Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2007-09

# Performance Analysis of a Palletizing System

M.F. van Amstel, E. van de Plassche, R. Hamberg,
M.G.J. van den Brand, J.E. Rooda

ISSN: 1872-1567

SE Report: Nr. 2007-09
Eindhoven, June 2007
SE Reports are available via http://se.wtb.tue.nl/sereports

**Abstract**

When designing the layout of the material handling system for a warehouse there is a need for the analysis of overall system performance. Since warehouses are typically very large and complex systems it is infeasible to build a simulation model for the entire system. Our approach is to divide the system into subsystems that are small enough to be captured in simulation models. These models can then later be assembled to acquire a simulation model of the entire system.

In this case study we assess the feasibility of this approach by creating a simulation model of a part of a warehouse and verify whether it can be used to embed it in a larger simulation model. The subsystem we use for our case study is a container unloading and automatic palletizing system. This system is chosen because it has already been studied extensively using another simulation tool. We also do a performance analysis of this system in order to come to an optimal layout for this subsystem as well as to reproduce the results of the earlier study for validation. For our performance analysis we created a $\chi$ model of the unloading and palletizing area. The process algebra $\chi$ has been extensively used for modeling and simulation of real-time manufacturing systems. Our case study is also used as a means to assess the suitability of $\chi$ for modeling and simulation in a logistics environment.

Our experiments resulted in roughly the same outcomes as the earlier study. It turns out that for the required throughput the layout chosen in that study is optimal. We also concluded that $\chi$ is perfectly suitable for modeling logistic systems. Considering the extensive time it takes to run simulations of a rather small part of a warehouse using $\chi$, we conclude that it is infeasible to perform simulations of entire warehousing systems by integrating the simulation models of all subsystems into one simulation model. To overcome this problem, aggregate modeling can be used.

# Contents

# Chapter 1
# Introduction

Modern warehouses are very large and complex systems. When designing the layout of the material handling system of a warehouse the customer has to be convinced that a proposed solution meets his required performance demands. It is necessary that this can be established by means of a performance analysis. Because of its size and complexity it is infeasible to do a performance analysis of an entire warehousing system whilst keeping all the details. To overcome this problem we consider a warehouse as a system composed of multiple subsystems, which are sufficiently small to capture in a simulation model suitable for performance analyses. These simulation models can then be combined such that a performance analysis of the warehousing system as a whole can still be performed.

This report describes the performance analysis of a subsystem in a warehouse: container unloading and automatic palletizing. The goal of this analysis is twofold. First, we will assess the feasibility of constructing a simulation model for an entire warehousing system by assembling the simulation models of smaller subsystems. Second, we attempt to come to an optimal layout for this subsystem by comparing performance indicators like throughput and flow time of different layouts. This specific unloading and palletizing subsystem has been selected because it was studied before by other means (see [DGHV01]) and it has been built in the real world. Hence, our layouts are variations of the real world layout.

For the performance analysis we use a simulation model which we create using $\chi$ [vBMR+06, VR06]. The process algebra $\chi$ is specifically designed for modeling manufacturing systems where the focus lies on processing rather than on transportation. We also shortly evaluate the applicability of $\chi$ in a logistics environment where the emphasis is more shifted towards transportation.

The analysis has been conducted in the context of the *FALCON* project [FAL06]. The FALCON project addresses the development of techniques and tools for the design and implementation of professional logistics systems. As a research driver, the project concentrates on a new generation of distribution centers and warehouses with a maximum achievable degree of automation.

The remainder of this report is structured as followed. Chapter 2 gives an overview of a generic warehouse architecture to provide the necessary context. In Chapter 3 the function

of the receiving area is elaborated as well as the requirements for the palletizing system are given. Chapter 4 describes the architecture and implementation of our $\chi$ model and gives a short reflection on the modeling process. Chapter 5 describes the experiments we performed using our simulation model. Conclusions and directions for further research are formulated in Chapter 6.

# Chapter 2
# Warehouse Architecture

This chapter gives an overview of warehouse activities illustrated by a reference architecture. Also, some light is shed on the most important parameters that should be considered in a warehouse design.

## 1 Warehouses

In a warehouse goods are received, stored and sent, optionally supplemented by manipulation of the arrangement and time in which the goods are requested. Warehouses exist in all shapes and sizes, ranging from simple storage at the end of a production line to enormous plants where airplanes exchange time-critical parcels in transit. Due to differences in business aspects and product characteristics, no two warehouse are the same.

In this report, we consider warehouses in a distributing logistics chain. In this type of ware-

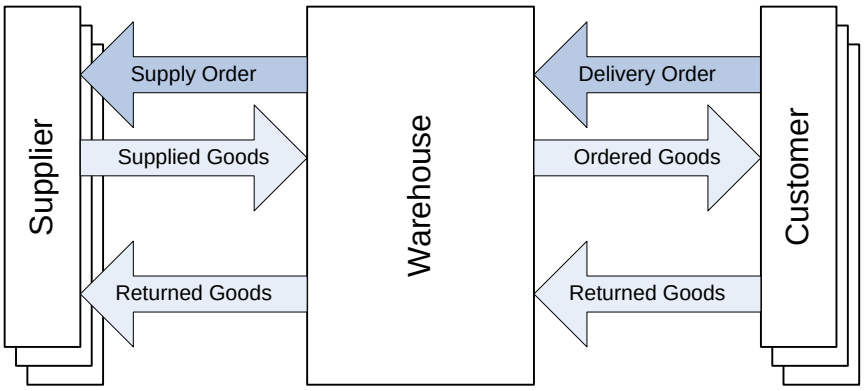


Figure 2.1: Warehouse in a distributing logistics chain

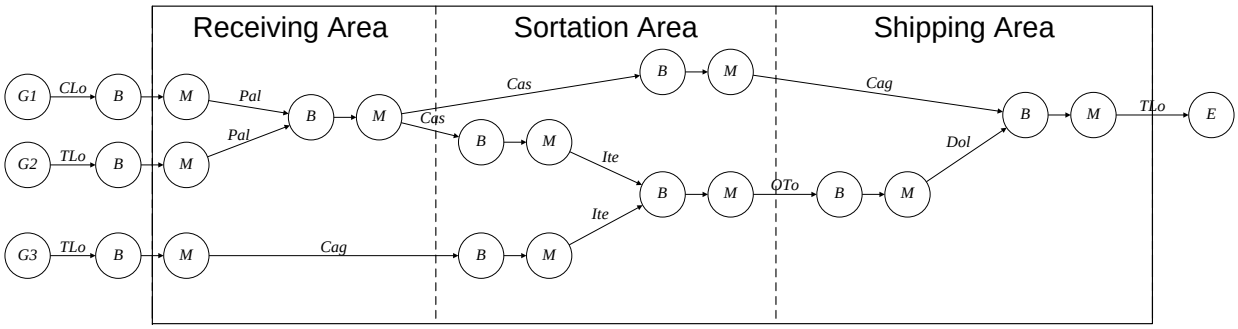


Figure 2.2: Global processes in a warehouse

houses the main objective is balancing the asynchronous supply and demand of goods as shown in Figure 2.1. This kind of warehouses may require massive bulk storages when supply and demand are strongly out of phase.

# 2 Reference Warehouse

When designing and analyzing a warehouse, its internal processes should be known. For this reason, a reference warehouse has been defined (see Figure 2.2). In this chapter the internals of the warehouse under consideration are described.

In general, three major process areas in warehouses can be identified:

1. Receiving area
2. Sortation area
3. Shipping area

The following descriptions of receiving, sortation and shipping area are related to Figure 2.2. The arrows are labeled with units of transportation.

The receiving area comprises the reception of goods from suppliers and, optionally, returns from customers. The received goods may reside in shipping containers, on pallets or in cages, but may also be supplied in bulk. The receiving area transforms received goods into internal warehouse storage units. The considered receiving area receives goods supplied in cases stored in containers (*CLo*, names are shown in Figure 2.2), pallets (*Pal*) supplied in trucks (*TLo*) and loose items packed in cages . Usually, this area does not contain a storage facility other than internal synchronization buffers with the next area. The received goods are checked and verified with the delivery orders.

The sortation area contains the major part of generic goods storage. Storage of goods in the reference warehouse is done in the form of pallet storage, case storage and storage on item level. Items are the smallest form of goods handled in a warehouse. After receiving an order, this area will transform the generic, non-customer order specific goods collection into customer order specific goods collections. This is called *order picking* and can be performed on all levels of internal units. The cases (*Cas*) are picked into cages (*Cag*). Order totes (*OTo*) are introduced to contain the picked items (*Ite*). The reference warehouse only handles picking of cases and items, implying a de-stacking of pallets.

Order consolidation, packaging and marshalling of shipments is done in the shipping area. Here, goods belonging to an order are grouped into shipping units, awaiting shipment to the customer. Order totes are grouped to form dollies (*Dol*) and, together with the cages form truck loads (*TLo*).

The reference warehouse can be classified as *large*, with a throughput of around one million order lines per day, each containing one to several items. The typical buffer capacity of this warehouse is equivalent to $10^5$ pallet loads.

# 3 Design Approach

The design of a warehouse is strongly affected by the following indicators [LR06] as they have a high influence on performance and investment costs:

- Throughput $\delta$
- Flow time $\varphi$
- Floor space $fs$

Derived from the overall flow time (i.e., the time that goods reside in the warehouse) and throughput, the minimal buffer capacity can be determined. Consequently, a distinction can be made in flow time for goods received and moved into the buffer, resting time in the buffer, and flow time for goods extracted from the buffer and sent to the customer. As a consequence of the above required performance, initial investment costs and operational costs determine the feasibility and level of automation in a warehouse.

Designing and analyzing a warehouse requires good understanding of relevant processes and a thorough knowledge of possible solutions. Because of the number of concurrent processes within a warehouse, a simple quantification of design space budget per process can not be given (e.g. cost, floor consumption, required capacity etc.). For this reason, theoretical descriptions and models can give insight in the quality of a design. The level of achievable detail of these analyses is however limited by simulation processing power. Analyzing the reference warehouse as a whole on a detailed level requires the availability of sufficient details of all processes and is deemed to be a lot of work.

In this report, only a portion of the receiving area of the total warehouse is discussed in detail. The analyzed area transforms container loads arriving at the warehouse dock doors into pallet loads used in the bulk store. The goods flowing through this area are boxes.

# Chapter 3

# The Receiving Area

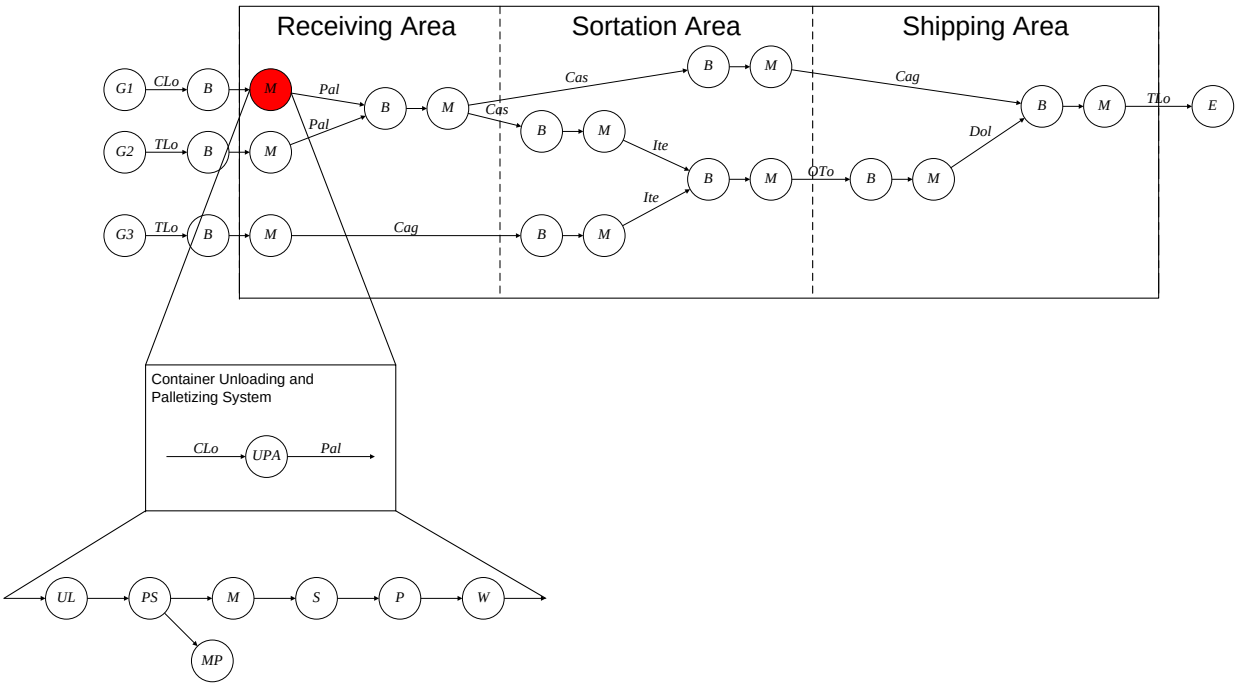


Figure 3.1: The unloading and palletizing area in the context of the complete warehouse.

In this study a part of the receiving area has been modeled to analyze the performance of the automatic palletizing process. In Figure 3.1 it is indicated how this relates to the schematic, functional view on the complete warehouse. In this chapter a description of the area under study is given, along with its behavior, i.e., a high level specification or set of requirements

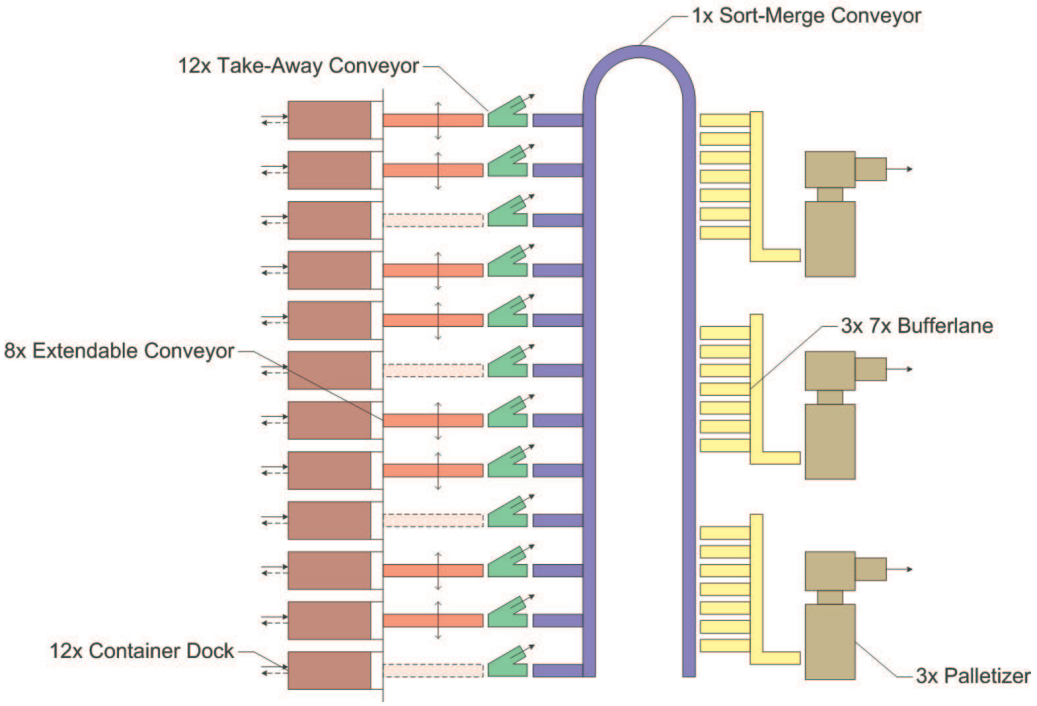


Figure 3.2: Layout of the relevant part of the receiving area

(expressed in natural language). This serves as a starting point for the modeling activities described in Chapter 4.

# 1 Process

The main activity performed in the receiving area is palletization. The part of the receiving area relevant for our model (the palletization area) is schematically depicted in Figure 3.2. The goods enter this area in containers and leave it on wrapped pallets.

