TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

ETSEIB

UPC

Simulation of cycle and throughput of Shuttle Storage Systems

Master Thesis - Industrial Engineer

Author: Antoni Castells i Sanabra

Tutor: Univ. Prof. Dipl. - Ing. Dr. techn. Georg Kartnig

Date: September.2013

ABSTRACT

This project intends to extract and analyse the results of functioning of Shuttle Storaging Systems. Shuttle Storage Systems are rack-storing facilities defined by the goods transporting devices, which are a combination of a lift and one shuttle at each floor. The lift is responsible for transporting the goods between the floor and the corresponding floor, and the shuttle then takes the goods to the corresponding position on the floor for storage. The combination of these transporters, which are independent thanks to mid-process buffers, is what will define the total throughput of the system. Their independence, however, makes it too complex to extract easy analytical solution for the system. Therefore, the system will be modelled and simulated with a discrete process simulator: ARENA Simulation Software. This software will grant the ability of monitoring closely the functioning of the system and will report simulation results that will help understand and explain the functioning of the system.

INDEX

1. INTRODUCION 11

2. STATE OF THE ART OF STORAGE SYSTEMS 13

2.1. EVOLUTION OF STORAGE SYSTEMS 13

2.2. DESCRIPTION OF THE SYSTEM OF STUDY 16

2.2.1. PHYSICAL DISTRIBUTION 16

2.2.2. TRANSPORTERS 19

2.2.3. CYCLES OF GOODS 20

2.2.4. LOGICAL FEATURES 21

2.2.4.1. ARRIVALS AND DEPARTURES OF CONTAINERS 21

2.2.4.2. LIFT DOUBLE CYCLE 23

2.2.4.3. DEADLOCK SITUATION 23

2.2.4.4. SHUTTLE LOGIC 24

2.2.4.5. OUTPUT SEQUENCE 25

3. ARENA SIMULATION SOFTWARE 26

4. MODEL CONSTRUCTION 29

4.1. INITIAL MODEL 30

4.1.1. PRIOR ARENA SIMULATION SOFTWARE CONCEPTS 30

4.1.2. ADAPTATION OF FEATURES 32

4.1.2.1. WARM UP PERIOD 33

4.1.2.2. STORAGE POSITION 33

4.1.2.3. TRANSPORTERS INDEPENDENT MOVEMENT 34

4.1.2.4. OUTPUT SEQUENCE GUARANTY 35

4.1.2.5. ACCELERATION ADAPTION 35

4.1.3. INITIAL MODEL CONSTRUCTION 40

4.1.3.1. BOX ARRIVALS STATION 40

4.1.3.2. Floor ## Transfer Station (I) 45

4.1.3.3. Position #### Station 49

4.1.3.4. Floor ## Transfer Station (II) 52

4.1.3.5. Box Departure Station 56

4.1.3.6. Order Process Station 56

4.1.3.7. Lift Operator Station 59

4.1.3.8. Shuttle Operator Station 60

4.1.3.9. Animation 61

4.2. Intermediate Model 61

4.2.1. Prior ARENA Simulation Software Concepts 62

4.2.2. Intermediate Model Construction 63

4.2.2.1. Position Panel 63

4.2.2.2. Floor Panel 63

4.2.2.3. System Panel 64

4.3. Final Model 64

4.3.1. Final Model Construction 65

4.3.1.1. Position Panel 65

4.3.1.2. Floor Panel 65

4.3.1.3. System Panel 66

5. SIMULATION ANALYSIS AND RESULTS 68

5.1. Simulation Parameters 68

5.2. Analysis of the results 69

5.2.1. Lift cycle times 69

5.2.1.1. Results at constant height 71

5.2.1.2. Results at constant length 74

5.2.2. Throughput 77

5.3. Comparison with Matlab 78

6. OPTIMITZATION AND IMPROVEMENTS 82

6.1. INPUT – OUTPUT PRIORITIZATION 82

6.2. SHUTTLE LOGIC OPTIMIZATION 83

6.3. OPTIMIZED RESULTS 88

7. CONCLUSIONS AND OUTLOOK 92

8. ACKNOWLEDGEMENTS 95

9. REFERENCES 96

LIST OF IMAGES

Image 1: Traditional warehouse ……. 13

Image 2: Rack storing …… 14

Image 3: Forklift transporter ……. 14

Image 4: Automated racking systems …… 15

Image 5: Run Setup Dialog Window ……... 68

LIST OF FIGURES

Figure 1: Standard Conainer 17

Figure 2: Rack dimensions 17

Figure 3: Rack floor top view 18

Figure 4: Rack storage system floor top view 19

Figure 5: Input Cycle Flowchart 20

Figure 6: Output Cycle Flowchart 20

Figure 7: Arrivals Logic Flowchart 21

Figure 8: Departure Logic Flowchart 22

Figure 9: Double Cycle Flowchart 23

Figure 10: ARENA Simulation Software hierarchy structure 27

Figure 11: Velocity – Time plot..... 36

Figure 12: Box Arrivals Station Flowchart (I) 40

Figure 13: Box Arrivals Station Flowchart (II) 42

Figure 14: Box Arrivals Station Flowchart (III) 44

Figure 15: Floor ## Transfer Station (I) Flowchart (I) 46

Figure 16: Floor ## Transfer Station (I) Flowchart (II) 47

Figure 17: Floor ## Transfer Station (I) Flowchart (III) 48

Figure 18: Position #### Station Flowchart (I) 50

Figure 19: Position #### Station Flowchart (II) 50

Figure 20: Position #### Station Flowchart (III) 52

Figure 21: Floor ## Transfer Station (II) Flowchart (I) 54

Figure 22: Floor ## Transfer Station (II) Flowchart (II) 55

Figure 23: Floor ## Transfer Station (II) Flowchart (III) 55

Figure 24: Box Departure Station Flowchart (I) 56

Figure 25: Order Process Station Flowchart (I) 57

Figure 26: Order Process Station Flowchart (II) …… 58

Figure 27: Lift Operator Station Flowchart (I) …… 59

Figure 28: Lift Operator Station Flowchart (II) …… 59

Figure 29: Shuttle Operator Station Flowchart (I) …… 60

Figure 30: Model Animation …… 61

Figure 31: Template Logic Window of modules from Floor Panel …… 65

Figure 32: Template Logic Window of modules from Systems Panel …… 66

Figure 33: User Interface Window …… 67

Figure 34: Double cycle of the lift …… 70

Figure 35: Double-Cycle at a constant height of 20 floors …… 71

Figure 36: Double-Cycle times at constant height …… 73

Figure 37: Between cycle waiting time at constant height …… 74

Figure 38: Double-Cycle times at a constant length of 40 positions …… 75

Figure 39: Double-Cycle times at constant length …… 76

Figure 40: Between Cycle waiting time at constant length …… 77

Figure 41: Throughput at constant height …… 77

Figure 42: Throughput at constant length …… 78

Figure 43: MATLAB simulation results. Double-Cycle times at constant length …… 80

Figure 44: ARENA simulation results. Double-Cycle times at constant length …… 80

Figure 45: Double-Cycle times evolution depending on SOP at 10 positions …… 85

Figure 46: Double-Cycle times evolution depending on SOP at 20 positions …… 85

Figure 47: Double-Cycle times evolution depending on SOP at 30 positions …… 85

Figure 48: Double-Cycle times evolution depending on SOP at 40 positions …… 86

Figure 49: Double-Cycle times evolution depending on SOP at 50 positions …… 86

Figure 50: Between Cycle waiting time evolution depending on SOP at 50 positions …… 87

Figure 51: Waiting time evolution between Double-Cycles depending on SOP at 50 positions.. 87

Figure 52: Double-Cycle times at constant height …… 88

Figure 53: Between Cycle waiting times at constant height …… 89

Figure 54: Waiting times between Double-Cycles at constant height …… 89

Figure 55: Double-Cycle times at constant length …… 90

Figure 56: Between Cycle waiting times at constant length …… 90

Figure 57: Waiting time between Double-Cycles at constant length …… 91

LIST OF TABLES

Table 1: Standard container sizes ……. 17

Table 2: Distances between positions and floors …… 18

Table 3: Lift parameters …….. 19

Table 4: Shuttle parameters ……. 20

Table 5: Simulation Parameters ……. 68

Table 6: Lift possible travelling times ……. 71

1. Introduction

Industrial storing is nowadays known as a very important feature of manufacturing cycles. Its development has not only supposed an improvement of time and space efficiency on storehouses, but it has also played a key role in adapting the manufacturing cycles to the current supply-and-demand mechanisms.

The introduction of technology to the field of industrial storing has supposed a remarkable quality increase. Technologies have been implemented on various aspects of storing, such as transporting devices, and there is still a long way for development. Although industrial storehouses offer a wide range of possibilities, there are some features that are the base of most industrial storehouses.

One of the most worldwide spread industrial storing configuration, due to its space effective configuration, is the shelve storehouse. Different rows of shelves offer multiple storing places, divided in floors and columns. However, the capacity scope of a shelve storehouse depends greatly on the good's transporting device. Although the most common transporter is the worldwide known forklift truck, it has a limited height range and a necessarily human-dependant operating. Therefore, the development of adapted shelve-vehicles supposed a boost on the use of shelve storehouses, even though the technology used is much more advanced representing also a much higher budget investment.

Shelve-vehicles can be described in few words as fully automated transporting devices with two translational degrees of freedom, meaning that they can move along the shelve as well as up and down, which gathers under only one transporter all the coordinates of a shelve. The increment of storing scope in height, combined with high transporting times and computerized system update and control, has not only placed the shelve storehouse with automated shelve-vehicles in the front-line of industrial storing configurations, but also grants a wide range of future development possibilities.

The Shuttle Storage System is considered one development alternative derived from shelve-vehicles. The Shuttle Storage System replaces the traditional shelve-vehicle for a combination transporters: on one hand, a lift – granting the up and down translational degree of freedom – and, on the other hand, one shuttle per each floor – granting the translational degree of freedom along the shelve –. Although goods are brought to the floors by the lift and then stored in their corresponding positions by the shuttles, the movements of the lift and the shuttle are totally independent thanks to internal buffers at each floor.

Because traditional shelve-vehicles have been operating for some decades, they have obviously been analysed and studied to extract information regarding its operation and cycle times. However, Shuttle Storage Systems, although they have already been developed and manufactured for some years, do not possess much background information regarding its operation and cycle times. This fact is not only because of its recent implementation, but also because the independence between transporters complicates the analysis of the system.

The following project has the goal to study and analyse the operation, throughput and cycle times of Shuttle Storage Systems. To achieve these goals, the system will be graphically modelled and simulated using the ARENA Simulation Software[1]. The system will be modelled in different sizes and different configurations to extract as many relevant information and coherent conclusions.

The following project has also been developed with the collaboration of the company KNAPP[2] – which have developed and manufacture Shuttle Storage System, among others – and in parallel with another project related to the study and analysis of Shuttle Storage Systems, developed by Herr Lukas Fleischer, focusing its study on numerical simulations with MATLAB[3]. The global purpose of both projects, with the collaboration of the company KNAPP, is to obtain, study and unite reliable conclusions regarding Shuttle Storage Systems from three independent branches: real results from the company, numerical simulation and graphical simulation.

This project is developed as the Master Thesis of the Industrial Engineering studies of the Escola Tècnica Superior d'Enginyeria Industrial de Barcelona (ETSEIB) at the Universitat Politècnica de Catalunya (UPC) and executed at the Technische Universität Wien (TU Wien) during the second semester of the year 2012-2013.

[1] More information at http://www.arenasimulation.com/
[2] More information at http://www.knapp.com/
[3] More information at http://www.mathworks.de/products/matlab/

2. State of the art of storage systems

2.1. Evolution of storage systems

The storage of goods has historically played a key role on the economic development and industrial growth of cities, regions or countries. The activities associated to the storage of goods concern practically to all the steps of any good's supply chain. The storage of goods is used by manufacturers, distributors, transport companies, wholesalers… and stored goods go from raw materials to finished products.

Thanks to the Industrial Revolution, that took place from the mid 1700s to the start of 1800s, the concept of storage of goods suffered a considerable modernization, where the traditionally called warehouses erupted and became part of the usual urban landscape, specially next to railway stations or harbours, in order to highly increase the quantity imported and exported goods, as well as boosting international trade. As a matter of fact, during that time, bigger fortunes were made by merchants who invested in warehouses than the manufacturers themselves.

Image 1: Traditional warehouse

Commerce required the major need for storage facilities. The initial basic goals in storage were basically protection from weather, security from theft and isolation from destructive animals. However, storage facilities also served as a reservoir to accommodate seasonal and fluctuating demand.

From that point on, the storage of goods and its corresponding facilities have been on constant growth and upgrade, in parallel with the non-stopping development and creation of up-to-day technology.

When talking of storage of goods, various components have been defined and modernized over time. The distribution of goods in the facilities where they are stored, on one hand, is one of the keys of storing and is closely related to the storage capacity of the facility.

The different distribution alternatives have evolved from basic stacked goods to complex automated high-racks, though the most common distribution nowadays is composed of simple racks. The facility layout is a very similar concept and it is important in achieving greater efficiencies. Minimizing travel times between different locations within the warehouse can greatly improve productivity.

Image 2: Rack storing

On the other hand, the vehicles or storing transporters that carry the goods in and out of the facility also condition very greatly the global characteristics of the storehouse. These vehicles embrace from human handling to specific automated devices, going through the worldwide used forklift-truck. Other components of storage facilities to be taken under account are the material handling devices, such as containers or the commonly used pallet racks, and the storing and unstoring strategies of goods.

Image 3: Forklift transporter

The application of technology to all those components allows the improvement of storehouses on many aspects, such as the capacity of the storehouse, the velocity of storing or unstoring, the number and of personnel required or the human error factor. The need of technology is, however, specific for each storage purpose.

Although very advanced technology has been applied for storing purposes, the most common features in storing facilities are pallet racks, simple racks and forklift trucks. Storehouses are developed each time more automated. The use of conveyors, automatic transporters and cranes is, nowadays, very common in quite modern storage facilities. It is every time more usual for companies to initially invest in automated storage facilities that will require less human resources in global, granting the assurance of reducing the human error factor and in most cases incrementing efficiency.

However, the differential step regarding storage of goods was given with the implantation of computer-based intelligence. Programmable logic controllers and logistics automation software

allow coordinating the automated storing and unstoring of goods and the maintaining an updated inventory of stored items.

This computer-based intelligence application goes in parallel with modern automated transporting devices. Those automated transporting devices store and unstore goods automatically under the computer's control. Two main variants have been developed: on one hand, an operator is carried by the device and is responsible of the action of storing or unstoring the goods, or on the other hand, all actions are carried out without the need of personnel at all, only an operator or a supervisor (but requires higher technology and investment).

The implementation of automated transporting devices have provided to the storing of goods systems a substantial gain. Whereas with traditional human-operated transporting vehicles had limitations regarding the maximum height of storage, an important advantage has been achieved in that aspect with the use of automated transporting devices. Racks with heights of 20 or 30 meters are commonly and easily implemented, which represents an increase of the capacity of the system due to the high vertical efficiency.

Computer-controlled storing system's facilities with automated transporting devices are usually distributed in racks. Each rack is divided in separate floors and separate positions for storing the goods. The automated transporting devices travel between rows of racks. The transporting devices can either operate on only one rack, but mainly on both racks at each side of the device. As previously commented, rack-distributed facilities suppose an important improvement on efficiency, thanks to the recapturing of floor space, and allow to keep an updated inventory control.

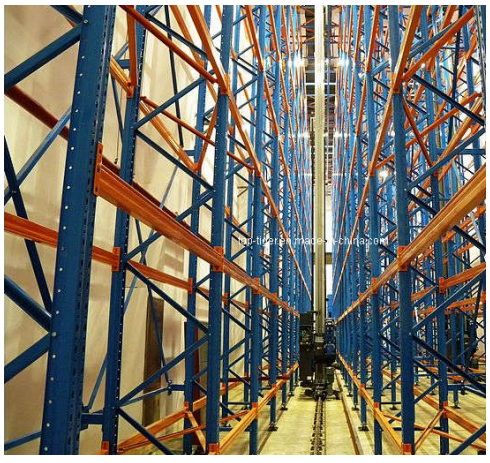

Image 4: Automated racking systems

The use of computer-controlled systems has traditionally being implemented with the use of automated rack transporters. The main feature of this transporter is that it has two translational degrees of freedom: the transporter can travel along the length of the rack as well as the height of the rack, needing only one transporter to cover all the area of the rack. Therefore, the transporter can travel from any point of the rack to any other point of the rack freely.

The arrivals and departure of goods to the racks are independent to the storing and unstoring process, and can be made for example by conveyors or manual work. Entities are deployed at a queue where they will be picked up by the automated rack transporters to be stored.

However, there are other combinations of transporters that substitute the automated rack transporter. There has recently been developed a new storing strategy that comprises a transporter with the height degree of freedom, or lift, and one transporter per each floor of the rack with the length degree of freedom.

This new configuration called Shuttle Storage System is a fairly new rack-storing alternative to the traditional automated rack transporters. **The use of two different and independent transporters per each storing or unstoring cycle leads to the initial conclusion that throughput of the system will be higher, because entities can be picked up faster because of the first transporter lower cycle time.** However, the logic combination and prioritizing of the transporters can vary, as well as the physical configuration such as pick-up point or number of lifts, making that initial conclusion a hypothesis.

2.2. Description of the system of study

In order to carry out an accurate study and extracting coherent conclusions for Shuttle Storage Systems, the system to study has been very carefully designed and delimited. This following system has been designed according to time-limitation and feasibility parameters, although different physical and logical configurations could have been implemented and could offer a wider scope of results. Rack storage is a flexible world and offers many alternatives.

