MASTER

HPP design with CPN tools

Wu, D.

Award date:
2005

Link to publication
Table of Content

Executive Summary ........................................................................................................................1
Project organization.........................................................................................................................2

1. Research design ..........................................................................................................................4
   1.1 Introduction ...........................................................................................................................4
   1.2 Research objective and methods ........................................................................................5
   1.3 Research questions definition ............................................................................................6
   1.4 Thesis structure ...................................................................................................................6

2. Hierarchical Production Planning ..........................................................................................7
   2.1 Introduction ...........................................................................................................................7
   2.2 Key points in modeling ........................................................................................................8

3. CPN and CPN Tools introduction ...........................................................................................9
   3.1 Simulation tools ..................................................................................................................9
   3.2 CPN Tools ..........................................................................................................................10
      3.2.1 CPN—Extension of Classic Petri Nets ..................................................................10
      3.2.2 CPN Tools introduction ..........................................................................................12

4. Engineer-to-Order design ........................................................................................................14
   4.1 Case description ................................................................................................................14
   4.2 Architecture design ............................................................................................................16
   4.3 Algorithm design ..............................................................................................................17
   4.4 CPN model .......................................................................................................................19
      4.4.1 Main level ..............................................................................................................19
      4.4.2 Initialize ............................................................................................................... ..21
      4.4.3 Generate demands ..................................................................................................21
      4.4.4 Acquire capacity .....................................................................................................21
      4.4.5 Plan & control ........................................................................................................21
      4.4.6 Execute assembly ...................................................................................................24
      4.4.7 Report .....................................................................................................................24
   4.5 Simulation results and analysis .........................................................................................25
      4.5.1 Validation ...............................................................................................................25
      4.5.2 Speed .................................................................................................................... ..27
   4.6 Conclusion .........................................................................................................................28

5. Evaluations .............................................................................................................................29
   5.1 Observational method .......................................................................................................29
   5.2 Experimental method ........................................................................................................30
   5.3 Conclusions and future work ............................................................................................31

Appendix A: source code for first CPN application .................................................................35
Appendix B: source code for second CPN application ..............................................................39
Appendix C: manual of production planning model .................................................................48
Appendix D: manual of Engineer-to-Order model .................................................................51
Appendix E: Production planning design ................................................................................54
Executive Summary

Hierarchical Production Planning (HPP) was originally proposed by Anthony (1965), and then developed by Bitran, Hax and Meal (BHM) in the early 1970s (Winter (1989)). Meal (1984) analyzes and describes the characteristics of HPP in more detail. De Kok and Fransoo (2003) make use of HPP concept and position the Supply Chain Operations Planning (SCOP) problem in a hierarchical framework. SCOP is to coordinate the release of materials and resources in the supply network under consideration such that customer service constraints are met at minimal cost (De Kok and Fransoo (2003)). However, there is a need for a simulation environment to efficiently develop and simulate the performance of such concepts.

The objective of our project is to investigate whether CPN Tools can be used for the development of a hierarchical modeling environment for logistics control concepts. This could also be regarded as the first step to build a simulation environment which can simulate and analyze the performance of SCOP problem in HPP concept.

In our project, we are trying to answer this question by two test cases. The first case is a production planning example which has already been implemented by Delphi. We want to mimic this Delphi application by using CPN Tools. Actually, the main purpose of this case is to get familiar with CPN Tools and test the feasibility of the second case. This part is put in the Appendix E of this thesis. Based on the experience obtained from the first case, we can start to test the second case, which is using CPN Tools to build a simulation environment based on the production control framework given by Bertrand et al (2003). In this simulation environment, some basic activities can be performed, e.g. make planning of the resource; execute the planning already made; calculate the utilization of the resource. In this test case, we only focus on the design of such HPP structure, and the validity of CPN Tools for that kind of design, but not on the design of effective and efficient decision algorithms.

Finally, we will assess CPN Tools using in modeling HPP concept. The evaluation can help us to answer the question whether CPN Tools can be used for the development of a hierarchical modeling environment for logistics control concepts.
Project organization

Technische Universiteit Eindhoven

The Technische Universiteit Eindhoven was established in 1956. It now encompasses 9 scientific departments: Architecture, Building and Planning, Electrical Engineering, Chemical Engineering and Chemistry, Applied Physics, Technology Management, Mechanical Engineering, Mathematics and Computer Science, Biomedical Engineering, and Industry Design, providing 10 academic Bachelor programmes, 19 Master programmes, 10 postgraduate design programmes, 3 first degree teacher-training programmes in mathematics, physics and chemistry, as well as various other postgraduate courses and programmes.

TU/e has approximately 3000 employees, 220 professors, 6800 students, 200 postgraduate students, 450 PhD students, 20,000 graduate engineers, 1000 graduate design engineers and has awarded about 2000 PhD's. The university participates in the European university networks CESAER, Santander and CLUSTER and collaborates with universities all over the world.

The Master’s research project is supervised by the Operations, Planning, Accounting, and Control (OPAC) and the Information Systems (IS) sub departments of the Technology Management department at the Technische Universiteit Eindhoven.

Participants Master’s research project

Student

The Master’s research project is managed by Di Wu and Xi Lu. Both of them are the Master students of Business Information System program in Technische Universiteit Eindhoven. They are responsible for the day-to-day development of the Master’s research project, which is under guidance of two academic advisors of TU/e.

Di Wu is the author of this thesis. He is responsible for the CPN model design based on Integrated Approach in the first test case. In the Engineering-to-Order case, he is responsible for the detailed planning part.

Xi Lu is responsible for the CPN model design based on Aggregation-decomposition Approach in the first test case. In the Engineer-to-Order case, he is responsible for the aggregated planning part.

Di Wu Email: W.Du@student.tue.nl
Xi Lu Email: X.Lu@student.tue.nl
Technische Universiteit Eindhoven
Prof. dr. A.G. de Kok acts as an advisor from the Operations, Planning, Accounting, and Control sub-department of the Technology Management department at the Technische Universiteit Eindhoven.

Prof. dr. ir. Wil van der Aalst acts as an advisor from the Information Systems sub-department of the Technology Management department at the Technische Universiteit Eindhoven.

Dr. Jansen - Vullers, M.H. acts as an advisor from the Information Systems sub-department of the Technology Management department at the Technische Universiteit Eindhoven.
1. Research design

This chapter introduces the research field of testing CPN Tools. Section 1.1 gives the research background. Section 1.2 presents the objective of this project and methods used in reaching the project objective. Section 1.3 defines the research questions. Section 1.4 provides an overview of the whole report.

1.1 Introduction

Decisions with regard to the different components of planning of supply chain operations are difficult to make due to the scheduling problem, the inventory problem, and the aggregate capacity planning problem have hardly been interconnected while maintaining their own characteristics (De Kok and Fransoo (2003)). One way to deal with that is to organize these decisions in a hierarchical manner.

Hierarchical Production Planning (HPP) is originally proposed by Anthony (1965), and then developed by Bitran, Hax and Meal (BHM) in the early 1970s (Winter (1989)). Meal (1984) analyzes and describes the characteristics of HPP in more detail.

De Kok and Fransoo (2003) further investigate HPP concept mainly based on the research result of Meal (1984), and then they position Supply Chain Operations Planning (SCOP) problem in a hierarchical framework, shown in Figure 1.1. SCOP is to coordinate the release of materials and resources in the supply network under consideration such that customer service constraints are met at minimal cost (De Kok and Fransoo (2003)).

However, there is a need for a simulation environment to efficiently develop and simulate the performance of such concepts. On one hand, the simulation environment can help the researchers study this structure. On the other hand, it can validate this structure and find some drawbacks.
1.2 Research objective and methods

In order to design and build this simulation environment, we need to find an appropriate simulation tool. The objective of our project is to investigate whether CPN Tools can be used for the development of a hierarchical modeling environment for logistics control concepts. This could also be regarded as the first step to build the simulation environment which can simulate and analyze the performance of SCOP problem in HPP concept.

In our project, we used a logistics case, namely Engineer-to-Order (E2O) design case, to test whether CPN Tools is suitable for HPP concept. E2O is to make decision of resource release in the context of Engineer-to-Order production situation. Bertrand et al. have already designed a hierarchical planning structure. According to this structure, we can make use of CPN Tools to design a model which can simulate this E2O environment. After that, in order to validate our CPN model, some statistics should be analyzed.

Based on the evaluation of our model, finally we can find the answer for the question: whether CPN Tools can be used for the development of a hierarchical modeling environment for logistics control concepts.
1.3 Research questions definition

In this thesis, the main question that needs to be answered is whether CPN Tools can be used for the development of a hierarchical modeling environment for logistics control concepts, which has already been defined as the objective of this master project. In order to give a comprehensive answer, several sub-questions are also given as follows:
- What is Hierarchical Product Planning concept?
  What are the characteristics of HPP?
  What are the challenges of modeling HPP?
- Why did we choose CPN Tools?
  What are the characteristics of different categories of simulation tools?
  What is CPN Tools?
  What is the main function of CPN Tools?
- Is CPN Tools valid to develop a hierarchical modeling environment for logistics control concepts?
  What is the generic hierarchical planning model in the context of an Engineer-To-Order assembly stage?
  How to integrate decision functions in this generic hierarchical planning structure?

1.4 Thesis structure

The remainder of this thesis is organized as follows. In Chapter 2, a brief introduction of HPP concept is described, as well as the difficulties of modeling HPP is also pointed out. In Chapter 3, a general introduction to CPN and CPN Tools is given. In Chapter 4, we describe the Engineer-to-Order case in detail. Our description includes the case introduction and depiction of the design, implementation, as well as evaluation of the CPN model. Finally in Chapter 5, as the final conclusion we will answer the question of whether CPN Tools can be used for the development of a hierarchical modeling environment for logistics control concepts based on the evaluation of the CPN Tools.
2. Hierarchical Production Planning

In this chapter, we will give a brief introduction of HPP concept at beginning. And then we will explain the difficulties in modeling HPP problem.

2.1 Introduction

Production planning is a fundamental problem in the operation of a manufacturing enterprise. Normally, the task is to determine the type and quantity of the products to produce, to meet uncertain demand in the future time periods under consideration of minimal cost. Traditionally, this problem can be solved using a monolithic model, but it often requires very large-scale mathematical programming model.

Another problem is decisions with regard to the different components of planning of supply chain operations are difficult to make due to the scheduling problem, the inventory problem, and the aggregate capacity planning problem have hardly been interconnected while maintaining their own characteristics (De Kok and Fransoo (2003)).

Hence it becomes necessary to develop alternative techniques which has strong computational ability and can also develop near optimal solutions. Decomposition techniques are one way to solve such large-scale models.

The idea for Hierarchical Production Planning was captured formally by Anthony (1965). He introduced three levels of hierarchical control: Strategic Planning, Management Control, and Operation Control. Based on this idea, Hax and Meal (1975) formalize the hierarchical production planning concept.

Hierarchical Production Planning (HPP) is such an approach based on decomposition technique that separates the whole planning problem into distinct sub-problems according to length of planning horizon, time and cost. Typically, at the higher level of HPP, it represents the planning for aggregated problems related to total manpower requirements and total product-line demand, with longer lead times, longer planning horizons, e.g. several months. In contrast, at the lower level of HPP, it provides a more detailed plan with shorter lead times and shorter planning horizons related to individual items, machines and workers.

The related optimization models are solved in sequence starting from the top level and ending at the bottom level. The optimized solution of the model getting at a given level will be transmitted to the model of the subsequent level. These solutions reflect the constraints which are transferred from one level to the next lower level. The
solution of the bottom level, i.e. the detailed production plan, is implemented on the shop floor. Sometimes, higher levels also need the feedback from lower levels in order to adjust and improve the optimized solutions of upper levels. A better HPP approach would be allowed for more feedback from the detailed level to the aggregated level, because in this case, more information will be used in the optimization models in order to improve the accuracy of the solutions.

Typically, a HPP approach has these advantages as follows: [17]
1. Reduces complexity and thus renders the planning problem solvable within reasonable time
2. Copes better with disturbances since only part of the problems have to be solved again in case of random events
3. Fits with the management hierarchy, since it is possible to design the hierarchical production management system in such a way that each of its level corresponds to a level of the management hierarchy
4. Reduces the need for detailed information in long and medium term planning
5. Allows the use of different criteria at each managerial level.

2.2 Key points in modeling

As discussed in the last section, HPP is an approach to decompose a complex planning problem into several small planning problems mainly according to the length of lead time. So we can know that the time concept is a quite important factor in model HPP problem.

Although HPP decompose the very large-scale mathematical programming model, to solve the related optimization models, a quite strong computation power is still required. It implies that the tools in modeling HPP problem must have enough capability to deal with mathematics problems.

In logistics field, to simulate a reality environment is quite difficult not only because of the complex process but also due to the quite huge different type of data. It is a challenge to express each data clearly for the simulation tools.

