindler A.G.*, 1-4.

Lohri, R., 2000. Traffic Simulation. *Configuration Rules - Schindler A.G.,* 1-3.

Martinez, J., Muro, P., 1987. Figureing validation and software implementation of production systems using high level Petri nets. *Int. Conf. Robotics and Automat.,* 1180.

Matsuno, M., & S, Chen, L., 2006. Petri Net based description for systematic understanding of biological pathways. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* , 89.

Medeiros, A. D., Weijters, A., & Aalst, W. v., 2005. Genetic Process Mining: A Basic Approach and its Challenges. *First International Workshop on Business Process Intelligence , BPI'05*, s. 46-57. Nancy, France.

Mulyar, N., & Aalst, W. v., 2005. *Patterns in colored Petri nets.* Eindhoven: Department of technology management, Eindhoven University of technology.

Mulyar, N., & Aalst, W. v., 2005. Towards a Pattern Language for Colored Petri Nets. *Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools* , 39-48. Aarhus, Denmark: volume 576 of DAIMI.

Recker, J., Mendling, J., Aalst, W. v., & M. Rosemann., 2006. Figure-Driven Enterprise Systems Configuration. *Proceedings of the 18th International Conference on Advanced Information Systems Engineering , CAiSE'06* , s. 369-383. Springer-Verlag, Berlin: volume 4001 of Lecture Notes in Computer Science.

Vanderfeesten, I., Aalst, W. v., & Reijers, H., 2005. Figureing a Product Based Workflow System in CPN tools. *Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools , CPN 2005,* , s. 99-118. Aarhus, Denmark: volume 576 of DAIMI.

Wiendahl, H., & Worbs, J., 2003. Simulation based analysis of complex production systems with methods of non linear dynamics. *Journal of Materials Processing Technology 139* .

Wong, K. S., Parkin, R. M., & Coy, J., 2007. Integration of the Cimosa and high-level coloured Petri net Figureing techniques with application in the postal process using hierarchical dispatching rules. *Proceedings of the I MECH E Part B Journal of Engineering Manufacture* , 775-786.

Zulch, G., Jonsson, U., & Fisher, J., 2002. Hierarchical Simulation of complex production systems by coupling of Figures. *International Journal of Production Economics,* 77 .

Zurawski, R., & Zhou, M., 1994. Petri Nets and Industrial Applications: A Tutorial. Transactions on Industrial Electronics*, VOL. 41, NO. 6* , 561.

CURRICULUM VITAE

Ömer Gökçek was born in Istanbul in 1981. He graduated as a Mechanical Engineering from ITU in 2003. He has started to have a masters degree in ITU Industrail engineering faculty in 2003. He did the military obligation in 2006. He is now working in the Schindler Turkeli A.ş. as technical support and purchasing engineer.