Specific values and dimensions have been set according to the information provided by the company KNAPP. Specifically, the values make reference to the KNAPP Shuttle Storage System OSR Shuttle 32.

2.2.1. Physical Distribution

In first place, before specifying the main parameter values of the system, it is important to specify the size of the smallest entity of the system that, in this case, conditions all dimensions of the system.

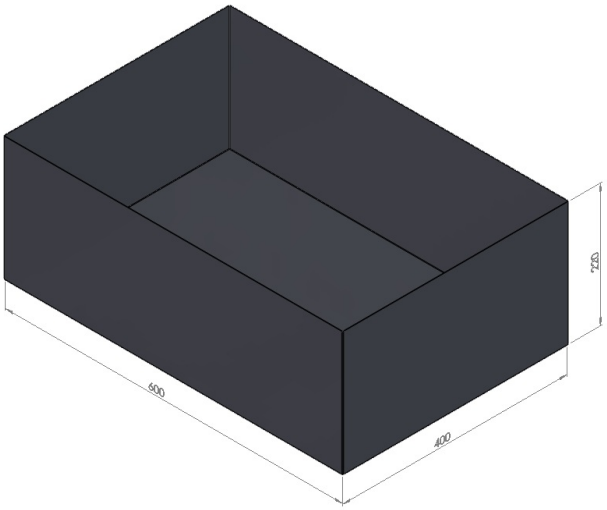

Figure 1: Standard Container

For this study, goods enter the system and are stored in the same standardized container. The size of this container is relative to the rest of the system's dimensions, because it conditions the size of the storing positions and consequently the size of the rack itself, as well as the transporters.

Standard Container Size	
Height	220 mm
Length	600 mm
Width	400 mm

Table 1: Standard container size. Information provided by KNAPP

The most important feature when talking about physical distribution of rack systems would be the size of the racks. However, due to the nature of the study of this project, the size of the rack will be left as a variable to extract important information varying both the height and length of the rack. The distances between positions and floors inside the rack are, however, constant values on the study, as shown in *Figure 2*, and are conditioned by the container dimension values in *Figure 1*.

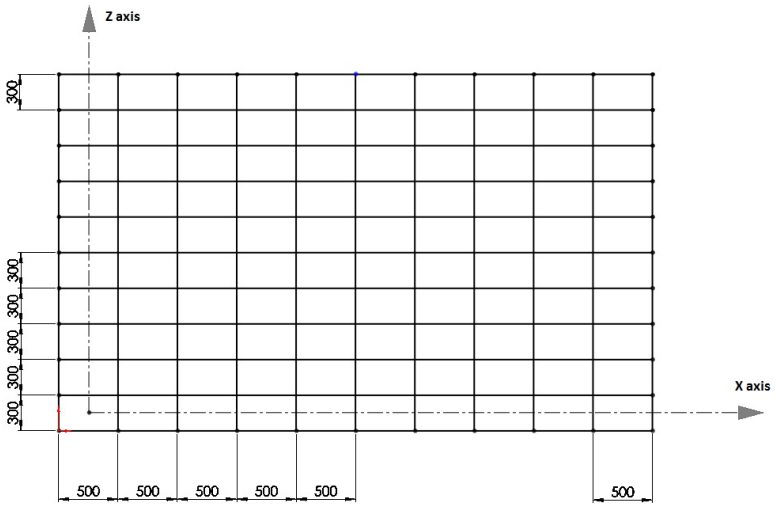

Figure 2: Rack dimensions

Distance between floors	300 mm
Distance between length positions	500 mm

Table 2: Distances between positions and floors. Information provided by KNAPP

As shown in *Figure 2*, referential axis X and Z have been defined as length of the rack and height of the rack respectively.

The first positions at each floor, or positions at X = 0, do not represent a storing position of the rack as all the other positions. Those positions are the loading and unloading buffers of the transporters and are essential to the proper behaviour of the system, as it is shown in *Figure 3*. Thanks to basically the buffers, independence between transporters is granted, which supposes the great difference of Shuttle Storage Systems from traditional rack systems.

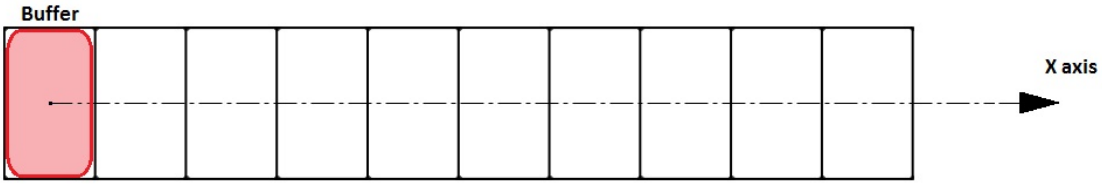

Figure 3: Rack floor top view

The system to analyse is composed of two parallel racks. Both racks, however, are considered as a pack because they are both handled by the same set of transporters, or shuttles, which travel between the racks. This high efficiency configuration is adopted for this system, but systems with only one rack per transporter also exist. As explained, the first positions of each floor on both racks are used as buffers. The fact of having two racks for a single set of shuttles also grants the system a great advantage. Buffers on one side of the shuttles will be used only for goods entering the rack storage system, and buffers on the other side of the shuttles will be only used for goods departing from the rack storage system, as shown in *Figure 4*.

Figure 4: Rack storage system floor top view

A lift transporter carries out the entrance and departure of goods from the system. This lift will pick up and deploy entities from the arrivals/departure station placed at the ground floor, or Z = 0, to the corresponding floor buffer, either input or output.

2.2.2. Transporters

As previously explained, the Shuttle Storage System is composed of two different types of transporters. Unlike the traditional automated rack transporters, which covers both length and height degrees of freedom, in Shuttle Storage Systems each degree of freedom is covered by a different transporter.

On one hand, the lift covers de height degree of freedom. When working under input operation, the lift will load containers from the arrivals/departure station at Z = 0, travel to the corresponding floor and unload the containers at the input buffer. The procedure is the same when working under output operation, travelling from the desired floor to the arrivals/departure station. The lift operates under the parameters shown at *Table 3*.

Lift parameters	
Velocity:	5 m/s
Acceleration:	7 m/s^2
Load time:	1,4 s
Unload time:	1,4 s

Table 3: Lift parameters. Information provided by KNAPP

On the other hand, the shuttles cover the length degree of freedom. As previously commented, there are as many shuttles as floors are in the system. When working under input operation, the shuttle will load containers from the input buffer at X = 0, travel to the corresponding position and unload and store the containers. The procedure is the same when working under output operation, unstoring and loading containers from their position, travelling to the output buffer at X = 0 and unloading it. The shuttles operate under the parameters shown at *Table 4*.

Shuttle parameters	
Velocity:	2 m/s
Acceleration:	2 m/s^2
Load time:	5 s
Unload time:	6,5 s

Table 4: Shuttle parameters. Information provided by KNAPP

2.2.3. Cycles of goods

From their own point of view, goods will undergo two different cycles on the system, the input and the Output Cycle.

On one hand, the Input Cycle comprises all actions between the loading into the lift and the unloading of the shuttle and storage. In first place, all actions regarding the lift take place (loading, travelling to corresponding floor and unloading). At that point, there will be a certain waiting time at the input buffer, where the container waits until the shuttle picks it up. Then all actions regarding the shuttle take place (loading, travelling to corresponding position and unloading).

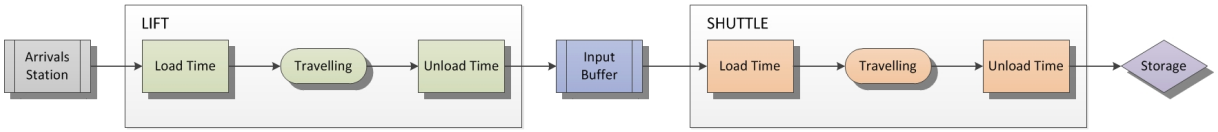

Figure 5: Input Cycle Flowchart

On the other hand, the output cycle comprises the same actions but in the opposite direction, which means from the storing position to the arrivals/departure station.

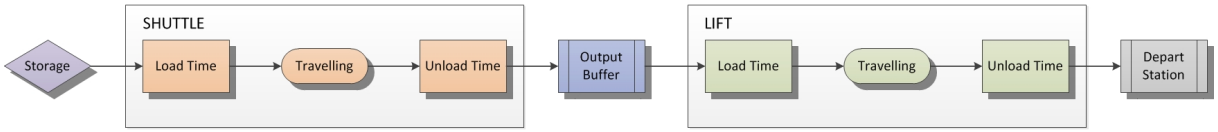

Figure 6: Output Cycle Flowchart

2.2.4. Logical features

As it can be seen in *Section 2.2.3*, the cycles of good require the use of both transporters. In a full-time operating system, where goods need to be stored and unstored continuously, the coordination of both is key for the well functioning an efficiency of the system. Also, many containers may require the need of a transporter at the same time, specially the lift, and the choice of the transporter has to respond to a certain defined logical characteristics.

Although the variety of possibilities of configuration of the system is wide, some decisions have been supposed to limit and control the functioning of the system as well as resembling as close as possible the functionality features of real Shuttle Storage Systems, in this case related to the company KNAPP.

2.2.4.1. Arrivals and departures of containers

The arrivals and departures of containers are both located at the same physical location. This location corresponds to the first floor of the rack, which is the ground floor on Z = 0. This means that the lift will pick up and deploy containers at Z = 0 equally.

Regarding the arrivals of containers, it is important that lift has always goods to pick up for storage at any time. The nature of this project is the study of cycle times and throughput, and therefore the rate of arrivals is, to some degree, irrelevant as long as the input queue is always full.

In order to ensure the existence of the input queue, it is defined that in the queue there will always be as many containers as available storing positions. As a matter of fact, each container will already have its storage position defined when entering the queue. An available storing position is the position that has no container assigned to it. When an entity enters the queue because there is an available storing position, the storing position is restricted. A storing position becomes available again when its container is requested for output and picked up by the transporter, and in that moment a new container will enter the queue with the position assigned.

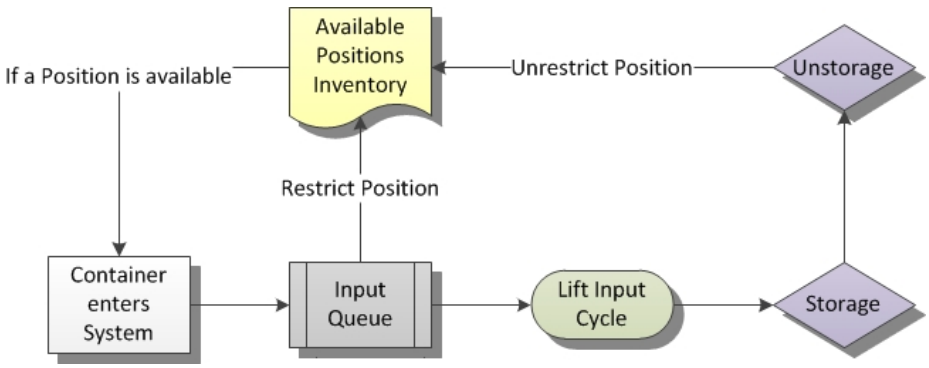

Figure 7: Arrivals Logic Flowchart

Regarding the departure of containers, the concept of orders is implemented. Orders enter the system and request a specific container. This request is turned into a signal sent to the container and the transporters to begin its output cycle. Once the signal is sent, the order is free to depart from the system.

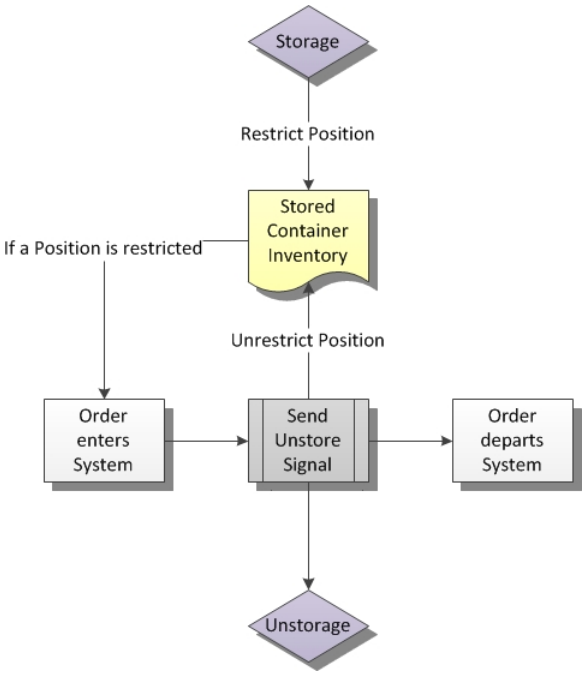

Figure 8: Departure Logic Flowchart

Just as implemented with the arrivals of containers, there must always be orders requesting containers in order to keep the system and the transporters working at full operation. In order to ensure this, any position of the rack occupied by a container will allow the entrance of its corresponding order. The fact that orders arrive as containers are stored does not guarantee that the container will be instantly removed at all, as it will be explained in *Section 2.2.4.5.*

Note that the available position inventory and the stored container inventory are not updated at the same time. While the first only refers to the availability of positions for its assignation to a container entering the queue, the second one refers to the real storage of positions. However, due to the nature of the logic of arrivals and departures of containers, their values might lead to misunderstandings. In first place, all positions will always appear as unavailable for storage at the first inventory because when containers depart from a position, it is instantly assigned to a new container entering the queue. In second place, although containers may be stored in the positions, its status on the stored inventory might not reflect it because it was instantly unrestricted by the requesting order. This fact is due to the fact of making the system work at full capacity, ensuring the continuous work of the transporters. Containers fill to the maximum the input queue and orders enter the system instantly when a container is stored.

2.2.4.2. Lift Double Cycle

The most important feature on the system to underline is the **forcing of double cycling of the lift**. This feature states that the operation of the lift will alternate consecutively Input Cycles and Output Cycles, naming double cycle the totality of both cycles.

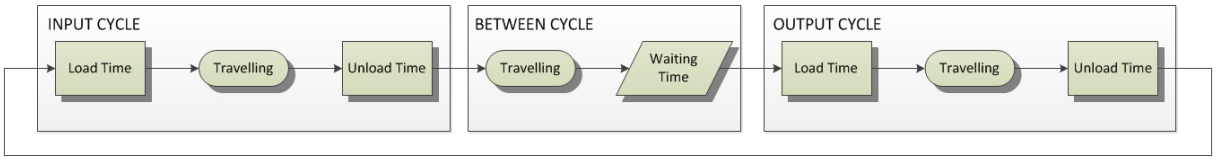

Figure 9: Double Cycle Flowchart

As explained in *Section 2.2.4.1*, there are containers always available for the lift to input. The Input Cycle of the lift encompasses the actions between the arrivals station and the input container's desired floor (loading, travelling and unloading). At that point, the lift will move to the predefined floor to wait (if necessary) and pick up the corresponding output container. This time interval has been defined as Between Cycle, or time between the Input and the Output Cycles. Finally, the Output Cycle of the lift encompasses the actions between the output's desired floor and the departure station (loading, travelling and unloading).

2.2.4.3. Deadlock situation

It is also essential to underline another very important logical limitation. As seen at *Section 2.2.4.2*, in the Input Cycle of the lift when it arrives to the destination floor, no waiting time is contemplated. Although it is be realistic to think that the input buffer could be full when the lift arrives, this option is not allowed in the system. In other words, the lift will never start an Input Cycle to a destination floor that has the input buffer full.

The fact of restricting Input Cycles to floors with a full input buffer is due to a very strong reason: **to prevent the deadlock situation**. There is one chance of deadlock situation between transporters in this system. If both buffers are full and both lift and shuttle are transporting an entity to the buffers, system will enter a deadlock. This is solved by not allowing one of the transporters to travel to the buffer if this one is full. The choice of the lift is due to strictly reasons of reality resemblance, in order to configure the system as similar to KNAPP.

However, this has a disadvantage in return. If shuttles do not pick up goods from input buffers in time and consequently when the lift has to start an Input Cycle no input buffers are available, there is going to be some delay between double cycles. However, this time delay is only expected to occur on

systems composed of few floors, and also greatly depends on the logic configuration given to the shuttles.

2.2.4.4. Shuttle logic

In comparison with the lift, which strictly obeys the double cycle configuration, the logic regarding the shuttles is more complex and offers a wider range of possibilities. As the lift, the shuttle is configured to do Input Cycles and Output Cycles, but the shuttle can also remain idle if no containers affect his floor at that time. The Input Cycle of the shuttle encompasses all actions between the input buffer and the storage of a container (loading, travelling and unloading). The Output Cycle of the shuttle encompasses all actions between the storage position and the output buffer (loading, travelling and unloading).

A shuttle will start an Input Cycle if the corresponding input buffer is full or is going to be full. This means that either a container is waiting at the input buffer or a container has been picked up by the lift at the arrivals station for input and it is going to travel to its floor. This is a timesaving strategy, because it reduces the container's waiting time on the input buffer.

A shuttle will start an Output Cycle if there is a container requesting it for output and the output buffer is empty. If the output buffer is full, the shuttle will not start an Output Cycle because it will have to wait until it is emptied by the lift. Meanwhile, if the output buffer is full and the shuttle has started an Output Cycle, the shuttle is unavailable to carry other operations out, basically an Input Cycle. This should be avoidable, especially in systems of few floors, because need to empty the input buffers for the well behaviour of the lift double cycling.

However, there might be the chance where the shuttle will be able to start both input and output shuttle. In this case, **the priority is to start the Output Cycle in first place**.