These three points mentioned above construct the difficulties in modeling HPP problem. In the following chapter, we can see if CPN and CPN Tools can solve these difficulties.
3. CPN and CPN Tools introduction

In this chapter, we will give a brief introduction of CPN Tools, which includes the characteristic and the main function of CPN Tools. Besides we also explain the reason why we choose CPN Tools as our research object. Section 3.1 gives a general introduction of simulation tools. Section 3.2 gives a quite detailed introduction of CPN Tools which includes the basic concepts and how to use it.

3.1 Simulation tools

In the process modeling field, simulation is a kind of technique which can be used in modeling systems, such as manufacturing systems. The Oxford English Dictionary describes simulation as:

"The technique of imitating the behavior of some situation or system (economic, mechanical, etc.) by means of an analogous model, situation, or apparatus, either to gain information more conveniently or to train personnel."

In other words, simulation is the technique of building a model of a real or proposed system so that the behavior of the system under specific conditions may be studied. One of the key powers of simulation is the ability to model the behavior of a system as time progresses [12].

Simulation tools are the software for building a simulation model. In principle, simulation tools were classified into two categories: simulation languages and simulation packages [9]. The simulations languages are typically extensions of programming languages with facilities for timed and stochastic behavior. It can generate pseudo-random numbers and variants from different probability distributions. Simulation packages are graphical applications with a graphical interface and a simple statistical gathering capability for the analysis of the results. Using a simulation package, people can assemble a simulation model from predefined components.

Simulation languages are flexible but require coding. Simulation packages provide graphical modeling environment but are less flexible and quite limited because only predefined components can be glued together.

There are also tools which are in between the traditional simulation language and simulation package. These tools combine the advantage from simulation languages and simulation packages. They not only provide a graphical modeling environment but also support the design of new components. CPN Tools is such kind of tool.
3.2 CPN Tools

It’s impossible to discuss about CPN Tools without the introduction of Color Petri Nets. So at the beginning of this sub-section, a general introduction of the CPN concept is necessary. Afterward, we will describe the characteristics of CPN Tools, followed by how to use CPN Tools.

3.2.1 CPN—Extension of Classic Petri Nets

Van de Aalst (2004) defines a colored Petri net (CPN) is a Petri net in which every token has a certain value. It implies that CPN is an extension of Classical Petri Nets. On the one hand it inherits all characteristics of Classical Petri Nets. On the other hand the extension has been made aims at the deficiencies of Classical Petri Nets. These extensions are expressed as follows [10]:
1. The extension with color increases the expressive power of Petri nets considerably.
2. The extension with time adds a dimension to the Petri nets. As a result, new analysis methods focusing on time become possible.
3. The extension with hierarchy allows for the construction of large and complex models but does not complicated analysis.
4. The extension with programming language provides the primitives for the definition of data types and the manipulations of data values.

A CPN model is always created as a graphical drawing and Figure 3.1 contains a simple example [24] of CPN model. This model is used to simulate a simple transport process. The left part models the sender, the middle part models the network, and the right part models the receiver.

Figure 3.1 a simple transport protocol [24]
The CPN model shown in Figure 3.1 is composed of seven places (drawn as ellipses or circles), five transitions (drawn as rectangular boxes), a number of directed arcs connecting places and transitions, and some inscriptions next to the places, transitions, and arcs.

The places are used to represent the states of the system. Each place contains a set of markers called tokens. In contrast to Classic Petri nets, each of these tokens carries a data value, which belongs to a given type. These types are called the colour set of the place and written below the place, e.g. the places “NextSend”, “C” and “D” have the colour set NO; the places “A”, “B” and “Packets to send” have the colour set NO * DATA.

The inscription at the upper right side of place “NextSend” specifies that the initial marking of this place consists of one token with the value 1. It indicates that data packet number 1 is the first data packet going to be sent.

The inscription at the upper left side of place “PacketsToSend”:

\[
1'(1,"COL") ++ \\
1'(2,"OUR") ++ \\
1'(3,"ED ") ++ \\
1'(4,"PET") ++ \\
1'(5,"RI ") ++ \\
1'(6,"NET")
\]

specifies that the initial marking of this place consists of six tokens with the data values: (1,"COL" ), (2,"OUR"), (3,"ED "), (4,"PET"), (5,"RI "), (6,"NET").

The transitions represent the events that can take place in the system. When a transition occurs, it removes tokens from its input place and adds tokens to its output places. Same as Petri Nets, one transition can only occur when all of its input places have at least one token.

The arcs describe how the state of the CPN changes when the transitions occur. These arc expressions are written in the CPN ML programming language and are built from variables, constants, operators, and functions.

Based on the characteristic mentioned above, CPN can be applied in the area of communication protocols, distributed systems, imbedded systems, automated production systems, work flow analysis and VLSI chips [13].

CPN can be used in practice because there are mature and well-developed tools to support it. CPN Tools is one out of those mature and well-developed tools.
3.2.2 CPN Tools introduction

CPN Tools is a tool, developed by the University of Aarhus, Denmark, for editing, simulating and analyzing Colored Petri Nets. As a major redesign of the Design/CPN tool (a popular widespread tool for Colored Petri Nets), CPN Tools is the new standard tool for CPN. The new GUI is based on advanced interaction techniques, such as toolglasses, two-handed input by means of a mouse and a trackball, marking menus and a new metaphor for managing the workspace. Using these techniques, CPN Tools gets rid of traditional ideas about user interfaces, such as pull-down menus, scrollbars and even selection, while provide the same or more powerful functionality. CPN Tools requires an OpenGL graphics accelerator and will run on all major platforms (Windows, Unix/Linux, MacOS) [14].

Compared with classic Petri Nets, the main innovation of CPN is the extension with programming language, i.e. primitives for the definition of data types and the manipulations of data values. CPN Tools use CPN ML, which is obtained by extending Standard ML in three different ways [15].
• CPN data types, called colorsets.
• CPN variables.
• Reference variables with a specified scope.

The first extension adds syntactic sugar for colorset declarations. This makes it very easy to declare the most common kinds of color sets. It also means that the user can include a number of predefined functions, just by mentioning their name in a color set declaration. CPN Tools provides a mechanism for user-defined data types. For historic reasons these user-defined data types are called colorsets. Each colorset represents a set of values and is identified by an alphanumeric identifier. The colorsets must be declared in global, local, and temporary declaration nodes. CPN Tools automatically generates a set of pre-declared constants, operations, and functions for each colorset declared. [16]

CPN Tools can define the colorsets of Unit, Integers, Real, Boolean, String Enumerated Value etc. CPN Tools also provides three kinds of type constructors which can construct new and complex data type, namely product, record and list. Product and record are quite similar. All things expressed in terms of records can also be expressed in products and vice versa. However, the difference still exists. We will discuss about that in later chapters.

The second extension of Standard ML allows the user to introduce typed variables, as a part of the declarations. The third extension is made to be able to define the scope of reference variables.

In order to further explain the function of CPN Tools and how to use it, we will
introduce a simple example traffic light, which is shown in Figure 3.2. The user interface is a window, containing an index on the left side and a workspace on the right. In the workspace, a graphical model is shown. Places ‘red’, ‘green’ and ‘orange’ present three states of traffic light. Between every two places, a transition is connected in order to express the action should be taken. The arc connected between place and transition is responsible for data transfer. Data type and place type are defined in Declaration located in the left side. Compared with classic Petri net traffic light model, this model can determine how long the traffic light is red, green or orange through setting a delay to the transition when produce tokens. The delay is denoted by @+., which is shown in Figure 3.2

In order to run this model, we need to first drag the simulation tools from the index to the workspace, and then select and apply one of the simulation tools from the tool palette. The first button is to rewind the simulation to initial state; the second button is to stop a simulation; the third button is to fire a single transition; the fourth button is to fire the specified number of transitions; the fifth button is to fire the specified number of transitions without intermedia process. The number of transitions can be expressed by step shown in index. Time in index is to record the current system time. For instance, the Figure below represents that the state of traffic light at time 30 and system have already executed 44 steps.

So far, we can only see a fundamental function of CPN Tools. With the introduction of two test cases, a detail explanation will be given in Chapter 4 and Appendix E.
4. Engineer-to-Order design

This chapter introduces the design and implementation of a CPN model for a HPP concept in the context of engineering to order situation. Section 4.1 gives the case description in detail. Section 4.2 shows the architecture of model. Section 4.3 describes the algorithm used in this model. Section 4.4 introduces the data structure of the model. Section 4.5 shows the model implementation.

4.1 Case description

The Manufacturing Resources Planning (MRP II) is a production control system which can be used for supporting the resource coordination decision making. However, MRP II has its own limitations and can not be employed in every situation.

Sari (1981) classified four different production situations, namely

(1) make-to-stock: converts lower-level components and raw materials all the way to end-items in anticipation of customer orders

(2) assemble-to-order: converts lower-level components and raw materials to a predetermined level of manufacture, and configures to customer order upon receipt of a customer order

(3) make-to-order: obtains very few, perhaps no lower-level materials until after receipt of a customer order

(4) engineer-to-order: knows very little about what to order or manufacture until after receipt of a customer order and development of engineering specification

The MRP II production concept has especially been developed for the “make to stock” and (in specific situations) for the “assemble-to-order” situation [6, 7], but it is not suited for Engineer-To-Order (E2O) situation. Bertrand et al. (1993) have developed a production control framework considering the specific characteristics of E2O, and defined three main processes, which are Engineering, Component production and Assembly.

Based on HPP concept, a Hierarchical Planning Model [21] is shown in Figure 4.1
At the operational level, there are planning procedures within the engineering department, for the production of components, and within the assembly phase. Above that, there is a work order release function. The work order release function is fed by order acceptance.

At tactical level, one has to deal with capacity planning. For the engineering department, one has to decide how many engineers of each type are needed. For the production of components, one has to decide on reserved capacities and prices specified in contracts with subcontractors. For the assembly department, one has to decide on how many assembly employees of each type are needed.

In our test case, we only focus on assembly stage rather than the whole process due to two reasons. Firstly, our aim is to test the validation of CPN Tools using in HPP concept, and the CPN model which simulates assembly stage is sufficient for us to answer this question. Another reason is because of the time limitation. Under this condition, the Hierarchical Planning Model can be reduced into capacity acquisition in assembly department and operational planning in assembly, these two models.

Assembly capacity includes assembly Specialist Capacity (SC) and assembly Regular Capacity (general assembly personnel). Assembly specialists are needed for the critical steps in assembling and testing of the capital good, and they can also do regular work. Assembly SC can not be taken place by other capacity due to they need long time technique training. We assume SC is always enough due to department “Order acceptance” will not accept any orders when SC is not enough.

With regard to Regular Capacity, we need to distinguish it into Own Regular Capacity (ORC) and Contingent Capacity (CC). The ORC can be changed over time, but with a high cost; the CC is more costly per unit, but can be varied with much lower costs.
The using of CC aims at two points. One is to fully utilize ORC, and the other is to overcome the shortage of ORC and make sure each order can be released without delay.

In Figure 4.2, we can see the distribution of demands for assembly capacity. The bold line represents ORC which decided by higher level. At some point ORC is redundant, but at some point ORC is in shortage. We can use the redundancy of period t1 to make up the shortage of period t2 in the way of doing part of t2 work in t1. However, there is still a possibility that at some point, we can not find any redundancy to overcome the shortage. In this case, we need to make use of CC.

\[
\begin{align*}
\begin{array}{cccc}
t1 & t2 & t3 & t4 \\
40 & 60 & 80 & 100 \\
\end{array}
\end{align*}
\]

**Figure 4.2 Contingent Capacity**

At the higher level, capacity acquisition, the main task is to make a planning of how much general assembly personnel are required up to a horizon (e.g. 4 weeks)

At the bottom level, operational planning, the main task is to make a planning of how much CC are required up to a horizon (e.g. 1 week. This horizon must be shorter than the upper level’s horizon). Another task is to decide how to distribute ORC, SC and CC to each order.

### 4.2 Architecture design

Based on the description above, we can conclude the main working process of this simulation environment. Initially, system will generate the demands. The order arrival follows Poisson distribution and the order volume is mixed-Erlang distributed. Between the order arrival and assembly start, there is a fixed time interval (e.g. 4 weeks leadtime for engineering, 4 weeks leadtime for components) for completing engineering phase and component phase. After getting the demands, system can deal with the capacity acquisition and then calculate the level of required Own Capacity up to a planning horizon. Thereafter, system will deal with operational planning level. Finally, it is statistics work which includes average cost and utilization of each capacity.
From the exposition above, it can be concluded that the architecture of this simulation environment should be drawn as shown in Figure 4.3

Figure 4.3 architecture

The whole environment is split into planning world and execution world. Planning world is consisted of three main functional modules: Demand Generation, Own Capacity Planning, Operational planning modules, while execution world is constituted by Report and Simulation modules. In this paper, we only focus on Contingent Capacity Planning modules. Lu Xi (2005) described other modules in detail.

4.3 Algorithm design

In Contingent Capacity Planning module, we need to solve two problems. The first is to calculate contingent capacity. The other problem is Capacity Distribution. Here ORC, SC and CC need to be distributed.