Containers ready for unloading are positioned at one of the container positions in the receiving area. How containers are allocated to these unloading positions is beyond the scope of this analysis. Therefore we will assume that there is an infinite number of containers waiting to be assigned to the first available unloading dock. Each container typically contains hundreds of boxes, the type of which belongs to a limited number of stock keeping units (SKU's). A SKU is a unique article number.

When a container is ready to be unloaded, one of the extendable conveyors will be assigned to it whenever it becomes available. An extendable conveyor can move to any unloading position as long as it is not in use and does not traverse another extendable conveyor. The operator working on the extendable conveyor will unload the container. This is done by first announcing the SKU type of the first box that will be unloaded to the control system, followed by the physical unloading. When the operator has unloaded all boxes of an SKU type, he will announce the next SKU type and start unloading again. This is repeated until the container has completely been depleted.

After being transported by the extendable conveyor, boxes reach the take-away conveyor (in short: pre-sorter). For each SKU it is known whether the boxes[1] can be palletized automatically. For boxes that can be palletized automatically, a reservation request is sent to the palletizers. If there is a palletizer that has enough buffer lanes available to buffer boxes for a whole pallet, all boxes of this pallet-to-be will be sent to this palletizer via the sort-merge conveyor (in short: sorter). Alternatively, when not enough buffer lanes are available or the characteristics of the SKU forbid it to be palletized automatically, the boxes of the pallet-to-be are sent to the manual exit located near the pre-sorter. Here, an operator will manually stack the boxes on pallets. The process of manual palletizing is beyond the scope of this analysis.

There are more unloading positions than there are palletizers and it should be possible for all boxes to travel to any palletizer. Therefore, boxes are merged on a conveyor and sorted out into the appropriate palletizer buffer lanes. This occurs at the sorter. This sorter allows boxes entering the system at any unloading position to travel to any of the palletizers.

When all boxes for a pallet are present in the appropriate buffer lanes the palletizer can create a new pallet. It will start by self-adjusting its stacking algorithm depending on the characteristics of the SKU it has to handle (box sizes, number of layers, stability, etc.). Boxes will be fed from the buffer lanes to the palletizer and the actual palletizing can start.

The final step in the receiving area is wrapping the newly-stacked pallets with foil to ensure stability in the processes to follow.

# 2 Requirements

This section contains the requirements for the palletizing concept as extracted from [DGHV01]. In [DGHV01] two possible solutions are described. We only consider the solution using layer-palletizers. Note that in this section the layout is considered to be fixed. In the experiments described in Chapter 5 we will also vary the layout parameters.

Tables with detailed requirements can be found in Appendix A.

As described in Section 3.1, the receiving area covers the following operations (see also Figure 3.3):

1. Unloading boxes (U),
2. Pre-sorting boxes to manual or automatic palletizing (PS),
3. Merging automatically palletizable boxes on a take-away conveyor (M),
4. Sorting automatically palletizable boxes to buffer lanes (S),
5. Automatically palletize automatically palletizable boxes (P),
6. Manually palletize non-automatically palletizable boxes (MP),
7. Wrapping full pallets (W).

[1] Throughout the document SKU means boxes containing items of a certain SKU type.

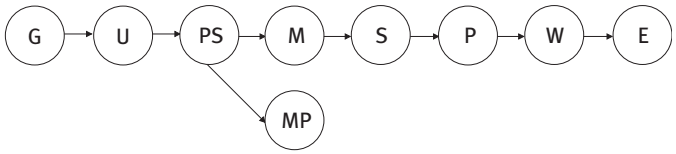


Figure 3.3: Process

## 2.1 Container Docking and Extendable Conveyors

To unload the containers, twelve unloading positions and twelve take-away conveyors are available. The unloading positions are connected to the take-away conveyors by eight moveable extendable conveyors. Containers can only be unloaded when the unloading position is connected to a take-away conveyor, this means only eight containers can be unloaded simultaneously. The other four unloading positions are used to replace empty containers for full ones. At start-up, the extendable conveyors are placed at unloading positions 2, 3, 5, 6, 8, 9, 11, and 12.

When a container is empty, the extendable conveyor used for unloading it will move to the nearest unloading position that has a full container available. If multiple full containers are available at the same time at the same distance, the extendable conveyor will go to the unloading position left from its current position. Note that extendable conveyors cannot cross each other.

The time needed to change an extendable conveyor from one unloading dock to another is less than the time needed to replace an empty container with a full one. The durations used for these operations are also taken from [DGHV01], they are listed in Table 3.1.

| Operation | Duration (seconds) |
|---|---|
| Changing extendable conveyor between doors | 120 |
| Replacing container at a door | 500 |

Table 3.1: Operation times

## 2.2 Unloading Containers

When a container and an extendable conveyor are in place, an operator will start unloading the container. If a container contains boxes of more than one SKU the operator will first unload all items of SKU 1, then all items of SKU 2, etc.

Before an operator can start unloading the boxes of an SKU, the WMS/WCS will have to be informed what SKU is going to be unloaded. This is done by the operator via a computer terminal near the extendable conveyor and takes two minutes. Once a new SKU is reported, the operator will unload all boxes of this SKU at a constant rate. The unloading rates are fixed, but differ for small and large boxes.

## 2.3 Pre-sorting

At the end of the extendable conveyors, a luffing conveyor is used to pre-sort boxes to either manual or automatic palletizing. (A luffing conveyor can direct boxes in two directions.) Manual palletizing is outside the scope of this study.

There are two possible reasons to send boxes of a certain pallet-to-be to manual palletizing:

1. The shape of boxes of that SKU type is such that it is impossible to automatically palletize them.
2. It is impossible to make a buffer reservation at any of the palletizers because none of them has enough free buffer lanes available to buffer a complete pallet load of those boxes.

Boxes for a pallet that can be palletized automatically, are allocated to the required number of buffer lanes and travel to them via conveyors.

## 2.4 Palletizing

There are three layer palletizer machines and each of these machines has seven buffer lanes feeding boxes to it. A palletizing machine can palletize only one pallet at a time. Table 3.2 shows the equipment capacities.

| Subsystem | Amount | Design capacity |
|---|---|---|
| Layer palletizer machine | 3 | 1500 boxes/machine/hour |
| Buffer lanes | 7 per machine = 21 | 500 boxes/lane/hour |
| Wrappers | 1 per machine = 3 | 90 pallets/wrapper/hour |

Table 3.2: Layer palletizer equipment capacities

It is required that the system can automatically palletize 3400 boxes per hour on average.

## 2.5 Operating Concepts

SKU's are allocated dynamically to a palletizer, i.e., boxes unloaded at any of the twelve unloading positions can be handled by any of the palletizing machines. The allocation of an SKU to a palletizer machine depends on the required number of buffer lanes to buffer a full pallet load of boxes of that SKU. If multiple palletizers have sufficient buffer lanes available, the system shall select the palletizer with the maximum number of free buffer lanes. Consideration should also be taken to balance allocation of SKU's requiring a lot of buffers with SKU's requiring little buffers. This requires a degree of planning to maximize automatic palletization and avoid boxes being manually palletized due to insufficient buffer lanes being available.

Under normal conditions the system loads precisely enough boxes of an SKU into the allocated buffer lanes to create a full pallet prior to palletizing. However, if an operator informs the system of an SKU changeover, the palletizer finishes the buffered boxes of the previous SKU as a partial pallet.

Furthermore, there are a number of factors which should also be taken into account:

- Overflow of buffers must be avoided as recirculation around the sorter has not been accounted for.
- Pallet pattern changeover should be minimized to reduce changeover time.

- The system needs to have access to a database which stores SKU palletizing properties, for example for determining the required number of buffer lanes.

# Chapter 4
# Modeling the Receiving Area

This chapter describes the steps to come to the $\chi$ simulation model. We start by designing an architecture for the receiving area in terms of communicating parallel processes. Thereafter we describe the processes itself in more detail. The last section of this chapter is devoted to the modeling process.

## 1 Architecture

We create the model architecture in two phases. First we design a global architecture which we later refine to a more detailed architecture.

The requirements of Section 3.2 state specific amounts of extendable conveyors, palletizers and buffer lanes per palletizer. We abstract from these numbers as the goals of this project is to find the optimal combination of these structural parameters.

### 1.1 Global Architecture

In Figure 4.1 the global architecture of the system is depicted. Each of the components of the model is an autonomous process that communicates over synchronous channels with other processes. Notice the resemblance with the process steps depicted in Figure 3.3. However, the division of the process steps from Figure 3.3 among $\chi$ processes is somewhat different: unloading and pre-sorting is performed in the unload area, merging and global sorting is performed by the sorter, and detailed sorting, palletizing and wrapping is performed in the palletize area.

In Figure 4.1 it can be seen that a generator process and two exit processes are added. We do this to create a stand-alone simulation system. Notice also the presence of a database. The database is added because it is necessary to provide the simulation model with the required data and its presence is one of the requirements, see Appendix A.2.4.

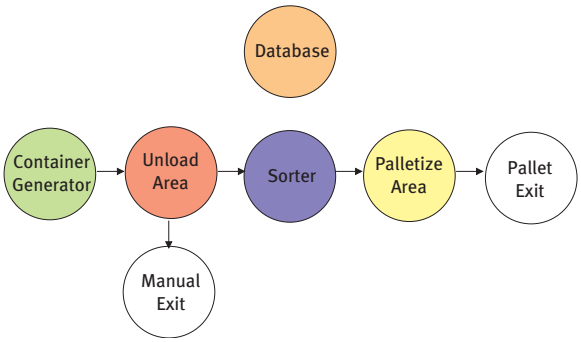


Figure 4.1: Global architecture

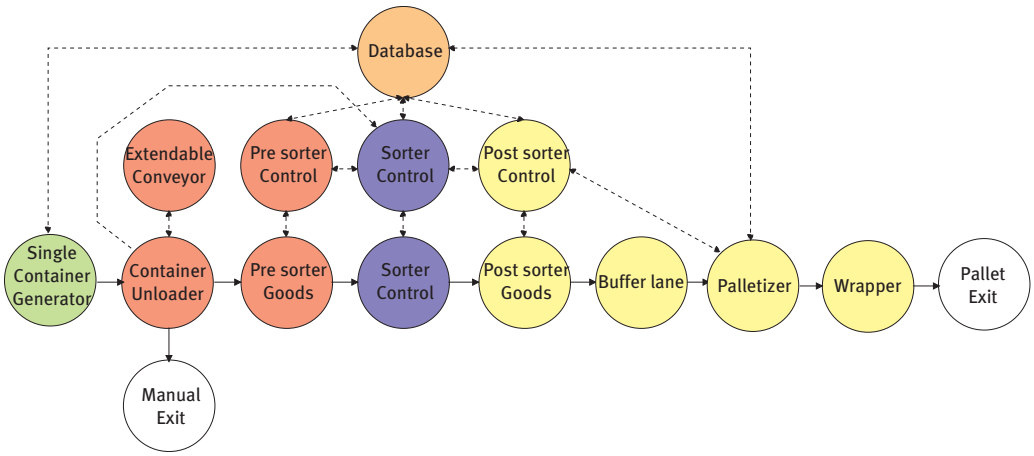


Figure 4.2: Detailed architecture

## 1.2 Detailed Architecture

The detailed architecture is depicted in Figure 4.2. The ordinary arrows represent material flow and the dashed arrows represent data flow.

In the real world, material and control are two separated flows, therefore we also decide to split the material and control flow in our model where possible. Another reason for this separation is that goods in the system are subject to delays, whereas control data can travel infinitely fast. All control processes are again roughly divided into two parts, one that handles reservations at the palletizers and one for taking routing decisions.

The container generator of Figure 4.1 consists of multiple 'single container generators'. Each of these 'single container generators' represents an infinite row of containers standing in line at an unloading position waiting to be unloaded. Because a single container generator is coupled to an unloading position, there need to be as many container generators in the system as there are unloading positions.

The unload area from the global architecture is a set of 'single unload areas' and a set of extendable conveyors. A 'single unload area' consists of a container unloader and a pre-sorter. Because pre-sorters cannot be shared there needs to be a one–to–one mapping between pre-sorters and container unloaders. The extendable conveyors are added as separate processes. This is done because the coordination of extendable conveyors is independent of the con-

tainer unloaders but not timeless.

The palletize area is also divided into multiple 'single palletize areas'. Each 'single palletize area' consists of a post-sorter, a number of buffer lanes, a palletizer, and a wrapper. The post-sorter is introduced for simplifying buffer reservations and to make the system more modular. The post-sorter does not exist in the original system, as can be seen in Figure 3.2. Therefore the post-sorter does not incur any delay on the boxes traveling through it. In the original system, the 21 buffer lanes are directly connected to the sorter in three groups of seven lanes (see Figure 3.2). Our post-sorter handles this division in groups. In this way the sorter does not have to know to which palletizer each buffer lane belongs, it just forwards any reservation request to the post-sorters. Moreover, it is now possible to simply add a palletizer with its own buffer lanes and post sorter without having to adapt the internals of the sorter. It is even relatively easy to create heterogeneous palletizers by adding or removing any number of buffer lanes.

# 2 Processes

This section describes in detail how all the processes depicted in Figure 4.2 are implemented in a $\chi$ model. The model is created using $\chi$ version 1.0, the listing is found in Appendix B.

## 2.1 Database

The database is initialized with two input files, one contains SKU data and the other the division of boxes over containers. The database can be queried for the following information:

- contents of a container,
- pallet factor of an SKU (the number of boxes on a full pallet),
- length of an SKU,
- whether an SKU can be palletized automatically,
- the time it takes to automatically palletize $n$ boxes of a certain SKU, where $n$ is an input parameter.

The data we use to initialize the database can be found in Appendix D.

## 2.2 Single Container Generator

The single container generator process represents a single unloading position. It generates container contents from a data set.

All instances of the single container generator are connected to two other processes, a container identifier generator (*CidG*) and a pallet identifier generator (*PidG*) (see Figure 4.3). These two processes generate unique identifiers for containers and pallets respectively. A container is generated as follows. First, a container identifier is acquired from the container identifier generator. This identifier is then used to retrieve the contents of a container from the database. Containers are generated in zero simulation time, this implies there are always containers available for unloading and thus the system is always under peak load.

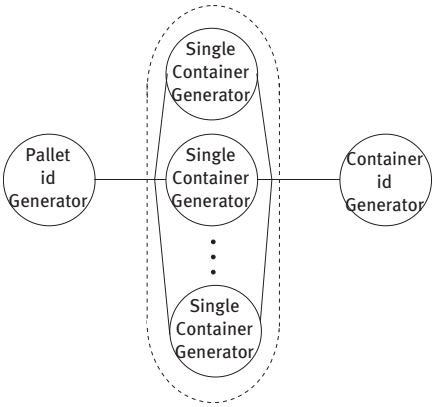


Figure 4.3: Container generator

The single container unloader can be replicated as often as needed to simulate multiple unloading positions. Note that for every unloading position an unloading area is required, see Section 4.1.2.

We choose to create pallets (virtually) before boxes go into the palletizing system. So actually, a container is a collection of virtual pallets. This is done to simplify routing and reservation decisions further on in the process. Because pallets are created beforehand, a pallet identifier is added to a box in the container generator process.

## 2.3 Container Unloader

A container unloader starts the unloading process by acquiring a container and an extendable conveyor. The latter is possible as a parallel control process, part of the container unloader administers the assignment of an extendable conveyor to an unload position. When an extendable conveyor is in place the unloading can start. Boxes travel from an unloading position via an extendable conveyor to a pre-sorter.

In the container unloader the control and material flow are decoupled. For every new virtual pallet the data about the boxes it contains is sent to the pre-sorter to make a reservation at a palletizer for the boxes belonging to that virtual pallet.

When the last box of an SKU is encountered, this box is tagged with an identifier. This identifier is also sent to the controller of the sorter such that it can distinguish the box from other boxes. Whenever a box of a new SKU type is encountered during unloading a delay of 120 seconds is incurred, simulating the time it takes an operator to report an SKU changeover.

When a container is completely unloaded a delay of 500 seconds is incurred before the next container can be unloaded. This delay simulates changing a container at an unloading position.