As previously said, it is important to quickly empty input buffers for allowing the well behaviour of the lift double cycling. Remember that if no input buffers are available in the system, the lift will not be allowed to start an Input Cycle and will have to wait. Although it might seem then more appropriate that the priority of the shuttles resides on emptying the input buffer, this lift block situation will only occur in systems with few floors. Furthermore, the velocity and acceleration of the lift are much higher than the velocity and acceleration of the shuttles. It is necessary that output containers arrive at the output buffer in a suitable time schedule for the lift to pick up. Because the lift travels much faster that the shuttles, prioritizing bringing an output to the output buffer is considered optimal. Although there is risk of have a time delay between lift double cycles due to

having all input buffers full, the waiting time of the lift by output buffer waiting for the container is greatly reduced.

2.2.4.5. Output Sequence

As explained in *Section 2.2.4.1*, orders enter the system and send unstorage signals to the stored containers. For the proper functioning of the system and to maintain deterministic choices for the transporters, containers have to depart from the system in the same order as they have been requested. This will condition the choice for a shuttle in a floor with various requested containers and also for the lift with various output buffers full. Randomness in those choices is with this strategy avoided, as well as non-logical strategies such as shortest-distance choice.

3. ARENA Simulation Software

ARENA Simulation Software is the main software tool used for the development of the following project. The main purpose of ARENA Simulation Software is to model and simulate both simple and complex discrete processes, and then generate very detailed output information.

The software was developed by Systems Modeling Corporation on 1982, which was acquired by Rockwell Automation at 2000, when the ARENA Simulation Software was considered to be the number one simulation and modelling package in the industry.

It was then incorporated as part of Rockwell Automation's production facility design environment. Rockwell Automation, which was at that point a manufacturing software applications and tools business incorporated with this agreement a brand new range of services to their customers, such as scheduling and simulation applications.

The modelling of a discrete process in ARENA Simulation Software is implemented by building flowcharts. ARENA provides graphical simulation modelling and analysis **modules**, that are combined to build a fairly wide variety of simulation models. Those modules are programmed with different actions and parameters that will represent the real functioning of the system. There is a wide variety of models and are typically grouped under **panels**.

ARENA Simulation Software is based on the simulation language SIMAN. Behind the modelling of any process with ARENA, SIMAN code is written and responsible of the simulation, and even can be reached, read and manually modified. As a matter of fact, in the initial versions of ARENA Simulation Software, the modelling of processes was generated by editing manually the corresponding lines of code. With the continuous enhancements of the software, ARENA provides an easy-to-use simulation tool, while it is all still based on the powerful and flexible SIMAN language.

ARENA Simulation Software maintains its modelling flexibility by being fully **hierarchical**, as shown in *Figure 10*[4]. Modules provided by ARENA are programmed with SIMAN code. Depending on the complexity of the module's function (and consequently the SIMAN code behind it), its hierarchy level is defined. Modules grouped under the same panel share normally same hierarchy level.

[4] KELTON, DAVID M; SADOWSKY, RANDALL P; SADOWSKI, DEBORAH A. *Simulation with Arena*. Page 11.

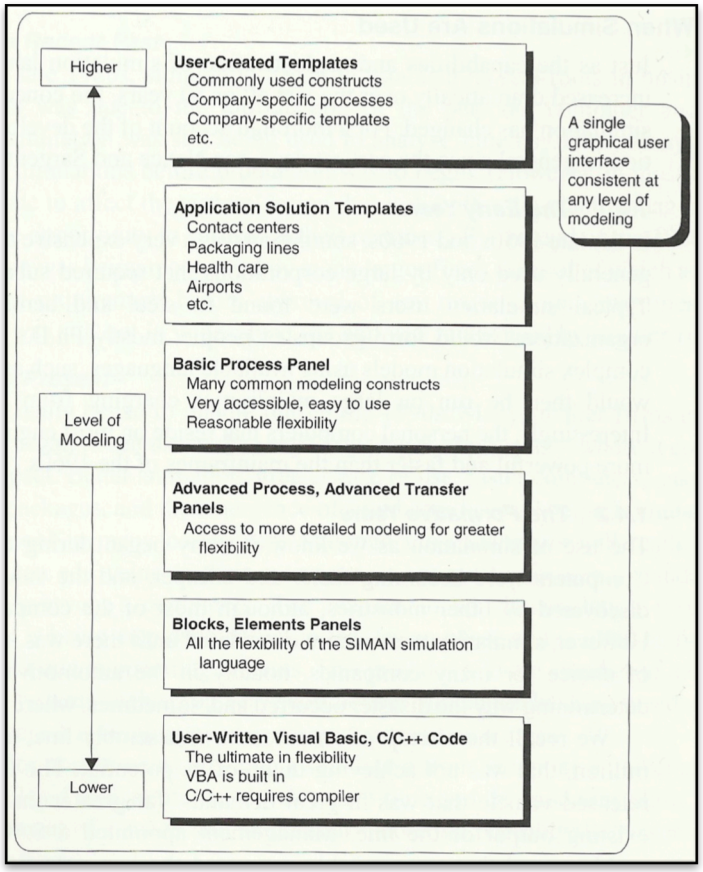

Figure 10: ARENA Simulation Software hierarchy structure

While building a model, modules from different panels can be used, allowing the user to become more or less flexible depending on the specific needs of the model. The Blocks and Elements panels are the lowest hierarchy panels, and grant the flexibility of SIMAN code.

Simulations with ARENA Simulation Software are entity-based simulations. Entities, that normally simulate goods or real objects, run through the flowcharts undergoing all the actions and processes of the modules. Entities are created at the starting point of the flowchart and are usually disposed at the end.

Note that the variety of possibilities is very wide. Entities would, on a very simplistic explanation, simulate goods travelling through a process system. However, many other uses can be given to entities, such as triggering operators at some instant or starters of machinery failures. This conclusion regarding the wide range of possibilities with entities can be extended to the ARENA Simulation Software in global. Although many processes and modules have been designed for a default purpose, the user can explore them for defining special configurations or personalizing systems.

It is important to point out another important distinction between modules: logical modules, data modules and animation modules. The first two types of modules are indispensable for the system to operate, and the animation modules are just for graphically representing the flow of entities under activity. The module flowcharts are composed of logical modules, which are the responsible of the actions undergoing in the system. On the other hand, the data modules do not interact in the system as setting actions for the flowcharts. Data modules have the role of listing all the different objects operating in the system. Each list of objects has a specific data module, and some examples of those objects are entities, variables, attributes, resources, transporters or conveyors.

With a successfully built model, ARENA Simulation Software offers multiple options to simulate it. There are many parameters that can be modified to simulate the model with the wanted configuration and to obtain the required output results. Run times, warm-up periods and base-time units are parameters referred to time. Because the animation if a powerful tool in ARENA, parameters related to the animation can be set, such as Frame per Second or run the simulation according to a real time parameter. Finally, ARENA also allows running simulation with different replications, which means simulating the model with the same parameters but setting new pseudo-random numbers.

4. Model construction

The construction of the required Shuttle Storage System model using ARENA Simulation Software has been developed from the beginning, in order to generate a total personified knowledge of the system's model and operation, which guarantees the desired real system operating activity.

Although it can seem rather slow to carry on, there are also relevant timesaving advantages that make this procedure worthy. The personified knowledge of the model allows the quick implementation of modifications and expansions, as well as granting the ability to detect easily malfunctions or mistakes. Knowing perfectly the system also allows the required information and statistical data to be efficiently extracted for its posterior analysis and study.

In order to provide an easy and clear way of describing the model, its construction has been divided into three steps. Each step defines a provisional model, which has been taken at that specific time of the model's construction as a useful tool for the model's description. This fact becomes especially important in this particular model due to the intelligence complexity of the transporters' combination as well as high-skilled software handling.

Each step is a totally well functioning model, which means it can be simulated without trouble and it realistically represents the desired parts of the system. However, depending on the depth of the model construction of the step, the results provided from the simulation might not be at all relevant, supposing that the modelled parts of the system do not provide a realistic functioning of the system because they are only a part of a global model.

In particular, the three steps in which our model construction has been divided are Initial Model (*Section 4.1*), Intermediate Model (*Section 4.2*) and Final Model (*Section 4.3*):

The Initial Model's description is by far the most important. The Initial Model represents a low-scale version of a Shuttle Storage System, developed using low hierarchy software functions, in order to achieve a very personified response from the model. Therefore, in this section is where all intelligence of the system is described.

The difference between the Initial Model and the Intermediate Model does not contribute to any model's intelligence modification. The Intermediate Model represents the same low-scale version of the Shuttle Storage System that the Initial Model represented, but with a very crucial modification: while the Initial Model was created using low hierarchy functions, the purpose of the Intermediate Model is to create very high hierarchy functions that enclose parts of the model. This step is

necessary to simplify the model before enlarging it to more realistic sizes and then simulate and analyse relevant results.

The Final Model is the enlarged version of the Intermediate Model. It is easily developed thanks to the simplification granted by the very high hierarchy functions created on the Intermediate Model. The size of the Final Model is not to be fixed, because it is a very important variable for the different simulations and analyses of the model.

4.1. Initial Model

The so-called Initial Model is a well functioning representation of the Shuttle Storage System, where the most important feature is that the material transporting activities are properly modelled as its realistic functioning. However, it is just the first step towards a totally appropriate realistic model. The fact that the Initial Model is not a proper model is because it is a low-scale model, where a Shuttle Storage System with only two floors and eight storage positions in each floor are modelled. Therefore, no relevant information or statistical data can be obtained from this model's simulation.

The Initial Model has been developed using low hierarchy functions of the ARENA Simulation Software. In this software, low hierarchy is often associated to very personified models, where each different function or operation is closely defined. Therefore, the Initial Model is the result of a step-to-step development, where the well functioning model has been achieved by gradually adding functions, operations or associated activities such as data collection to the model.

The purpose of the Initial Model development has been to achieve the correct operation response of the model, especially from the point of view of the material transporting activities, independently of the number of floors or storage positions. Its simplicity also allows describing easily its functioning features clearly. Therefore, all of the important basic modelling features of the system will be described on this following section.

4.1.1. Prior ARENA Simulation Software Concepts

As explained in *Section 3*, the software is an entity-based software. It means that during the simulation, each entity follows the model path and is affected by all its corresponding modules. Nevertheless, the model path is usually not unique and entities follow different routes according to the values of variables or time. Variables are very important on the construction and operation of models and it has to be clear that there are two kinds. On one hand, there is a data module that assigns individual values to each entity: the *Attribute Module*. On the other hand, there are also global entity-independent variables: the *Variable Module*. Both *Attributes* and *Variables*, although its

initial values can be specified, have to be set in the model to allow the simulation's proper functioning or can be changed as the simulation runs.

The two data modules introduced belong to the *Basic Process* template. As specified in *Section 3*, templates are groups of modules, both data modules and logic modules. The modules under the same template share features, topics or just hierarchy levels. The templates used to develop the Initial Model are the *Basic Process* template, the *Advanced Process* template and the *Advanced Transfer* template, although not all the modules under the templates are used on the model's construction. Also very flexible modelling was required, so also modules from the *Blocks* template and the *Elements* template have been used. The first two templates basically embrace the most generic modules used in model construction, but in two different hierarchy levels. The *Advanced Transfer* template includes the modules referred to defining physical or logical locations, transporting vehicles and conveying. Note that most of the non-topic modules are represented with a yellow shape, such as rectangles or rhombuses. The *Blocks* and *Elements* templates, which contain the lowest hierarchy modules at SIMAN language level, are represented as non-colour rectangles.

In many modules, the evaluation of expressions or conditions is required. To evaluate an expression or a condition is to define whether its content is true or false, and then to refer it to a value. As a matter of fact, when an evaluated expression is true, the expression as a whole is valued as a 1, and when an evaluated expression is false, the expression as a whole is valued as a 0. To understand this is key for acknowledgment of the system's intelligence, because multiple expressions, conditions and combinations of expressions are evaluated. It is also fundamental to have a clear idea of the basic Truth Tables, because they will be systematically used:

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table 1 – AND table

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table 2 – OR table

The fact that the basis of the system is the transportation of entities between different physical locations (storage space, buffers, input zone…), a classification of those physical locations has been created. Therefore, in order to efficiently describe the Initial Model, each different location has been separately described. In ARENA Simulation Software, physical locations can be modelled using the

station concept:

> *The **Station module** defines a station (or a set of stations) corresponding to a **physical or logical location where processing occurs**. [...] The station (or each station within the defined set) has a matching Activity Area that is used to report all times and costs accrued by the entities in this station. This Activity Area's name is the same as the station.*

Although above is only described the *Station Module*, there is a group of related logical modules: the *Enter Module*, the *Leave Module*, the *PickStation Module* and the *Route Module*, which all belong to the *Advanced Transfer* template. Those modules are referred to the definition of physical stations and the different connecting activities between stations, such as the destination to be chosen by an entity and the means of transportation to be used. The visual representation of those modules on the ARENA Simulation Software is red rectangles.

It is also basic for the development of the model to have a good command on all aspects regarding transporters. Transporters are responsible of carrying entities between stations, which in this particular case of study represents both the lift and shuttles. Therefore, special attention has to be taken for all modules referred to transporters. The transporter-related modules are on the *Advanced Transfer* template and the logical ones are visually represented by blue rectangles. The *Request Module* and the *Free Module* are the two important logical modules for the model, while the *Transporter Module* and the *Distance Module* are the important data modules. The mentioned logical modules attach and detach transporters to entities, while the data modules define the required properties of the transporter and distances between stations.

Finally, remark that many modules have been implemented only for data recording purposes, such as entity counters or time intervals. Those modules, that do not affect in any case the logic of the system, are marked with a grey rectangle and will not be described in the following sections. All the data recorded, although many was recorded just for control reasons to ensure the coherent functioning of the system and keep track on the required data, will be discussed in *Section 5* when describing and analysing the results.

4.1.2. Adaptation of features

Some features of the system will be modelled in specific ways according to the software needs. This adaption of features may not resemble the operation of the system in real life, but it will allow the model to simulate it correctly.

4.1.2.1. Warm Up period

As explained in *Section 3*, simulations with ARENA Simulation Software are entity-based simulations. Entities, that normally simulate goods or real objects, run through the flowcharts undergoing all the actions and processes of the modules.

Because in this system entities can only be created when the simulation starts, the system will initially be empty of containers. No stored containers will be on the rack at the initial instant. According to *Section 2.2.4.1*, the unstoring request of a container is sent at the same instant that it is stored. It would seem very complicated to simulate the smooth operating of the system in this case, because the first input container transported by the lift in the Input Cycle will be the same container transported by the lift in the Output Cycle, because no other containers are on the system. This fact would be recurrent, and the operating, cycle times and throughput will not be correct.

In order to correct this and simulate the system correctly, a warm up time has been implemented. This warm up time is an interval between the initial instant of the simulation until a certain predefined time value. This warm up time has the goal of only filling the system with stored containers. Therefore, the logic controlling the lift transporter is different, because double cycling cannot be forced and as many Input Cycles of the lift as possible will be done in a row. Also orders for requesting output containers are not allowed during the warm up period. The warm up period will last until it is guaranteed that some containers are stored in the system.

At this point, the regular operating of the system will start. Lift will start the double cycle logic and orders will be allowed in the system to request output containers. Note that orders will the randomly choose the stored containers to output, ensuring the proper functioning of the system. The data extracted from the warm up period is, under the point of view of this study, irrelevant, and only data extracted after the warm up period will be analysed.

4.1.2.2. Storage position

Regarding the storage positions, a little consideration needs to be pointed out. As seen in *Section 2.2.1*, the system is configured with two racks in parallel. Because the same positions on each rack share all the parameters regarding the shuttles (distances and loading/unloading times) and there is no apparent logical differences, both positions will be considered as one position with a capacity of two containers. This consideration has to be taken under account specially regarding the inventory variables of the system, where each cell of the rack will have a minimum value of 0 if empty and a maximum value of 2 if both racks are full.

4.1.2.3. Transporters independent movement

As explained in *Section 3*, simulations with ARENA Simulation Software are entity-based simulations. Entities, that normally simulate goods or real objects, run through the flowcharts undergoing all the actions and processes of the modules. These actions also refer to the allocation, transporting and release of transporters. This means that an entity (representing a container) needs to be on control of the transporter for setting the corresponding operation. In other words, an input container is going to request the lift, is going to make it travel to the corresponding floor and then is going to free it.

However, there are some transporter actions that lack of the need of a container. Specifically, this occurs in two occasions. In first place, when the lift finishes an Input Cycle, is going to automatically travel to the corresponding floor for output. In second place, when a shuttle starts an Input Cycle, it is going to travel to the input buffer. Both actions happen with independence of the status of the container that is going to be picked up, meaning that the container may not be at the buffer when this actions start. Therefore, the model flowcharts for requesting the transporter need to be alternative and not depend on the passing of the respective container to trigger the actions.

For its modelling, separate flowcharts have been implemented. Those flowcharts do not refer to any physical location on the system, because only affect the logic of the system. The use of those flowcharts can resemble operators on the system that control the transporters from the outside. The flowcharts, which will be described in *Section 4.1.3.7.* and *Section 4.1.3.8.*, will generate a special entity that will undergo all the transporter-related actions when required.

Whereas that when the shuttle is controlled by an operator it will only travel to the input buffer, when the lift is controlled by an operator it has to choose the floor where the next output container will be picked up, according to the Output Sequence. This cannot be simulated so easily, because it happens with independence of the output container. The lift has to know previously the sequence of floors to pick up output containers. To keep track of the output sequence, when orders enter the system and send the unstore signal to the containers, they will store the value of the output container's floor in a list, so the lift can check on the list what is the next floor to travel to. Each time an Output Cycle is carried away, the first value of the list is deleted. The list must be as long as the maximum number of storage positions on the system, that is the maximum number of requested containers that could still be in the system.

4.1.2.4. Output sequence guaranty

Output containers have to depart from the system in the same order has they have been requested, as explained in *Section 2.2.4.5.* The choice of what container to pick up is conditioned by this fact. Therefore, transporters are ruled by an output priority that is in accordance with the output sequence of the containers.