The decisions are based on several conditions which are:
1. The amount of contingent capacity must be enough for completing order before leadtime. Initially, we assume the contingent capacity is infinite, so in our model, it is not necessary to think about the case that the order can not be finished in time.
2. We also need to consider minimizing the cost on contingent capacity. It implies that we need to think a way to use own capacity sufficiently. In that case, contingent capacity can achieve minimization.
3. The priority of each order is based on its own leadtime. It means the shorter of the leadtime, the higher of the priority.
4. For the time period, we define one week as a period. We also define that the interval between planning is one period which means if we are at time week t, we will make a plan for week t+1. Planning made in week t, will be executed in week
5. We also considered about the leadtime of Contingent Capacity. We make an assumption that all Contingent Capacity’s leadtime is one week. Under this assumption, we could omit leadtime in the whole process design, because as mentioned above the plan made in week t is for the execution of week t+1.

6. The input of this process is from Demand Generation and Own Regular Capacity acquisition, these two processes. From Demand Generation, we can get how many orders are in processing, as well as work activities and leadtime for each order.

7. The output is sent to Execution process. It includes the amount of Contingent Capacity and how to distribute Capacity for each order. In this model, we assume that Contingent Capacity and Regular Capacity only have difference on cost without any other difference.

Model description:

- At beginning, we need to think about how much regular assembly capacity we really have (reg_r) in each period. Besides own regular assembly capacity, specialist assembly capacity can also do regular assembly work. For each period, only specialists who are not occupied in special work can do regular work. In order to make the model simple, we make an assumption that for each order, all the special work will be finished in first period. Under this condition, we can easily get the amount of the specialist capacity left (spe_l) which can do regular work in each period.

\[ spe_l = SPE - \text{sum}(spe) \]

SPE is total amount of SC, and sum(spe) is total SC required for each order. Now we can calculate how much regular assembly capacity we really have.

\[ reg_r = reg + spe_l \]

reg is the amount of ORC

- Then we need to optimize the amount of contingent capacity over planning horizon. The core idea of optimization is to fully utilize own regular assembly capacity under the condition of releasing order without delay. No delay means every order should be completed before leadtime.

After receiving new orders, we need to calculate how much workload we need to finish in each period. We assume we split the whole order into work activities. The quantity of work activities we need to finish in each period depends on the amount of own regular assembly capacity (reg_r). We also define:

\[ W_{a_{i,t}} \]: denote total work activities of order i not finished yet at period t

\[ L_{t,i} \]: denote leadtime of order i at period t

At period t, if we got enough reg_r, which satisfies reg_r >= sum ( W_{a_{i,t}} / L_{t,i} ) (i=1..to n) then we do not need contingent capacity (cc) at all. Under this condition, each order can finish at least W_{a_{i,t}} / L_{t,i} , then we can achieve the
demand of finishing before leadtime. Full utilization is another demand. For left part of \( \text{reg}_r \) ( \( \text{reg}_r \) - sum ( \( \frac{W_{a_{i,t}}}{L_{i,t}} \)) ( i=1..to n)), we will assign it to the order which has highest priority.

At period t, if \( \text{reg}_r \geq ( \frac{W_{a_{i,t}}}{L_{i,t}} ) \) (order j has the highest priority), we still do not need contingent capacity. The reason is because the order with highest priority can be finished before leadtime, and after that we will get more capacity for other orders. Maybe the capacity is not enough, in that case, we will need contingent capacity, but it is the plan will be made in future not in period t. To assign \( \text{reg}_r \) is still based on order priority. Each order can get maximum \( \frac{W_{a_{i,t}}}{L_{i,t}} \) capacities

If \( \frac{W_{a_{i,t}}}{L_{i,t}} > \text{reg}_r \) (order j has the highest priority), it implies order j can not be finished before leadtime. In that case, we must hire contingent capacities and the amount is:

\[
cc = \frac{W_{a_{j,t}}}{L_{j,t}} - \text{reg}_r.
\]

Only the order has highest priority can get capacity and work in processing situation, other orders are in waiting situation.

- Last step is combining all the result get in this process and deliver it to next process (Execution). In Contingent Capacity acquisition, we get two results. One is the amount of Contingent Capacity and the other is how to arrange the distribution of capacity for each order.

### 4.4 CPN model

In this section we explain the content of the developed CPN model. We first show the top level view of the model, as well as discuss the color sets. Further, we explain each sub-page in general. Finally, we will pay attention to describe sub-page “plan & control” in detail.

#### 4.4.1 Main level

The main level of the CPN model is depicted in Figure 4.3. It can express the whole working process of the simulation environment in a high level view. The main structure is composed of 4 places and 6 substitutions (transitions).
The place “capacity order” contains complete information about the planning of how much ORC are required as one single token of color CAP_ORDs, a list of CAP_ORD. CAP_ORD is defined as product D * VOL, where D expresses the time period and VOL expresses how much ORC is available in time D. For instance if CAP_ORD = (117, 100), it means from time 117 up to one month horizon, 100 ORC are supposed to be required.

The place “statistic” contains a token of color STAT defined as product WK*MON*PERCENT*PERCENT. The color STAT is composed of an integer containing the time indicator (week), and an integer showing average cost, followed by two integers which can indicate the utilization of SC and ORC.

The place “demands” contains the information about the future demand as one single token of color ORDs, a list of ORD defined as product ID*VOL*VOL*D*D, where the first is the unique ID of an order, the second element is the volume of an order for Specialist, the third is the volume for Regular, the forth is its start time and the last is its lead time. The volume of an order is described as its capacity requirement (number of hours working).

The most complex color within this model is the token in the place “assembly plan”. It is defined as D*VOL*ASS_PLs and composed as follows: D expresses the time period and VOL expresses how much CC are required in time D. ASS_PLs is a list ASS_PL, and ASS_PL is defined as product ID*VOL*VOL*VOL*VOL, where ID is the order’s unique number, first VOL expresses how much SC are allocated to this order, second VOL expresses how much ORC are allocated to this order, third VOL expresses how much CC are allocated to this order and the last VOL is how much workload left for this order.

The whole process start from the transition “initialize” followed by the transition “generate demands”. After the place “demands” get a token, the transition “acquire
capacity” can be fired. It produces two tokens which one is delivered to the place “capacity order” and the other is returned to the place “demands”. And then the transition “plan & control” can be fired. It products three tokens and deliver them respectively to the place “assembly plan”, “demands” and “capacity order”. Now the transition “execute assembly” can be fired. Finally, the transition “report” records all information and produces a token for the place “statistic”.

In the next sections, the sub-pages for the transitions “initialize”, “generate demands”, “acquire capacity”, “plan & control”, “execute assembly”, and “report” will be clarified.

4.4.2 Initialize

As the first step of the whole simulation environment, ‘Initialize’ is executed at the beginning of each time running this simulation environment. Unlike other transitions, it is only executed for one time. The main task of “initialize” is to set the parameters of the environment, such as unit cost for each kinds of capacity (see Appendix D), start run time for each sub-page, and the quantity of Specialist Capacity. After completing “initialize”, the real simulation process can start.

4.4.3 Generate demands

Lu Xi (2005) designed and implemented this sub-page. In this sub-page, the only task is to simulate orders arrival which includes the arrival time and the order volume. We define the order arrival follows Poisson distribution and the order volume follows Erlang distribution.

4.4.4 Acquire capacity

This is a high level planning module, and responsible for the task of making aggregated planning. We define lead times and planning horizon for this level is one month which means the planning made at the beginning of this month will be executed in next month. Lu Xi (2005) described this sub-page in more detail.

4.4.5 Plan & control

Figure 4.4 shows diagram of Operational Control level of the simulation environment implemented by CPN Tools. The start time of this sub-page is 7*week+1 which have already been defined in transition “initialize”. That is due to the lead times for each order in Engineering and Components phases is 8*week, the detailed planning horizon is 1*week and the system start time is 1, so the start time of making detailed planning should be no earlier than (8*week -1*week+1).
This sub-page contains four transitions which construct the whole process of making detail planning. First two transitions are used to filter data obtained from the transition “Acquire capacity” and “generate demands”, and produce new tokens saved in the place “T_capa_order” and “T_demands”. System time is the criterion to measure the data is useful or not. For instance: if the value of the token in “capacity order” is [(117, 13), (145, 13)] which means in time 117 and 145, the ORC should be reach level of 13. And if the current system time is 113 which mean the system need to making planning for time 113+1*week equals to 120. Then the place “T_capa_order” will get the token with value [(117, 13)] rather then [(145, 13)], because only [(117, 13)] can express the ORC level of time 120. The value of the token in the place “demands” includes real demands in the future up to several weeks, but the system only needs the demands of next week in order to make planning. So the place “T_demands” will get a token with value of demands for next week.

After that, the transition “translate” is enabled. On the one hand, it calculates how much capacity is going to be allocated, and the result is saved as a token in the place “T_capacity”. On the other hand, it reorders all the orders according the priority which has been discussed in Section 4.3. “Translate” has four inputs: place “T_capa_order”, “capacity”, “T_demands” and “left_demands”; and two outputs: place “T_demands” and “T_capacity”

The place “T_capacity” contains the information about the amount and changing situation of each kind of capacity. It defined as product SPE_C*OWN_C*CON_C. SPE_C is defined as product VOL*VOL. First VOL represents the quantity of specialist capacity at the beginning of current period e.g. one week. Second VOL expresses the quantity of specialist capacity at the ending of current period. OWN_C and CON_C are defined as product VOL*VOL. The last VOL is the amount change for the capacity in current period, and the remainders are similar as VOL in SPE_C. Capacity is also described as the number of hours working.
The place “T_capacity” is derived from the place “T_capa_order” and “capacity”. The token in the place “T_capacity” expresses how much capacity can do regular work. As mentioned before, it is composed of ORC and the extra SC which do not need to do regular work. The place ‘T_demands’ is derived from the place “T_demands” and “left_demands”. The token in the place “T_demands” is a list of all orders, which have already been started but not finished yet, and the sequence is based on the priority of each order. The order with higher priority is set in front of this list. The order priority only depends on the lead times, which means the shorter of the lead times, the higher of the priority.

Finally, system will run the transition “dis_Capacity”, which has two inputs: the place “T_demands” and “T_capacity”; and two outputs: the place “assembly plan” and “left_demands”. The transition “dis_Capacity” is a complicated process, so we create a substitution transition in order to show this complicated process.

Figure 4.5 shows the sub-page of the transition “dis_Capacity” which shows the detail of how to distribute capacity to each order. Firstly, the transition “dis_C” distinguishes two situations which one is expressed by the place “capacity of C1” and “demands of C1”, with the condition of enough capacity; and the other is expressed by the place “capacity of C2” and “demands of C2”, with the condition of shortage of capacity. Based on these two situations, the transition “dis_C1” and “dis_C2” will calculate how much CC is required and how to fulfill each order respectively. Afterward, the transition ‘combine’ will combine the results obtained from two situations. The transition ‘combine’ has four outputs: the place “left_demands” presents the parameters of uncompleted orders in order to make the following period planning cycle; the place “available capacity”, “capa of each order” and “current condition of each order” provide parameters to the transition “cal_Result” to decide how to distribute capacity to each order. The final result is shown in place “assembly plan”
4.4.6 Execute assembly

In this sub-page, according to the planning of both high level and bottom level, system will simulate the situation of order fulfillment and capacity level change. The result will be delivered to “Report”.

4.4.7 Report

This sub-page is the last step of each process cycle which responsible for calculating average cost for each period, and the utilization for Specialist Capacity and Own Regular Capacity. In Lu Xi (2005), a more detailed description is given.
4.5 Simulation results and analysis

In this section, we will firstly test the validation of our CPN model based on several groups of statistics. These statistics are related to the changing of demands. After that, we will test the execution speed of this CPN model.