## 2.4 Extendable Conveyors

The extendable conveyors act on the basis of their current position (i.e., a certain container unloader position) and its target position. If these positions are different, the conveyor first announces to the current container unloader that it is leaving, and moves towards the target position, which takes 120 seconds. Subsequently, in both cases, it announces its availability

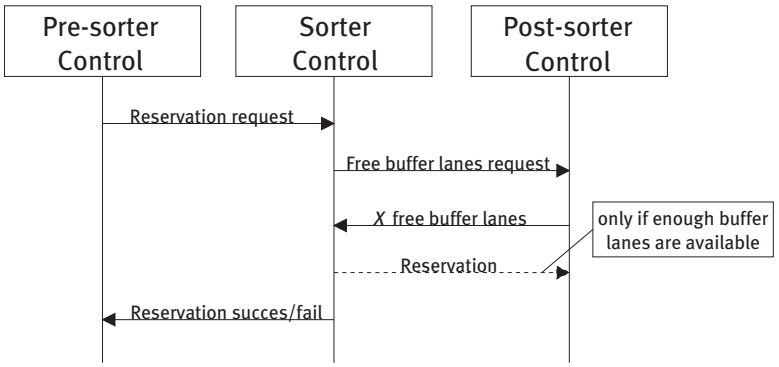


Figure 4.4: Reservation process

to the container unloader where it resides. All extendable conveyors are initialized with the same current and target positions, which are given in Section 2.1 of Chapter 3.

After the release signal is received from the unloading position, the extendable conveyor computes its new target position. It does this by sending a request to the unloading position to the left asking whether a conveyor is needed or not; if the answer is negative, it will send a similar request to the unloading position to the right. Obviously, when a confirming answer is received, the conveyor will set its target position to the corresponding unloading position. The procedure is executed either in left-right order or in right-left order, both with 50% probability.

## 2.5 Pre-sorter

In the pre-sorter process boxes are pre-sorted to automatic or manual palletizing. The pre-sorter is the first process in the chain that has a separate control and material process. The control process is again divided into two functional parts, one part is concerned with reservations and the other one with routing.

When the container unloader announces that boxes for a new virtual pallet will be coming, the pre-sorter first checks whether the characteristics of the SKU allow it to be palletized automatically by querying the database. If it can be palletized automatically, the pre-sorter tries to make a reservation at a palletizer by sending a message to the sorter. The sorter will respond whether the reservation was successful. If no reservation has been made, either due to the SKU's characteristics or due to insufficient palletizer capacity, this will be stored internally. The messages exchanged between the processes involved in the reservation process are depicted in the sequence diagram of Figure 4.4.

When a box arrives at the material process, the controller is asked whether the box has to be palletized automatically or manually. If the controller was unable to make a reservation, the box will travel to manual palletizing, else it will travel to the sorter.

## 2.6 Manual Exit

The manual exit process is nothing more than a simple exit process. It accepts boxes that cannot be palletized automatically, either due to their characteristics or due to insufficient capacity of the automatic palletizers.

Note, because manual palletizing is not taken into consideration this process will always be

able to accept boxes such that it does not influence the rest of the process.

## 2.7 Sorter

The sorter consists of a separate control and material process. Its task is to make reservations at the palletizers and to route boxes from the pre-sorters to the appropriate post-sorter.

The sorter can receive reservation requests from pre-sorters. Upon receipt of such a reservation request, a signal is sent to all post-sorters. The post-sorters will respond how many buffer lanes are available at the palletizer they belong to. If there is no palletizer available with enough free buffer lanes, the sorter will respond to the pre-sorter that a reservation cannot be made. If there are one or more palletizers that have enough available buffer capacity, the sorter will select one with the largest number of free buffer lanes and make a reservation there to spread the work over the available palletizers. In case multiple palletizers have the largest number of free buffer lanes, the one with the lowest identifier is selected. When a reservation is made, the pre-sorter is informed that a reservation was successfully made. See also Figure 4.4.

The sorter can also receive a message from an unloader containing the identity of the last box of a certain SKU type. Whenever such a last box is encountered in the material process of the sorter, the post-sorter that box will travel to is informed that the buffers assigned to the virtual pallet to which the box belongs are filled when that box has arrived.

When a box arrives at the material process of the sorter it asks the controller to which post-sorter it should go. The controller answers and the box is sent to the correct post-sorter with a constant delay of 60 seconds simulating the time the box resides on the conveyor.

We abstract from the merge and sort algorithm in the sorter as a more detailed model of this algorithm results in a too low performance of the simulation model. However, in practice it turns out that the sorter is a possible bottleneck for overall system performance. To simulate limited capacity, a minimum time distance of 0.6 seconds is incurred between subsequent boxes. To ensure that this delay does not cause preceding processes to block, boxes can always be accepted by the sorter. They are put in a queue and extracted according to the aforementioned timing rules. This limits the capacity of the sorter to 3600/0.6 = 6000 boxes/hour. Of course this value can be changed, simulating different sorter capacities.

## 2.8 Post-sorter

The post-sorter is the last process in the chain with a separate control and material process. The post-sorter is concerned with buffer reservations and directing boxes to the appropriate buffer lanes.

To initiate a buffer reservation, the sorter sends a message to all post-sorters. The post-sorters respond with how many buffer lanes its palletizer has available. Thereafter the sorter *may* respond with a message indicating the number of buffers that need to be reserved for boxes of a certain SKU type. If such a message arrives, the post-sorter will administer this reservation. See also Figure 4.4.

The post-sorter can also receive a message from the sorter containing the identity of the last box of a certain SKU type. Whenever such a last box is encountered in the material process of the post-sorter it is known that all boxes for a pallet are present in a subset of the buffer lanes. Now the palletizer will be informed that it can flush all buffers assigned to that pallet.

On reception of a box, the post-sorter asks its controller to which buffer lane the box has to be sent to. The controller checks its reservations and responds. Thereafter the box travels to the appropriate buffer lane with zero delay simulating the non-existence of the post-sorter.

### 2.9 Buffer Lane

The buffer lane process consists of two parts, one part is used to store boxes in the buffer and the other part is used to empty the buffer. The boxes arrive in a buffer lane via the post-sorter it is connected to. This post-sorter also indicates when a buffer lane can be emptied such that the boxes can be sent to the palletizing machine in order to create a pallet. Arrival is modeled through addition to a queue, while departure is modeled as emptying the complete queue at once.

### 2.10 Palletizer

The palletizer creates pallets from boxes in buffer lanes.

As soon as a palletizer receives a message from its post-sorter that a pallet can be created, it stores this information. Buffer lanes are emptied and pallets are created using this information whenever the palletizer is available. The database is queried to find out how long palletizing of the boxes will take. To simulate the palletizing process a delay is incurred. Complete pallets are sent to the wrapper.

### 2.11 Wrapper

The wrapper accepts pallets, wraps them, and sends them to the exit process. Wrapping a pallet takes forty seconds.

Ultimately, this process is not modeled at all as it is just another additional delay at the end of the model, not changing any relevant properties of the model as such.

### 2.12 Pallet Exit

The pallet exit process is, like the manual exit process, an ordinary exit process. It accepts wrapped pallets.

# 3 Reflections on the Modeling Process

This section summarizes the modeling process we went through. First another representation is described in which the behavior of our model is captured. Second we describe where our simulation model deviates from the requirements. Thereafter the evolution of the model throughout the modeling process is explained. Last we describe our experiences with the $\chi$ tool set.

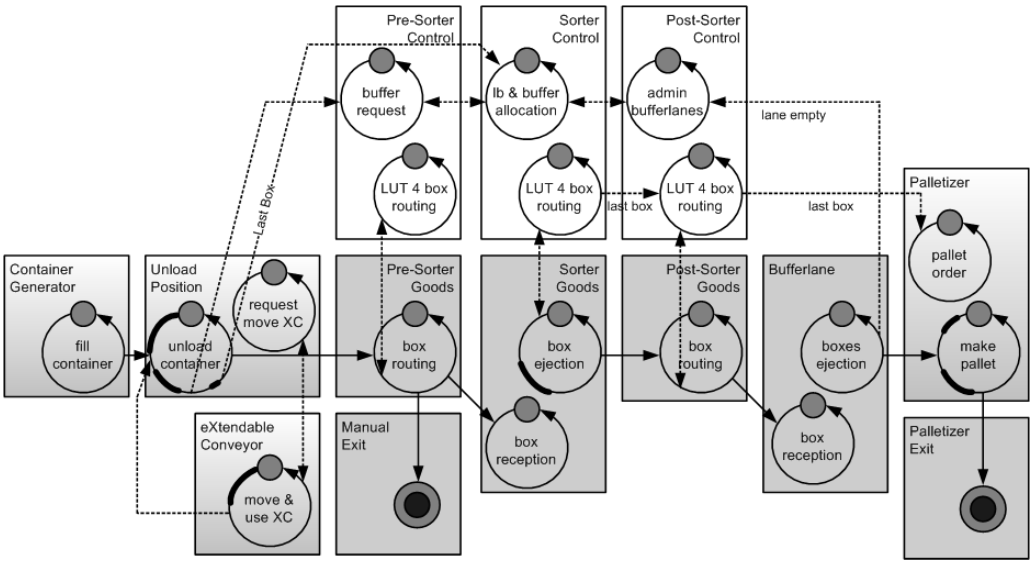


Figure 4.5: A representation of the processes with a sketch of their dynamical behavior. Circular/repetitive time lines represent independent processes, while thick line segments represent a time delay. Dotted arrows indicate control flows, while solid arrows depict transported goods. Grey boxes are involved in goods flows, while white boxes concern control. Shaded boxes are involved in both types of flows.

## 3.1 Model Dynamics

As an alternative to the prosaic description of Section 4.2 a representation that visualizes the temporal behavior of the processes and their interactions is given in Figure 4.5. This representation has been introduced because it is very hard to understand and to reason about system behavior. It should help finding modeling errors early in the modeling process.

The prosaic description of the sub-processes can be mapped onto the blocks in Figure 4.5, although some simplification had to be performed in order not to clutter the diagram too much. Independent processes, represented by circles, inside the sub-processes communicate via shared data, while between sub-processes communication is done via synchronous communication channels.

## 3.2 Deviations from Intended Behavior

Our $\chi$ model does not strictly adhere to all the requirements stated in Section 3.2. In this section the deviations from the requirements are pointed out and motivated.

In Section 3.2.5 it is stated that consideration should be taken to balance the allocation of SKU's requiring a lot of buffers with SKU's requiring little buffers to palletizers. Most SKU's require only one buffer lane for buffering a full pallet load, simply because the buffer lanes are long enough. Therefore it hardly ever occurs that a virtual pallet cannot be allocated to a palletizer due to inefficient allocation planning. The control overhead induced by a more sophisticated planning algorithm does not outweigh its benefits in these rare occasions. Therefore we have decided not to implement this load balancing.

Also, it is stated in Section 3.2.5 that pallet pattern changeover should be minimized. In our model virtual pallet loads are assigned to buffer lanes based only on the amount of free

buffer lanes available. It is true that a performance gain could be achieved by implementing this rule, although this effect is expected to be quite small.

Another requirement we do not implement concerns the extendable conveyors. In Section 3.2.1 it is stated that extendable conveyors have a preference for moving to the left. In our model, the extendable conveyors choose a movement direction (left or right) non-deterministically. As a result the utilization of the unloading positions is less skewed to one side.

## 3.3 Model Evolution

The $\chi$ model as listed in Appendix B has evolved throughout the modeling process. In this section we will explain the most important evolutions.

Multiple attempts have been made to model the behavior of the pool of extendable conveyors. The very first attempt was just a pool from which any unloading position can request an extendable conveyor when it needs one. In this solution any extendable conveyor can serve any unloading dock, so they can also cross each other. This is not an acceptable solution as it does not adhere to the requirements, because extendable conveyors are not allowed to cross each other. The next few attempts all use a controller to direct the conveyors to their positions. This solution does adhere to the requirements, but it should be able to model this in a simpler way. This idea is mainly inspired by the natural language description of the behavior of the extendable conveyors, which is just a few sentences. Such a natural language description is most easily given from the perspective of one extendable conveyor, therefore we decided to try to model the behavior also from the perspective of one of the conveyors. This attempt results in the final, decentralized, model of the extendable conveyors.

The first versions of the model use some hard-coded input data to test the behavior of our model. One of the requirements is to use data from a database to perform the simulation experiments. Therefore we add an interface to an external database. The database can be queried in an object-oriented like fashion [BME+07].
The shift from hard-coded input to a database introduced another problem. The hard-coded input is tailored such that a container always contained at most one pallet load of SKU's. In the database this is not always the case. Therefore we decide to create virtual pallets upfront instead of adapting the complete scheduling algorithm in our model. This virtual pallet approach has as main advantage that reservations at the palletizers are made upfront, both for full pallet loads or for incomplete pallets whenever necessary.

As there is no generic front-end available for $\chi$ to organize and visualize simulation output, we have found our own solution for this. First we inserted print statements at interesting points in the code. This is a workable solution, although it has as major drawback that it clutters the code. Subsequently we create a logbook to which all processes can write output. This has as major advantage that all output is collected in one process such that the code is less cluttered. A disadvantage of it is that all processes need an interface to the logbook. This way of dealing with logging resembles an aspect-oriented approach [KLM+97].

The last adaptation of our model comprises the introduction of parameters that can be changed just before compile time. This has as advantage that it is relatively easy to start a batch of experiments using a preprocessor instead of repeatedly adapting the model. Also, automatically adapting these parameters is less error prone as dependent parameters are adapted simultaneously and yields faster execution models.

## 3.4 Model Refinement

The time it takes to run the model for a fair amount of simulation time is quite long, or to be more precise, in many (experimental) conditions the simulation runs for a number of days. Partly, the refinement process in the model causes this, because after most discussions more detailed behavior (as opposed to less) is added to the processes, which takes more time to simulate. Only a few times a better modeling approach was seen, that actually improved model simulation performance. On the other hand, refinement is necessary in order to build understanding of the processes, their models, their intended and actual behaviors.

Application of the model *as such* in a larger context is hard to imagine for two reasons. First, the larger model would be unusably slow and secondly, the model would be unusably complex.

This calls for adequate abstraction of such models, which could be achieved through aggregate modeling [LA07].

## 3.5 Pros and Cons of the $\chi$ Tool Set

In this subsection a few sentences are devoted to our experiences with the $\chi$ tool set. This section is therefore rather subjective.

The $\chi$ language is fairly easy to learn, however the underlying formalism is a lot more difficult to apprehend. Reasoning about parallel processes is not trivial. It is however a very strong formalism, which is expressed by the fact that operational concepts can be written in $\chi$ very concisely. This results in a high speed of modeling.

It is very easy to vary the structure of the model. In our case adding or removing a buffer lane is just a matter of changing a few numbers. This is made possible by the powerful concept of channel bundles. Unfortunately constant values are not yet supported by $\chi$. Because of this, adding or removing a buffer lane requires the change of a few values, rather than one constant. We solved this by creating a preprocessor which changes constants throughout the code of the simulation model by the appropriate values, see Appendix C.

Another powerful and indispensable concept, is the concept of simulation time. This enables the simulation of days of *real* time in just a few seconds. Unfortunately our model turned out to be of such complexity that simulation takes more time than is actually simulated. The use of simulation time ensures that the platform on which the model runs and hence, the execution time of the simulation program, do not influence the time-dependent results of the conducted experiments.

A drawback of the $\chi$ formalism is the absence of facilities for model verification and validation. Once a model has passed the compilation process, no feedback is given anymore. Therefore it is very difficult to establish whether the intended behavior is modeled correctly. Also, the discovery of deadlocks and livelocks can be difficult. Once a deadlock is encountered this is reported to the user. However, the absence of deadlocks cannot be guaranteed. It is possible that a deadlock only occurs in rare occasions which may not occur during every simulation. Livelocks are not reported at all, the user has to conclude for himself that (and why) a livelock has occurred. It should however be noted that translations from $\chi$ to SPIN, UPPAAL and mCRL2 exist. These formalisms can be used for verification and validation purposes.

The output of the simulation of a $\chi$ model is returned in plain text format. Therefore other

tools are needed to generate graphical output. This requires to think about specifying the output such that it can be used as input for those other tools. Also, inserting output statements clutters the code of a simulation model. It is however possible to generate some graphical output in the form of Gantt charts and automata from a $\chi$ simulation model.