To model this feature, order counters have been implemented. Each time an order sends an unstore signal to a container, the counter is updated (increased by one) and the requested container will have the value of the counter assigned as output sequence. This way, only containers with the correct output sequence value will be allowed to request the transporter, respecting this way the output sequence.

Note that there is a global counter for the lift, but also particular counters for each shuttle. All transporters will start with an output priority value of 1. Only the container with the output sequence value of 1 (which means it is the first container requested for output) will be able to request the corresponding transporters. Each time an Output Cycle is carried away, the priority value of the transporter is increased by one, and will be requested by the next container in the output sequence.

4.1.2.5. Acceleration adaption

In *Section 2.2.2*, the velocity and acceleration values of the transporters have been specified. However, the type of transporters used in the model calculates the travelling times based only on distance and velocity. This means that the software does not simulate acceleration and deceleration of the transporters.

In order to implement correctly the velocity and acceleration for the calculation of times, the velocity value required has been overwritten by a formula that calculates the average velocity of the velocity cycle (including acceleration and deceleration steps).

$$v_{av} = \frac{dist}{t_{total}}$$

Lift calculus[5]

Lift parameters	
Velocity:	5 m/s
Acceleration:	7 m/s^2

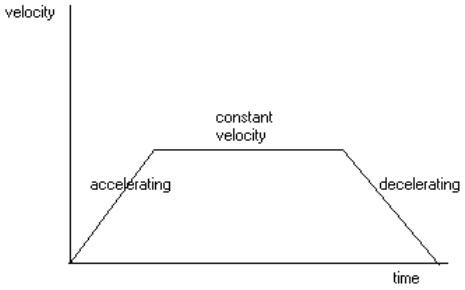

Figure 11: Velocity – Time plot

As it can be seen in *Figure 11*, the standard velocity of the lift requires a certain time of acceleration to be achieved, as well as a certain time of deceleration before it stops. This cycle in *Figure 11* represents the velocity evolution of all the travelling cycles of the transporters in the system.

$$t_{acc} = \frac{|v_f - v_i|}{a} = \frac{\left|5000\,mm/s - 0\,mm/s\right|}{7000\,mm/s^2} \approx 0{,}71s$$

$$z_{acc} = \frac{1}{2} a \cdot t_{acc}^{\,2} = \frac{1}{2} 7000\,mm/s^2 \cdot 0{,}71^2 s^2 \approx 1786mm$$

This time and distance values are required to achieve the standard velocity of 5000^{mm}/s. Also, the values referring the deceleration are:

$$t_{dec} = t_{acc} \approx 0{,}71s$$

$$z_{dec} = z_{acc} \approx 1786mm$$

Thanks to these values, a distinction is made between those cycles in which the standard velocity will be achieved and those cycles where only acceleration and deceleration will occur.

$$z_{dec} + z_{acc} \approx 3572mm$$

Therefore, the lift will only achieve the standard velocity if it is going to travel more than 3300mm, the value corresponding to 11 floors.

If the lift is going to travel **a distance of more than 11 floors**, the lift will operate as shown in the *Figure 11*. The total travelling time is, in this case, divided in three different values, which are acceleration time, deceleration time and time at constant velocity.

[5] Values extracted from *Table 3* on *Section 2.2.2.*

$$t_{total} = t_{acc} + t_{v.c.} + t_{dec} = t_{v.c.} + 1{,}42s$$

To calculate the time at constant velocity, the following formulas are used.

$$t_{v.c.} = \frac{z_{v.c.}}{v} = \frac{dist - 3572mm}{5000\,mm/s}$$

$$z_{v.c.} = dist - 3572mm$$

To calculate then the average value of the velocity:

$$v_{av} = \frac{dist}{t_{total}} = \frac{dist}{1{,}42s + \frac{dist - 3572mm}{5000\,mm/s}}$$

However, if the lift is going to travel **a distance equal or smaller than 11 floors**, the lift will not operate as shown in the *Figure 11*, but with a triangular distribution instead. The total travelling time is, in this case, divided in two different values, which are acceleration time and deceleration time. The values of the acceleration and deceleration time are in this case variable, and depend on the global distance.

$$t_{total} = t_{acc} + t_{dec} = 2 \cdot t_{acc}$$

To calculate the acceleration time, the following formulas are used.

$$z_{acc} = \frac{dist}{2}$$

$$z_{acc} = \frac{1}{2} a \cdot t_{acc}^{\,2} \Rightarrow t_{acc} = \sqrt{\frac{2 \cdot z_{acc}}{a}} = \sqrt{\frac{dist}{a}}$$

$$t_{total} = t_{acc} + t_{dec} = 2 \cdot t_{acc} = 2 \cdot \sqrt{\frac{dist}{a}} = 2 \cdot \sqrt{\frac{dist}{7000\,mm/s^2}}$$

To calculate then the average value of the velocity:

$$v_{av} = \frac{dist}{t_{total}} = \frac{dist}{2 \cdot \sqrt{\frac{dist}{7000\,mm/s^2}}}$$

Shuttle calculus[6]

Shuttle parameters	
Velocity:	2 m/s
Acceleration:	2 m/s^2

As it can be seen in *Figure 11*, the standard velocity of the shuttle requires a certain time of acceleration to be achieved, as well as a certain time of deceleration before it stops. This cycle in *Figure 11* represents the velocity evolution of all the travelling cycles of the transporters in the system.

$$t_{acc} = \frac{|v_f - v_i|}{a} = \frac{|2000\,mm/s - 0\,mm/s|}{2000\,mm/s^2} = 1s$$

$$x_{acc} = \frac{1}{2} a \cdot t_{acc}^{\,2} = \frac{1}{2} 2000\,mm/s^2 \cdot 1^2 s^2 = 1000mm$$

This time and distance values are required to achieve the standard velocity of 2000^{mm}/s. Also, the values referring the deceleration are:

$$t_{dec} = t_{acc} = 1s$$

$$x_{dec} = x_{acc} = 1000mm$$

Thanks to these values, a distinction is made between those cycles in which the standard velocity will be achieved and those cycles where only acceleration and deceleration will occur.

$$x_{dec} + x_{acc} = 2000mm$$

Therefore, the shuttle will only achieve the standard velocity if it is going to travel more than 2000mm, the value corresponding to 4 positions.

If the shuttle is going to travel **a distance of more than 4 positions**, the shuttle will operate as shown in the *Figure 11*. The total travelling time is, in this case, divided in three different values, which are acceleration time, deceleration time and time at constant velocity.

$$t_{total} = t_{acc} + t_{v.c.} + t_{dec} = t_{v.c.} + 2s$$

[6] Values extracted from *Table 4* on *Section* 2.2.2.

To calculate the time at constant velocity, the following formulas are used.

$$t_{v.c.} = \frac{z_{v.c.}}{v} = \frac{dist - 2000mm}{2000\,{}^{mm}/_{s}}$$

$$z_{v.c.} = dist - 2000mm$$

To calculate then the average value of the velocity:

$$v_{av} = \frac{dist}{t_{total}} = \frac{dist}{2s + \frac{dist - 2000mm}{2000\,{}^{mm}/_{s}}}$$

However, if the shuttle is going to travel **a distance equal or smaller than 4 positions**, the shuttle will not operate as shown in the *Figure 11*, but with a triangular distribution instead. The total travelling time is, in this case, divided in two different values, which are acceleration time and deceleration time. The values of the acceleration and deceleration time are in this case variable, and depend on the global distance.

$$t_{total} = t_{acc} + t_{dec} = 2 \cdot t_{acc}$$

To calculate the acceleration time, the following formulas are used.

$$x_{acc} = \frac{dist}{2}$$

$$x_{acc} = \frac{1}{2} a \cdot t_{acc}^{\,2} \Rightarrow t_{acc} = \sqrt{\frac{2 \cdot x_{acc}}{a}} = \sqrt{\frac{dist}{a}}$$

$$t_{total} = t_{acc} + t_{dec} = 2 \cdot t_{acc} = 2 \cdot \sqrt{\frac{dist}{a}} = 2 \cdot \sqrt{\frac{dist}{2000\,{}^{mm}/_{s^2}}}$$

To calculate then the average value of the velocity:

$$v_{av} = \frac{dist}{t_{total}} = \frac{dist}{2 \cdot \sqrt{\frac{dist}{2000\,{}^{mm}/_{s^2}}}}$$

It is very important to have clear that the transporter-related modules only allow one velocity expression. To model this, each time a movement transporter-related module (that requires a velocity value) should be found, the container finds first different branches to choose according to

the distance it is going to travel and, consequently, the formula it is going to use. This guides the container to the module with the correct formula. After these modules, the branches reunite to follow the flowchart as one. This should be in mind every time the container meets a movement transporter-related modules, because it will not be explained in detail again in the model construction description.

4.1.3. Initial Model Construction

As previously specified, a separation between stations has been made in order to be concise and effective while describing the system. However, although each station will be separately described, it has to be taken under account that many logical actions are interrelated between stations and that stations are connected with transporter paths. Therefore, there is never total independence between the functioning of stations.

The order of stations to be described will follow the path of goods. Therefore, first the flowchart path followed by goods in the Input Cycle is followed, and after the flowchart path in Output Cycle is described. Afterwards, supplementary flowcharts will be described, as well as the animation used.

4.1.3.1. Box Arrivals Station

Description: The "Box Arrivals" Station represents the location from where containers enter the system until they leave with the lift to the desired floor. The main actions that occur in this station are the creations containers, the assignation of a destination storage position and the request of the lift.

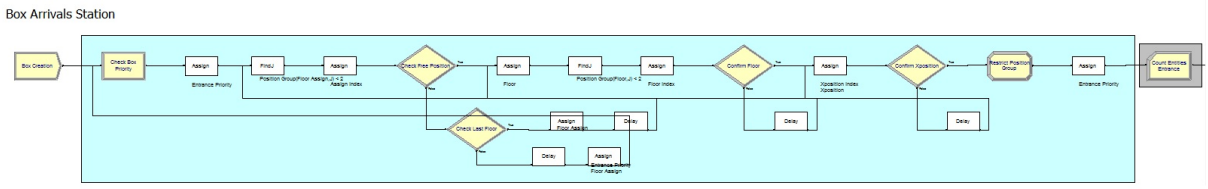

Figure 12: Box Arrivals Station Flowchart (I)

The arrival of containers to the system is modelled with a *Create Module* (from the *Basic Process* template), whose main function is to create containers with a specific inter-arrival time. The created entities, named "Box", represent the containers to be stored and are defined in the data *Entity Module* (from the *Basic Process* template). Because the focus of the model analysis is the functioning and times of the means of transportation, the rate of arrivals is to some degree irrelevant. Therefore, a group of containers will be created at instant t = 0 of the simulation and will generate a queue.

Note that it has been taken under account that the number of entities created at the beginning will be, in any case, fairly higher than the throughput of the system. Specifically, 10.000 entities will be created for a 10.000 second simulation (as it will be specified at *Section 5*), which guarantees constant input supply queue with enough capacity for not running out of inputs during the simulation. For more information, *see Section 2.2.4.1*.

The first operation that a recently created container has to go through is the definition of the position where it will be stored. This process is defined in the modules grouped under the blue box on *Figure 12*. The process adopted to solve this is by generating an array variable. This matrix represents constantly which positions from the storage system are available for storage and can be assigned to the containers. This variable, defined in the data *Variable Module* (from the *Basic Process* template), is called "Position Group" and has as many rows as floors has the system – in this particular case, two – and has as many columns as storage locations has each floor – in this particular case, eight –. "Position Group" initial values for all the cells are 0, which means that the position is empty. It's maximum value is 2, meaning that containers on both racks have been assigned for that position.

Regarding the actions in the modules encompassed under the blue box on *Figure 12*, containers will enter the blue box one at a time. In first place, they will scan the "Position Group" matrix to find any available position. If no position is found, meaning that all positions in the system have already been assigned to other containers, the current container will be looped to the queue a will trigger a certain delay before another container is allowed to enter the blue box modules, giving time to the system to output the corresponding containers. However, if there is at least one position available, random floor and position values will be assigned to the container and, after checking that they are indeed available, restricted from the "Position Group" matrix. The random assignation is implemented with the following expressions representing uniform distributions and the values are assigned to two values from the data *Attribute Module* (from the *Basic Process* template): "Floor" and "Xposition".

Expression (1):

Floor = ANINT(UNIF(1,UBOUNDROW("Position Group")))

Expression (2):

Xposition = ANINT(UNIF(1,UBOUNDCOL("Position Group")))

Once each container has “Floor” and “Xposition” successfully defined, the cell value corresponding to that specific position has to be restricted from “Position Group”. This ensures that no other container can have that same location assigned until the previous container has been unstored. To restrict the position from “Position Group”, the value corresponding to the “Floor” row and to the “Xposition” column has to be increased by 1.

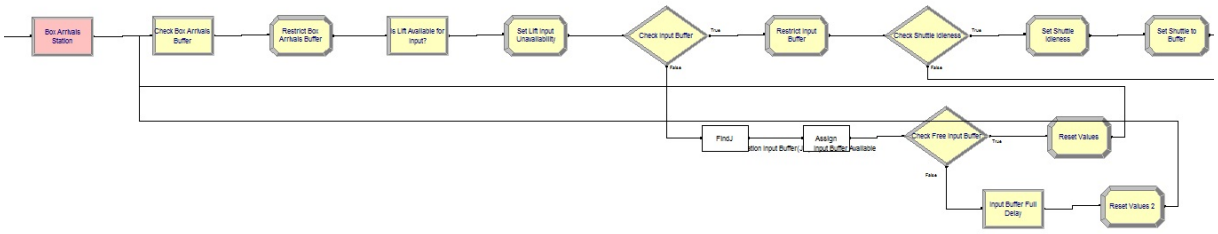

Figure 13: Box Arrivals Station Flowchart (II)

After the successful assignment of a position, a *Station Module* is found. The module only has the function of defining a current physical location of the containers and all the undergone actions. That is to say that all actions happen at the same physical location until a transporting unit such as a transporter or a conveyor moves the container to another physical location. This module sets the name of the station to “Box Arrivals”.

At this point, all containers with an assigned storage position form a second queue, in which they wait for the lift to pick them up. In first place, a theoretical buffer has been modelled. The purpose of this buffer is, on one hand, to allow entities to enter the following modules only one at a time, and, on the other hand, to define a variable from data the *Variable Module* named “Box Arrivals Buffer” that reflects the state of the theoretical buffer. Its value can either be 1 – when a buffer is available – or 0 – when a buffer is full –.

As explained in *Section 2.2.4.2*, one of the system’s most important required feature is to prioritize the “Lift” double cycles. It is meant that the “Lift” prioritizes transporting consecutively containers in input and containers in output. Two global variables from the *Variable Module* have been implemented: “Lift Input Availability” and “Lift Output Availability”. The value of the variables can be 0 – which means that the transporter is available and has priority to carry out the operation – or 1 – which means that the transporter has no priority for the operation –. In case that there are no requesting entities for the “Lift”, another variable has been defined named “Lift Idleness” and its values can be 0 – which means that the “Lift” has no requesting entities – or 1 – which means that the “Lift” is on action –. The initial values of these variables are “Lift Input Availability” = 1, “Lift

Output Availability" = 1 and "Lift Idleness" = 0.

The first container of the queue, that is the only container that occupies the "Box Arrivals Buffer", meaning that is the only one with possibility to request the lift, will be held until the conditions regarding "Lift Input Availability" and "Lift Idleness" are met:

Condition (1):

(Lift Input Availability == 0) || (Lift Idleness == 0)

As it will be observed around the entire model, precautions regarding the request of the "Lift" have been very thoughtful. In order not to stress the logic and decision of the "Lift" with constant multiple container requests from different stations, the use of the availability variables prior described is combined with various *Hold Modules* to ensure that the "Lift" will only be requested by one container at a time.

As soon as a container meets the lift conditions, the "Lift" becomes unavailable for any container request. Therefore, it is necessary to restrict the input availability and idleness of the lift.

The next step is referred to the deadlock situation solution. As explained on *Section 2.2.4.3*, the lift is not allowed to input a container with a destination floor in which the corresponding input buffer is full. Therefore, the input buffer of the current container is checked and, if buffer is not full, it continues its path through the flowchart. However, if the respective input buffer is full, the container will take an alternative path and in first place scan if there is any input buffer empty. If there is any input buffer available, the current container will be looped to the "Box Arrivals Buffer" queue and another container will be instantly allowed to enter the "Box Arrivals Buffer" and take its place. This action will be repeated until a container with an empty input buffer at the corresponding floor is found. However, if the first scan reveals that no input buffers are empty, the container will be looped to the queue but with a specific time delay, in order to give time to the shuttles to empty the input buffers.

When a container meets the input buffer condition, the status of the input buffer for the corresponding floor is restricted, so no other containers can start an input buffer to that floor until it is emptied by the shuttle. To model the buffers, two instances from the *Variable Module* have been defined. These global variables, under the names of "Transfer Station Input Buffer" and "Transfer Station Output Buffer", are vectors whose numbers of rows correspond to the number of floors of the system – in this particular case, two –. Each cell represents the state of the corresponding buffer, and its value can be either 1 – when a buffer is available – or 0 – when a buffer is full –.

In resume, at this point, the appropriate container will be occupying the "Box Arrivals Buffer" with the lift awaiting an input container to start the Input Cycle. The next actions should be regarding the loading of the container to the lift. However, before starting these actions, the container will scan the status of its corresponding shuttle at the destination floor. If the shuttle is currently idle, it will be assigned to travel to the buffer position X = 0 in order to save time.