4.5.1 Validation

Case 1
The CPN model designed in this case can be used to calculate the average cost of the capacity and the utility ratio of each capacity. As for each order, it has a fixed 12 weeks lead time, and all the plans e.g. aggregated plan and detailed plan are made at least four weeks after received an order. So we can know the uncertainty of the demands will not affect the result e.g. average cost and utility ratio. It implies that the modification of SD (demand deviation) will not change the result.

| Parameters setting: simulation length = 520 (10 years); interarrival = 7; SC maintaining cost = 10; ORC maintaining cost = 5; ORC changing cost = 3; CC maintaining cost = 8; CC changing cost = 1; SC initial volume = 5; ORC initial volume = 10; ED = 20; |
|---|---|---|---|---|---|
| SD=1 | 1 | 2 | 3 | 4 | 5 |
| AC | 160 | 156 | 166 | 158 | 163 | 160.6±2.9 |
| URS | 80 | 82 | 83 | 82 | 79 | 81.2±1.2 |
| URO | 85 | 87 | 88 | 88 | 86 | 86.8±1.0 |
| SD=2 | 1 | 2 | 3 | 4 | 5 |
| AC | 163 | 158 | 166 | 168 | 165 | 164±2.8 |
| URS | 81 | 84 | 81 | 81 | 79 | 81.2±1.3 |
| URO | 88 | 89 | 88 | 88 | 86 | 87.8±0.8 |
| SD=5 | 1 | 2 | 3 | 4 | 5 |
| AC | 162 | 155 | 163 | 157 | 169 | 161.2±4.0 |
| URS | 81 | 80 | 80 | 79 | 81 | 80.2±0.6 |
| URO | 87 | 86 | 88 | 86 | 87 | 86.6±0.7 |
| SD=10 | 1 | 2 | 3 | 4 |
| AC | 164 | 161 | 162 | 155 | 162 | 160.8±2.5 |
| URS | 79 | 79 | 81 | 84 | 79 | 80.4±1.6 |
| URO | 85 | 85 | 87 | 89 | 86 | 86.4±1.2 |

Table 4.1

<table>
<thead>
<tr>
<th>AC</th>
<th>URS</th>
<th>URO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[157.7,163.5]</td>
<td>[80.0,82.4]</td>
</tr>
<tr>
<td>2</td>
<td>[162.2,166.8]</td>
<td>[79.9,82.5]</td>
</tr>
<tr>
<td>3</td>
<td>[157.2,165.2]</td>
<td>[79.6,80.8]</td>
</tr>
<tr>
<td>4</td>
<td>[158.3,163.3]</td>
<td>[78.8,82.0]</td>
</tr>
</tbody>
</table>

Table 4.2

In Table 4.1, we try to test if the modification of SD will change the result, e.g. the
average cost (AC), the utility ratio of SC (URS) and the utility ratio of ORC (URO). For the parameters setting, one rule [22] need to be paid attention to is that the ORC maintaining cost should be more expensive than CC maintaining cost and ORC changing cost should be cheaper than CC changing cost.

We set four different values to SD, and get four groups of results with 90% confidence interval shown in Table 4.1. After analysis, we find their mean is almost equal and their confidence interval is overlap. It indicates that in our model the modification of SD will not changing the result, which is coincide with the idea we mentioned above.

**Case 2**

It is well known that if the demand is increase, the totally cost will increase as well. In normal case, it also leads to the higher utility rate of each kind of capacity. So we can make a hypothesis that if our model is correct, the results (AC, URS and URO) will be increased with the demands increasing. The demands can be represented by two terms, one of which is the volume of the order and the other is the frequency of the order. In this case we will test the changing of “ED” (mean of demand) at beginning, and then followed by to test the changing of “interarrival” (frequency of order).

| Parameters setting: simulation length = 520 (10 years); interarrival = 7; SC maintaining cost = 10; ORC maintaining cost = 5; ORC changing cost = 3; CC maintaining cost = 8; CC changing cost = 1; ORC initial volume = 10; SD = 5; |
|---|---|---|---|---|---|---|---|
| ED=20 | SC=5 | AC | 162 | 155 | 163 | 157 | 169 | 161.2±4.0 |
| URS | 81 | 80 | 80 | 79 | 81 | 80.2±0.6 |
| URO | 87 | 86 | 86 | 86 | 87 | 86.4±0.4 |
| ED=12 | SC=3 | AC | 101 | 94 | 100 | 100 | 98 | 98.6±2.1 |
| URS | 77 | 76 | 79 | 77 | 76 | 77.0±0.9 |
| URO | 85 | 85 | 86 | 86 | 85 | 85.4±0.4 |
| ED=24 | SC=6 | AC | 200 | 198 | 196 | 183 | 193 | 194±4.9 |
| URS | 82 | 81 | 84 | 82 | 83 | 82.4±0.8 |
| URO | 88 | 86 | 89 | 89 | 88 | 88.0±0.9 |

**Table 4.3**

| Parameters setting: simulation length = 520 (10 years); interarrival = 7; SC maintaining cost = 10; ORC maintaining cost = 5; ORC changing cost = 3; CC maintaining cost = 8; CC changing cost = 1; ORC initial volume = 10; SD = 5; |
|---|---|---|---|---|---|
| 1 | 157.2,165.2 | 79.4,80.8 | 86.0,86.8 |
| 2 | 96.5,100.7 | 76.1,77.9 | 85.0,85.8 |
| 3 | 189.1,198.9 | 81.6,83.2 | 87.1,88.9 |
| Result | No overlap | No overlap | No overlap |

**Table 4.4**
Parameters setting: simulation length = 520 (10 years); SC maintaining cost = 10; ORC maintaining cost = 5; ORC changing cost = 3; CC maintaining cost = 8; CC changing cost = 1; SC initial volume = 5; ORC initial volume = 10; ED = 20; SD = 5;

<table>
<thead>
<tr>
<th>interarrival</th>
<th>AC</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Final result(90% conf. interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>l = 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>162</td>
<td>155</td>
<td>163</td>
<td>157</td>
<td>169</td>
<td>161.2±4.0</td>
</tr>
<tr>
<td></td>
<td>URS</td>
<td>81</td>
<td>80</td>
<td>80</td>
<td>79</td>
<td>81</td>
<td>80.2±0.6</td>
</tr>
<tr>
<td></td>
<td>URO</td>
<td>87</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>87</td>
<td>86.4±0.4</td>
</tr>
<tr>
<td>l = 14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>105</td>
<td>98</td>
<td>103</td>
<td>104</td>
<td>103</td>
<td>102.6±2.0</td>
</tr>
<tr>
<td></td>
<td>URS</td>
<td>65</td>
<td>65</td>
<td>66</td>
<td>63</td>
<td>60</td>
<td>63.8±1.8</td>
</tr>
<tr>
<td></td>
<td>URO</td>
<td>71</td>
<td>72</td>
<td>74</td>
<td>70</td>
<td>67</td>
<td>70.8±1.9</td>
</tr>
<tr>
<td>l = 21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>84</td>
<td>90</td>
<td>85</td>
<td>86</td>
<td>90</td>
<td>87.0±2.1</td>
</tr>
<tr>
<td></td>
<td>URS</td>
<td>55</td>
<td>58</td>
<td>51</td>
<td>59</td>
<td>56</td>
<td>55.8±2.3</td>
</tr>
<tr>
<td></td>
<td>URO</td>
<td>61</td>
<td>65</td>
<td>58</td>
<td>64</td>
<td>61</td>
<td>61.8±2.0</td>
</tr>
<tr>
<td>l = 28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>82</td>
<td>79</td>
<td>79</td>
<td>74</td>
<td>77</td>
<td>78.2±2.2</td>
</tr>
<tr>
<td></td>
<td>URS</td>
<td>49</td>
<td>49</td>
<td>50</td>
<td>45</td>
<td>50</td>
<td>48.6±1.5</td>
</tr>
<tr>
<td></td>
<td>URO</td>
<td>57</td>
<td>53</td>
<td>55</td>
<td>51</td>
<td>57</td>
<td>54.6±1.9</td>
</tr>
</tbody>
</table>

Table 4.5

<table>
<thead>
<tr>
<th></th>
<th>AC</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[157.2, 165.2]</td>
<td>[79.4, 80.8]</td>
<td>[86.0, 86.8]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>[100.6, 104.6]</td>
<td>[62.0, 65.6]</td>
<td>[68.9, 72.7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>[84.9, 89.1]</td>
<td>[53.5, 58.1]</td>
<td>[59.8, 63.8]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[76.0, 80.4]</td>
<td>[47.1, 50.1]</td>
<td>[52.7, 56.5]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>No overlap</td>
<td>No overlap</td>
<td>No overlap</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6

Obviously, the results shown in Tables 4.3-4.6 are totally in line with the hypothesis which is proposed before.

Based on the case 1 and case 2, we validate our model from two aspects. However it can prove our model is completely correct. If we want to further evaluate our model, we can use other ways. For instance, a state space analysis is a good approach to validate a CPN model; or we can use our model in practice to test if it is correct, e.g. employ it in a company. But in our thesis, we will stop at here.

4.5.2 Speed

Besides the validation, we also test the response time of this CPN model. The result is shown in the Table 4.2. From the result, we can notice that the performance of CPN application is pretty preferable. All the simulation spends less than 6 seconds even if we expand the simulation length to 2000.
The test environment:
Hardware: Intel Pentium 4(M) 2.00GHz; 512MB DDR
Software: Windows XP Professional; CPN Tools v.1.4.0
Parameters setting is same as in Table 4.1

<table>
<thead>
<tr>
<th>run</th>
<th>200</th>
<th>300</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (1)</td>
<td>0.91</td>
<td>1.18</td>
<td>1.47</td>
<td>2.64</td>
<td>5.53</td>
</tr>
<tr>
<td>time (2)</td>
<td>0.93</td>
<td>1.13</td>
<td>1.44</td>
<td>2.62</td>
<td>5.62</td>
</tr>
<tr>
<td>time (3)</td>
<td>0.97</td>
<td>1.15</td>
<td>1.50</td>
<td>2.69</td>
<td>5.42</td>
</tr>
<tr>
<td>time (4)</td>
<td>0.94</td>
<td>1.20</td>
<td>1.49</td>
<td>2.58</td>
<td>5.50</td>
</tr>
<tr>
<td>time (5)</td>
<td>0.97</td>
<td>1.15</td>
<td>1.52</td>
<td>2.70</td>
<td>5.72</td>
</tr>
<tr>
<td>average</td>
<td>0.94</td>
<td>1.16</td>
<td>1.48</td>
<td>2.65</td>
<td>5.56</td>
</tr>
</tbody>
</table>

Table 4.7
In Appendix E, we have proven that the response time of CPN model follows linear distribution, and we will prove it again for this model. Using these statistics, we only want to verify that the execution speed of the CPN model is acceptable.

4.6 Conclusion

In this chapter, we described CPN Tools in modeling a HPP concept in the context of Engineer-to-Order situation. This developed CPN model can be used in a quite generic situation and the related statistics discussed in the Section 4.5 can prove that our model is correct. The test for the response time is also acceptable. Now we can get the conclusion that CPN Tools can be used in modeling HPP in logistics concept.

In the next chapter, we will further prove our point based on the evaluation of CPN Tools.
5. Evaluations

In this chapter, we make an evaluation for CPN Tools in order to eventually give the answer to the question which introduced at the beginning of this project: whether CPN Tools can be used for the development of a hierarchical modeling environment for logistics control concepts.

The goodness and efficacy of an object can be rigorously demonstrated via well-selected evaluation methods [18, 19]. The selection of evaluation methods must be matched appropriately with the evaluation object, as well as the participant of the evaluation. In our evaluation, we choose observational and experimental method [4] to evaluate CPN Tools.

This chapter is organized as follows: Section 5.1 and Section 5.2 discuss the CPN Tools evaluation via observational method and experimental method respectively. Section 6.3 draws the final conclusion of this report.

5.1 Observational method

Observational method [4] is an evaluation technique, in which one or more persons watch the performance, utility and efficacy of an evaluation object and record perceptions on a measurement tool. It provides useful insight with little expertise required. Counter to its simplicity, as the observation is influenced by observer, subjectivity sometimes will make the evaluator get wrong conclusion. Since observational method is the simplest method using in evaluation, we will first use it to evaluate the CPN Tools.

Based on the observation of the model developed by CPN, a strong feeling is generated that it is particularly well suited for complex systems due to it has an intuitive graphic representation.

The Figure 5.1 shown below is the interface of Delphi application which we use CPN Tools to mimic. Obviously, it can only provide the result of each simulation run (e.g. average cost). In contrast, due to CPN Tools provide a way which can simulate a working process step by step, the application implemented by CPN Tools can describe each step of the whole working process quite clearly, as well as the state of each component which is composed of the whole system. For instance, in first test case, the user can monitor inventory level change situation in anytime; in second test case, as planning level involve multiple planning decision, it is really important to reflect lead time and result for each planning decision. CPN’s graphical representation can cope with it nicely. So based on the observational method, we can conclude that CPN Tools...
is good at modeling HPP in logistics concepts as it not only can calculate the final result but also can depict the intermedia step.

5.2 Experimental method

Experimental method [4] is an evaluation technique, in which the participants judge the performer, utility and efficacy of an evaluation object based on their experience and the theoretical knowledge. In this section, we will make use of experimental method to assess the strength and weaknesses of CPN Tools in modeling and analysis of HPP concept. Our experience is obtained during the two CPN models design and implementation.

According our experience we conclude the advantage of CPN Tools using in HPP as follows:

1. **Good performance on data expression**
   We have already clarified that in order to simulate a logistics environment, we need to deal with the expression of thousands of products, hundreds of order etc. As CPN adds a certain data value to each token and each data value belongs to a given type, which make it is possible to represent different products or different orders.

2. **Strong computation power**
CPN Tools make use of CPN ML programming language which leads to the fact that it has a strong computation power. For each of our two developed models, it requires to solve one or more optimization models. As we have already validated our models and get the result that the execution speed is acceptable, we can point out that CPN Tools solve these optimization models pretty well, which means it has enough capability to deal with this kind of mathematics problems.

3. The support of modeling time.
As discussed in the Chapter 2, in HPP models, a key criterion to separate the whole planning problem is depending on the length of planning horizon and time. According to this, we can notice that the time of each activity in these models is very important. For instance, in E2O design, the lead time of aggregated planning is four weeks and the lead time of detailed planning is one week. As CPN can define the delay of each activity, we can easily express that how long each kind of planning take. Another example is the detailed planning should take place three weeks after an aggregated planning was made. Using CPN Tools, this problem can also be easily solved.