# Chapter 5
# Experiments

The receiving area has been described and a $\chi$ model of it has been built, the primary goal of the latter is to analyze the performance properties of the receiving area as a function of its structural and behavioral design. The well-known parameters to characterize performance properties have been introduced in Chapter 2 and will be treated in the subsections of this chapter. First, however, the rationale for the series of experiments that has been carried out is described.

## 1 Series of Experiments

The experimental set-up is primarily chosen such that the influence of structural parameters on the performance figures can be measured. The relevant structural parameters are:

- The number of extendable conveyors $c$,
- The number of buffer lanes at each palletizers $b$, and
- The number of palletizers $p$.

The first of these, the number of extendable conveyors, is the most determining factor for representing the input capacity of the total area. This is related to the fact that this number is directly coupled to the number of human operators that unload the boxes from the containers. The second parameter, the number of buffer lanes, is most relevant in characterizing the level of decoupling between front-end and back-end of the system (front-end and back-end are the parts before and after the sorter, respectively). After all, this number of lanes determines how much work can be done upfront without considering the palletizers, and how much work can be stored for the palletizers without knowing what is happening at the unloading positions. The last parameter, the number of palletizers, is the most determining factor for representing the output capacity of the total area.

A large number of parameters is not varied, but are chosen to represent the reality as closely as possible. These include the number of static unloading positions, the time it takes a human operator to unload boxes, the time it takes to palletize different types of boxes, the time it takes to change containers, the time it takes to move extendable conveyors, the time an operator needs for announcing an SKU changeover, the traveling time of boxes through the system, and the length of each buffer lane. The central sorter deserves some specific attention. It is designed to have an overcapacity, although not being infinite. That is, its behavior is taken constant, but neither trivial nor detailed, which was necessary to render the simulation model both realistic and workable.

We vary the three structural parameters mentioned above to fill a three-dimensional space with points at regular intervals around the working point that is available from the original design of an existing receiving area of a warehouse. Representing this working point with $(c, b, p) = (8, 7, 4)$, where $c$ represents the number of extendable conveyors, $b$ the number of buffer lanes, and $p$ the number of palletizers, the experiments cover the set $\{(c, b, p) | c \in \{6, 8, 10\}, b \in \{3, 5, 7, 9\}, p \in \{3, 4, 5\}\}$. The variation in the number of buffer lanes $b$ is taken to lower values, because in earlier experiments we have seen that the original design has some overcapacity in that parameter as well.

The stochastic input data to the model, i.e., the choice of containers and their contents, is chosen to be an identical array for each of the experiments that has been done. The series of containers is 76 long, which are selected to avoid containers with purely manually palletizable boxes and to achieve an average pallet factor (number of boxes per pallet) of about 23.

# 2 Experimental Results

In each run of of the simulation model 25 hours of operation are simulated. This is taken identical to the experiments described in [DGHV01]. As the model uses deterministic input, two of the same runs will result in the same outcome, therefore none of the experiments are replicated. In order to remove the influence of any start-up noise, only the last 24,5 hours of the simulated time are used to calculate the performance statistics.

## 2.1 Throughput

Throughput $\delta$ is one of the most important characteristics for the receiving area of a warehouse. Therefore, it is the first dependent variable that is considered in the analysis of the experimental data.

### Average Throughput

In Table 5.1 an overview of the throughput data is given. The numbers are given in pairs, which denote the throughput for the palletizer outputs alone, and for the combined output of palletizers and 'manual' exits, respectively. First of all, it can be seen that the total throughput is linear with the number of extendable conveyors ($\delta \approx 1200 \cdot c$), although this decreases a little for larger values of $c$. This decrease from linearity can be explained by the fact that in the case of more extendable conveyors, fewer position changes can be made, and therefore, conveyors have to wait at a position rather than move to a neighboring position. The former option costs more time and decreases the receiving area's throughput.

The palletizer output numbers are also visualized in Figure 5.1. The general increase of

| $c$ | $b$ | $p = 3$ | $p = 4$ | $p = 5$ |
|---|---|---|---|---|
| 10 | 9 | 2754/5899 | 3698/5902 | 4429/5906 |
| | 7 | 2722/5906 | 3618/5911 | 4330/5909 |
| | 5 | 2581/5912 | 3365/5912 | 3989/5917 |
| | 3 | 2122/5917 | 2626/5919 | 3040/5915 |
| 8 | 9 | 2789/4788 | 3578/4792 | 3871/4803 |
| | 7 | 2735/4792 | 3497/4794 | 3846/4803 |
| | 5 | 2536/4797 | 3188/4796 | 3637/4801 |
| | 3 | 1985/4797 | 2449/4801 | 2820/4803 |
| 6 | 9 | 2598/3597 | 2810/3602 | 2818/3603 |
| | 7 | 2524/3596 | 2798/3601 | 2818/3603 |
| | 5 | 2309/3603 | 2662/3602 | 2786/3603 |
| | 3 | 1764/3605 | 2086/3605 | 2351/3606 |

Table 5.1: Average throughput (measured in boxes/hours) over nearly 25 hours of simulation time. The pairs of numbers indicate throughput $\delta$ for the palletizer output only and palletizer output combined with the 'manual' exit. The standard errors of these means are all less than 50.

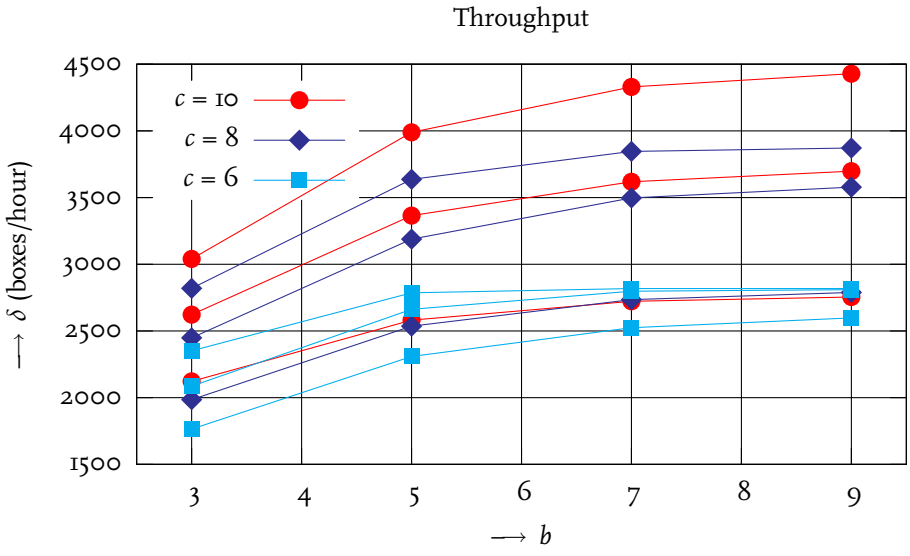


Figure 5.1: Average throughput (measured in boxes/hours) over nearly 25 hours of simulation time. The horizontal axis denotes the number of buffer lanes. Different plotting symbols are used for different numbers of extendable conveyors, while different numbers of palletizers result in several lines with identical plotting symbols.

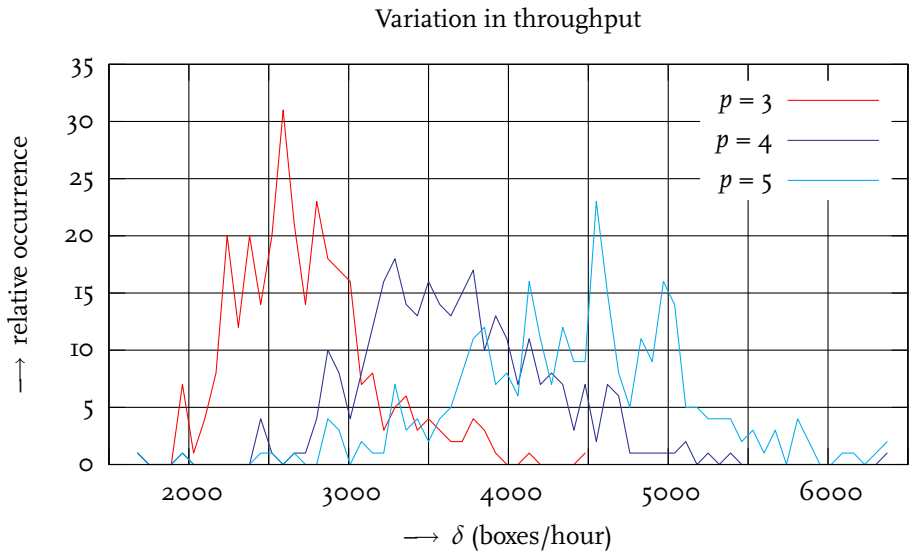


Figure 5.2: Variation in throughput, visualized through a histogram in which the relative occurrence of a range of throughput values is the dependent variable. The throughput values are measured each 5 minutes, while the binsize (throughput quantization unit) is taken 70. This graph only represents the case where the number of extendable conveyors is 10 and the number of buffer lanes is 9.

throughput with higher values of $c$, $b$, and $p$ is clear, although saturation effects are very clear as well. Saturation means that the curves approach a flat asymptote, a limiting value, instead of increasing indefinitely. Variation in the number of palletizers is visualized with identical plotting symbols in order not to further clutter the graph — the resulting lines lie higher with higher values of $p$.

The balance between front-end and back-end capacities is visible from this graph. For $c = 6$ all three curves more or less coincide, while for $c = 8$ the two curves for $p = 4$ and $p = 5$ coincide, and for $c = 10$ all three curves are clearly separated. From this can be inferred that for $c = 2 \cdot p$ the system is well balanced. This relation is weaker for smaller values of $b$, indicating that a stronger coupling between input and output disturbs the simple rule that was found. This conjectured rule-of-thumb $c = 2 \cdot p$ (in order to get a balanced system) has been assumed as a first order approximation in the original design of the receiving area as well.

The influence of the number of buffer lanes is such that the original design is quite optimally chosen again. Leveling of the throughput is visible at $b = 7$ for most curves.

### Variation in Throughput

An important question that relates to throughput is how the throughput capacity is varying over time. This has been considered in one simple case, where the number of palletizers is varied and the other structural parameters have been kept constant ($c = 10$, $b = 9$). This results in the histogram pictured in Figure 5.2.

The center-of-gravity (i.e., the weighted average) of these curves on the $x$-axis is equal to their averages as mentioned in Table 5.1. The curves are quite symmetrical around this point, but

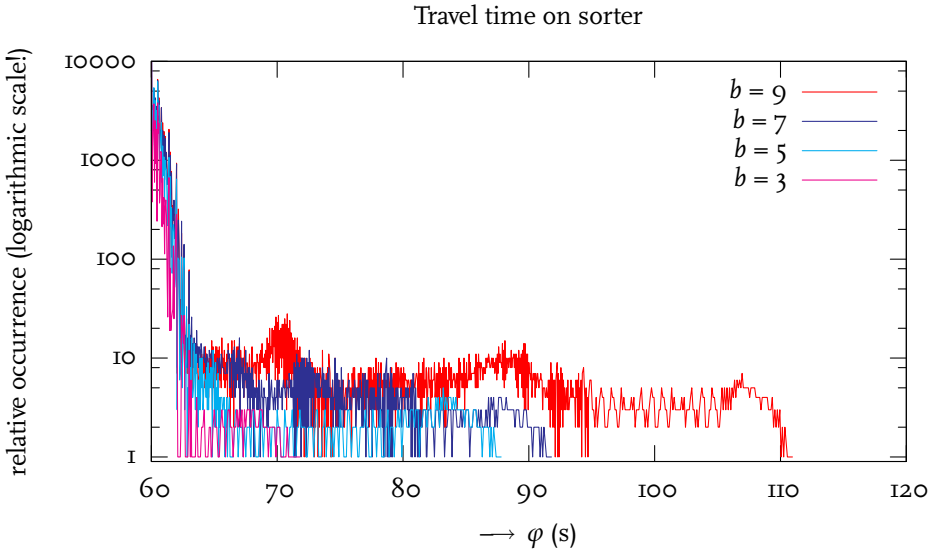


Figure 5.3: Flow time graph for the sorter subprocess. Note that the vertical axis denotes relative occurrence on a logarithmic scale. The numbers of extendable conveyors and palletizers are both taken maximal, i.e., $c = 9$ and $p = 5$.

the higher the average throughput is, the higher the spread of the curve. This means that the system load exhibits more variation for higher throughput. The size of this effect is not very large.

## 2.2 Flow Time

Flow time ($\varphi$) of boxes in the system is one of the few variables that has the property that they add linearly over sub-processes, provided all processes are taken into account. Therefore, it is an important variable to study in order to characterize processes, especially when they are going to be considered in the context of a larger system.

Flow time exhibits quite straightforward properties in our model, at least for the first parts of the model. In the front-end part, boxes either travel 4.3 or 8.6 seconds, depending on their size. A flow time graph would add nothing to this observation, except for rating how many large boxes versus small boxes there are.

The central sorter is modeled in such a way that travel time through the system is incorporated and a maximum capacity is obeyed. The travel time is set to 60 seconds, while the capacity is bounded to 6000 boxes per hour, i.e., one per 0.6 seconds. In Figure 5.3 the flow time is displayed for boxes passing the sorter. The largest strain and therefore, also the largest variations on flow times on the sorter occur when the sorter becomes the limiting factor of the complete system. Hence, in order to study these largest effects, the front-end and back-end capacities are taken maximal, i.e., $c = 9$ and $p = 5$, because in other situations the sorter has a clear overcapacity. To see this, compare its capacity of 6000 boxes per hour with the left-hand side figures in Table 5.1. But even in this case, the flow time graph deviates only little from the 60 seconds delay that is imposed on every box.

To indicate that the effect of the first two parts of the model (front-end and sorter) on the flow time is quite trivial, the total flow time is compared to the flow time in the back-end part of

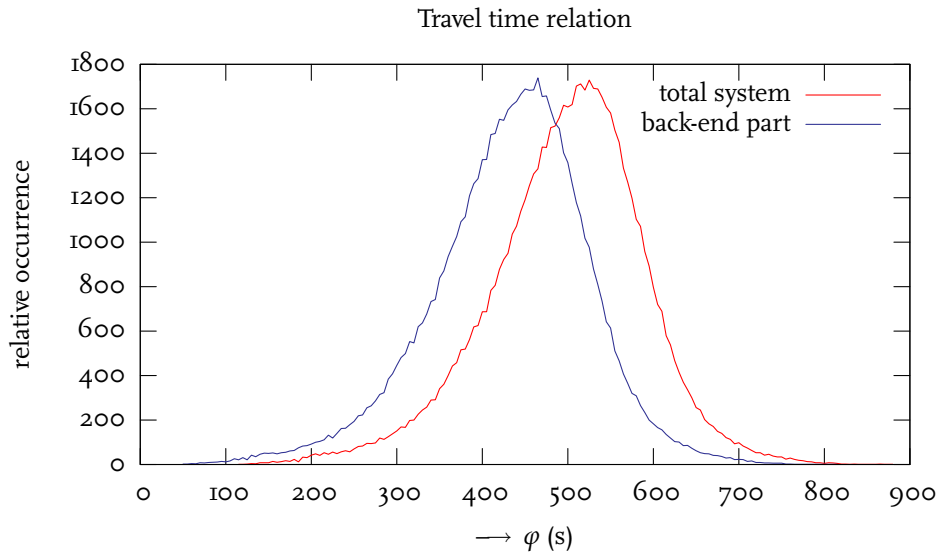


Figure 5.4: Example of flow time graphs for the total process and the palletizing subprocess. The first experiment is taken, i.e., the numbers of extendable conveyors and palletizers are $c = 9$ and $p = 3$.

the model. This is mainly a shift, as can be seen in Figure 5.4.

It is interesting to see how the total flow time changes with different parameter settings. This is shown in Figure 5.5, where all parameter settings are displayed in a matrix, with the number of palletizers varying over rows, the number of extendable conveyors varying over columns, and the number of buffer lanes varying within each plot, shown as different curves.