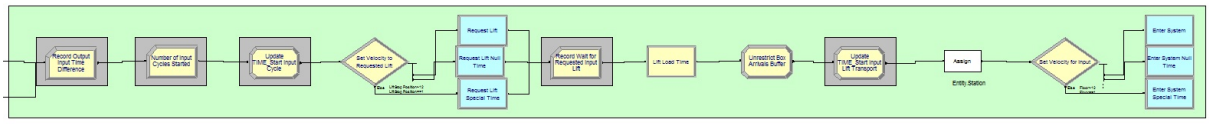

Figure 14: Box Arrivals Station Flowchart (III)

In next place, the lift-transporting unit must be defined. This has been done using the data *Transporter Module* (from the *Advanced Transfer* template). The transporter has been named "Lift", defined as a Free Path transporter type and with a velocity of 5000 ^{mm}/s.

The containers then come across a logical *Request Module* (from the *Advanced Transfer* template). This module belongs to the transporter-related group of modules. In particular, the *Request Module* assigns a transporter – in this particular case, the "Lift" – to the requesting container. Note that the requesting container will wait at the *Request Module* until the transporter is assigned and moved to the physical location of the *Request Module* – in this particular case, the "Box Arrivals" Station –. However, because double cycling is forced, the "Lift" will already be located at the station after deploying an output container.

Once container and "Lift" are united, the load time is modelled with a logical *Delay Module* (from the *Advanced Process* template). The load time of the "Lift" is 1.4s.

At this point, containers are ready to depart with the "Lift" to the required destination floor. Therefore, the state of the "Box Arrivals Buffer" has to be updated, in order to let other entities can enter.

The lasts modules of the "Box Arrivals" Station are obviously related to the transfer of containers to other stations. The module used to carry out this operation is the logical *Transport Module* (from the group of transporter related modules of the *Advanced Transfer* template). This module chooses the destination station of the containers that pass through. The destination floor, which has been specified on the initial modules, is composed of various stations, but all containers entering and departing from the floor converge in the "Floor ## Transfer Station".

The combination of container and transporter is, at this point, leaving the system to the pre-defined destination. However, the transporter's features only include the velocity of transportation and, in order to calculate the corresponding travel time, distances between stations need to be defined. This operation is modelled by the data *Distance Module* (from the *Advanced Transfer* Template), where the following matrix is defined:

Initial Station	Final Station	Distance
Box Arrivals	Floor 01 Station Transfer	0 mm
Box Arrivals	Floor 02 Station Transfer	300 mm
Floor 01 Station Transfer	Floor 02 Station Transfer	300 mm
Box Arrivals	Box Departure	0 mm
Floor 01 Station Transfer	Box Departure	0 mm
Floor 02 Station Transfer	Box Departure	300 mm

Distance Table 1 - Lift

Note that the Distance matrix includes all the possible connections available for the "Lift", even to those stations that have not been described yet.

4.1.3.2. Floor ## Transfer Station (I)

Description: The "Floor ## Transfer" Station represents the location on each floor where entities swap from the "Lift" to the shuttles that transport them to the desired storing positions and vice versa. Because this is a bidirectional location, the two flow directions have been separately described and in this section only the actions referred to the entities on the input step will be explained. The main actions that occur in this station are the release of the "Lift", an intermediate buffer and the request of the shuttle.

Note that the "Floor ## Transfer" Station will be generically described, although there is a "Floor ## Transfer" Station for each floor – in this particular case, two – and each station works exactly the same. The ## is in each floor the related floor number, and all modules in that station will have the same reference as the floor. Therefore, all ## in modules in the same floor transfer station will be the same number, for example 01 or 02 for floor's number 1 or 2.

As said, "Floor ## Transfer" Station has been divided into two different sections: *4.1.2.2- Floor ## Transfer Station (I)* and *4.1.3.4- Floor ## Transfer Station (II)*. The first section models the flow of entities entering the system to be stored and the second section models the flow of those leaving

the system because they are being unstored. Because the operations for both cases are different, two different module flowcharts have been modelled, although they have a common entering path.

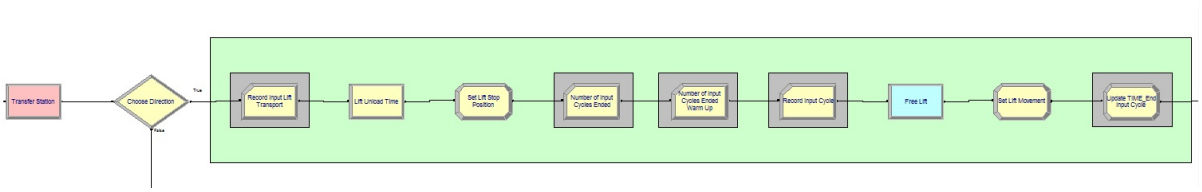

Figure 15: Floor ## Transfer Station (I) Flowchart (I)

The first module of the "Floor ## Transfer" Station has to be necessarily related to stations and its in-between transportation. The arrival of entities to the station is done by transporters and a clear route is required. Therefore, the station needs to be properly defined and set as an arrival port of entities. In order to do this, a logical *Enter Module* (from the *Advanced Transfer* template) has been defined. The function of this module is to define the new physical location and to embrace the arrival of entities.

Containers arriving to the "Floor ## Transfer" Station can be either on the input step of in the output step. Therefore, a value from the *Attribute Module* has been set under the name of "Direction". "Direction" can have only two values: 0 – when a container is in input direction – and 1 – when a container is in output direction –. "Direction" is by default set to 0 when entities are created.

In order to take the correct flowchart path, "Direction" is evaluated in a logical *Decide Module*, where entities choose its corresponding route. In this section, only the path of those entities with "Direction" = 0 will be described, which means that are under the input direction.

Note that both the "Floor ## Transfer Station" and "Floor ## Choose Direction" modules are common on the input and on the output flowcharts. This means that they are both described in *Section 4.1.3.2- Floor ## Transfer Station (I)* and in *Section 4.1.3.4- Floor ## Transfer Station (II).*

At this point, containers enter a group of modules where all the actions regarding the lift happen, under a green rectangle on *Figure 15*. Note that the "Floor ## Transfer" Station is, in reality, a platform where the "Lift" and the shuttles deploy and pick up entities at the same time. This exchange is carried out thanks to intermediate buffers, where entities reside after being deployed and await to be picked up by its corresponding transporter. Remark that the capacity of these buffers is of only one container. These intermediate buffers are very useful, because they grant independence between both transporters and make the system more efficient. In every floor there is one buffer for entities on the input direction and another buffer for entities on the output direction.

Remark that the act of accessing the input buffer and restricting it from other entities was carried away before the lift departure to the floor, although it might not be the strict physical representation. It is important to have buffer restricted before any other container tries to access it, even if the current lift-transported unit has still not been deployed to the buffer itself.

In first place, entities encounter modules related to the release of the "Lift". In first place, a *Delay Module* simulates the time required for the lift to unload the entities, which is of 1.4s.

At this point, the lift is ready to be released for the Between Cycle. To carry this operation out correctly, different actions need to be done. In first place, the current position of the lift needs to be saved, in order to properly calculate travelling times between stations for the next step. The next step of the lift is the Between Cycle. The Between Cycle, as explained in *Section 4.1.2.3*, is carried away without the use of the container. This means that the transporter is moving without having any container allocated. As explained in *Section 4.1.2.3*, to carry this sort of operations out, transporter operators have been designed. Therefore, the signal for the operator to start the Between Cycle has to be sent.

The module of the group referred to the release of the transporter is a *Free Module* (from the group of transporter-related modules from the *Advance Transfer* template). The module erases the attachment between container and transporter, allowing the container to continue its path along without the transporter and frees the transporter from the container, allowing it to be assigned to other requesting entities.

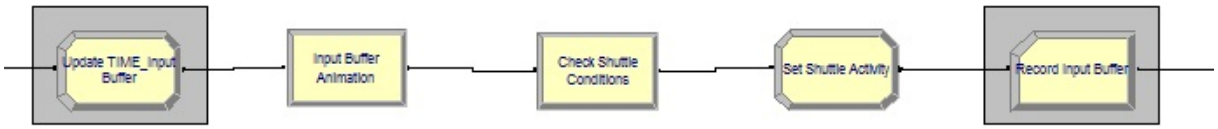

Figure 16: : Floor ## Transfer Station (I) Flowchart (II)

Once the lift has been released, the entities need to request the shuttle in order to proceed to the storing in the desired location. The interval of time between the release of the lift and the allocation to the shuttle is the time where the container remains in the input buffer. The container will remain in the input buffer until the conditions for requesting the shuttle are met. Those conditions make reference to the idleness of the shuttle – the container can not request the shuttle if it is still under operation – and to the priority of the shuttle – the shuttle must be allowed to be requested by an

input –. If the conditions are met, then the container is allowed to advance to the shuttle related modules.

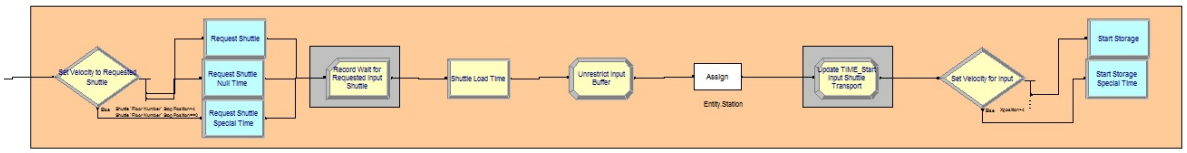

Figure 17: : Floor ## Transfer Station (I) Flowchart (III)

In first place, the shuttle-transporting unit must be defined. This has been done using the data *Transporter Module* (from the *Advanced Transfer* template). The transporter has been named "Shuttle ##", defined as a Free Path transporter type and with a velocity of 2000 ^{mm}/s. There will be as many shuttles as floors have the system.

In the next step, a *Request Module* is used, in order to assign the "Shuttle ##" to the container. Then, a *Delay Module* simulates the loading time of the container to the shuttle, which is of 5s.

At this point, the entities are ready to depart with the shuttle to the desired position, but before departing, "Transfer Station Input Buffer" has to be set again to its available status for the corresponding floor, allowing entities to enter the "Floor ## Transfer" Station for input. This is done with the an *Assign Module* and the following expression:

The modelling of the departure is done with a *Transport Module*, the same way it was done to model the floor assignation while transporting entities with the "Lift". The module will evaluate the "Xposition" attribute to set the destination storing positions – in this particular case, there are eight possibilities –.

The combination of container and transporter is, at this point, leaving the system to the pre-defined destination. However, the transporter's features only include the velocity of transportation and, in order to calculate the corresponding travel time, distances between stations need to be defined. This operation is modelled by the data *Distance Module* (from the *Advanced Transfer* Template), where the following matrix is defined:

Initial Station	Final Station	Distance
Floor ## Transfer	Position ##01	500 mm
Floor ## Transfer	Position ##02	1000 mm
Floor ## Transfer	Position ##03	1500 mm
Floor ## Transfer	Position ##04	2000 mm
Floor ## Transfer	Position ##05	2500 mm
Floor ## Transfer	Position ##06	3000 mm
Floor ## Transfer	Position ##07	3500 mm
Floor ## Transfer	Position ##08	4000 mm
Position ##01	Position ##02	500 mm
Position ##01	Position ##03	1000 mm
Position ##01	Position ##04	1500 mm
Position ##01	Position ##05	2000 mm
Position ##01	Position ##06	2500 mm
Position ##01	Position ##07	3000 mm
Position ##01	Position ##08	3500 mm
Position ##02	Position ##03	500 mm
Position ##02	Position ##04	1000 mm
Position ##02	Position ##05	1500 mm
Position ##02	Position ##06	2000 mm
Position ##02	Position ##07	2500 mm
Position ##02	Position ##08	3000 mm
Position ##03	Position ##04	500 mm
Position ##03	Position ##05	1000 mm
Position ##03	Position ##06	1500 mm
Position ##03	Position ##07	2000 mm
Position ##03	Position ##08	2500 mm
Position ##04	Position ##05	500 mm
Position ##04	Position ##06	1000 mm
Position ##04	Position ##07	1500 mm
Position ##04	Position ##08	2000 mm
Position ##05	Position ##06	500 mm
Position ##05	Position ##07	1000 mm
Position ##05	Position ##08	1500 mm
Position ##06	Position ##07	500 mm
Position ##06	Position ##08	1000 mm
Position ##07	Position ##08	500 mm

Distance Table 2 – Shuttle ##

Note that the Distance matrix includes all the possible connections available for the “Shuttle ##”.

4.1.3.3- Position #### Station

Description: The “Position ####” Station represents the locations where entities are stored. The main actions that occur in this stations are the release of the “Shuttle ##”, the storage of the container until it has to be removed, and the following request of the “Shuttle ##” for removal.

Note that the “Position ####” Station will be generically described, although there is a “Position ####” Station for each position – in this particular case, sixteen – and each station works exactly the same. The #### is in each position related to both “Floor” number and “Xposition” number, where

the first two ## are related to “Floor” and the last two ## are related to “Xposition”. All modules in the station will have the same reference number. As an example, the fifth position on the second floor will have a reference number of 0205 for all modules.

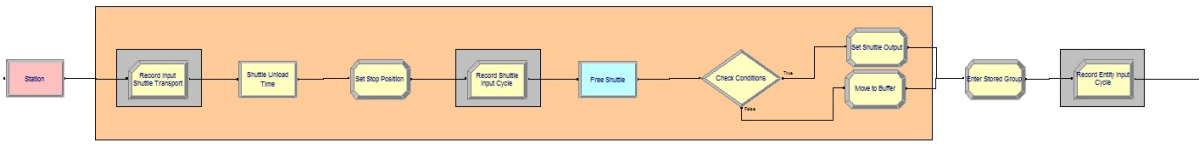

Figure 18: Position #### Station Flowchart (I)

The first module of the “Position ####” Station has to be necessarily related to station and its in-between transportation. The arrival of entities to the station is done by transporters and a clear route is required. Therefore, a logical *Enter Module* has been defined.

The first step for containers is entering a group of modules where all actions related to the unload and detachment from the shuttle are described, and are grouped under a orange rectangle. In first place, the unload time of the “Shuttle ##” is modelled with a logical *Delay Module* with a time of 6.5s. Then, the current position of the shuttle needs to be saved, in order to properly calculate travelling times between stations for the next step. The following *Free Module* represents the release of the “Shuttle ##” from the container. It erases the attachment between container and transporter.

Before storing, the container will come across the modules to set the next step of the shuttle. As explained in *Section 2.2.4.4*, the shuttle does not follow a constant cycle like the lift, but the next step depends on the status of the system. Therefore, the modules will evaluate the proper conditions and the variables will be assigned. There are three possibilities: to start an Output Cycle, to start an Input Cycle of to become idle. That decision depends on the status of the buffers. Remark that the priority for the system is to start an Output Cycle if the output buffer is free rather than an input buffer.

Figure 19: Position #### Station Flowchart (II)

At this point, “Box” is to be stored indefinitely until it is asked to be removed.

In first place, a new instance from the data *Variable Module* has been defined to have a constantly updated report of the state of the storage positions. Although it is already known which positions are going to be occupied thanks to "Position Group", the new variable "Stored Group" represents exactly the interval from where a container is stored until it is removed with the shuttle. "Stored Group" will have the same size as "Position Group", which is as many rows as "Floor" – in this particular case, two – and as many columns as "Xposition" – in this particular case, eight –. Each cell of the matrix can have two values: 0 – when the storage position is empty – and 1 – when the storage position is full –. Remark that the initial values of the matrix are set to 0, because the model works under the hypothesis that the system is initially empty.

The following module is the responsible of the storage. To model the storage period, a *Hold Module* has been implemented. Containers will reside on this module until the certain condition is met. This condition acts as the unstore trigger for stored containers.

To model the unstoring orders, a new instance from the data *Variable Module* has been implemented. Depending on the values of the variable, containers will meet or not the *Hold Module* conditions and trigger its output step. The new variable, named "Unstore Signal", will have the same size as "Position Group", which is as many rows as "Floor" – in this particular case, two – and as many columns as "Xposition" – in this particular case, eight –. Each cell of the matrix can have two values: 0 – when entities do not have to be unstored – and 1 – when entities have to be unstored –. Remark that the initial values of the matrix are set to 0, because the model works under the hypothesis that the system is initially empty.

As long as the value of the "Unstore Signal" cell continues being set to 0, containers will reside on the module without advancing, simulating the storage period. Therefore, the condition to be met on "Position #### Storage" to trigger the output step is:

Condition (1):

Unstore Signal(Floor,Xposition) == 1

The assignment of orders to "Unstore Signal" will be properly described on *Section 4.1.2.6.*

Containers at this point are ready to depart from the system. They have been properly stored for the desired time, but are now are requested to be unstored. The first unstore step is to leave the "Position ####" Station with the corresponding "Shuttle ##", and being deployed again in the "Floor ## Transfer" Station. The first step is to request the shuttle, but however, certain conditions need to be met first. As explained in *Section 2.2.4.5*, output containers have to depart from the system with

the same order as they are requested. For this reason, containers have assigned an output sequence, and transporters (both lift and shuttle) will only pick up output containers according to the correct output sequence. Therefore, the first condition for requesting the shuttle is that it is the container's right time to output according the output sequence. This means that from all the containers in the same floor that have been requested to output, the current container was requested first. If the container meets the condition, then conditions regarding the shuttle need to be met, such as shuttle idleness and shuttle output prioritization, as explained when requesting shuttle for input in *Section 4.1.3.2.*

Containers are now ready to depart from the system. However, because the storing position is going to been emptied, all variables related will be set again to its initial values, in order to make it available for new entities. Also, "Direction" will be set to the value that indicates that the container is on the output step, so it will follow the correct flowchart path on "Floor ## Transfer" Station.