5.3 Conclusions and future work

In this thesis, we have tried to answer the question whether CPN Tools can be used in modeling HPP problem in logistics concepts, through the design and implementation of the two CPN models. In Chapter 2, we proposed three difficulties in modeling HPP problem. Through the design and implementation of a CPN model for E2O case, we found that each of these difficulties is overcome by CPN Tools.

Based on the experience obtained in the study process about CPN Tools, we can conclude that CPN Tools has the following advantage which is particular suitable for modeling HPP problem:
1. The support of making hierarchical model and modeling time of activity make it possible to simulate HPP concept.
2. Intuitive graphical presentation let the user can clearly observe each step of the whole working process.
3. The support of lot of data type expression and the combination of CPN ML programming language increases the expressive power and computational power of the HPP model.

Of course, there should be some other features for CPN Tools. As the limitation of our research, we can only leave it as the future work. For instance we can make a comparison between CPN Tools and other simulation tools (e.g. Arena), and then to discuss their pros and cons. As in our project, what we have designed are generic concepts which are independent of any specific tool. They should be also suitable for other simulation tools. Another future work is to verify if this hypothesis is correct.
We can see that a lot of works are still remaining, nevertheless by our CPN model we think we have already reach our goal. CPN Tools can be used for the development of a hierarchical modeling environment for logistics control concepts.
References


Appendix A: source code for first CPN application

**Standard declarations**
- `colset E = with e;`
- `colset INT = int;`
- `colset INTs = list INT;`
- `colset BOOL = bool;`
- `colset STRING = string;`
- `colset U = unit;`

**Simulation declarations**
- `colset No = int;`
- `colset Nos = list No;`
- `var i: INT;`
- `colset STOCK = record Number: No* inv: INT * bak: INT;`
- `colset STOCKs = list STOCK;`
- `colset SS = record Number: No * h: INT * p: INT;`
- `colset SSs = list SS;`
- `colset safe_stock = product No*INT;`
- `colset safe_stocks = list safe_stock;`
- `colset PLI = record Number: No * Pro: INT;`
- `colset PLIs = list PLI;`
- `colset Dem_Actual = record Number: No * Dem: INT;`
- `colset Dem_Actuals = list Dem_Actual;`
- `colset Dem = product INT*Dem_Actuals;`
- `colset Dems = list Dem;`
- `var d: Dems;`
- `colset RESULT = record no: INT * cost: INT;`
- `colset STAT = product INT*INT;`
- `colset UT = U timed;`
- `colset H = list INT;`
- `colset P = list INT;`
- `var u: U;`
- `var numbers: Nos;`
- `var s: STOCKs;`
- `var dem_acs: Dem_Actuals;`
- `var d1, d2: Dem_Actuals;`
- `var pli: PLIs;`
- `var rst: RESULT;`
- `var cost: INT;`

**Simulation Functions**
- `fun init_stock(n:Nos) =`
  - `if length(n)>0 then [([Number=hd(n), inv=0, bak=0])]`^`
  - `init_stock(tl(n))`
else[];

fun init_cata(x:INT,y:INT) =
if x=0 then []
else [y-x+1] ^^
init_cata(x-1,y);

fun init_result(l:Nos)=
{no=0, cost=0}

fun vol_gen() =
round(erlang(round(Math.pow(E1,2.0)/Math.pow(s1,2.0)), E1/Math.pow(s1,2.0)), E1/Math.pow(s1,2.0));

fun gen_demand(x:Nos)=
if length(x)>0
then [{Number=hd(x), Dem=vol_gen()}] ^^
gen_demand(tl(x))
else[];

fun fore(x:Dem_Actuals,y:real)=
if length(x)>0
then [{Number=(#Number(hd(x))), Dem=round((1.0+normal(0.0,y*y))*real((#Dem(hd(x)))))}] ^^
fore(tl(x),y)
else[];

fun fore_De(x:Dems,y:real)=
if length(x)=0 then []
else if (#1(hd(x))) = (IntInf.toInt(time()) div 7)+ 1 orelse
(#1(hd(x))) = (IntInf.toInt(time()) div 7)+ 2
then [(#1(hd(x)),fore((#2(hd(x))),y))] ^^ fore_De(tl(x),y)
else [] ^^ fore_De(tl(x),y);

fun sum(x: Dem_Actuals,y:safe_stocks)=
if length(x)=0 then 0
else (#Dem(hd(x)))+(#2(hd(y)))+sum(tl(x),tl(y));
fun cal_p(x: PLIs, y: H) = 
  if length(x) = 0 then 0 
  else (#Pro(hd(x))) * hd(y) + cal_p(tl(x), tl(y));

fun posi(x: INT, y: INT) = 
  if x > y then x - y 
  else 0;

fun demf(x: STOCKs, y: Dem_Actuals, z: Dem_Actuals) = 
  if length(z) = 0 then [] 
  else [{Number = (#Number(hd(z))), 
          Dem = posi((#Dem(hd(z))), posi((#inv(hd(x))), (#Dem(hd(y))))), 
          Inv = (#inv(hd(y)))}] 
  ^^ demf(tl(x), tl(y), tl(z));

fun cal(x: Dem_Actuals, y: STOCKs) = 
  if length(x) = 0 orelse length(y) = 0 then [] 
  else [{Number = (#Number(hd(x))), 
          Dem = posi((#Dem(hd(x)))+(#inv(hd(y))), (#Dem(hd(y))))}, 
          Inv = (#inv(hd(y)))] 
  ^^ cal(tl(x), tl(y));

fun dis_A(x: Dem_Actuals, s: safe_stocks) = 
  if length(x) = 0 then [] 
  else [{Number = (#Number(hd(x))), Pro = (#2(hd(s))) + (#Dem(hd(x)))}, 
          Pro = (#2(hd(s))) ] 
  ^^ dis_A(tl(x), tl(s));

fun dis_B_1(x: Dem_Actuals, y: Dem_Actuals, z: INT, s: safe_stocks) = 
  if length(x) > 0 and also z - (#Dem(hd(y)))- (#2(hd(s))) >= 0 
  then [{Number = (#Number(hd(x))), Pro = (#Dem(hd(x))) + (#2(hd(s))) + (#Dem(hd(y)))}, 
          Pro = (#2(hd(s))) + (#Dem(hd(y))) ] 
  ^^ dis_B_1(tl(x), tl(y), z - (#2(hd(s))) - (#Dem(hd(y))), tl(s)) 
  else if length(x) > 0 
  then [{Number = (#Number(hd(x))), Pro = (#Dem(hd(x))) + (#2(hd(s))) + z}, 
          Pro = (#2(hd(s))) + z ] 
  ^^ dis_B_1(tl(x), tl(y), 0, tl(s)) 
  else [];

fun dis_B(x: Dem_Actuals, y: Dem_Actuals, z: INT, s: safe_stocks) = 
  if z - sum(x, s) >= sum(y, s) - z 
  then dis_B_1(x, y, sum(y, s) - z, s) 
  else dis_B_1(x, y, z - sum(x, s), s);

fun dis_D(x: Dem_Actuals, z: INT, s: safe_stocks) = 
  if length(x) > 0 and also z - (#Dem(hd(x)))- (#2(hd(s))) >= 0 
  then [{Number = (#Number(hd(x))), Pro = (#Dem(hd(x))) + (#2(hd(s)))}, 
          Pro = (#2(hd(s))) ] 
  ^^ dis_D(tl(x), z - (#Dem(hd(x))) - (#2(hd(s))), tl(s)) 
  else if length(x) > 0 and also z - (#Dem(hd(x)))- (#2(hd(s))) < 0
fun dis(x:Dem_Actuals,y:Dem_Actuals,z:INT,s:safe_stocks)=
if z>=sum(x,s) andalso z>=sum(y,s)
then dis_A(x,s)
else if z>=sum(x,s) andalso z<sum(y,s)
then dis_B(x,y,z,s)
else if z<sum(x,s)
then dis_D(x,z,s)
else [];

fun stock_upd(x: STOCKs, y: PLIs, z: Dem_Actuals)=
if length(z)>0 then
[{
  Number=(#Number(hd(x))), inv=
  posi((#Pro(hd(y)))+(#inv(hd(x))),(#bak(hd(x)))+(#Dem(hd(z)))),
  bak=posi(#bak(hd(x)))+(#Dem(hd(z))),(#Pro(hd(y)))+(#inv(hd(x))))}
  ^^stock_upd(tl(x),tl(y),tl(z))
else [];

fun cost_cacu(x: STOCKs,y:H,z:P)=
if length(x)>0 then
(#inv(hd(x)))*hd(y)+(#bak(hd(x)))*hd(z)
+cost_cacu(tl(x),tl(y),tl(z))
else 0;

fun result_fin(x: RESULT, y: INT)=
{no=(#no(x)+1), cost=y};

fun gen_real(x: Dems)=
if length(x)= 0 then []
else [#2(hd(x))];
Appendix B: source code for second CPN application

Declarations

Standard declarations

colset E = with e;
colset ET = E timed;
colset BOOL = bool;
colset INT = int;
colset STRING = string;
val week = 7;

Parameter settings

val t_pl2exe = week;
val t_cap2exe = 4*week;
val interarrival = 7.0;
val ec_lt = 8;
val ass_lt = 4*week;
val ED = 20.0;
val SD = 5.0;
val cap_co = 3;
val cap_init = ((5,5),(10,10,0),(0,0,0));
val uc_init = (10,5,3,4,1);

Simulation declarations

Colors

colset D = int;
colset WK = int;
colset VOL = int;
colset ID = int;
colset IT = ID timed;
colset MON = int;
colset PERCENT = int;
colset ORD = product ID*VOL*VOL*D*D;
colset ORDs = list ORD;
colset SPE_C = product VOL*VOL;
colset OWN_C = product VOL*VOL*VOL;
colset CON_C = product VOL*VOL*VOL;
colset CAP = product SPE_C*OWN_C*CON_C;
colset CAP_ORD = product D * VOL;
colset CAP_ORDs = list CAP_ORD;
colset ASS_PL = product ID*VOL*VOL*VOL*VOL;
colset ASS_PLS = list ASS_PL;
colset OPR_PL = product D*VOL*ASS_PLS;
colset OPR_PLS = list OPR_PL;
colset WT = product ID*VOL*VOL*D;
colset WTs = list WT;
colset WIP = product ID*VOL*VOL*VOL*D;
colset WIPs = list WIP;
colset HIS = product ID*D;
colset HISs = list HIS;
colset COS = product MON*MON*MON*MON*MON;
colset COS_T = product WK*MON;
colset UTIL = product WK*PERCENT*PERCENT;
colset STAT = product WK*MON*PERCENT*PERCENT;
colset CAP_TEM = product VOL*VOL*VOL;
colset CAP_DIS = product ID*VOL*VOL;
colset CAP_DISs = list CAP_DIS;
colset SPE = VOL;

Variables
var i,j,k : ID;
var t,t1,t1 : D;
var v,v1,v2,v3,v4,v5 : VOL;
var u_cos : COS;
var ol1,ol2,ol3,ol4 : ORDs;
var wt : WTs;
var col,col1,col2,col3 : CAP_ORDs;
var opl : OPR_PL;
var cap : CAP;
var cos : COS_T;
var wip : WIPs;
var his : HISs;
var m : MON;
var p1,p2 : PERCENT;
var stat : STAT;
var ct : CAP_TEM;
var cd : CAP_DISs;
var spe : SPE;
var leadtime,a : INT;

Globals
globref cost_stat = 0.0;
globref util_stat1 = 0.0;
globref util_stat2 = 0.0;