A number of observations can be made from these graphs in order to build up a better understanding of the studied system. First of all, the curves shift to the right if the number of buffer lanes increases. That is, the average flow time is larger when more buffer space is available inside the system. This only holds, however, if the input capacity exceeds the output capacity as can be seen in Figure 5.5. If it is the other way around, the complete system is just working at nominal throughput of the back-end, independent of the number of buffer lanes. This situation occurs for 5 palletizers with either 8 or 6 extendable conveyors, and for 4 palletizers with 6 extendable conveyors. Again, indirectly it might be incurred that $c = 2 \cdot p$ belongs to an approximately balanced system.

The shape of the curves is not constant. When the palletizers have little buffer capacity, they will cause nominal behavior for a queueing system that delivers output in a batched manner. The curves resemble a $(\varphi - \varphi_0)^{-1}$ relation, with some limiting effect around $\varphi_0$. The heuristic variable $\varphi_0$ denotes the lower bound of the flow time curves. When they have large buffer capacity, the flow times get larger and spread over a larger range of values. Note, that for some combinations, the curves flatten off much stronger. This can be seen for the experiments where 9 lanes have been combined with $c = 2 \cdot p$, the balanced situations. The front-end and back-end both vary around a mutual equilibrium, and as such, sometimes the palletizers catch up with stored work, while at other times they need much longer times.

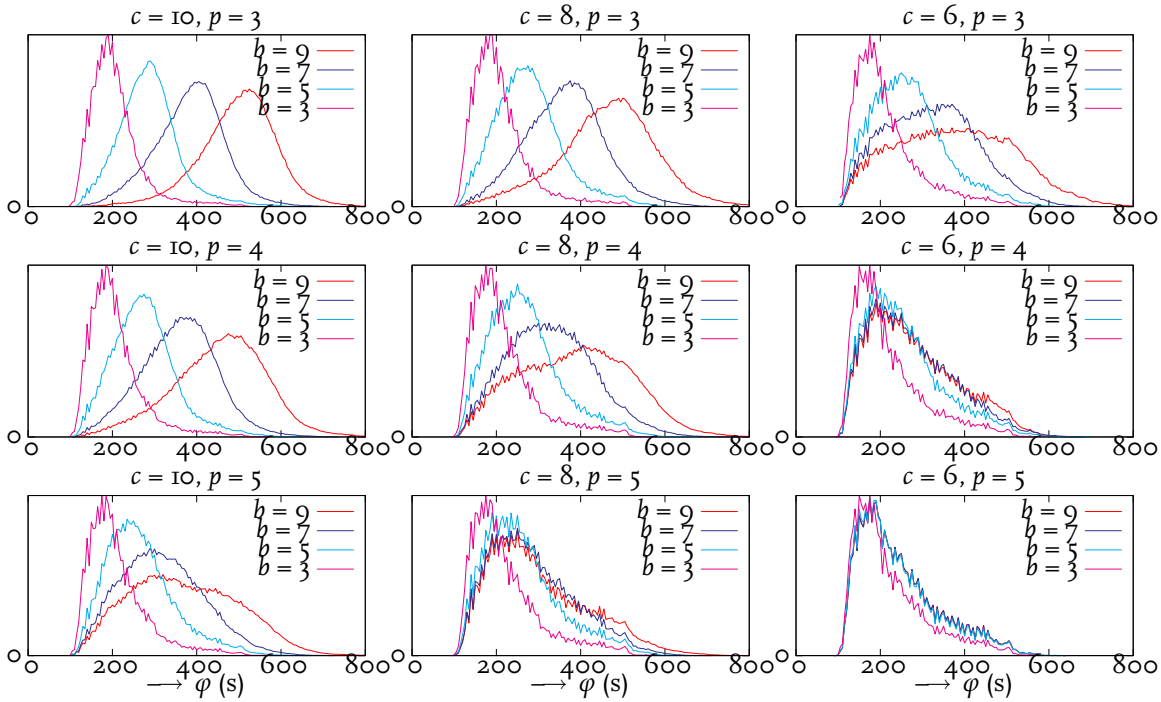


Figure 5.5: Total flow times of automatically palletized boxes as a function of all parameter settings in the experiments that have been run. Over the rows $p$ varies, over the columns $c$ varies, while different values for $b$ are shown in each subgraph. The surface under the curves is equal to the left-hand side figures in Table 5.1.

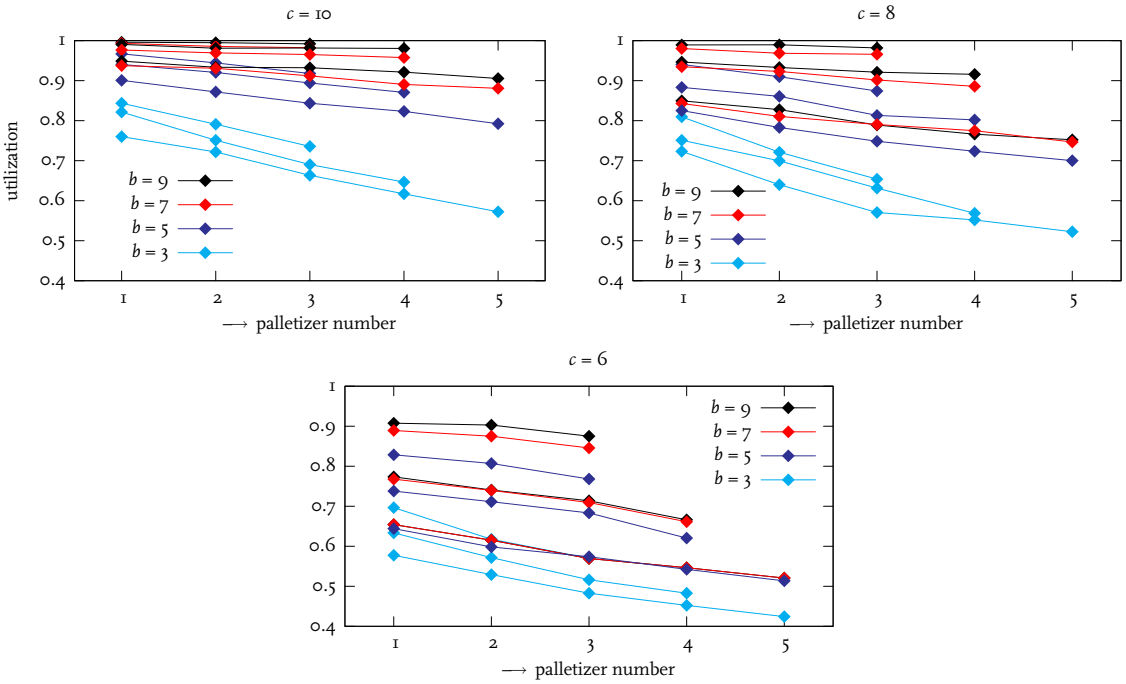


Figure 5.6: Utilization of the palletizers for all different experimental settings. The 3 subgraphs indicate different values of $c$. For each palletizer (indicated on the $x$-axis) the utilization can be read on the vertical axis.

## 2.3 Work-in-Process

In a stationary situation, work-in-process (wip) is related to flow time and throughput by means of Little's law [Lit61]: $w = \varphi \cdot \delta$. On a infinitesimal scale, this can be interpreted as mass conservation, i.e., what goes in, must come out[1]. It can be measured in an independent way in our experiments, and graphs which show $w$ over time give some further insight in the dynamic behavior of the total system. This is however so little, that the graph is left out in this report.

It appears that our model has abstracted from considering $w$. Timing is incorporated in the model, but distances and velocities are not. To study $w$ in more detail, we feel that a more detailed model has to be made that includes a lot more actual design decisions about the physical components that will be applied.

## 2.4 Utilization

### Palletizers

Utilization of the palletizers is an important variable to study, as it indicates how much the system is effectively used. In Figure 5.6 the curves of average utilization for each of the palletizers is plotted.

From the figures it can be concluded that the first palletizers always get the highest load. This

[1] An infinitesimal part of a system can be seen as a volume $\partial V = \partial A \cdot \partial \ell$, which is the product of an area (through which work flows) and a length. Definitions lead to expressions for throughput $\delta = \rho \cdot v \cdot \partial A$, flow time $\partial \varphi = \partial \ell / v$, and work-in-process $\partial w = \rho \cdot \partial V$, in which $\rho$ denotes work density and $v$ the velocity with which the work flows. They can be seen to relate through $\partial w = \delta \cdot \partial \varphi$, which is isomorphic with Little's law.

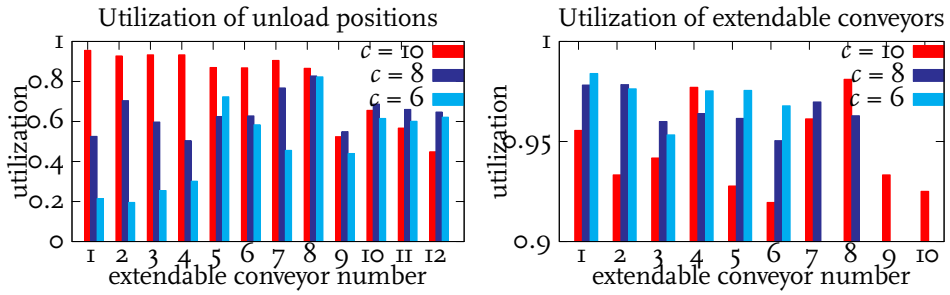


Figure 5.7: Utilization of the unloading positions and extendable conveyors. Only the number of extendable conveyors is varied. Note the difference in scale of the $y$-axes.

is caused by the fact that for equal numbers of free buffer lanes the palletizer with the lowest index is selected first. Hence, this is a modeling artefact, as the selection procedure could have been chosen differently as well.

For lower numbers of buffer lanes the load decreases. This can be understood when one realizes that in the case of fewer buffer lanes, less work in advance can be scheduled to a palletizers and hence, the chance that such a palletizer runs out of work is higher.

Lastly, if more palletizers are involved in the system, the load per palletizer is lower. This seems trivial: the work spreads over more palletizers and hence, each palletizer has less work to do. However, the sum of work that is done also increases, as the sum of the utilizations grows. Only for the right-hand graphs ($c = 6$), and larger number of buffer lanes, also the sum of utilizations is fairly constant, which means that all the work that can be dispatched to the palletizers is actually handled (i.e., there is overcapacity in the back-end).

### Unloading Positions & Extendable Conveyors

In Figure 5.7 the utilization in the front-end of the system is displayed. Only the number of extendable conveyors is varied, because the graphs are independent of the other structural parameters of the model. This can be explained by the fact that the manual exit causes the back-end to have infinite capacity from the viewpoint of the front-end part (see Section 4.2.6).

The unloading positions are used less if there are fewer extendable conveyors. All in all, their utilization is quite low, as the number of extendable conveyors does not match the number of unloading positions. Further, it might be observed that the balance of utilization over the different unloading positions is far from optimal. This could be avoided by other scheduling schemes for the extendable conveyors.

The utilization of the extendable conveyors is very high. The variation in utilization merely indicates the variation in the unload times of the containers, and the scheduling of the extendable conveyors which can result in 500 seconds waiting for a new container or 120 seconds moving from one position to another.

# Chapter 6
# Conclusions and Future Work

## 1 Conclusions

This report has focussed on a simulation-based performance analysis of a container unloading and palletizing process. This system is already operational, therefore our initial layout closely resembles the real-world layout. We varied the number of extendable conveyors, the number of buffer lanes, and the number of palletizers to come to an optimal solution. From the simulation results it can be concluded that the number of extendable conveyors is most determinant for the input capacity and the number of palletizers is the most determinant factor for the output capacity of the palletizing area. When the input capacity exceeds the output capacity, an increase in the number of buffer lanes increases the average flow time of boxes in the system. For a smaller number of buffer lanes, the average throughput decreases. This is because less work can be scheduled in advance. When the output capacity exceeds the input capacity, the system works at nominal capacity of the palletizers and is therefore nearly independent of the number of buffer lanes. Based on the required average unloading and automatic palletizing capacity of 3400 boxes/hour, the best fitting solution has the following structure parameters: 8 extendable conveyors, 4 palletizers, and 7 buffer lanes per palletizer. This conclusion is consistent with the findings described in an earlier study [DGHV01].

We proposed a divide–and–conquer approach to do a performance analysis of a warehouse. The running time for the simulation of the $\chi$ model of one, rather small subsystem of a warehousing system takes more time than is actually simulated. Therefore we can assume that doing a performance analysis of an entire warehousing system by integrating the simulation models of the subsystems into one large simulation model is not a workable solution. The running time of such a simulation will probably be way too long. Not only this time complexity is a problem, it is also very hard to comprehend the behavior of such a complex, integrated system. We already needed another representation to capture the dynamics of the palletizer model to help us understand the behavior of this small system (see Figure 4.5 on Page 24). Therefore, other techniques are required to do a performance analysis of larger systems. Aggregate modeling [LA07] appears to be a good candidate for this, because aggregate models are very small, while the characteristics of all subsystems are maintained in the larger simulation model.

The use of the $\chi$ formalism proves practical for use in a logistical environment. The fact that it uses parallel processes allows to think locally about behavior, i.e., from the perspective of one process. Still, the effects of interoperation of parallel processes is difficult to comprehend. The principle *simulation time* is indispensable when analyzing systems of this kind. It allows to completely ignore calculation overhead and thus really focusses on the physical system performance.

# 2 Directions for Further Research

The unloading and palletizing system is only the first of a number of warehouse processes that have to be analyzed. There are a lot more subsystems that need to be modeled in order to come to a performance analysis of an entire warehousing system. As we have shown it is infeasible to do this in a straightforward way by putting simulation models together. To allow modeled systems to be integrated into a larger simulation model, aggregate models are needed. The palletizing system could be aggregated as two processes with a buffer in between: one process representing the unloading processes (the input), the other representing the palletizing processes (the output). The exact configuration and the precise characteristics of the aggregate model is left open for further research.

As can be concluded from the graphs in Figure 5.6, the load balancing between extendable conveyors is not optimal. More advanced scheduling schemes for the positions of the extendable conveyors may decrease these utilization differences. This may also lead to a higher overall system performance as the input capacity increases.

The presented approach starts from an existing design and predetermined process policies that eventually determine the system's behavior. Ultimately, one would like to work the other way around: given required system behavior, the process policies can be determined.

Apart from coming to a more complete and optimal model it is also interesting to research other uses for the $\chi$ model and for $\chi$ models in general. One possibility is to automatically generate diagrams like the one depicted in Figure 4.5. Another interesting research direction is to generate (parts of the) controller code for the different physical parts directly from the $\chi$ model.

# Bibliography


[AWK03] *GAWK: Effective AWK Programming: A User's Guide for GNU Awk*, third edition, 2003.

[BME+07] G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen, and K. A. Houston. *Object-Oriented Analysis and Design with Applications*. Object Technology Series. Addison Wesley Professional, Longman, third edition, 2007.

[DGHV01] R. Debets, D. Gristy, E. Hessel, and M. Veenman. Automatic palletising solution analysis. Multi-discipline report 94293–086–11201–EN–B, Vanderlande Industries, Veghel, 2001.

[FAL06] FALCON website. `http://www.esi.nl/falcon/`, 2006.

[Fri06] J. E. F. Friedl. *Mastering Regular Expressions*. O'Reilly, Sebastopol, third edition, August 2006.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242, Jyväskylä, Finland, June 1997. Springer-Verlag, Berlin.

[LA07] E. Lefeber and H. D. Armbruster. Aggregate modeling of manufacturing systems. SE Report 2007-02, Eindhoven University of Technology, Eindhoven, 2007.

[Lit61] J. D. C. Little. A proof of the queueing formula $l = \lambda w$. *Operations Research*, 9:383–387, May–June 1961.

[LR06] E. Lefeber and J. E. Rooda. Modeling and analysis of manufacturing systems. SE Report 2006-01, Eindhoven University of Technology, Eindhoven, 2006.

[vBMR+06] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers. Syntax and consistent equation semantics of hybrid chi. *Journal of Logic and Algebraic Programming*, 68(1–2):129–210, June–July 2006.

[VR06] J. Vervoort and J. E. Rooda. *Learning Timed $\chi$ 1.0*. Technische Universiteit Eindhoven, Department of Mechanical Engineering, Systems Engineering Group, Eindhoven, November 2006.