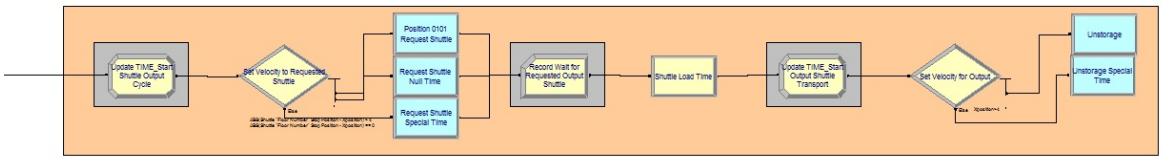

Figure 20: Position #### Station Flowchart (III)

If all output conditions are completed, then the modules referred to the output shuttle cycle are met. In the next step, a *Request Module* is used, in order to assign the "Shuttle ##" to the container. Then, a *Delay Module* named "Position #### Shuttle Load Time" simulates the loading time of the container to the shuttle, which is of 5s.

To model the departure of the transporter with the container, a logical *Leave Module* (from the transporter-related group of modules from the *Advanced Transfer* template) has been implemented. Because all containers departing from this station have the same destination, this module is much simpler compared to the *Transport Module*, where expressions had to be evaluated in order to assign the correct destination. In this case, only the destination "Floor ## Transfer" Station needs to be defined.

4.1.3.4. Floor ## Transfer Station (II)

Description: The "Floor ## Transfer" Station represents the location on each floor where entities swap from the "Lift" to the shuttles that transport them to the desired storing positions and vice versa. Because this is a bidirectional location, the two flow directions have been separately described

and in this section only the actions referred to the entities on the output step will be explained. The main actions that occur in this station are the release of the "Shuttle ##", an intermediate buffer and the request of the "Lift".

Note that the "Floor ## Transfer" Station will be generically described, although there is a "Floor ## Transfer" Station for each floor – in this particular case, two – and each station works exactly the same. The ## is in each floor the related floor number, and all modules in that station will have the same reference as the floor. Therefore, all ## in modules in the same floor transfer station will be the same number, for example 01 or 02 for floor's number 1 or 2.

As said, "Floor ## Transfer" Station has been divided into two different sections: *4.1.3.2- Floor ## Transfer Station (I)* and *4.1.3.4- Floor ## Transfer Station (II)*. The first section models the flow of entities entering the system to be stored and the second section models the flow of those leaving the system because they are being unstored. Because the operations for both cases are different, two different module flowcharts have been modelled, although they have a common entering path.

The first module of the "Floor ## Transfer" Station has to be necessarily related to stations and its in-between transportation. The arrival of entities to the station is done by transporters and a clear route is required. Therefore, the station needs to be properly defined and set as an arrival port of entities. In order to do this, a logical *Enter Module* (from the *Advanced Transfer* template) has been defined with the name of "Floor ## Transfer Station". The function of this module is to define the new physical location and to embrace the arrival of entities.

Entities arriving to the "Floor ## Transfer" Station can be either on the input step of in the output step. Therefore, a value from the *Attribute Module* has been set under the name of "Direction". "Direction" can have only two values: 0 – when a container is in input direction – and 1 – when a container is in output direction –. "Direction" is by default set to 0 when entities are created.

In order to take the correct flowchart path, "Direction" is evaluated in a logical *Decide Module* named "Floor ## Choose Direction", where entities choose its corresponding route. In this section, only the path of those entities with "Direction" = 1 will be described, which means that are under the input direction.

Note that both the "Floor ## Transfer Station" and "Floor ## Choose Direction" modules are common on the input and on the output flowcharts. This means that they are both described in *Section 4.1.3.2- Floor ## Transfer Station (I)* and in *Section 4.1.3.4- Floor ## Transfer Station (II).*

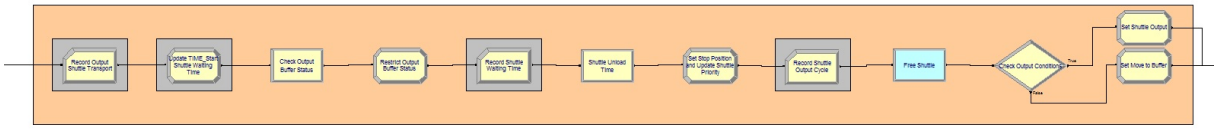

Figure 21: Floor ## Transfer Station (II) Flowchart (I)

At this point, entities enter a group of modules where actions regarding the shuttle happen. Note that the "Floor ## Transfer" Station is, in reality, a platform where the "Lift" and the shuttles deploy and pick up entities at the same time. This exchange is carried out thanks to intermediate buffers, where entities reside after being deployed and await to be picked up by its corresponding transporter. Remark that the capacity of these buffers is of only one container. These intermediate buffers are very useful, because they grant independence between both transporters and make the system more efficient. In every floor there is one buffer for entities on the input direction and another buffer for entities on the output direction.

Remember that to model the buffers, two instances from the *Variable Module* had been defined, under the names of "Transfer Station Input Buffer" and "Transfer Station Output Buffer". Remark that, right before departing "Position ####" Station, the value "Transfer Station Output Buffer (Floor,1)" has already been set to unavailable, which guarantees that no other entities can enter the station as a representation of the occupation of the output buffer.

In first place, the status of the output buffer will be checked. According to the logic of the model, the shuttle cannot bring an output container if the output buffer is full. Therefore, the status of the buffer will be empty. However, it will be then restricted, as the output container enters the output buffer.

Next, the first modules that entities encounter are related to the release of the "Shuttle ##". In first place, a *Delay Module* simulates the time required for the lift to unload the entities, which is of 6.5s. The last module of the group is a *Free Module*, which erases the attachment between container and transporter.

As it happened in the "Position" station, the container will come across the modules to set the next step of the shuttle. As explained in *Section 2.2.4.4*, the shuttle does not follow a constant cycle like the lift, but the next step depends on the status of the system. Therefore, the modules will evaluate the proper conditions and the variables will be assigned. There are three possibilities: to start an Output Cycle, to start an Input Cycle of to become idle. That decision depends on the status of the buffers. Remark that the priority for the system is to start an Output Cycle if the output buffer is free

rather than an input buffer.

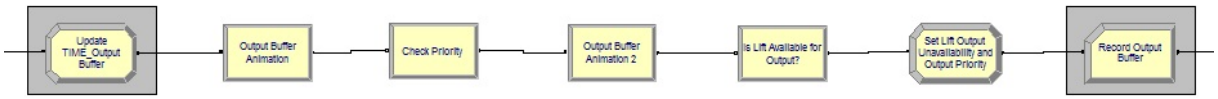

Figure 22: Floor ## Transfer Station (II) Flowchart (II)

Once the container has been unloaded and freed from the shuttle, it is technically in the output buffer, waiting to request the lift. In order to request the lift, certain conditions need to be met. As explained in *Section 2.2.4.5*, the output sequence needs to be respected. This means that containers depart from the system in the same order that they are requested. Only one container will meet the output condition. However, this container will still be held until the lift conditions are met. Those conditions make reference to the "Lift Idleness" and to the "Lift Output Availability".

Condition (1):

(Lift Output Availability == 0) || (Lift Idleness == 0)

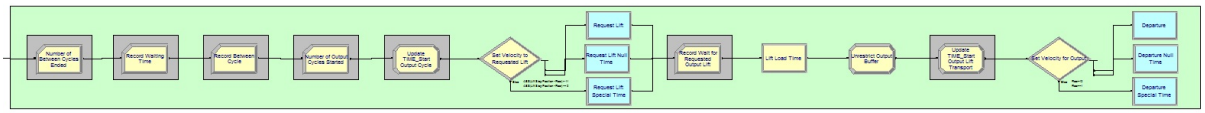

Figure 23: Floor ## Transfer Station (II) Flowchart (III)

The following modules represent all the actions regarding the lift allocation and departure for unstore. The containers come across a logical *Request Module*, which belongs to the transporter-related group of modules. In particular, the *Request Module* assigns a transporter – in this particular case, the "Lift" – to the requesting container. Once container and "Lift" are united, the load time is modelled with a logical *Delay Module* called "Floor ## Lift Load Time". The load time of the "Lift" is 1.4s.

At this point, containers are ready to depart from the station. Therefore, in order to allow the unstoring of containers, the current output buffer will be set again to available.

To model the departure of the transporter with the container, a logical *Leave Module* named has been implemented, because all entities departing from this station have the same destination. In this case, only the destination "Box Departure" Station needs to be set as destination.

4.1.3.5. Box Departure Station

Description: The "Box Departure" Station represents the location where containers are deployed and depart from the system. The main actions that occur in this station are the release of the lift and the disposal of entities.

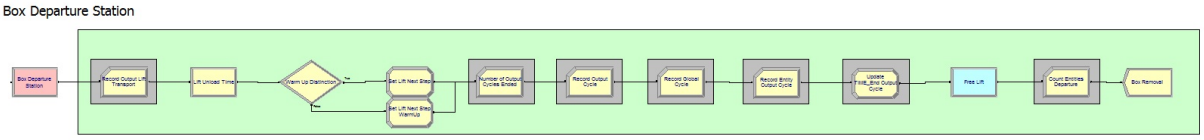

Figure 24: Box Departure Station Flowchart (I)

The first module of the "Box Departure" Station has to be necessarily related to station and its in-between transportation. The arrival of entities to the station is done by transporters and a clear route is required. Therefore, a logical *Enter Module* has been defined.

At this point, containers enter modules grouped under a green rectangle that make reference to actions regarding the "Lift". Containers encounter modules related to the release of the "Lift". In first place, a *Delay Module* called "Lift Unload Time" simulates the time required for the lift to unload the entities, which is of 1.4s.

In the next module, the lift is ready to be released for starting a new Input Cycle. To carry this operation out correctly, different actions need to be done. In this case, the current position of the lift does not need to be saved, in order to properly calculate travelling times between stations for the next step, because it will always be at the ground floor on the arrivals/departure station. The next step of the lift is a new Input Cycle. Therefore, the proper values to the lift-related variables, such as Lift Input Availability, need to be assigned.

The last module of the group is a *Free Module* that erases the attachment between container and transporter.

The following module is the last module that "Box" encounters. It is a logical *Dispose Module* (from the *Basic Process* template). The action of the module is to delete entities from the system, because they have finished their path.

4.1.3.6. Order Process Station

Description: The "Order Process" Station represents the location where orders enter the system, are treated and then leave the system. The main actions that occur in this station are the creation of

entities, the assignation of “Box” to remove and the disposal of entities.

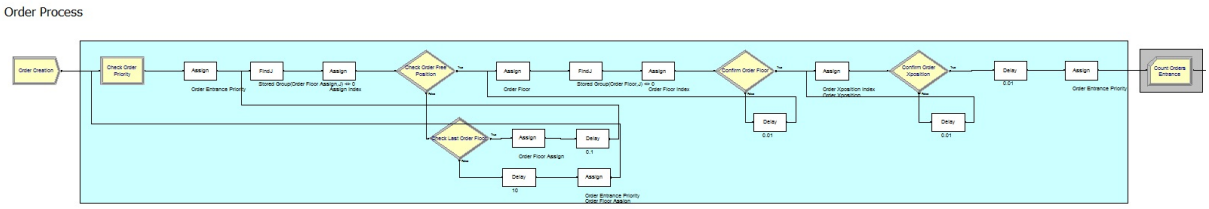

Figure 25: Order Process Station Flowchart (I)

As explained in *Section 2.2.4.1*, in order to represent the unstore process, the concept of orders has been developed. The orders are documents that arrive to the system and determine which “Box” has to be unstored.

Note that the Order Process station is not a real physical station. Unlike all the other stations previously described that were determined by station-related modules and joined with transporters, this station only affect the system logically, that means that its actions will changing variables and parameters.

The arrival of orders to the system is modelled with a *Create Module* named “Order Arrivals”. The created entities, named “Order”, represent the orders that determine the “Box” to be removed and are defined in the data *Entity Module*. Because the focus of the model analysis is the functioning and times of the means of transportation, the rate of arrivals is to some degree irrelevant. Therefore, a group of orders will be created at instant t = “Warm Up” of the simulation and will generate a queue. Note that it has been taken under account that the number of entities created at the beginning will be, in any case, fairly higher than the throughput of the system. Specifically, 10.000 entities will be created for a 10.000 second simulation (as it will be specified at *Section 5*), which guarantees constant supply for the order queue with enough capacity for not running out of orders during the simulation. More information regarding departure of containers is at *Section 2.2.4.1.*

The first operation that a recently created “Order” has to go through is the definition of the “Box” to be unstored. As a matter of fact, what will be defined is the position where the currently stored entity will be removed. This process is defined in the modules grouped under the blue box on *Figure 25*. To solve this, the “Stored Group” array variable plays a key role. “Stored Group”, as it was defined on *Section 4.1.3.3- Position ####*, represents the state of the storing positions. Therefore, only those cells from “Stored Group” with a full state will be permitted for assignment.

Regarding the actions in the modules encompassed under the blue box on *Figure 25*, orders will

enter the blue box one at a time. In first place, they will scan the "Stored Group" matrix to find any stored position. If no stored position is found, meaning that no positions in the system have a container stored that still has not been requested, the current order will be looped to the queue and will trigger a certain delay before another order is allowed to enter the blue box modules, giving time to the system to input at least one container. However, if there is at least one position with a non-requested stored container, random floor and position values will be assigned to the order and, after checking that they is indeed a stored container, unrestricted from the "Stored Group" matrix. The random assignation is implemented with the following expressions representing uniform distributions and the values are assigned to two values from the data *Attribute Module* (from the *Basic Process* template): "Order Floor" and "Order Xposition".

Expression (1):

Order Floor = ANINT(UNIF(1,UBOUNDROW("Stored Group")))

Expression (2):

Order Xposition = ANINT(UNIF(1,UBOUNDCOL("Stored Group")))

Once each order has "Order Floor" and "Order Xposition" successfully defined, the cell value corresponding to that specific position has to be restricted from "Stored Group". This ensures that no other order can have that same container assigned. To restrict the position from "Stored Group", the value corresponding to the "Order Floor" row and to the "Order Xposition" column has to be decreased by 1.

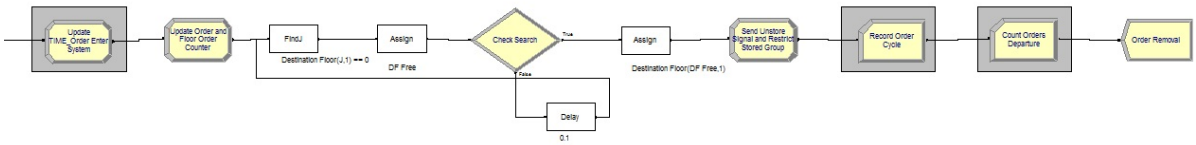

Figure 26: Order Process Station Flowchart (II)

In the next step, the counters to guarantee the Output Sequence will be updated, as specified in *Section 4.1.2.4.* Then, the value of the floor is saved in the output floor sequence list, to allow the correct development of Between Cycles, as seen in *Section 4.1.2.3.* Then, the signal to start the unstoring process has to be sent. To carry this out, "Unstore Signal" has to be updated. Also, the defined position has to be restricted from the "Stored Group" so it cannot be assigned to other entities. This "Stored Group" update is equivalent to affirm that the position is no longer occupied.

At this point, all actions regarding "Order" have been done. Its path is to conclude. Therefore, a *Dispose Module* is defined, where entities are deleted from the system.

4.1.2.7. Lift Operator Station

Description: The "Lift Operator" Station represents the location where the "Lift" is controlled by an external entity in order to carry out a movement with independence of containers. In particular, the action held by the "Lift Operator" Station is the movement of the "Lift" on the Between Cycle.

Note that the "Lift Operator" Station is not a real physical station. Unlike all the other stations previously described that were determined by station-related modules and joined with transporters, this station only affect the system logically, that means that its actions will changing variables and parameters.

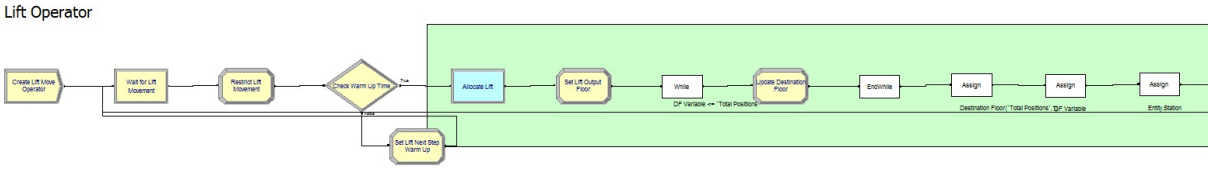

Figure 27: Lift Operator Station Flowchart (I)

To externally control the "Lift", only one entity is needed. This entity is created with a *Create Module* (from the *Basic Process* template). The entity will remain at a *Hold Module* at the beginning of the flowchart until the Between Cycle starts. At this point, the entity will receive the signal and advance throughout the whole flowchart, undergoing the Between Cycle pertinent actions, and will be looped to the *Hold Module* again when finished, awaiting for the start of a new Between Cycle. If the system is still on the Warm Up period, the entity will be directly looped to the *Hold Module* but assigning the proper values to the lift-related variables.

In first place, the entity will take control of the "Lift" and will read the list of output floors regarding the output sequence, as seen in *Section 2.2.4.5*. It will save the current value, and then update the list, deleting the value taken and making room for the next order.