Functions
fun interarrivaltime() =
round(exponential(1.0/interarrival));
(((IntInf.toInt(time()) div 7)+ec_lt)*7)+5;

fun vol_gen() =
  round(erlang(round(Math.pow(ED,2.0)/Math.pow(SD,2.0)), ED/Math.pow(SD,2.0)));

fun ord_gen(x:ID, y:VOL) =
  [(x, y div 10, y, st_gen(), ass_lt)];

fun scope([], ORDs) = [] |
scope(xi::x) =
  if (#4xi>(IntInf.toInt(time())+4*week))
    andalso (#4xi<(IntInf.toInt(time())+8*week))
  then xi::scope(x)
  else scope(x);

fun cap_fore(x:CAP_ORDs)=
  if length(x)>0 then
    if #1(hd(x)) = (IntInf.toInt(time())+4)
      then (#2(hd(x))) else cap_fore(tl(x))
  else #1(#2cap_init);

fun dem_sum([], ORDs) = 0 |
  dem_sum(xi::x) = #3xi + dem_sum(x);

fun acq_cap(x:VOL, y:VOL)=
  if Int.abs((x div 4)-y) <=  cap_co then y
  else if (x div 4) > (y+cap_co) then y+cap_co
  else y-cap_co;

fun capo_gen(x:VOL)=
  [((IntInf.toInt(time())+4+4*week),x)];

fun tran3(x:ORDs)=
  if length(x)=0 then 0
  else if #4(hd(x))=IntInf.toInt(time())+11
    then (#2(hd(x)))+tran3(tl(x))
  else 0+tran3(tl(x));

fun tran (x:ORDs, y:ORDs)=
  if length(x)=0 and also length(y)=0 then []
  else if length(y)=0 then tran(tl(x),[hd(x)])
  else if length(x)=0
    then y
  else ( #5(hd(x)))<=(#5(hd(y)))
then tran(tl(x),hd(x)::y)
else tran(tl(x),hd(y)::tran([hd(x)],tl(y)));

fun tran1 (x:ORDs)=
if length(x)>0
then [((#1(hd(x))),0:VOL,0:VOL,0:VOL,(#3(hd(x))))] ^^ tran1(tl(x))
else [];

fun tran2(x: ORDs)=
if length (x) = 0 then []
else if (#4(hd(x)))= IntInf.toInt(time())+11
then [hd(x)] ^^ tran2(tl(x))
else [] ^^ tran2(tl(x));

fun tran4([],ORDs)=[[]]
tran4(xi::x)=
[(#1xi),(#2xi),(#5xi)] ^^ tran4(x);

fun translate([],ORDs)=[[]]
translate (xi::x)=
[ {ID=(#1xi),SPE=(#2xi),REG=(#3xi),Start=(#4xi),Lead=(#5xi) } ] ^^ translate(x);

fun cal(x:CAP_ORDs)=
if length(x)=0 then 0
else #2(hd(rev(x)));

fun aver(x:ORD)=
(#3(x)) div ((#5(x)) div 7);

fun sumaver(x:ORDs)=
if length(x)>0
then (#3(hd(x))) div ((#5(hd(x))) div 7)+ sumaver(tl(x))
else 0;

fun get_CC(x:ORDs,y:VOL)=
if (y-sumaver(x))>=0 then 0
else if (y-(#3(hd(x))) div ((#5(hd(x))) div 7))>=0 then 0
else (#3(hd(x))) div ((#5(hd(x))) div 7)-y;

fun minus(x:int,y:int) =
if x - y >=0 then x-y
else 0;

fun min(x,y) = if x<y then x else y;
fun get(x:ORDs)=
if length(x)>0 then
    [((#1(hd(x))),(#2(hd(x))),(#3(hd(x)))-aver(hd(x)),(#4(hd(x))),(#5(hd(x)))-7)]
    ^^get(tl(x))
else [];

fun get_A(x:ORDs, y:VOL)=
if length(x)>0 then
    [((#1(hd(x))),(#2(hd(x))),minus((#3(hd(x))),y),(#4(hd(x))),(#5(hd(x)))]
    ^^get_A(tl(x),minus(y,(#3(hd(x)))))
else[];

fun get_C(x:ORDs)=
if length(x)=0 then 0
else if length(x)=1 then aver(hd(x))
else if (#5(hd(x))) = (#5(hd(tl(x))))
then aver(hd(x))+get_C(tl(x))
else aver(hd(x));

fun get_B(x:ORDs,y:INT)=
if length(x) = 0 then 0
else if (#5(hd(x)))=y
then aver(hd(x))\+get_B(tl(x),y)
else 0;

fun get_D(x:ORDs, y:VOL)=
if length(x)>0 then
    [((#1(hd(x))),(#2(hd(x))),minus((#3(hd(x))),min(aver(hd(x)),y)),(#4(hd(x))),(#5(hd(x)))-7)]
    ^^get_D(tl(x),minus(y,(aver(hd(x)))))
else[];

fun update_ORDs(x: ORDs)=
if length(x)=0 then []
else if #3(hd(x))=0
then []^^update_ORDs(tl(x))
else [(#1(hd(x)),0,#3(hd(x)),#4(hd(x)),#5(hd(x)))]^^update_ORDs(tl(x));

fun cal_DIS(x:ORDs,y:ORDs)=
if length(x)>0 then
    [((#1(hd(x))),(#2(hd(x))),(#3(hd(x)))-(#3(hd(y))))]
    ^^cal_DIS(tl(x),tl(y))
else[];

fun A(x:CAP_TEM,y:CAP_DISs)=
  (#1(x)-(#3(hd(y))),#2(x),#3(x));

fun B(x:CAP_TEM,y:CAP_DISs)=
  (0:VOL,(#2(x))-((#3(hd(y)))-(#1(x))),#3(x));

fun C(x:CAP_TEM,y:CAP_DISs)=
  (0:VOL,0:VOL,((#3(x))-((#3(hd(y)))-(#1(x))-(#2(x))));

fun cal_dis(x:CAP_TEM,y:CAP_DISs,z:ORDs)=
  if length(y)=0 then []
  else if (#1(x)) >= (#3(hd(y))
  then [(#1(hd(y))),(#2(hd(y))),(#3(hd(y))),0:VOL,(#3(hd(z)))]^cal_dis(A(x,y),tl(y),tl(z))
  else if (#1(x))+(#2(x)) >= (#3(hd(y))
  then [(#1(hd(y))),(#2(hd(y)))+(#3(hd(y)))-(#1(x)),(#1(x)),0:VOL,(#3(hd(z)))]^cal_dis(B(x,y),tl(y),tl(z))
  else if (#1(x))+(#2(x))+(#3(x)) >= (#3(hd(y))
  then [(#1(hd(y))),(#2(hd(y)))+(#3(hd(y)))-(#1(x))+(#1(x)),((#3(hd(y)))-(#1(x))-(#2(x))),(#3(hd(z)))]^cal_dis(C(x,y),tl(y),tl(z))
  else [(#1(hd(y))),(#2(x)),(#1(x)),(#3(x)),(#3(hd(z)))];

fun guard(x:time)=
  IntInf.toInt(x)<>22 andalso (IntInf.toInt(x)-22) mod 28 <> 0;

fun guard1(x:time)=
  IntInf.toInt(x) mod 7 = 0;

fun prepare([]:CAP_ORDs) = [] |
prepare(xi::x) =
  if Int.toString(#1xi)=Time.toString(time())
  then xi::prepare(x)
  else prepare(x);

fun update(x:CAP_ORDs, y:VOL) =
  if length(x)=0 then y else #2(hd(x));

fun return([]:CAP_ORDs) = [] |
return(xi::x) =
  if Int.toString(#1xi)=Time.toString(time())
  then return(x)
else xi::return(x);

fun nr_bak([], ORDs) = [] |
    nr_bak(xi::x) =
    if Int.toString(#4xi)=Time.toString(time())
        then nr_bak(x)
        else xi::nr_bak(x);

fun get_asspl(x: OPR_PL) = #3x;

fun get_cc(x: OPR_PL) = #2x;

fun cap_up(x: CAP, y: CAP_ORDs, z: OPR_PL) =
    (#1x, 
    (update(prepare(y), #1(#2x)), update(prepare(y), #1(#2x)),
    update(prepare(y), #1(#2x))-(#1(#2x)),
    (get_cc(z), get_cc(z), get_cc(z)-(#1(#3x))));

fun trans_asspl([], ASS_PLs, y: WT) = [] |
    trans_asspl(xi::x, y) =
    if #1xi = (#1y) then [(#1xi, #2xi, #3xi, #4xi, #5xi, #4y)] else trans_asspl(x, y);

fun trans_asspl2([], ASS_PLs, y: WIP) = [] |
    trans_asspl2(xi::x, y) =
    if #1xi = (#1y) then [(#1xi, #2xi, #3xi, #4xi, #5xi, #6y)] else trans_asspl2(x, y);

fun comp_asspl([], ASS_PLs, y: ID) = false |
    comp_asspl(xi::x, y) =
    if #1xi = y then true else comp_asspl(x, y);

fun w_for([], ORDs) = [] |
    w_for(xi::x) =
    if Int.toString(#4xi)=Time.toString(time())
        then (#1xi, #2xi, #3xi, #4xi+(#5xi)-7)::w_for(x)
        else w_for(x);

fun w_bak([], WTs, y: OPR_PL) = [] |
    w_bak(xi::x, y) =
    if comp_asspl(get_asspl(y), (#1xi)) then w_bak(x, y)
    else xi::w_bak(x, y);

fun wip_wt([], WIPs, y: OPR_PL) = [] |
    wip_wt(xi::x, y) =
    if comp_asspl(get_asspl(y), (#1xi)) then wip_wt(x, y)
else (#1xi, 0, #5xi, #6xi)::wip_wt(x,y);

fun wip_for([], WTs,y:OPR_PL) = [] | wip_for(xi::x,y) =
    trans_asspl(get_asspl(y), xi)^^wip_for(x,y);

fun wip_up([], WIPs,y:OPR_PL) = [] | wip_up(xi::x,y) =
    trans_asspl2(get_asspl(y), xi)^^wip_up(x,y);

fun cap_using([], ASS_PLs, y:STRING) = 0 | cap_using(xi::x, y) =
    case y of
        "s" => #2(xi)+cap_using(x,y)
    | "o" => #3(xi)+cap_using(x,y)
    | "c" => #4(xi)+cap_using(x,y);

fun cap_occ(w:CAP, x:VOL, y:VOL, z:VOL)=
( (#1(#1w), #2(#1w)-x),
  (#1(#2w), #2(#2w)-y, #3(#2w)),
  (#1(#3w), #2(#3w)-z, #3(#3w))
);

fun wip_bak([], WIPs) = [] | wip_bak(xi::x) =
    if #5xi = 0
        then wip_bak(x)
    else xi::wip_bak(x);

(* fun cap_used([], WIPS, y:STRING) = 0 |
  cap_used(xi::x, y) =
    case y of
        "s" => #2(xi)+cap_used(x,y)
    | "o" => #3(xi)+cap_used(x,y)
    | "c" => #4(xi)+cap_used(x,y); *)

fun cap_rel(w:CAP, x:VOL, y:VOL, z:VOL)=
( (#1(#1w), #2(#1w)+x),
  (#1(#2w), #2(#2w)+y, #3(#2w)),
  (#1(#3w), #2(#3w)+z, #3(#3w))
);

fun wip_bak([], WIPs) = [] | wip_bak(xi::x) =
    if #5xi = 0
        then wip_bak(x)
    else xi::wip_bak(x);
fun compl([]:WIPs) = [] |
compl(xi::x) =
if #5xi = 0
then (#1xi, IntInf.toInt(time()))::compl(x)
else compl(x);

fun cap_chg_bak(x:CAP)=
(#1x,(#1(#2x), (#2(#2x)), 0),(#1(#3x), (#2(#3x)), 0));

fun cap_util_bak(x:CAP)=
((#1(#1x), #1(#1x)),(#1(#2x), #1(#2x), #3(#2x)),(#1(#3x), (#1(#3x)), #3(#3x)));

fun cost_calc(x:COS,y:CAP)=
((#1x)*(#1(#1y)))+
((#2x)*(#1(#2y)))+
((#3x)*Int.abs(#3(#2y)))+
((#4x)*(#1(#3y)))+
((#5x)*Int.abs(#3(#3y)));

fun util_calc(x:CAP,y:WK)=(y,
(((#1(#1x))-(#2(#1x)))*100)div(#1(#1x)),
(((#1(#2x))-(#2(#2x)))*100)div(#1(#2x)));

fun get_stat(x:real, y:int, z:int) =
(x*real(y-1)+real(z))/real(y);
Appendix C: manual of production planning model

In the following paragraphs you can find how to run the Production Planning Model. We start from loading the CPN model, followed by parameters setting, and ending at the execution of the model. To be able to run the model, you need to have installed at least CPN Tools version 1.4.0.

To load the CPN model for the Production Planning Model, right click and hold the right mouse button in the workspace. The marking menu as depicted in Figure C.1 will appear.

![CPN Tools (Version 1.4.0 - May 2005)](image)

**Figure C.1**

Next move the move cursor to Load Net. Now release the right mouse button. A dialog window will appear where you can choose a CPN model to load. Here we choose the CPN model of the Production Planning Model. Go to the location of the CPN model and choose to open it.

Now that the CPN model has been loaded, the settings for the simulation toolbox have to be changed. To do so, click on the triangle next to Tool box, then on the triangle next to Simulation, and finally on the triangle next to Play. The menu should look like the one depicted in Figure C.2. Now change the number of steps of the simulation to a large number, for instance 100000.
Next left click and hold on Simulation, move the mouse pointer into the work space and then release it. The simulation toolbox appears. Note that the value beneath the play button should be equal to the large number, 100000. The simulation toolbox is depicted in Figure C.3.

Before starting the simulation, parameters setting are necessary. To do so, click on the triangle next to “pp_v3.cpn”, then on the triangle next to “Declarations”, then on the triangle next to “Simulation Initializations”, and finally on the triangle next to each “val”. The menu should look like the one depicted in Figure C.4. Now you can change each parameter of this model.
To start the simulation, in this model, left click on the play button. The mouse pointer now has the play button next to it. Now move the mouse over to the model area as depicted in Figure C.5

Left click the mouse, and the simulation will start.
Appendix D: manual of Engineer-to-Order model

In the following paragraphs you can find how to run the Production Planning Model. We start from loading the CPN model, followed by parameters setting, and ending at the execution of the model. To be able to run the model, you need to have installed at least CPN Tools version 1.2.0. However CPN Tools 1.4.0 is recommended,

To load the CPN model for the Production Planning Model, right click and hold the right mouse button in the workspace. The marking menu as depicted in Figure D.1 will appear.