# Appendix A
# Detailed Requirements

This appendix lists the requirements extracted from Section 3.2 as well as some additional requirements that were necessary to fully model the system.

The requirements are grouped in two categories:

- High-level requirements
- Detailed requirements

All requirements have a unique identifier.

## 1 High-Level Requirements

| Identifier | Requirement |
|---|---|
| HLR-1 | Boxes must be unloaded. |
| HLR-2 | Boxes must be pre-sorted to manual or automated palletizing. |
| HLR-3 | Automatically palletizable boxes must be merged on take-away conveyors. |
| HLR-4 | Automatically palletizable boxes must be sorted in buffer lanes. |
| HLR-5 | Automatically palletizable boxes must automatically be palletized. |
| HLR-6 | Full pallets must be wrapped. |

Table A.1: High-level requirements

# 2 Detailed Requirements

## 2.1 Container Marshalling

| Identifier | Requirement |
|---|---|
| DR-1 | Twelve unloading positions are available. |
| DR-2 | Twelve take-away conveyors are available. |
| DR-3 | Eight moveable extendable conveyors connect the unloading positions with the take-away conveyors. |
| DR-4 | At start-up the extendable conveyors are placed at doors 2, 3, 5, 6, 8, 9, 11, and 12. |
| DR-5 | When a container is empty, the extendable conveyor used for unloading it will move to the nearest unloading position that has a full container available as soon as it is available. |
| DR-6 | If multiple full containers are available at the same time at the same distance, the extendable conveyor will go to the unloading position left from its current position. |
| DR-7 | An extendable conveyor only moves to another unloading position if a full container is available. |
| DR-8 | An extendable conveyor needs to be empty before it can be moved. |
| DR-9 | Extendable conveyors cannot cross each other. |
| DR-10 | Changing an extendable conveyor between doors takes 120 seconds. |
| DR-11 | Replacing a container at a door takes 500 seconds. |

Table A.2: Detailed requirements — Container Marshalling

## 2.2 Container Unloading

| Identifier | Requirement |
|---|---|
| DR-12 | Before an operator can start unloading, the WMS/WCS needs to be informed what SKU is being unloaded. |
| DR-13 | Informing the WMS/WCS of an SKU changeover takes 120 seconds. |
| DR-14 | An operator will start unloading when a container and an extendable conveyor are in place. |
| DR-15 | If a container contains more than one SKU, the operator will first unload all items of SKU 1, then all items of SKU 2, etc. |
| DR-16 | The operator will unload boxes at a constant rate. |

Table A.3: Detailed requirements — Container Unloading

### 2.3 Pre-sorting

| Identifier | Requirement |
|---|---|
| DR-17 | At the end of the extendable conveyors boxes are pre-sorted to automatic or manual palletizing. |
| DR-18 | A box will be manually palletized if any of the following situations occur:<br>• it is technically impossible to automatically palletize a boxes of that SKU type,<br>• it is impossible to make a buffer reservation at any of the palletizers because none of them has enough buffer lanes available to buffer a complete pallet load of those boxes. |

Table A.4: Detailed requirements — Pre-sorting

### 2.4 Layer Palletizing

| Identifier | Requirement |
|---|---|
| DR-19 | There are three layer palletizing machines. |
| DR-20 | Every layer palletizing machine has seven buffer lanes feeding boxes to it. |
| DR-21 | The length of such a buffer lane is 13.185 meters. |
| DR-22 | The capacities of the layer palletizing equipment are listed in Table 3.2 on page 15. |
| DR-23 | A layer palletizing machine can palletize one pallet at a time. |
| DR-24 | A buffer cannot be partially flushed. |
| DR-25 | The layer palletizing machine requires a full pallet load to be buffered prior to processing it. |
| DR-26 | On SKU changeover the palletizer shall complete the buffered boxes of the previous SKU's as a partial pallet. |
| DR-27 | Overflow must be avoided. |
| DR-28 | Pallet pattern changeover should be minimized. |
| DR-29 | An SKU will be allocated to the palletizer based on the required number of buffers (If $x$ lanes are needed, then $x$ buffer lanes need to be available). |
| DR-30 | If multiple palletizers have enough buffer lanes available, the one with the most free lanes is chosen. |
| DR-31 | If a palletizer with sufficient buffer lanes is available the system shall reserve buffer lanes the new SKU. |
| DR-32 | A database with SKU data is required. |