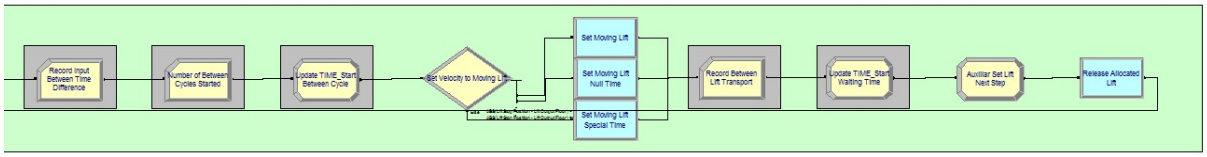

Figure 28: Lift Operator Station Flowchart (II)

The “Lift” will then be moved to the corresponding floor and the relation between entity and “Lift” will be erased. The lift-related variables will be then set to the proper values to allow the Between Cycle to continue properly, so the logic of the “Lift” is respected. Finally, the entity will be looped to the *Hold Module*, where it will await the new Between Cycle.

4.1.3.8. Shuttle Operator Station

Description: The “Shuttle Operator” Station represents the location where the “Shuttle” is controlled by an external entity in order to carry out a movement with independence of containers. In particular, the action held by the “Shuttle Operator” Station is the movement of the “Shuttle” to the input buffer. There is on “Shuttle Operator” Station per each shuttle, which means that there are as many “Shuttle Operator” Stations as floors have the system.

Note that the “Shuttle Operator” Station is not a real physical station. Unlike all the other stations previously described that were determined by station-related modules and joined with transporters, this station only affect the system logically, that means that its actions will changing variables and parameters.

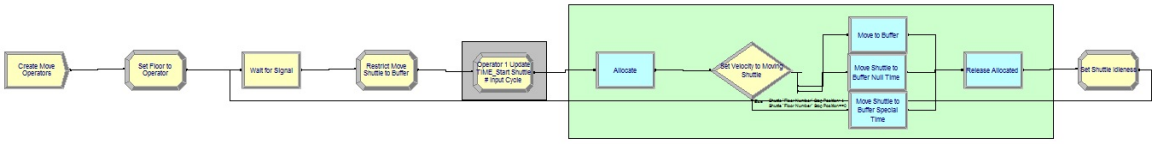

Figure 29: Shuttle Operator Station Flowchart (I)

To externally control the “Shuttle”, only one entity is needed. This entity is created with a *Create Module* (from the *Basic Process* template). After the creation, the entity will have the “Floor” attribute assigned, to associate each station to one specific shuttle. The entity will remain at a *Hold Module* at the beginning of the flowchart until the entity receives the respective signal and advance throughout the whole flowchart. The signal is sent to the “Shuttle Operator” Station when the “Shuttle” is starting an Input Cycle, and will be looped to the *Hold Module* again when the corresponding actions are finished.

In first place, the entity will take control of the “Shuttle”. It will move it to the input buffer, and then it will release it. The shuttle-related variables will then be updated, to allow the proper development of the Input Cycle of the shuttle.

4.1.3.9. Animation

The animation is an important feature of the system. As a matter of fact, it is one of the most relevant differential features of ARENA Simulation Software, when comparing it to other types of simulation.

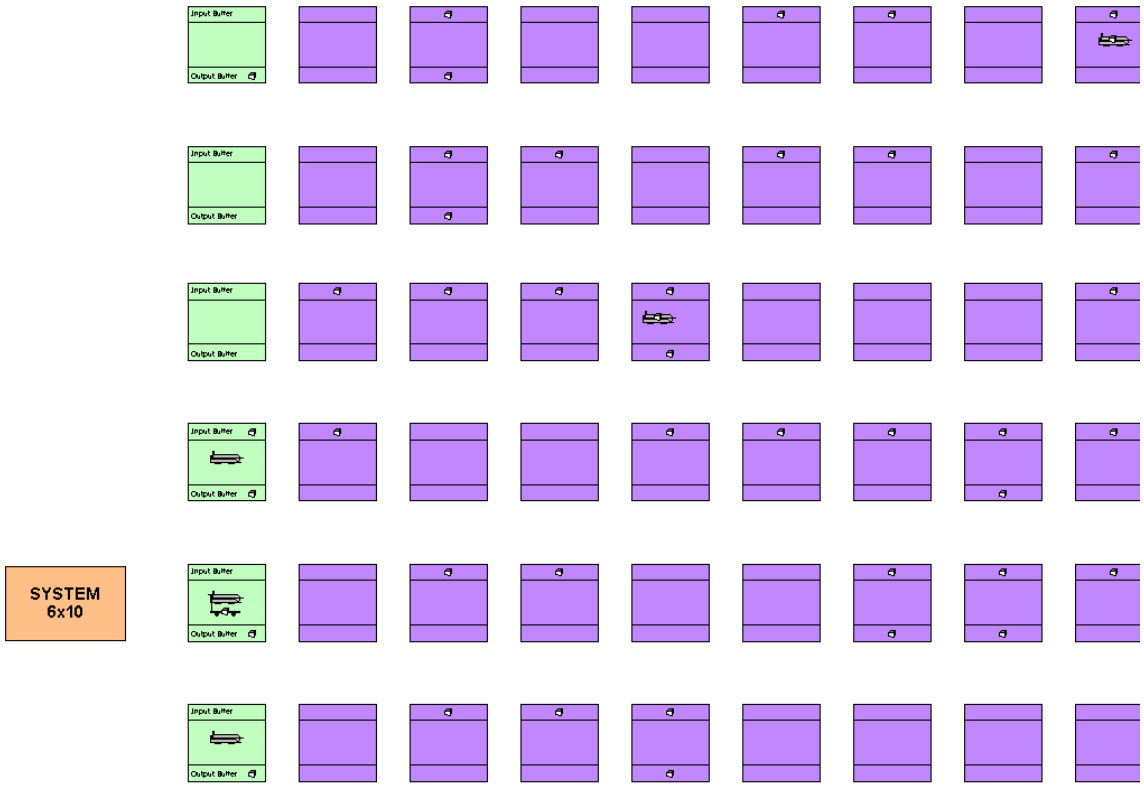

Figure 30: Model Animation

4.2. Intermediate Model

As previously explained, the difference between the Initial Model and the Intermediate Model does not contribute to any model's intelligence modification. The Intermediate Model represents the same low-scale version of the Shuttle Storage System that the Initial Model represented, but with a very crucial modification: while the Initial Model was created using low hierarchy functions, the purpose of the Intermediate Model is to create very high hierarchy functions that enclose parts of the model. This step is necessary to simplify the model before enlarging it to more realistic sizes and then simulate and analyse relevant results.

4.2.1. Prior ARENA Simulation Software Concepts

In ARENA Simulation Software, different modules are grouped in different panels. The Blocks and Elements panels enclose the modules with the lowest hierarchy level, which represent SIMAN language functions. All the other panels include modules that are configured with combinations of SIMAN language functions. As already discussed, high hierarchy modules may grant more user-specified needs and low hierarchy modules give more modelling flexibility to the user.

Although all those panels come attached with the software, ARENA Simulation Software gives the chance of configuring own modules and creating templates to the user. Therefore, in the current project, it is very useful to develop modules that include parts of the system.

There is one main reason for developing the model this way. ARENA Simulation Software is an entity-based simulation software, which means that entities go throughout the flowcharts undergoing all actions. For example, as explained in *Section 4.1*, the station where containers are deployed, stored up and then requested and picked up again is modelled in one flowchart. However, there must be as many of those flowcharts as position stations. In the Initial Model case, there are 2 floors with 8 positions per floor, meaning that there are 16 storing positions and, consequently, 16 flowcharts. Notice that, for the modules that have been used in creating the Initial Model, a name for each module is required. This name must be unique in the whole module.

Say, for instance, that each position flowchart is composed of 32 different modules. This flowchart needs to be implemented 16 times, and each module needs to be configured each time with, at least, a different name for the module. As it can be deducted, the development of the Initial Model was very time-consuming. Imagine then, trying to build a model composed of 20 floors and 50 positions per floor. It would be unviable.

Thanks to the Template Development Dialog, ARENA Simulation Software provides a very easy and user-friendly group of dialogs for successfully creating and development panels and modules. For the current project, only three dialogs will be pointed out.

In first place, the Template Logic Window is where the actions of the module are defined. To explain it in few words, it acts like a normal panel window, where the required flowchart has to be implemented.

In second place, the Template Animation Window is where the visual representation of the module is defined. This means, that when an instance of the defined module is inserted in a model, the appearance of the module and its corresponding animation can be set here.

Finally, the Template User Interface has to be created. When an instance of the module is inserted on a model, some parameters regarding that specific module need to be defined. The values of parameters affect and condition variables, attributes and expressions of the configured module. As a matter of fact, those parameters are names Operands. Operands are values defined at the module when it is inserted in a model, and this value is linked to variables and expressions inside the logic configuration of the module that has been defined.

4.2.2. Intermediate Model Construction

4.2.2.1. Position Panel

During the construction of the Intermediate Model, the Position panel has been implemented. There will only be one module under this panel, also named Position module. The position module will encompass all the modules of *Section 4.1.3.3*. Therefore, all the actions involved in the position station (container deployed by the shuttle, stored and picked up again by the shuttle) will be configured under only one module.

The operands implemented for the proper functioning are the number of the floor in which the position is allocated and the exact position number. Those operands allow to properly allocate and release the correct shuttle and to calculate the distances between different stations, basically.

The animation for this module is quite simple, because it is composed of a purple square that embraces the storing positions (or the place where the entity will be animated during the storage period) and the logical point of the station, that will be linked as a part of the path of the shuttle transporter.

4.2.2.2. Floor Panel

Under the Floor panel, only one module is implemented. This module is named “Floor 8 Pos”, because it consists of a floor that has eight storing positions in it. This module will encompass all modules in *Section 4.1.3.2*, *Section 4.1.3.4* and *Section 4.1.3.8,* as well as eight instances of the “Position” module from the Position panel, described in *Section 4.2.2.1.* Therefore, all actions involved in the “Floor 8 Pos” (all actions between the deployment by the lift of a container to the input buffer until the picking up the container by the lift at the output buffer) will be configured using only this module.

The only operand needed for ensuring the correct functioning of this module is the specific floor number that the module is going to represent.

Regarding the visual animation of the module, all the animation features of the Position module previously explained will be linked together as the path of the shuttle. Furthermore, another square will be added at the beginning representing the "Floor Transfer" Station, embracing the animations for both input and output buffers and defining the logical point of the station, that will be linked as a part of the path of the lift. Consequently, the visual animation represents a whole floor and the entire shuttle path, with the open link point where the lift path will be connected.

4.2.2.3. System Panel

Under the System panel, also only one module is implemented. This module is named "System 2x8", because it consists of two floors with eight storing positions at each floor. This module will encompass all modules in *Section 4.1.3.1, Section 4.1.3.5, Section 4.1.3.6* and *Section 4.1.3.7,* as well as two instances of the "Floor 8 Pos" module from the Floor Template, described in *Section 4.2.2.2*. Therefore, inside the "System 2x8" module, all the actions of the system happen.

The module "System 2x8" is the only module that has to be inserted in a model to make proper simulations. **The simulations are exactly the same as the Initial Model simulations, but with the difference that all the modules have been encompassed under a new high hierarchy module**.

The "Position" module and the "Floor 8 Pos" module (from the Position panel and the Floor panel respectively) do not have to be inserted in the model where the simulation is going to be ran, because there are inserted inside the logic of the "System 2x8" module.

The only operands needed in this module are the total size of the system (number of floors, number of positions per floor and total number of storing places), as well as the Warm-Up time (see *Section 4.1.2.1*). As a matter of fact, those values are also linked to the other templates with separate operands, whose function is not as essential as the operands previously defined but are required to maintain the sizes of the system.

The visual animation that goes with the module is, in this case, composed of the visual animation of both "Floor 8 Pos" modules, united with the path of the lift. Also, the logical station points of the arrivals/departure station are allocated at the same place as the first floor.

4.3 Final Model

As previously explained, the Final Model is the enlarged version of the Intermediate Model. It is easily developed thanks to the simplification granted by the very high hierarchy functions created on the Intermediate Model. The size of the Final Model is not to be fixed, because it is a very important

variable for the different simulations and analyses of the model.

4.3.1. Final Model Construction

For the construction and development of the Final Model, the starting point will be the panels and modules developed during the construction of the Intermediate Model on *Section 4.2.* So, all the explanations for this point will refer to modifications or additions to those modules.

4.3.1.1. Position Panel

The logical part of the "Position" module, in other words the flowchart that configures the logic of the module, remains exactly the same. Note that this module is the lowest module in the scale on the Intermediate Model construction, meaning that is place inside the Floor Panel modules that are placed inside the System Panel modules.

However, although enlarging the size of the model does not affect the Position module itself, it does indeed affect the shuttle transporter that travels between positions. The only modification that suffers the Position module is to expand the Distance matrix regarding the shuttle travelling times.

4.3.1.2. Floor Panel

For the expansion of the model, the Floor Panel is going to be greatly modified. In the Floor Panel modules is where the length variable is implemented. To do so, different modules with different lengths will be implemented, so they can be used with any height configuration indistinctly. Therefore, under the Floor Panel, modules like "Floor 20 Pos" or "Floor 40 Pos" can be found.

Therefore, although most of the flowcharts remain the same, the most important change takes place in the number of "Position" modules inserted. On *Figure 31*, an overall view of the Template Logic Window can be seen, with the different flowcharts previously explained and underneath the different "Position" modules to generate a floor with 30 positions.

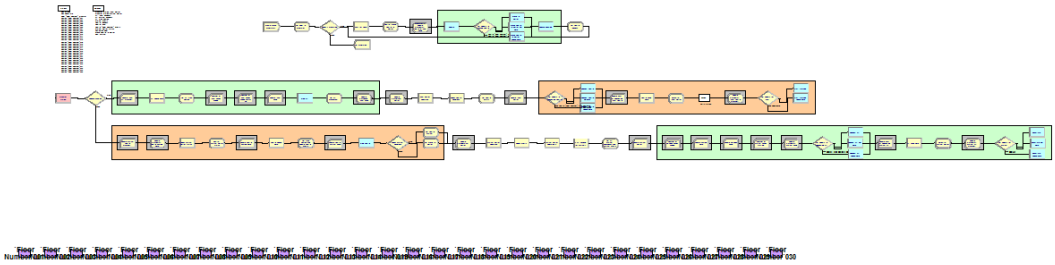

Figure 31: Template Logic Window of modules from Floor Panel

The two other important changes in the logic of the Floor templates is that the Distance matrixes of both transporters need to be expanded to meet all the travelling conditions depending on the size.

4.3.1.3. System Panel

For the expansion of the model, the System Panel is also going to be greatly modified. In the System Panel modules is where the height variable is implemented. To do so, different modules with different heights will be implemented, and will take all the length configurations available at the Floor Panel. Therefore, under the System Panel, modules like "System 12x20", "System 12x40" or "System 20x40" can be found.

Therefore, although most of the flowcharts remain the same, the most important change takes place in the number and type of "Floor" modules inserted. On *Figure 32*, an overall view of the Template Logic Window can be seen, with the different flowcharts previously explained and underneath the different "Floor" modules to generate a system of four floors.

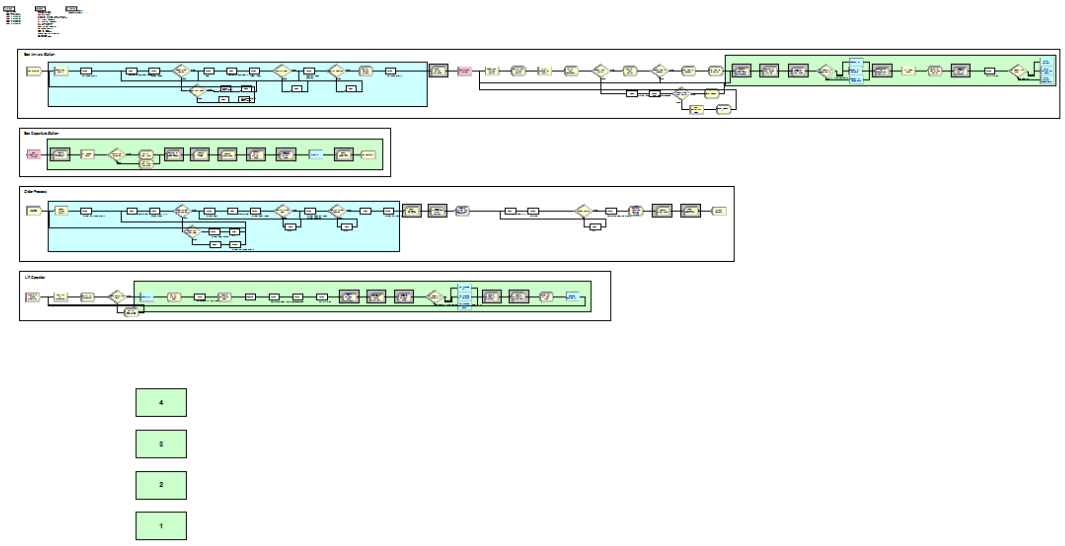

Figure 32: Template Logic Window of modules from System Panel

The only other modification that suffers the System module is to expand the Distance matrix regarding the lift travelling times.

Finally, the User Interface[7] of the module, which the user will have to configure before running a simulation, is shown in *Figure 33*. This is the User Interface of the System Panels modules, which is

[7] Shuttle Output Parameter will be discussed in *Section 6*.

the only that has to be configured by the user. The System Dimensions' Operand is referred to the dimensions of the system, and beneath the Warm Up time has to be implemented.

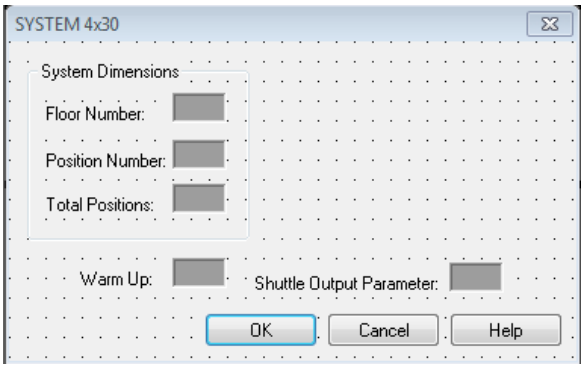

Figure 33: User Interface Window

As said, the fact the this is the only User Interface used is because it belongs to the only module that needs to be inserted in a model window to run simulations of the full system. The other pre-defined modules are inserted inside this module. Therefore, only the System Panel needs to be personally attached to ARENA Simulation Software, where all the different available system modules will be found, as shown in *Figure 33*.