![Figure D.1](Image)

Next move the move cursor to Load Net. Now release the right mouse button. A dialog window will appear where you can choose a CPN model to load. Here we choose the CPN model of the Engineer-to-Order. Go to the location of the CPN model and choose to open it.

Now that the CPN model has been loaded, the settings for the simulation toolbox have to be changed. To do so, click on the triangle next to Tool box, then on the triangle next to Simulation, and finally on the triangle next to Play. The menu should look like the one depicted in Figure D.2. Now change the number of steps of the simulation to a large number, for instance 100000.
Next left click and hold on Simulation, move the mouse pointer into the work space and then release it. The simulation toolbox appears. Note that the value beneath the play button should be equal to the large number, 100000. The simulation toolbox is depicted in Figure C.3.

Before starting the simulation, parameters setting are necessary. To do so, click on the triangle next to “e2o_4.cpn”, then on the triangle next to “Declarations”, then on the triangle next to “Parameter settings”, and finally on the triangle next to each “val”. The menu should look like the one depicted in Figure D.4. Now you can change each parameter of this model.
To start the simulation, in this model, left click on the play button. The mouse pointer now has the play button next to it. Now move the mouse over to the model area as depicted in Figure C.5

Left click the mouse, and the simulation will start.
Appendix E: Production planning design

This part introduces the CPN Tools by applying it to an existing software tool aiming at planning under non-stationary stochastic demand and finite capacity. Section 1 gives the motivation why we choose this case. Section 2 describes this case in detail. Section 3 expresses the model design which includes architecture design and data structure design, as well as the model implementation. Section 4 shows the simulation results. In Section 5, we give the conclusion of this Chapter.

1 Motivation

To test whether CPN Tools suits for modeling HPP concept, we need to make a comprehensive analysis mainly focused on the performance and functionality of CPN Tools. We choose “Planning under non-stationary stochastic demand and finite capacity” [8] and the related Delphi application as our test case due to several reasons:
1. This case was to get familiar with CPN Tools and to validate if it is feasible to continue with the E2O case which is a real HPP test case.
2. In conceptual level, there exist two known ways to solve this problem, which one is monolithic model upon which solving a complicated Linear Programming (LP) problem is based, and the other is a HPP model.
3. It has already been implemented by Delphi, so that the related application could provide a validation for our evaluation.

2 Case description

In this case, we consider a family of N products that are produced by a production unit. The production unit has a fixed capacity C per time unit (e.g. week). Demand for each product is stationary stochastic and demands in different periods are independent. It is defined as:

\[ d_i(t) \] demand for product \( i \) in period \( t \), \( i=1,\ldots,N, t \geq 1 \)

\[ D_i \] demand for product \( i \) in an arbitrary period, \( i=1,\ldots,N \)

The actual demand \( d_i(t) \) is mixed-Erlang distributed with mean \( E[D_i] \) and standard deviation \( \sigma(D_i), i=1,\ldots,N \).

The objective is to take decisions of how much to produce of each product so that average costs per time unit are minimized. The cost includes holding costs \( h_i \) and shortage costs \( p_i \) as follows:

\[ h_i \] cost per product \( i \) in inventory at the end of an arbitrary period, \( i=1,\ldots,N \)
\[ p_i \] cost per product \( i \) short at the end of an arbitrary period, \( i=1,\ldots,N \)
The planning model has already been implemented by Delphi [8]. Developers adopted a rolling schedule approach.

- At the beginning of each period, the state of the system can be observed, i.e. the actual net stocks of the products are known.
- Next the forecast of demand in future periods up to some horizon can be made. Forecast is based upon this formulation:

$$d_i(t, t + \tau) = (1 + f_i(t, t + \tau))d_i(t + \tau), t \geq 1, \tau \geq 0 \quad \text{formula (1)}$$

$$d_i(t, t + \tau) \quad \text{forecast of demand for product } i \text{ in period } t+\tau, \text{ made at the start of period } t, \ i=1,\ldots,N, \ \tau \geq 1$$

$$f_i(t + \tau) \quad \text{relative forecast error of demand for product } i \text{ in period } t+\tau, \text{ made at the start of period } t, \ i=1,\ldots,N, \ \tau \geq 1, \ t \geq 0$$

$$d_i(t + \tau) \quad \text{demand for product } i \text{ in period } t+\tau, \ i=1,\ldots,N, \ \tau \geq 1$$

$$f_i(t + \tau) \quad \text{is generated from a normal distribution with mean 0 and standard deviation } c_{i,\tau}$$

$$d_i(t + \tau) \quad \text{follows mixed-Erlang distribution with mean } E[D_i] \text{ and standard deviation } \sigma(D_i), \ i=1,\ldots,N.$$  

- Then developers formulate a mathematical programming problem that incorporates the relevant objectives and constraints and solve this to optimality. They defined two alternative solution methods, namely integrated approach and aggregation-decomposition approach.

Integrated approach is based on detailed forecasting and detailed inventory balance equations up to some planning horizon. The optimal solution is acquired by LP.

$$\min \sum_{\tau=1}^{\tau=0} \sum_{i=1}^{T} h_i I_i(t, t + \tau) + p_i B_i(t, t + \tau)$$

subject to

$$I_i(t, t + \tau) - B_i(t, t + \tau) = I_i(t, t + \tau - 1) - B_i(t, t + \tau - 1) + r_i(t, t + \tau) - d_i(t, t + \tau),$$

$$\tau = 0,\ldots, T - 1, i = 1,\ldots,N.$$  

$$\sum_{i=1}^{N} r_i(t, t + \tau) \leq C, \tau = 0,\ldots, T - 1.$$  

$$I_i(t, t + \tau), B_i(t, t + \tau), r_i(t, t + \tau) \geq 0, \tau = 0,\ldots, T - 1, i = 1,\ldots,N.$$  

55
Formula (2)

\[ I_i(t, t + \tau) \] forecast of physical stock at the end of period \( t+\tau \), made at the start of period \( t \), \( i=1,...,N \), \( \tau \geq 0 \)

\[ B_i(t, t + \tau) \] forecast of backlog at the end of period \( t+\tau \), made at the start of period \( t \), \( i=1,...,N \), \( \tau \geq 0 \)

\[ r_i(t, t + \tau) \] forecast of production of product \( i \) in period \( t+\tau \), made at the start of period \( t \), \( i=1,...,N \), \( \tau \geq 0 \)

After solving the LP problem, \( r_i(t, t + \tau) \) is yielded as the optimal solution.

Aggregation-decomposition approach is based on aggregate forecasting and aggregate inventory balance equations. The resulting aggregate problem is solved yielding the aggregate plan for the first period. The first period order release decisions are determined by allocating the aggregate plan among the different items. Three different rules are used for allocation.

In [8], authors made the comparison of the performance between two different approaches. The simulation results in [8] show that aggregation-decomposition approach is superior.

- Finally, the proposed decision will be implemented for the first period.
3 CPN Model

As mentioned earlier there are two approaches used in making decision. Essentially, the first approach is based on the monolithic planning concept and the second approach is based on the hierarchical planning concept. In our CPN model we will adopt the integrated approach. Lu Xi (2005) used the second approach to design this CPN model. In [8], De Kok acquired the conclusion that the Aggregation-decomposition approach is superior to the integrated approach. In our thesis, we will also compare these two approaches by testing two CPN models.

This section introduces the design and implementation of a CPN model based on the case described in Section 2. We first draw the main architecture and then discuss the color set of our model. After that the top level view of the model is described. The top level model contains four substitution transitions, which are also discussed. Finally, the model is evaluated according to the executed results and then we can get the conclusion if we can go on with our second test case.

3.1 Architecture design

![Figure E.1 model architecture](image)

Based on the case description in Section 2, we design the architecture of this Production Planning system shown in Figure E.1. At the higher level, the planning level, the system will make planning of how much to produce for each product. The decision based on the actual net stocks of the products, the forecast of demand and the actual capacity. At the lower level, the execution level, the system gets the results from planning level and calculates the average cost for both the holding cost and the shortage cost based on the actual demands.
3.2 Colors

Before showing the top level view of the model, we present some of the color sets used in the model. In Table E.1, we define some basic data type for this model.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type (color sets)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family</td>
<td>Int</td>
<td>the number of product</td>
</tr>
<tr>
<td>Horizon</td>
<td>Int</td>
<td>Planning horizon</td>
</tr>
<tr>
<td>H</td>
<td>list int</td>
<td>Cost of each product in inventory</td>
</tr>
<tr>
<td>P</td>
<td>list int</td>
<td>Shortage cost of each product</td>
</tr>
<tr>
<td>Cap</td>
<td>Int</td>
<td>Fixed capacity per time unit</td>
</tr>
<tr>
<td>No</td>
<td>Int</td>
<td>unique ID of a product</td>
</tr>
<tr>
<td>run</td>
<td>Int</td>
<td>Simulation run length</td>
</tr>
<tr>
<td>INT</td>
<td>Int</td>
<td>Quantity</td>
</tr>
</tbody>
</table>

Table E.1

For some complex data type, we can make use of ‘list’ and ‘record’ constructs to define them.

STOCK: Data type ‘STOCK’ is to describe the net stocks of the product. It is defined as: record Number: No* inv: INT * bak:INT. ‘Number’ used to distinguish different products. ‘inv’ and ‘bak’ are used to express the quantity of inventory and backlog respectively.

PLI: ‘PLI’ is the result of planning level. It indicates the decision of how much to produce of each product. It is define as: record Number:No * Pro:INT. The rule of ‘Number’ is same as in ‘STOCK’. ‘Pro’ is to express the quantity need to produce.

Dem_Actual: It is used to express demands for each product, and defined as: record Number:No * Dem:INT. ‘Dem’ is the quantity of demand.

RESULT To record the simulation result of each run. It is defined as: record no:INT * cost:INT; ‘no’ indicate the current simulation run. ‘cost’ is the total cost of inventory and shortage.

3.3 Top-level view

The top level of the CPN model is depicted in Figure E.2. It contains three places and four transitions. The main working stream of this model is composed of these three places and four transitions.

The first step in the execution of this model is the firing of the transition “initialize”. When it fires, the transitions “plan” and “demand_gen” are executable. After they fires, the places “plan” and “demand” will get one token respectively. The token in the place “plan” contains a list of production plans for each kind of product. The token in the place “demand” contains a list of real demands for each product. Next the
transition “execute” is fired, and then produce a token with value of average cost for each product saved in the place “statistic”.

In the next sections the sub pages for the transitions “initialize”, “demand generate”, “planning” and “execute” will be clarified.

3.4 Initialization

As the first step of the whole simulation environment, ‘initialize’ is executed at the beginning of each time running this simulation environment. Unlike other transitions, it is only executed at the very beginning, and never executed again. The main task of ‘initialize’ is to set the parameters of the environment. Parameters are classified into two categories: static parameters and dynamic parameters. A static parameter is different from a dynamic parameter as it is unchangeable after the system running, which means system change will not give influence for this kind of parameter. A static parameter is defined by setting a constant, which is shown in Figure E.3. In this model, it includes number of product family, length of planning horizon, fixed capacity, run length of simulation, mean demand and standard deviation of Erlang distribution for each product item, safety stock, inventory holding and shortage cost, and $c_0$ standard deviation of normal distribution for demand forecast.
The dynamic parameter initialization is shown in Figure E.4. It includes stock initialization and result data initialization. The token in place “start” and “ready” doesn’t contain real information. They are only needed to ensure the transition “cata_def” and “initialize” only fired when it is allowed to.

At beginning, other three places do not contain any token, but with the execution these places will have tokens which do contain information to parameterize the model. The place “prod_cata” will contain a list of Unique IDs for each product. The place “stock” will contain a list of data which can describe the current stock of each product. These data are not obtained by execution but provided by the user that starts the case. The place “result_data” will contain integer value which denoted the average cost. Clearly, the initial value of this variable is zero.

After completing “Initialize”, the real simulation process can start.
3.5 Planning

The task in this sub-page is to decide how much to produce for each product.

\[ p_i(t) \]

Production for product \( i \) in period \( t, i=1,\ldots,N, t\geq1 \)

It starts at the first day of each week, which is expressed as @+(1*day). Firstly, according formula (1) discussed in Section 1, transition “demand_fore” forecast future demand up to planning horizon (in our model, we set it as 2 weeks, so we will get \( d_i(t,t+0), d_i(t,t+1) \) and produce a token with value of forecast result to place “Dem_fore”. Then transition “pre_dis” can forecast how much are required of each product in current period based on demand forecast and current stock situation expressed by place “stock”.

\[ r_i(t) = d_i(t,t+0) - I_i(t,t-1) + B_i(t,t-1) + ss \]

requirement for product \( i \) in period \( t, i=1,\ldots,N, t\geq1 \)

\( ss \) denotes the safety stock

The result is saved in place “Current_Dems” as a value of a token.