Table A.5: Detailed requirements — Layer Palletizing

# Appendix B

# $\chi$ Model

This appendix contains the $\chi$ 1.0 code of our simulation model. Notice the DEFINE declarations in the beginning of the listing. This is not standard $\chi$ code, but directives for our preprocessor (see Appendix C).

```
from standardlib import *

DEFINE %NCONT 76
DEFINE %NCUL 12
DEFINE %NXC  8
DEFINE %NBUF 7
DEFINE %NPAL 4
DEFINE %MINLENGTH_LARGE 600
DEFINE %UNLOADTIME_LARGE 8.6
DEFINE %UNLOADTIME_SMALL 4.3
DEFINE %OPERATORCHANGETIME_SKU 120.0
DEFINE %XCCHANGETIME 120.0
DEFINE %CONTAINERCHANGETIME 500.0
DEFINE %BOXTRAVELTIME 60.0
DEFINE %SORTERHOURCAPACITY 6000.0
DEFINE %LANELENGTH 13185
DEFINE %PALLETIZERCHANGETIME_SKU 10.0
DEFINE %PALLETCHANGETIME 15.0

type box =    (nat, nat, nat, real, real) //(id, SKU, pid,
                                            // timestamp(abs), timestamp(rel))
 , boxdata =   box
 , container = (nat, nat, nat)          //(id, sku, # of SKU)
 , pallet =    (nat, [box])             //(id, list of boxes)
 , xc_index =   nat                     // extendable conveyor index
 , ul_index =   nat                     // unloader index
 , ecs_data =  (bool, ul_index)         // extendable conveyor store
 , spid =      (nat, nat)               //(sku id, pallet id)
 , skudata =   (nat, nat, nat, bool, nat, real) //(id, palletfactor, skulength,
                                                 // autopalletize?,
                                                 // layers per pallet,
                                                 // time per layer)
 , logentry = (string, string, nat, real) //(process, variable, logtype, value)
```

```
func digit ( val n: nat) -> string = |[ ret <"0", "1", "2", "3", "4", "5", "6", "7", "8", "9">.n ]|

func nat2str (val n: nat) -> string =
|[ var s: string = ""
:: ( n = 0 -> s := "0" | n > 0 -> skip )
 ; (n > 0) *> ( s:= digit (n mod 10) ++ s; n:= n div 10 )
 ; ret s
]|

func name4me(val s: string, n: nat) -> string = |[ ret s ++ nat2str(n) ]|

func getContents(val cid: nat, cdb: [container]) -> [container] =
|[ var x: [container] = [], c: container
:: (len(cdb) > 0)
    *>(  c := hd(cdb); cdb := tl(cdb)
      ; ( c.0  = cid -> x := x ++ [c]
        | c.0 /= cid -> skip
        )
      )
 ; ret x
]|

func getSkuData(val sku: nat, sdb: [skudata]) -> skudata =
|[ var s: skudata
:: (len(sdb) > 0)
    *>(  s := hd(sdb); sdb := tl(sdb)
      ; ( s.0  = sku -> ret s
        | s.0 /= sku -> skip
        )
      )
]|

func getPalletFactor(val sku: nat, sdb: [skudata] ) -> nat =
|[ ret getSkuData(sku, sdb).1 ]|

func getSkuLength(val sku: nat, sdb: [skudata]) -> nat =
|[ ret getSkuData(sku, sdb).2 ]|

func getAutoPalletize(val sku: nat, sdb: [skudata]) -> bool =
|[ ret getSkuData(sku, sdb).3 ]|

func getPalletTime(val sku: nat, sdb: [skudata], nr: nat) -> real =
|[var s: skudata = getSkuData(sku, sdb), a: nat
:: a := nr mod s.4;
   ( a > 0 -> ret s.5 * (nr div s.4 + 1)
   | a = 0 -> ret s.5 * (nr div s.4)
   )
]|

proc DB(val cdb: [container], sdb: [skudata],
        chan gc?: nat, c!: [container], gpf?, pf!: 2 # nat, gsl?, sl!: 3 # nat
           , gap?: nat, ap!: bool, gpt?: (nat, nat), pt!: real) =
|[ var cid, sku, nr: nat
:: *(  gc?cid;  c!getContents(cid, cdb)
    | (|, i<-0..1, gpf.i?sku; pf.i!getPalletFactor(sku, sdb))
    | (|, j<-0..2, gsl.j?sku; sl.j!getSkuLength(sku, sdb))
    | gap?sku; ap!getAutoPalletize(sku, sdb)
    | gpt?(sku, nr); pt!getPalletTime(sku, sdb, nr)
    )
]|

proc CidG(chan a!: nat) =
|[ var cid_max: nat = %NCONT, cid_min: nat = 1, cid: nat
:: cid := cid_min ; *( a!cid; cid := cid + 1;
                       ( cid >  cid_max -> cid := cid_min
```

```
                        | cid <= cid_max -> skip
                        )
                      )
]|

proc PidG(chan a!: nat) = |[ var pid: nat = 0 :: *( a!pid; pid := pid + 1 ) ]|

proc BidG(chan a!: nat) = |[ var bid: nat = 1 :: *( a!bid; bid := bid + 1 ) ]|

proc SConG(chan a!: [[box]], c?, p?: nat, gc!: nat, rc?: [container]
             , gpf!, rpf?: nat) =
|[ var cid: nat, ccs: [container], cc: container, pf: nat, bl, blp: [box]
     , bid, pid: nat, xss: [[box]]
:: *(  xss := []
    ; c?cid; gc!cid; rc?ccs
    ; (len(ccs) > 0)
       *>( cc := hd(ccs); ccs := tl(ccs)
         ; gpf!(cc.1); rpf?pf
         ; blp := []; pf > len(blp)+1 *> (blp := blp ++ [(0, cc.1, 0, 0.0, 0.0)])
         ; (cc.2 >= pf)
            *>( p?pid
              ; bl := [(0, cc.1, pid, 0.0, 0.0)] ++ blp
              ; cc.2 := cc.2 - pf
              ; xss := xss ++ [bl]
              )
         ; ( cc.2 > 0 -> p?pid
                       ; bl := []
                       ; (cc.2 > 0)
                          *>( bl := [(0, cc.1, pid, 0.0, 0.0)] ++ bl
                            ; cc.2 := cc.2 - 1
                            )
                       ; xss := xss ++ [bl]
           | cc.2 = 0 -> skip
           )
         )
    ; a!xss
    )
]|

proc ConG(chan a!: %NCUL # [[box]], gc!: nat, con?: [container]
            , gpf!, pf?: nat) =
|[ chan c, p: nat
:: (||, j<-0..%NCUL-1, SConG(a.j, c, p, gc, con, gpf, pf)) || CidG(c) || PidG(p)
]|

func init_pos_conv(val i: xc_index, n: nat) -> ul_index =
|[ ( (i >= 0 and i < n/3) -> ret 3*i
   | (i >= n/3 and i< 2*n/3) -> ret ( 3*(i- (n div 3)) + 1 )
   | (i >= 2*n/3 and i < n) -> ret ( 3*(i-2*(n div 3)) + 2 )
   )
]|

proc XC(val id: nat, chan l!: logentry, xcf!: %NCUL#void, xc!: %NCUL#nat
                       , xcd?: void, xcr!: %NCUL#nat, xca?: bool,
        val init_pos: nat) =
|[ var myName: string = name4me("XC", id), ulid: nat, pos: nat = init_pos
     , conv_needed: bool, target_pos: nat = init_pos
     , u: ->nat = uniform(0, 2), v: nat
:: *( ( target_pos /= pos -> xcf.pos!; delay %XCCHANGETIME
                                     ; pos := target_pos; xc.pos!id
      | target_pos  = pos -> xc.pos!id
      )
    ; l!(myName, "util", 1, 0.0)
    ; xcd?
    ; l!(myName, "util", 1, 1.0)
```

```
    ; conv_needed := false; v := sample u
    ; ( v>1 -> skip
      ; ( pos > 0 and not conv_needed -> xcr.(pos-1)!id; xca?conv_needed
         ; ( conv_needed -> target_pos := pos - 1 | not conv_needed -> skip )
        | pos = 0 or conv_needed -> skip
        )
      ; ( pos < %NCUL-1 and not conv_needed -> xcr.(pos+1)!id; xca?conv_needed
         ; ( conv_needed -> target_pos := pos + 1 | not conv_needed -> skip )
        | pos = %NCUL-1 or conv_needed -> skip
        )
    | v<2 -> skip
      ; ( pos < %NCUL-1 and not conv_needed -> xcr.(pos+1)!id; xca?conv_needed
         ; ( conv_needed -> target_pos := pos + 1 | not conv_needed -> skip )
        | pos = %NCUL-1 or conv_needed -> skip
        )
      ; ( pos > 0 and not conv_needed -> xcr.(pos-1)!id; xca?conv_needed
         ; ( conv_needed -> target_pos := pos - 1 | not conv_needed -> skip )
        | pos = 0 or conv_needed -> skip
        )
    )
  )
]|

func calcdelay(val sl: nat) -> real =
|[ ( sl <= %MINLENGTH_LARGE -> ret %UNLOADTIME_SMALL
  | sl > %MINLENGTH_LARGE -> ret %UNLOADTIME_LARGE
  )
]|

proc ConUnl(val id: nat, chan l!: logentry, a?: [[box]], bi?: nat, b!: box
                           , bd!: boxdata, xcr?: nat, xca!: %NXC#bool
                           , xcf?: void, xc?: nat, xcd!: %NXC#void, lfs!: nat
                           , gsl!: nat, rsl?: nat) =
|[ var myName: string = name4me("CU", id), xss: [[box]], xs: [box], x, xd: box
     , sl: nat, cur: nat = 0, curdelay: real, i, xcid: nat
     , conv_present: bool = false, pid: nat, bid: nat, nx: nat
:: *( ( xcr?i; (     conv_present -> xca.i!false
               | not conv_present -> xca.i!true; conv_present := true
               )
      | xcf?; conv_present := false
      )
    )
|| *( a?xss
    ; xc?xcid; conv_present := true
    ; l!(myName, "util", 1, 0.0); nx := 0
    ; (len(xss) > 0)
       *> ( xs := hd(xss); xss := tl(xss)
          ; xd := hd(xs); pid := xd.2; bd!xd; !!time, " cu: ", id, ", sku: ", xd.1, "\n"
          ; ( xd.1  = cur -> skip
            | xd.1 /= cur -> delay %OPERATORCHANGETIME_SKU
                           ; cur := xd.1; gsl!(xd.1); rsl?sl; curdelay := calcdelay(sl)
            )
          ; (len(xs) > 0)
             *> ( x := hd(xs); ( x.0 = 0 -> bi?bid; x.0 := bid
                               | x.0 > 0 -> skip
                               ); x.2 := pid; xs := tl(xs)
                ; ( len(xs)  = 1 -> bi?bid; xd := hd(xs)
                                    ; xd.0 := bid; xs := [xd]; lfs!bid
                  | len(xs) /= 1 -> skip
                  )
                ; x.3 := time; x.4 := time; delay curdelay; b!x; nx := nx + 1
                )
          )
    ; l!(myName, "util", 1, 1.0)
    ; xcd.xcid!
```

```
    ; delay %CONTAINERCHANGETIME
    )
]|

func inlistspid(val id: spid, ids: [spid]) -> bool =
|[ ret (+, x <- ids, x = id, 1) > 0 ]|

proc PreSortC(val id: nat, chan bd?: boxdata, ab!: (nat, boxdata), ba?: bool
                            , aom? :boxdata, tom!: bool
                            , gap!: nat, rap?: bool)=
|[ var myName: string = name4me("PreC", id), xd: boxdata
     , ap: bool, mids: [spid] = [], buf: bool, cur: boxdata
     , mid: spid = (0, 0), same: bool
:: *( bd?xd
    ; gap!xd.1; rap?ap
    ; ( not ap -> mids := mids ++ [(xd.1, xd.2)]
      | ap -> ab!(id, xd); ba?buf
                ; (     buf -> skip
                  | not buf -> mids := mids ++ [(xd.1, xd.2)]
                  )
      )
    | aom?cur; same := (cur.1, cur.2) = mid
    ; (     same -> skip
      | not(same) -> mids := mids -- [mid]; mid := (cur.1, cur.2)
      )
    ; tom!(inlistspid((cur.1, cur.2), mids))
    )
]|

proc PreSortG(val id: nat, chan l!: logentry, a?: box
                            , aom!: boxdata, tom?: bool, b!, m!: box)=
|[ var myName: string = "PreG", x: box, man: bool, t: real
::*(  a?x; aom!x; tom?man
    ; (  man -> t := time - x.3; m!x
                            ; l!(myName, "a/m", 2, 0.0)
                            ; l!(myName, "mx", 2, t)
      | not man -> t := time - x.3; x.4 := time; b!x
                          ; l!(myName, "a/m", 2, 1.0)
                          ; l!(myName, "ax", 2, t)
                          ; l!(myName, "EPTx", 0, t)
      )
  )
]|

proc PreSort(val id: nat, chan l!: logentry, a?, b! , m!: box, bd?: boxdata
                           , ab!: (nat, boxdata), ba?: bool, gap!: nat
                           , rap?: bool) =
|[ chan aom: boxdata, tom: bool
:: PreSortC(id, bd, ab, ba, aom, tom, gap, rap) || PreSortG(id, l, a, aom, tom, b, m)
]|

proc UnlA(chan l!: logentry, a?: %NCUL#[[box]], b!, m!: box
            , ab!: (nat, boxdata), ba?: %NCUL#bool, lfs!: nat
            , gsl!: nat, rsl?: nat, gap!: nat, rap?: bool) =
|[ chan bi: nat, xcr: %NCUL#nat, xca: %NXC#bool, xcf: %NCUL#void, xc: %NCUL#nat
      ,xcd: %NXC#void, x: %NCUL#box, xd: %NCUL#boxdata
:: (||, j<-0..%NCUL-1,
    (    ConUnl(j, l, a.j, bi, x.j, xd.j, xcr.j, xca, xcf.j, xc.j, xcd, lfs, gsl, rsl)
      || PreSort(j, l, x.j, b, m, xd.j, ab, ba.j, gap, rap)
    )
   )
|| (||, j<-0..%NXC-1, XC(j, l, xcf, xc, xcd.j, xcr, xca.j, init_pos_conv(j, %NCUL)) )
|| BidG(bi)
]|
```

```
func laneLength() -> nat = |[ ret %LANELENGTH ]|

func boxPerLane(val sl: nat) -> nat = |[ ret laneLength() div sl ]|

func lanesReq(val sl, pf: nat) -> nat =
|[ var bpl: nat = boxPerLane(sl)
:: ( pf mod bpl  = 0 -> ret pf div bpl
   | pf mod bpl /= 0 -> ret pf div bpl + 1
   )
]|

func box2PostS(val idsp: spid, stps: [(spid, nat)]) -> nat =
|[ var stp: (spid, nat)
:: (len(stps) > 0)
    *> ( stp := hd(stps); stps := tl(stps)
       ; ( stp.0  = idsp -> ret stp.1
         | stp.0 /= idsp -> skip
         )
       )
 ; !!"ERROR! This should never occur: box is not assigned to a postsorter\n"
]|

func selectPostS(val psas: %NPAL*nat) -> (bool, nat) =
|[ var i: nat = 0, res: (bool, nat) = (false, 0), max_i: nat = 0
:: (i <= %NPAL-1)
    *> ( ( psas.i >  max_i -> res := (true, i); max_i := psas.i
         | psas.i <= max_i -> skip
         ); i := i + 1
       )
 ; ret res
]|

func inlist(val id: nat, ids: [nat]) -> bool =
|[ ret (+, x <- ids, x = id, 1) > 0 ]|

func init_psas() -> %NPAL*nat =
|[ var init: %NPAL*nat, i: nat = 0
:: i<%NPAL *> ( init.i := 0; i := i + 1 ); ret init
]|

proc SortC(chan lg!: logentry, ab?: (nat, boxdata), rr!: %NPAL#nat, bfa?: %NPAL#nat
             , cr!: %NPAL#(spid, nat), ba!: %NCUL#bool, rps?: boxdata, aps!: nat
             , lfs?: nat, lbr!: %NPAL#nat
             , gsl!: nat, rsl?: nat, gpf!: nat, rpf?: nat) =
|[ var myName: string = "SorC", pre: nat, bd, bdm: boxdata, sku, pid: nat, lr: nat
     , psas: %NPAL*nat = init_psas(), as: bool, psid: nat, stps: [(spid, nat)] = []
     , l: nat, ls: [nat], ps: nat, lb: bool, sl, pf: nat
:: *(  ab?(pre, bd); sku, pid := bd.1, bd.2
    ; gsl!sku; rsl?sl; gpf!sku; rpf?pf; lr := lanesReq(sl, pf)
    ; (||, j <- 0..%NPAL-1, rr.j!lr; bfa.j?(psas.j))
    ; (as, psid) := selectPostS(psas)
    ; (  as -> stps := stps ++ [((sku, pid), psid)]; cr.psid!((sku, pid), lr)
      | not as -> skip
      )
    ; ba.pre!as
    |  lfs?l; ls := ls ++ [l]
    |  rps?bdm
    ; ps := box2PostS((bdm.1, bdm.2), stps)
    ; lb := inlist(bdm.0, ls)
    ; (     lb  -> ls := ls -- [bdm.0]; stps := stps -- [((bdm.1, bdm.2), ps)]
                 ; lbr.ps!bdm.0
      | not(lb) -> skip
      )
    ; aps!ps
    )
```

```
]|

proc SortG(chan l!: logentry, a?: box, rps!: boxdata, aps?: nat, b!: %NPAL#box) =
|[ var myName: string = "Sort", x, y: box, xs: [box] = [], psid: nat
     , tt, t: real, dt: real = 3600.0/%SORTERHOURCAPACITY
:: *( a?x; x.4 := x.4 + %BOXTRAVELTIME ; xs := xs ++ [x] )
|| tt := time + dt
  ; *( len(xs)>0 -> skip; y := hd(xs); xs := tl(xs)
      ; t := y.4 - time; ( t>0.0 -> delay t | t<=0.0 -> skip )
      ; t := tt - time; ( t>0.0 -> delay t | t<=0.0 -> skip ); tt := time + dt
      ; rps!y; aps?psid
      ; t := time - y.4; y.4 := time; b.psid!y
      ; l!(myName, "EPTx", 0, t); l!(myName, "x", 2, t)
      )
]|

proc Sort(chan l!: logentry, a?: box, b!: %NPAL#box, ab?: (nat, boxdata)
             , rr!: %NPAL#nat, bfa?: %NPAL#nat, cr!: %NPAL#(spid, nat)
             , ba!: %NCUL#bool, lfs?: nat, lbr!: %NPAL#nat
             , gsl!: nat, rsl?: nat, gpf!: nat, rpf?: nat) =
|[ chan rps: boxdata, aps: nat
:: SortC(l, ab, rr, bfa, cr, ba, rps, aps, lfs, lbr, gsl, rsl, gpf, rpf)
|| SortG(l, a, rps, aps, b)
]|

func lanesAv(val la: %NBUF*(spid, nat)) -> nat =
|[ ret (+, j <- 0..%NBUF-1, ((la.j).0).0 = 0, 1) ]|

func box2Lane(val idsp: spid, la: %NBUF*(spid, nat), sl: nat) -> nat =
|[ var i: nat = 0, bpl: nat = boxPerLane(sl), b: bool
:: (i < %NBUF)
    *> ( b := la.i.0 = idsp and la.i.1 < bpl
       ; ( b -> ret i | not b -> i := i + 1 )
       )
 ; !!"ERROR! This should never occur: box not assigned to lane\n"
]|

func initLa() -> %NBUF*(spid, nat) =
|[ var init: %NBUF*(spid, nat), i: nat = 0
:: i<%NBUF *> ( init.i := ((0, 0), 0); i := i + 1 ); ret init ]|

func resLane(val la: %NBUF*(spid, nat), rs: (spid, nat)) -> %NBUF*(spid, nat) =
|[ var i: nat = 0, nl: nat = rs.1
:: (i < %NBUF and nl > 0)
    *> ( ( la.i.0.0  = 0 -> la.i.0 := rs.0; nl := nl - 1
   | la.i.0.0 /= 0 -> skip
   )
 ; i := i + 1
 )
 ; ret la
]|

func inlistbd(val id: boxdata, ids: [boxdata]) -> bool =
|[ ret (+, x <- ids, x = id, 1) > 0 ]|

func skupal2Lanes(val psid: spid, la: %NBUF*(spid, nat)) -> [nat] =
|[ var i: nat = 0, ls: [nat] = []
:: (i < %NBUF)
    *>( ( la.i.0  = psid -> ls := ls ++ [i]
        | la.i.0 /= psid -> skip
        )
      ; i := i + 1
      )
 ; ret ls
]|
```

```
proc PostSortC(val id: nat, chan lg!: logentry, rr?: nat, bfa!: nat
                              , cr?: (spid, nat), le?: nat, fl!: [nat]
                              , rl?: boxdata, lbr?: nat
                              , gsl!: nat, rsl?: nat, al!: nat, cd?: void) =
|[ var myName: string = name4me("PoSC", id), r, nla: nat
     , la: %NBUF*(spid, nat) = initLa(), rs: (spid, nat), lid: nat
     , bd: boxdata, l: nat, lb: nat, lbs: [nat], lfp: bool, lf: [nat], sl: nat
:: *(  rr?r; nla := lanesAv(la)
    ; ( nla <  r -> nla := 0
      | nla >= r -> skip
      )
    ; bfa!nla
    |  cr?rs; la := resLane(la, rs)
    |  le?lid; la.lid := ((0, 0), 0)
    |  lbr?lb; lbs := lbs ++ [lb]
    |  rl?bd; gsl!(bd.1); rsl?sl
    ; l := box2Lane((bd.1, bd.2), la, sl); al!l; cd?; (la.l).1 := (la.l).1 + 1
    ; lfp := inlist(bd.0, lbs)
    ; (     lfp  -> lbs := lbs -- [bd.0]; lf := skupal2Lanes((bd.1, bd.2), la)
                  ; fl!lf
      | not(lfp) -> skip
      )
    )
]|

proc PostSortG(chan lg!: logentry, a?: box, rl!: boxdata, al?: nat, b!: %NBUF#box
                  ,cld?: %NBUF#void, cd!: void) =
|[ var myName: string = "PoSG", x: box, l: nat
:: *( a?x; rl!x; al?l; b.l!x; cld.l?; cd! )
]|

proc PostSort(val id: nat, chan l!: logentry, a?: box, b!: %NBUF#box, rr?: nat
                              , bfa!: nat, cr?: (spid, nat), le?: nat
                              , fl!: [nat], lbr?: nat, gsl!: nat, rsl?: nat
                              , cld?: %NBUF#void) =
|[ chan rl: boxdata , al: nat, cd: void
:: PostSortC(id, l, rr, bfa, cr, le, fl, rl, lbr, gsl, rsl, al, cd)
|| PostSortG(l, a, rl, al, b, cld, cd)
]|

proc BufferLane(val id1, id2: nat, chan l!: logentry, a?: box, cld!: void
                                    , b!: [box], le!: nat) =
|[ var myName: string = name4me("BufL", 10*id1+id2), x: box, xs: [box] = []
:: *( a?x
    ; ( len(xs) = 0 -> l!(myName, "util", 1, 0.0)
      | len(xs) > 0 -> l!(myName, "util", 1, 1.0)
      )
    ; xs := xs ++ [x]; cld!
    | b!xs; xs := []; le!id2; l!(myName, "util", 1, 1.0)
    )
]|

func getPalId(val xs: [box]) -> nat = |[ ret hd(xs).2 ]|

proc Pal(val id: nat, chan l!: logentry, a?: %NBUF#[box], b!: pallet, fl?: [nat]
                        , gpt!: (nat, nat), pt?: real) =
|[ var myName: string = name4me("Pal", id), i, j, pid: nat, lr, ls: [nat]
     , lss: [[nat]] = [], xs, ys: [box], t: real, last_sku: nat = 0
:: *( fl?lr; lss := lss ++ [lr] )
|| *( len(lss)>0 -> l!(myName, "util", 1, 0.0)
    ; ls := hd(lss); lss := tl(lss); ys := []; i:= 0
    ; (i < len(ls)) *> ( j:= hd(ls); a.j?xs; ys := ys ++ xs; ls := tl(ls) )
    ; ( last_sku  = hd(ys).1 -> skip
      | last_sku /= hd(ys).1 -> delay %PALLETIZERCHANGETIME_SKU
```

```
                              ; last_sku := hd(ys).1
      )
    ; pid := getPalId(ys); gpt!(hd(ys).1, len(ys)); pt?t
    ; delay t + %PALLETCHANGETIME; l!("Exit", "pT", 2, t)
    ; b!(pid, ys); l!(myName, "util", 1, 1.0)
    )
]|

proc SPalA(val id: nat, chan l!: logentry, a?: box, b!: pallet, rr?: nat
                           , bfa!: nat, cr?:(spid, nat), gsl!: nat, rsl?: nat
                           , lbr?: nat, gpt!: (nat, nat), pt?: real) =
|[ chan c: %NBUF#box, d: %NBUF#[box], le: nat, fl: [nat], cld: %NBUF#void
:: PostSort(id, l, a, c, rr, bfa, cr, le, fl, lbr, gsl, rsl, cld)
|| (||, j<-0..%NBUF-1, BufferLane(id, j, l, c.j, cld.j, d.j, le) )
|| Pal(id, l, d, b, fl, gpt, pt)
]|

proc PalA(chan l!: logentry, a?: %NPAL#box, b!: pallet, rr?: %NPAL#nat
             , bfa!: %NPAL#nat, cr?: %NPAL#(spid, nat), gsl!: nat, rsl?: nat
             , lbr?: %NPAL#nat, gpt!: (nat, nat), pt?: real) =
|[ (||, j<-0..%NPAL-1, SPalA(j, l, a.j, b, rr.j, bfa.j, cr.j, gsl, rsl, lbr.j, gpt, pt))
]|

proc PalExit(chan l!: logentry, a?: pallet) =
|[ var myName: string = "Exit", p: pallet, px: [box], x: box, t: real
:: *( a?p ; px := p.1
  ; l!(myName, "pf", 2, 1.0*len(px)); len(px) > 0
    *> ( x := hd(px) ; px := tl(px)
      ; t := time - x.4 ; l!(myName, "EPTx", 0, t) ; l!(myName, "x", 2, t)
      ; t := time - x.3 ; l!(myName, "EPTt", 0, t) ; l!(myName, "EPTt", 2, t)
       )
  )
]|

proc ManExit(chan a?: box) = |[ var x: box :: *( a?x ) ]|

model M(val cdb: [container], sdb: [skudata]) =
|[ chan l: logentry, a: %NCUL#[[box]], b, m: box, ab: (nat, boxdata)
      , ba: %NCUL#bool, c: %NPAL#box, d: pallet, bfa: %NPAL#nat
      , rr: %NPAL#nat, cr: %NPAL#(spid, nat), lfs: nat, lbr: %NPAL#nat
      , gc: nat, con: [container], gpf, pf: 2#nat, gsl, sl: 3#nat
      , gap: nat, ap: bool, gpt: (nat, nat), pt: real
:: ConG(a, gc, con, gpf.0, pf.0)
|| UnlA(l, a, b, m, ab, ba, lfs, gsl.0, sl.0, gap, ap)
|| Sort(l, b, c, ab, rr, bfa, cr, ba, lfs, lbr, gsl.1, sl.1, gpf.1, pf.1)
|| PalA(l, c, d, rr, bfa, cr, gsl.2, sl.2, lbr, gpt, pt)
|| DB(cdb, sdb, gc, con, gpf, pf, gsl, sl, gap, ap, gpt, pt)
|| ManExit(m) || PalExit(l, d) || logbook(l)
]|


func selindex(val element: string, maparray: [string]) -> nat =
|[ var i: nat = 0, s: string
:: len(maparray) > 0
       *> ( s := hd(maparray); maparray := tl(maparray);
            ( s  = element -> ret i
            | s /= element -> i := i + 1
            )
          )
 ; ret i
]|

func selvalue(val index: nat, maparray: [string]) -> string =
|[ var i: nat = 0
:: i<index *> ( maparray := tl(maparray); i := i + 1 )
```

```
 ; ret hd(maparray)
]|

func calc_var(val N: nat, cur_var, cur_mean, new_value: real) -> real =
|[ ( N > 0 -> ret (N - 1)/N * cur_var + (new_value - cur_mean)^2 / (N + 1)
   | N = 0 -> ret 0.0 )
]|

func calc_mean(val N: nat, cur_mean, new_value: real) -> real =
|[ ret ( N / (N + 1) * cur_mean + new_value / (N + 1) ) ]|

proc logbook(chan l?: logentry) =
|[ var i, vi: nat, meas: 1000*(real, real, nat, nat), process, variable: string
     , logtype: nat, value: real, inter_sample_time: real = 300.0, lt
     , t: real, tmap: [string] = []
:: *( l?(process, variable, logtype, value)
    ; vi := selindex(process ++ variable, tmap)
    ; ( vi = len(tmap) -> tmap := tmap ++ [process ++ variable]
                        ; meas.vi.0 := 0.0
                        ; meas.vi.1 := 0.0
                        ; meas.vi.2 := 0
                        ; meas.vi.3 := logtype
      | vi < len(tmap) -> skip
      )
    ; ( logtype = 0 -> !!time, " ", process, " ", variable, " ", value, "\n"
      | logtype = 1 -> t:= time
                     ; meas.vi.1 := meas.vi.1 + (t - meas.vi.0)*value
                     ; meas.vi.0 := t
                     ; ( value = 0.0 -> meas.vi.2 := 1
                       | value = 1.0 -> meas.vi.2 := 0
                       )
      | logtype = 2 -> meas.vi.1 := calc_var(meas.vi.2, meas.vi.1, meas.vi.0, value)
                     ; meas.vi.0 := calc_mean(meas.vi.2, meas.vi.0, value)
                     ; meas.vi.2 := meas.vi.2 + 1
      )
    )
|| *( delay inter_sample_time
    ; i := 0 ; lt := time
    ; (i < len(tmap))
        *>( ( ( meas.i.3 = 2 and meas.i.2 > 1 ) ->
!!lt, " ", selvalue(i, tmap), " ", meas.i.0, " ", meas.i.1, " ", meas.i.2, "\n"
            |   meas.i.3 = 1                    ->
!!lt, " ", selvalue(i, tmap), " ", (meas.i.1 + (lt - meas.i.0)*meas.i.2)/lt, "\n"
            |   meas.i.3 = 0                    -> skip
            | ( meas.i.3 = 2 and meas.i.2 < 2 ) -> skip
            )
          ; i := i + 1
          )
    )
]|
```

# Appendix C
# Constants Pre-processor

This appendix lists how the use of constants in a $\chi$ model can be achieved by a simple script, which is included in the shell initialization file, e.g., `.bashrc`. The script assumes the existence of some file `filename.chic` with the extension `.chic`, which is parsed to `filename.chi` that contains the $\chi$ model using AWK [AWK03].

The script picks up lines of the form '`DEFINE %VARIABLE value`' in the $\chi$ code and transforms occurrences of the form '`%VARIABLE([+-][1-9]+)?`' elsewhere in the code into the predefined value modified by the optional argument.[1]
The script takes the following form:

```
function chicp {
  filename=`basename $1 .chic`
  if [ -f $filename.chic ]; then
     if [ -f $filename.chi ]; then
        rm $filename.chi
     fi
     gawk '! /%[A-Z]+/ && ! /^DEFINE/ { print $0 }
           /^DEFINE/ { gs[$2]=$3 ; gsub($2,"") }
           /%[A-Z]+/ { for (i in gs) {
                         for ( j=1 ; j<=9 ; j++ ) {
                           mi = i"-"j ; gsub(mi,gs[i]-j)
                           mi = i"+"j ; gsub(mi,gs[i]+j)
                         }
                         mi = i ; gsub(mi,gs[i])
                       }
                       print $0
                     }' ${filename}.chic > ${filename}.chi
  fi
  ~/bin/chic ${filename}.chi
}
```

[1] The last part, `([+-][1-9]+)?`, is a regular expression [Fri06] meaning that the variable can optionally be followed by a + or - sign and a value of 1 or more digits.

# Appendix D
# Data Sets

| # | SKU nr. | # boxes | # | SKU nr. | # boxes | # | SKU nr. | # boxes | # | SKU nr. | # boxes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1020 | 2 | 2 | 3008 | 3 | 3 | 600 | 4 | 4 | 996 |
| 5 | 5 | 688 | 6 | 6 | 240 | 6 | 7 | 1215 | 7 | 8 | 720 |
| 8 | 9 | 72 | 8 | 10 | 610 | 9 | 11 | 3591 | 10 | 12 | 976 |
| 11 | 13 | 360 | 11 | 14 | 672 | 12 | 15 | 56 | 12 | 16 | 756 |
| 13 | 17 | 1200 | 14 | 18 | 540 | 14 | 19 | 450 | 14 | 20 | 72 |
| 14 | 21 | 80 | 14 | 22 | 288 | 14 | 23 | 108 | 15 | 17 | 1200 |
| 16 | 24 | 336 | 16 | 25 | 192 | 16 | 13 | 500 | 16 | 26 | 594 |
| 17 | 27 | 420 | 18 | 28 | 1287 | 19 | 29 | 1062 | 20 | 30 | 1344 |
| 21 | 31 | 1098 | 22 | 32 | 560 | 22 | 33 | 352 | 22 | 34 | 240 |
| 23 | 35 | 800 | 24 | 36 | 180 | 24 | 37 | 2080 | 25 | 38 | 405 |
| 25 | 39 | 432 | 25 | 40 | 24 | 26 | 41 | 702 | 27 | 42 | 752 |
| 28 | 43 | 105 | 28 | 44 | 1360 | 29 | 45 | 352 | 29 | 46 | 300 |
| 30 | 47 | 504 | 31 | 48 | 588 | 31 | 49 | 448 | 32 | 50 | 1645 |
| 33 | 51 | 500 | 33 | 52 | 345 | 34 | 53 | 2115 | 34 | 54 | 24 |
| 35 | 55 | 40 | 35 | 56 | 24 | 35 | 57 | 24 | 35 | 58 | 30 |
| 35 | 59 | 24 | 35 | 60 | 48 | 35 | 61 | 24 | 35 | 62 | 24 |
| 35 | 63 | 60 | 36 | 64 | 630 | 37 | 65 | 1856 | 38 | 66 | 612 |
| 39 | 67 | 72 | 39 | 68 | 2958 | 40 | 64 | 630 | 41 | 69 | 240 |
| 41 | 70 | 368 | 41 | 71 | 180 | 42 | 72 | 288 | 43 | 73 | 708 |
| 44 | 74 | 2046 | 45 | 75 | 670 | 46 | 76 | 360 | 46 | 77 | 744 |
| 46 | 78 | 25 | 46 | 79 | 360 | 47 | 80 | 1044 | 48 | 81 | 1840 |
| 49 | 12 | 976 | 50 | 82 | 672 | 51 | 83 | 297 | 51 | 84 | 495 |
| 51 | 85 | 336 | 52 | 24 | 288 | 52 | 86 | 60 | 52 | 87 | 225 |
| 52 | 88 | 120 | 52 | 89 | 84 | 53 | 65 | 1856 | 54 | 90 | 540 |
| 55 | 91 | 3024 | 56 | 92 | 600 | 57 | 93 | 1136 | 58 | 94 | 976 |
| 59 | 95 | 198 | 59 | 96 | 380 | 59 | 97 | 968 | 60 | 98 | 378 |
| 60 | 99 | 48 | 61 | 72 | 288 | 62 | 100 | 704 | 63 | 101 | 1650 |
| 64 | 102 | 660 | 65 | 103 | 1860 | 65 | 104 | 96 | 65 | 29 | 450 |
| 66 | 105 | 705 | 67 | 106 | 276 | 68 | 107 | 1292 | 68 | 108 | 2508 |
| 68 | 109 | 120 | 68 | 110 | 150 | 69 | 111 | 976 | 70 | 112 | 1044 |
| 71 | 113 | 80 | 71 | 46 | 240 | 72 | 114 | 780 | 73 | 94 | 976 |
| 74 | 115 | 324 | 75 | 36 | 180 | 75 | 116 | 660 | 75 | 117 | 1224 |
| 76 | 118 | 870 | | | | | | | | | |

Table D.1: The contents of the 76 containers in terms of SKU types and number of boxes.

| SKU | #/layer | $\ell$ (mm) | AP | layers | $t$ (s) | SKU | #/layer | $\ell$ (mm) | AP | layers | $t$ (s) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 20 | 919 | true | 4 | 10.5 | 2 | 64 | 449 | true | 8 | 20.3 |
| 3 | 12 | 543 | true | 4 | 10.5 | 4 | 12 | 467 | true | 4 | 10.5 |
| 5 | 16 | 541 | true | 4 | 10.5 | 6 | 24 | 391 | false | 0 | 0.0 |
| 7 | 45 | 419 | true | 9 | 22.5 | 8 | 15 | 815 | true | 3 | 8.0 |
| 9 | 12 | 751 | true | 2 | 5.2 | 10 | 10 | 617 | false | 0 | 0.0 |
| 11 | 63 | 406 | true | 9 | 22.5 | 12 | 16 | 530 | true | 4 | 10.5 |
| 13 | 20 | 495 | true | 4 | 10.5 | 14 | 16 | 492 | true | 4 | 10.5 |
| 15 | 28 | 436 | true | 7 | 17.8 | 16 | 12 | 441 | true | 4 | 10.5 |
| 17 | 15 | 924 | true | 3 | 8.0 | 18 | 36 | 287 | false | 0 | 0.0 |
| 19 | 30 | 523 | true | 6 | 15.2 | 20 | 72 | 530 | false | 0 | 0.0 |
| 21 | 80 | 322 | false | 0 | 0.0 | 22 | 48 | 398 | false | 0 | 0.0 |
| 23 | 36 | 365 | false | 0 | 0.0 | 24 | 48 | 419 | true | 8 | 20.3 |
| 25 | 48 | 421 | true | 8 | 20.3 | 26 | 99 | 370 | false | 0 | 0.0 |
| 27 | 15 | 624 | true | 3 | 8.0 | 28 | 33 | 701 | true | 3 | 8.0 |
| 29 | 18 | 782 | true | 3 | 8.0 | 30 | 32 | 558 | true | 4 | 10.5 |
| 31 | 18 | 706 | true | 3 | 8.0 | 32 | 16 | 439 | true | 4 | 10.5 |
| 33 | 16 | 480 | true | 4 | 10.5 | 34 | 24 | 411 | true | 6 | 15.2 |
| 35 | 32 | 419 | true | 8 | 20.3 | 36 | 30 | 695 | true | 3 | 8.0 |
| 37 | 40 | 558 | false | 0 | 0.0 | 38 | 15 | 650 | false | 0 | 0.0 |
| 39 | 16 | 530 | true | 4 | 10.5 | 40 | 12 | 670 | false | 0 | 0.0 |
| 41 | 18 | 579 | true | 6 | 15.2 | 42 | 16 | 571 | true | 4 | 10.5 |
| 43 | 15 | 632 | true | 3 | 8.0 | 44 | 40 | 574 | true | 5 | 13.4 |
| 45 | 32 | 500 | true | 8 | 20.3 | 46 | 12 | 769 | true | 4 | 10.5 |
| 47 | 18 | 1031 | true | 2 | 5.2 | 48 | 21 | 690 | true | 3 | 8.0 |
| 49 | 14 | 579 | true | 2 | 5.2 | 50 | 35 | 459 | true | 7 | 17.8 |
| 51 | 20 | 533 | true | 5 | 13.4 | 52 | 15 | 541 | true | 5 | 13.4 |
| 53 | 45 | 299 | false | 0 | 0.0 | 54 | 12 | 645 | true | 3 | 8.0 |
| 55 | 20 | 530 | true | 4 | 10.5 | 56 | 6 | 520 | true | 2 | 5.2 |
| 57 | 12 | 530 | true | 4 | 10.5 | 58 | 6 | 541 | true | 2 | 5.2 |
| 59 | 12 | 541 | true | 4 | 10.5 | 60 | 8 | 629 | true | 2 | 5.2 |
| 61 | 12 | 541 | true | 4 | 10.5 | 62 | 12 | 541 | true | 4 | 10.5 |
| 63 | 20 | 541 | true | 4 | 10.5 | 64 | 10 | 797 | true | 2 | 5.2 |
| 65 | 32 | 426 | true | 8 | 20.3 | 66 | 12 | 690 | true | 2 | 5.2 |
| 67 | 8 | 685 | true | 2 | 5.2 | 68 | 102 | 330 | false | 0 | 0.0 |
| 69 | 24 | 480 | true | 6 | 15.2 | 70 | 16 | 480 | true | 4 | 10.5 |
| 71 | 20 | 558 | true | 5 | 13.4 | 72 | 12 | 660 | true | 3 | 8.0 |
| 73 | 12 | 701 | true | 2 | 5.2 | 74 | 33 | 810 | true | 3 | 8.0 |
| 75 | 10 | 584 | true | 2 | 5.2 | 76 | 60 | 289 | false | 0 | 0.0 |
| 77 | 24 | 429 | true | 4 | 10.5 | 78 | 25 | 599 | true | 5 | 13.4 |
| 79 | 36 | 510 | true | 6 | 15.2 | 80 | 18 | 485 | true | 6 | 15.2 |
| 81 | 40 | 408 | true | 8 | 20.3 | 82 | 12 | 541 | true | 4 | 10.5 |
| 83 | 99 | 360 | false | 0 | 0.0 | 84 | 99 | 299 | false | 0 | 0.0 |
| 85 | 24 | 408 | true | 6 | 15.2 | 86 | 30 | 391 | false | 0 | 0.0 |
| 87 | 9 | 629 | true | 3 | 8.0 | 88 | 12 | 640 | true | 3 | 8.0 |
| 89 | 12 | 609 | true | 3 | 8.0 | 90 | 12 | 723 | true | 3 | 8.0 |
| 91 | 56 | 449 | true | 7 | 17.8 | 92 | 10 | 711 | true | 2 | 5.2 |
| 93 | 16 | 845 | true | 2 | 5.2 | 94 | 16 | 530 | true | 4 | 10.5 |
| 95 | 99 | 383 | false | 0 | 0.0 | 96 | 20 | 546 | true | 5 | 13.4 |
| 97 | 44 | 325 | false | 0 | 0.0 | 98 | 18 | 424 | true | 6 | 15.2 |
| 99 | 16 | 510 | true | 4 | 10.5 | 100 | 16 | 551 | true | 4 | 10.5 |
| 101 | 33 | 891 | true | 3 | 8.0 | 102 | 12 | 533 | true | 4 | 10.5 |
| 103 | 60 | 929 | false | 0 | 0.0 | 104 | 12 | 820 | true | 2 | 5.2 |
| 105 | 15 | 419 | true | 5 | 13.4 | 106 | 12 | 568 | true | 4 | 10.5 |
| 107 | 68 | 510 | false | 0 | 0.0 | 108 | 66 | 660 | false | 0 | 0.0 |
| 109 | 30 | 469 | true | 6 | 15.2 | 110 | 50 | 360 | false | 0 | 0.0 |
| 111 | 16 | 878 | true | 2 | 5.2 | 112 | 18 | 485 | true | 6 | 15.2 |
| 113 | 16 | 680 | true | 4 | 10.5 | 114 | 12 | 680 | true | 3 | 8.0 |
| 115 | 9 | 599 | true | 3 | 8.0 | 116 | 30 | 579 | false | 0 | 0.0 |
| 117 | 72 | 421 | true | 6 | 15.2 | 118 | 15 | 629 | true | 3 | 8.0 |

Table D.2: SKU data. Listed are respectively the number of boxes per layer, the length of each box, whether or not the boxes are automatically palletizable, the number of layers per pallet, and the time it takes to assemble one layer of a pallet.