Finally, transition “make plan” can make decision of how much to produce for each product. As indicated before, the plan is made under finite capacity \( C \); consequently we consider three conditions for finite capacity \( C \).
Case 1: \( C \geq r(t) \land C \geq d_i(t, t + 1) + ss \) then \( p_i(t) = r_i(t) \)

Case 2: \( C \geq r_i(t) \land C < d_i(t, t + 1) + ss \) then \( p_i(t) = r_i(t) + X_i \land \sum_{i=1}^{n} p_i(t) \leq C \)

If \( C - r_i(t) \leq d_i(t, t + 1) + ss - C \) then \( \sum_{i=1}^{n} X_i = C - r_i(t) \)

If \( C - r_i(t) \geq d_i(t, t + 1) + ss - C \) then \( \sum_{i=1}^{n} X_i = d_i(t, t + 1) - C \)

Case 3: \( C < r_i(t) \) then \( \sum_{i=1}^{n} p_i(t) = C \land p_i(t) \leq r_i(t) \)

Compared with LP used in Delphi application, several simplifications are made in our algorithm. First, we fixed the planning horizon \( T \) as 2. Then we also set \( h_1 = h_2 = h_3 = ... = h_n \) and \( p_1 = p_2 = p_3 = p_n \).

### Figure E.5 planning

#### 3.6 Demand generate

The task in this sub-page is to generate the real demands over all product items for every period at the end of that period. This sub-page is composed of 8 places and 3 transitions. The place “sta”, “start_d”, “C1” and “C2” used to control the working cycle, but do not contain real information. Given mean demands and standard deviations, transition “gen” can calculate actual demands for each item up to the
planning horizon using a function according to Erlang Distribution. The place “count” is to ensure transition “gen” only fires when it is allowed to, i.e. it only produce the demands up to the planning horizon. Transition “demand_gen” is fired at 5th day of each period. It provides real demand of current period for transition “execution”.

![Diagram]

**Figure E.6 demand generate**

### 3.7 Execution

Execution is the step of each simulation run, which is defined as one week. It starts at 6th day of each period. In this sub-page, cost calculation, stock update and produce final result are deal with in sequence.

First, the transition ‘cal_cost’ is enabled due to the place “start_E” and “result_data” have tokens which is acquired from “initialize” or last simulation run, and the place “plan of I” has already get the token from the substitution “plan”. It calculates total cost for current simulation run.

After then, the transition ‘execute’ according to the real demand releases orders and the updates stock levels. Finally, the transition ‘stat’ can calculate the average cost as the final result for the current simulation run.
4 Simulation results and analysis

In this section, some statistic data is shown in order to test the validation of CPN Tools in our model. On the one hand, these data can help us to verify if our model is correct. The result analysis of Delphi application in [8] can provide a criterion for our verification. On the other hand, the analysis of these data is aiming towards getting the conclusion if we will continue with the real HPP case using CPN Tools. The conclusion totally depends on the performance of CPN Tools in this test case.

According to the analysis of these data we can conclude if it is worth to continue with the second test case. The analysis is based on the comparison of Delphi application and CPN application. Functionality is discussed at the beginning, and then performance will be made comparison. Finally we will consider the efficiency of these two applications.

4.1 Functionality

The main functionality of the Delphi application is to calculate the average cost per time unit. Figure E.8 gives the interface of the Delphi application.
Figure E.8 interface of Delphi application

Figure E.9 interface of CPN application

Figure E.9 is the interface of CPN application. Same as the Delphi application, the CPN application can also provide the average cost per time unit. So we can get the conclusion that the CPN model can imitate the main function of the Delphi application.

4.2 Validation

We have already known that our CPN model can imitate the main functionality of the Delphi application, but we still need to verify if they are correct. As indicated in [8],
the aggregation-decomposition approach is superior to the integrated approach, which is represented by the cost of aggregation-decomposition approach is less than the integrated approach. So in our two models [20], if we can also obtain the same result, we can prove that our model is correct.

Initially, we choose three groups of parameters for the two models. The fixed parameters are shown below, and other parameters and results of aggregation-decomposition model are shown in Table E2 [20]

<table>
<thead>
<tr>
<th>horizon</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>simulation run</td>
<td>1000</td>
</tr>
<tr>
<td>E(D)</td>
<td>1000</td>
</tr>
<tr>
<td>s(D)</td>
<td>200</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
</tr>
<tr>
<td>p</td>
<td>10</td>
</tr>
<tr>
<td>safety stock</td>
<td>334</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No.</th>
<th>family</th>
<th>C0</th>
<th>c1</th>
<th>capacity</th>
<th>Demand Ratio (90% conf. interval)</th>
<th>Safety Stock Ratio (90% conf. interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0.05</td>
<td>0.20</td>
<td>2200</td>
<td>682.8±3.1</td>
<td>688.4±3.7</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.1</td>
<td>0.25</td>
<td>2400</td>
<td>778.9±5.8</td>
<td>792.3±6.7</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.1</td>
<td>0.25</td>
<td>3300</td>
<td>1228.8±10.5</td>
<td>1269.3±11.2</td>
</tr>
</tbody>
</table>

Table E2 [20]

In Table E3, we give the simulation results of the integrated approach model, as well as the final result based on the 90% confidence interval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Final result (90% conf. interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>987</td>
<td>1062</td>
<td>972</td>
<td>1060</td>
<td>1056</td>
</tr>
<tr>
<td>2</td>
<td>865</td>
<td>871</td>
<td>926</td>
<td>913</td>
<td>913</td>
</tr>
<tr>
<td>3</td>
<td>1640</td>
<td>1662</td>
<td>1676</td>
<td>1640</td>
<td>1654</td>
</tr>
</tbody>
</table>

Table E3

After compared the results for two CPN models in Table E4, we can see that the result from integrated approach is obviously smaller than aggregation-decomposition approach and there is no overlap between each group of result, e.g. [995.0, 1059.8] and [679.7, 685.9]; [995.0, 1059.8] [684.7, 692.1] etc.

<table>
<thead>
<tr>
<th>1</th>
<th>Integrated approach</th>
<th>995.0, 1059.8</th>
<th>aggregation-decomposition</th>
<th>Demand Ratio</th>
<th>Safety Stock Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>877.3, 917.9</td>
<td>[679.7, 685.9]</td>
<td>[773.1, 784.7]</td>
<td>[785.6, 799.0]</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1642.1, 1666.7</td>
<td>[1218.3, 1239.3]</td>
<td>[1258.1, 1280.5]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table E4 comparison of results

Based on Table E4, we can get the conclusion that aggregation-decomposition is superior to integrated approach, which is the same as in [8]. It can prove our models are correct.
In order to further prove our model is correct, we also make use of other parameters to test the performance of our CPN model. In [8], de Kok defined the formula of getting an optimal safety stocks. In our validation, we use our model to calculate the costs with the changing of the safety stocks aiming at to find an optimal point which can minimizes costs. If the point we found is in line with the point obtained by the formula given in [8], then we can prove our model is correct again.

Below are the fixed parameters for this test.

family = 2
horizon = 2
simulation run = 1000
cap = 2200
E(D) = 1000
s(D) = 200
h = 1
p = 10
c_i = 0.25

Using these parameters in the Delphi application [8], we get the optimal safety stock equals to 334.

<table>
<thead>
<tr>
<th>No.</th>
<th>Safety stock</th>
<th>avg. cost 1</th>
<th>avg. cost 2</th>
<th>avg. cost 3</th>
<th>avg. cost 4</th>
<th>avg. cost 5</th>
<th>Final result (90% conf. interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>600</td>
<td>1636</td>
<td>1728</td>
<td>1619</td>
<td>1661</td>
<td>1626</td>
<td>1654.0±32.5</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>1300</td>
<td>1303</td>
<td>1263</td>
<td>1296</td>
<td>1256</td>
<td>1283.6±16.3</td>
</tr>
<tr>
<td>3</td>
<td>334</td>
<td>1333</td>
<td>1242</td>
<td>1234</td>
<td>1288</td>
<td>1231</td>
<td>1265.6±32.4</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>1736</td>
<td>1788</td>
<td>1568</td>
<td>1426</td>
<td>1647</td>
<td>1633.0±105.5</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>2256</td>
<td>2333</td>
<td>2464</td>
<td>2105</td>
<td>2176</td>
<td>2266.8±102.6</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3438</td>
<td>3408</td>
<td>3089</td>
<td>3411</td>
<td>3615</td>
<td>3392.2±139.0</td>
</tr>
</tbody>
</table>

Table E.5 result of CPN Tools

In Table E5, we can find the only overlap appears between the lines 2 and 3, e.g. [1267.3, 1299.9] and [1233.0, 1298], which means that these two parameters changing doesn’t lead to quite big changing for the cost. However, due to the mean of line 3 is smaller than line 2, we still can get the optimal point is when safety stock equals to 334. In order to clearly observe the changing trends, a graphical chart is drawn in Figure E.10.
From both of the two validations described above, we can declare that our model is correct.

4.3 Efficiency

In this part, the efficiency evaluation is mainly aiming at the execution speed. As well known, speed is the bottleneck of process modeling tools, because this kind of tool focuses on clear process description rather than getting final result as soon as possible. Lots of researchers dislike using process modeling tools to make the simulation test only because they feel that the long response time can not be acceptable. Unlike other process modeling tools, CPN Tools has the capabilities of a high-level programming language, which lead us to believe that the application developed by CPN Tools must be faster than other application. In order to validate our point, we will make the evaluation for the speed of this CPN application.

As explained in Chapter 3, CPN Tools provides three ways to execute a simulation model. For the speed test, as we want to know how fast it can be, we only choose fastest manner of three execution ways.

The test environment is:
Hardware: Intel Pentium 4(M) 2.00GHz; 512MB DDR
Software: Windows XP Professional; CPN Tools v.1.4.0
Parameter setting: family = 2; horizon = 2; cap = 2200; E(D) = 1000 ; s(D) = 200 ; h = 1 ; p = 10; c_i = 0.25; ss=334
Table E.6 shows the response time of two applications. Each simulation length is run 5 times to get the average value in order to reduce error. From these data, we can get two conclusions. One out of them is that the speed of CPN application is really fast which can prove our point is correct. The other one is that the result is following a linear formula: \( a + b \times x \) (\( x \) represents simulation length).

Based on the data in Table E.6, we can deduce that \( a = 0.464 \); \( b = 0.00107 \). In order to verify if these two deductions are correct, we need to test two sets of data, which one is the response time for one run and the other is the response time for 2000 run. If our deductions are correct, the one result should be a little bigger than 0.464 and the other result should be around 2.604. The table below shows the result.

<table>
<thead>
<tr>
<th>Simulation run</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (1) CPN/Delphi</td>
<td>0.567</td>
<td>0.680</td>
<td>0.798</td>
<td>1.029</td>
<td>1.532</td>
</tr>
<tr>
<td>time (2) CPN/Delphi</td>
<td>0.570</td>
<td>0.671</td>
<td>0.801</td>
<td>1.022</td>
<td>1.535</td>
</tr>
<tr>
<td>time (3) CPN/Delphi</td>
<td>0.567</td>
<td>0.677</td>
<td>0.801</td>
<td>1.016</td>
<td>1.541</td>
</tr>
<tr>
<td>time (4) CPN/Delphi</td>
<td>0.573</td>
<td>0.677</td>
<td>0.810</td>
<td>1.019</td>
<td>1.539</td>
</tr>
<tr>
<td>time (5) CPN/Delphi</td>
<td>0.576</td>
<td>0.683</td>
<td>0.807</td>
<td>1.025</td>
<td>1.545</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.571</strong></td>
<td><strong>0.678</strong></td>
<td><strong>0.803</strong></td>
<td><strong>1.022</strong></td>
<td><strong>1.538</strong></td>
</tr>
</tbody>
</table>

### Table E.6 speed comparison of CPN and Delphi application

Table E.7

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.466</td>
<td>0.472</td>
<td>0.469</td>
<td>0.472</td>
<td>0.469</td>
<td>0.472</td>
<td>0.469</td>
<td>0.472</td>
<td>0.472</td>
<td>0.472</td>
<td>0.471</td>
</tr>
<tr>
<td>2000</td>
<td>2.572</td>
<td>2.569</td>
<td>2.575</td>
<td>2.581</td>
<td>2.572</td>
<td>2.572</td>
<td>2.575</td>
<td>2.569</td>
<td>2.572</td>
<td>2.581</td>
<td>2.574</td>
</tr>
</tbody>
</table>

Obviously, according to the statistics in Table E.7, we can get the conclusion that our deductions are correct and the response time of this CPN application is linear.

### 5 Conclusion

Based on the evaluation of this CPN model aiming at the functionality, performance and efficiency, we can conclude that CPN Tools can be employed in modeling HPP for logistics control concepts. The reason is described as follows:

- As shown in the first case, using CPN Tools, we completely design a model which can imitate the main functionality of Delphi application.
- The simulation results are acceptable which can persuade us to continue our test of CPN Tools in real HPP concept.
- Intuitive graphical presentation let the user can clearly observe each step of the whole working process.
- The execution speed is acceptable.