5. Simulation analysis and results

5.1 Simulation Parameters

Simulating the system in an appropriate way will give the necessary results to extract information from the simulations that will be helpful to understand the system. Then, to simulate the system appropriately, two different branches need to be taken under account.

In first place, proper data recorders have been implemented in the model during its construction. Those data recorders will provide the different time intervals during the simulation and entity counters. The implementation of those recorders was done while thinking about the analysis and results section, so the results would be appropriate.

In second place, the different runs of simulations with different variables have to be done under the same simulation parameters, in order to obtain the correct results for the study. If the simulation parameters were changed, the simulation results would be different and not consistent. To set the simulation parameters correctly, the Dialog Window in *Image 5* has to be defined. This Dialog Window is allocated in the Run Menu from ARENA Simulation Software.

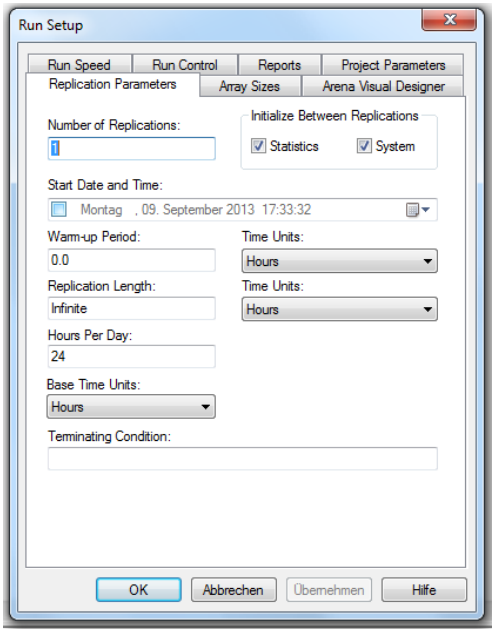

Image 5: Run Setup Dialog Window

Number of Replications:	**4**
Replication Length:	10.000
Time Units:	Seconds
Base Time Units:	Seconds

Table 5: Simulation parameters

The values not specified on *Table 5* will remain as the default values set by the software.

Number of Replications is the parameter that makes the software simulate the same model, with the same parameters, but changing the randomness. This means that all the numbers that have been set

randomly, like the uniform distributions that set the storing positions, will be different in each simulation. The results will be therefore different, but should be close. In order to obtain a better approximate result for the simulations of a model, 4 replications will be made and only the average of the replications will be used. Although a higher number of replications means a more accurate average value of the results, four replications will grant the security of working on enough accurate results.

Replication Length and Time Units make reference to the run time of the simulation. In other words, how long will the simulation last. This parameter is set to 10.000 seconds. This values is high enough to ensure the steady state of functioning, encloses the warm up period explained in *Section 4.1.2.1.* Note that both lift and shuttles have high-speed values, 5^{m}/s and 2^{m}/s respectively. The high values of the velocity turn into not many seconds the transporting of containers in and out of the system.

Base Time Units parameter sets the units with which time will be measured. This parameter does not alter the system logically, but the most convenient units should be picked up. Note that both lift and shuttles have high-speed values, 5^{m}/s and 2^{m}/s respectively. The high values of the velocity turn into not many seconds the transporting of containers in and out of the system. Therefore, seconds were chosen as Base Time Units.

5.2. Analysis of the results

The model, whose logical features were described in *Section 2.2*, construction specifications were explained in *Section 4* and simulation parameters listed in *Section 5*, usually reports a default set of results after a simulation. Those reports can be personalized depending on the needs or purpose of the user.

For the current study, personalized results are reported. Those results focus on all the time intervals of the working entities (for cycle times) and on counting the number of entities (for throughput).

5.2.1. Lift cycle times

The nature of the system relies on the nature of the transporters. The transporters are in charge of taking containers in and out of the system. Although there is one shuttle per each floor, there is only one lift in charge of bringing in and taking out containers alternatively. Furthermore, **there will be always containers to bring in and to take out of the system, generating in the lift the bottleneck of the system and ensuring constant functioning of the lift**.

However, although the lift will always have entities to bring in or take out, there are time intervals where the lift will have to wait for certain conditions of the system to be met. As described in *Section 2.2.4.2*, the double cycle of the lift is composed of the following times shown in *Figure 34*.

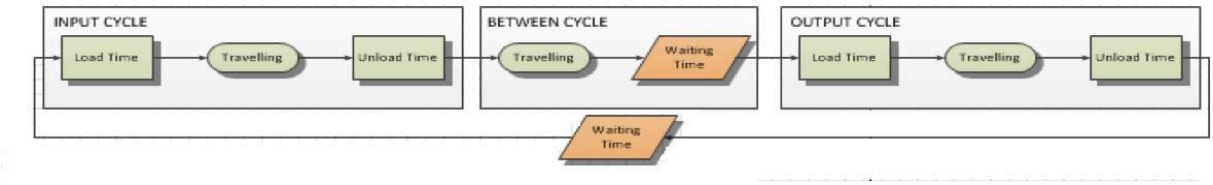

Figure 34: Double cycle of the lift

Regarding the simulation results, the values of the different separate times of the global double cycle in *Figure 34* are analysed as the average of all the recorded values of that time at each double cycle of the lift. Furthermore, remark that four replications of each model are going to be simulated. The values of the times are, then, the average value of the averaged values of the four replications.

From the different separated times of the global double cycle of the lift, three different kinds can be separated. In first place, there are four times with constant value: the load and unload times of the lift. In second place, the travelling times of the lift need to be distinguished. Those values have different maximum values depending on the number of floors. However, thanks to the randomness of the system and the long run time that ensures steady state functioning, the travelling times tend to stabilize at the medium value. Those values are, then, only conditioned and determined by the number of floors of the system. In third place, there are two waiting times in the double cycle of the lift. Those waiting times are, a priori, independent of the system. On one hand, there is the Between Cycle waiting time, which is waiting for an output container still not on the output buffer. On the other hand, in case that there are no available input buffers when an Input Cycle has to start, the lift will wait. This second waiting time is, however, not included as part of the global double cycle of the lift. Therefore, there is going to be special focus on those unpredictable waiting times.

As previously explained, the size of the model to simulate will be considered as a variable parameter of the simulation. The size is composed of the height and the length. To start analysing the model, a group of models with constant height and a group of models with constant length have been simulated. The constant values chosen are a height of 6 meters (20 floors) and a length of 20 meters (40 positions). With these simulations, the global behaviour of the system, as well as the waiting times, will be analysed.

5.2.1.1. Results at constant height

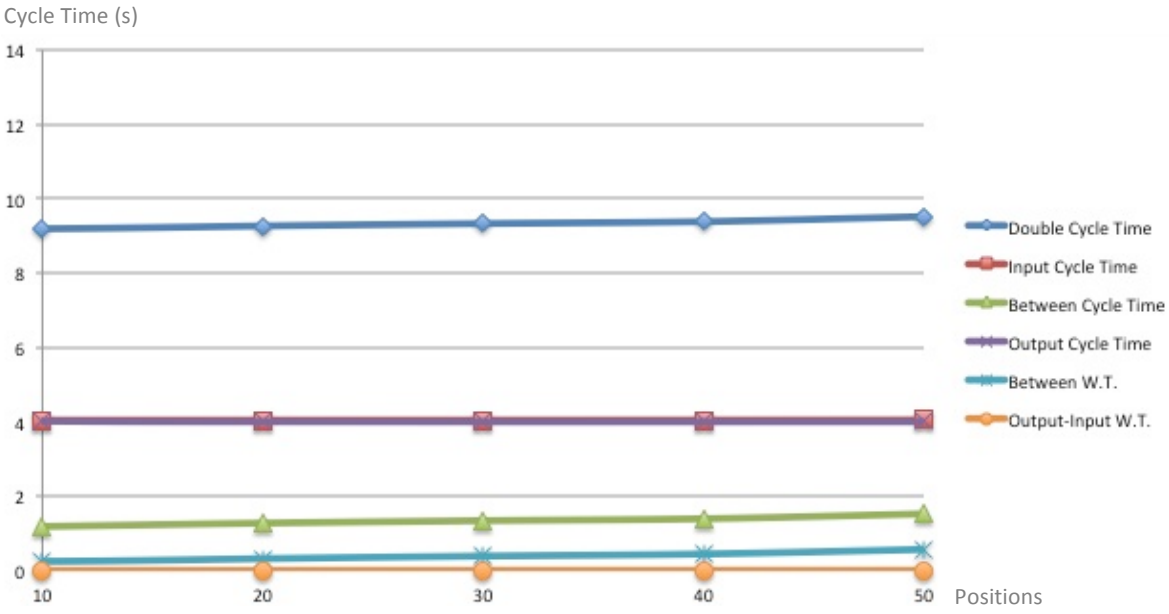

Figure 35: Double-Cycle times at a constant height of 20 floors

In first place, as seen in *Figure 35*, remark that the Input Cycle and Output Cycle of the lift times are stable around 4 seconds. This is because all the values are constant: the loading and unloading time are constant values and the travelling time is stable because the number of floors is fixed.

$$time_{input} = time_{load} + time_{travel} + time_{unload} = 1{,}4s + time_{travel} + 1{,}4s$$

$$time_{output} = time_{load} + time_{travel} + time_{unload} = 1{,}4s + time_{travel} + 1{,}4s$$

The different possible travelling times of the lift are:

DESTINATION	DISTANCE (mm)	TIME (s)
1	0	0,0000
2	300	0,4140
3	600	0,5855
4	900	0,7171
5	1200	0,8281
6	1500	0,9258
7	1800	1,0142
8	2100	1,0954
9	2400	1,1711
10	2700	1,2421
11	3000	1,3093
12	3300	1,3732
13	3600	1,4256

14	3900	1,4856
15	4200	1,5456
16	4500	1,6056
17	4800	1,6656
18	5100	1,7256
19	5400	1,7856
20	5700	1,8456

Table 6: Lift possible travelling times

Considering that, thanks to the randomness of the cycle and the reach of the steady state of functioning, the lift will travel to all the floors indistinctly, the travelling time will tend to be the average of all the travelling times.

$$time_{travel} = \frac{\sum_{i} time_i}{floors} = 1{,}2757s$$

$$time_{input} = 1{,}4s + time_{travel} + 1{,}4s = 2{,}8s + 1{,}2757s = 4{,}0757s$$

In next place, the Between Cycle suffers a soft increase parallel to the soft increase of the Between Cycle waiting time. Notice that the waiting time increases with the length of the rack, which means that shuttles need more time to travel all the distance. Notice that the travelling time is stabilized as it happens in the input and Output Cycles.

Finally, with a size rack of 6 meters (20 floors), there is never waiting time between double cycles. There are always enough available input buffers, so the flow of input containers never has to stop.

Therefore, without any waiting time between double cycles, the global cycle time is only affected by the increase of the Between Cycle waiting time, as reflected in the parallelism of the graphs. All the other values are either constant or stabilized.

In the following figures, there will be represented the behaviour of the important times of different constant heights simultaneously.

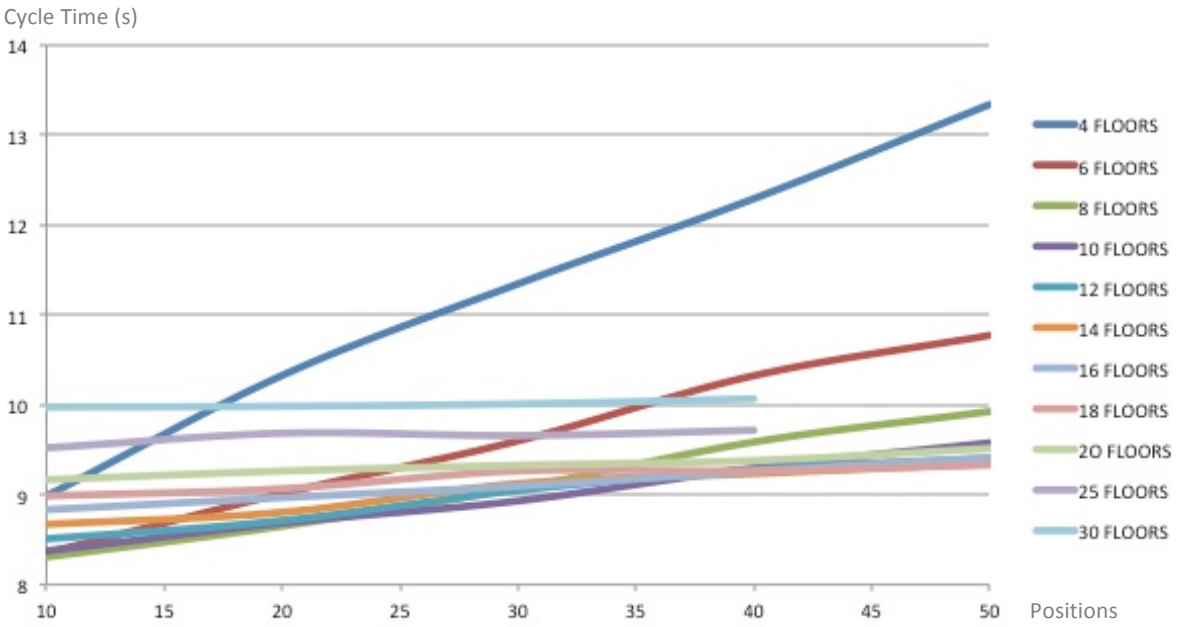

Figure 36: Double-Cycle times at constant height

In first place, in order to give consistency to the initial supposition, remark that all values of waiting times between cycles are null for all the simulated heights except for the simulation with only four floors. In that case, the waiting time between double cycles suffers a slight increase as the length also increases (because shuttles require higher travel times). However, the value is undoubtedly negligible, because with the highest length of 50 positions (25m), the value is approximately 0,01s.

As it can be seen in *Figure 36*, the behaviour of the double cycle time of the lift increases with the increment of the length of the rack. Because theoretically the travel time of the lift to not vary (because the height is constant), the increment can only be due to the increment of Between Cycle's waiting time. As seen in *Figure 36*, the curves of the Between Cycle waiting time resemble the curves of the cycle time.

Note also that the increments are more pronounced with a smaller number of floors. This has a reasonable explanation: if there are fewer floors, the lift will visit the output buffers more often, because the range of choice is smaller. If this visit rate is too high, the shuttles might not have enough time to continuously supply the output buffer in time, causing the Between Cycle waiting time. Furthermore, this time will be increased if the length of the rack is longer, because the travelling times of the shuttle will be higher.

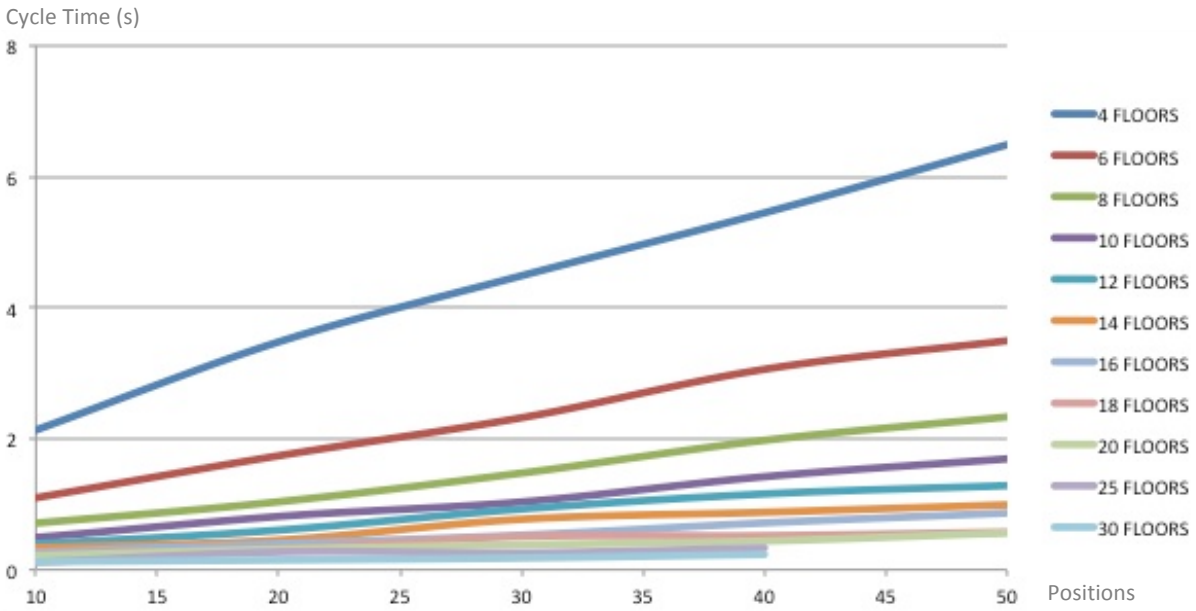

Figure 37: Between Cycle waiting time at constant height

To conclude this study, the following statement has been resolved. **The times of the double cycle of the lift in a system with constant height will increase with the increment of the length of the rack**. This increment is, however, direct consequence of the increment of the Between Cycle waiting time, because shuttles require more time to transport containers to the output buffer.

5.2.1.2. Results at constant length

This study is complimentary to the study with constant height, and the main purpose is to define better the behaviour of both waiting times (in the Between Cycle and between the input and the Output Cycle).

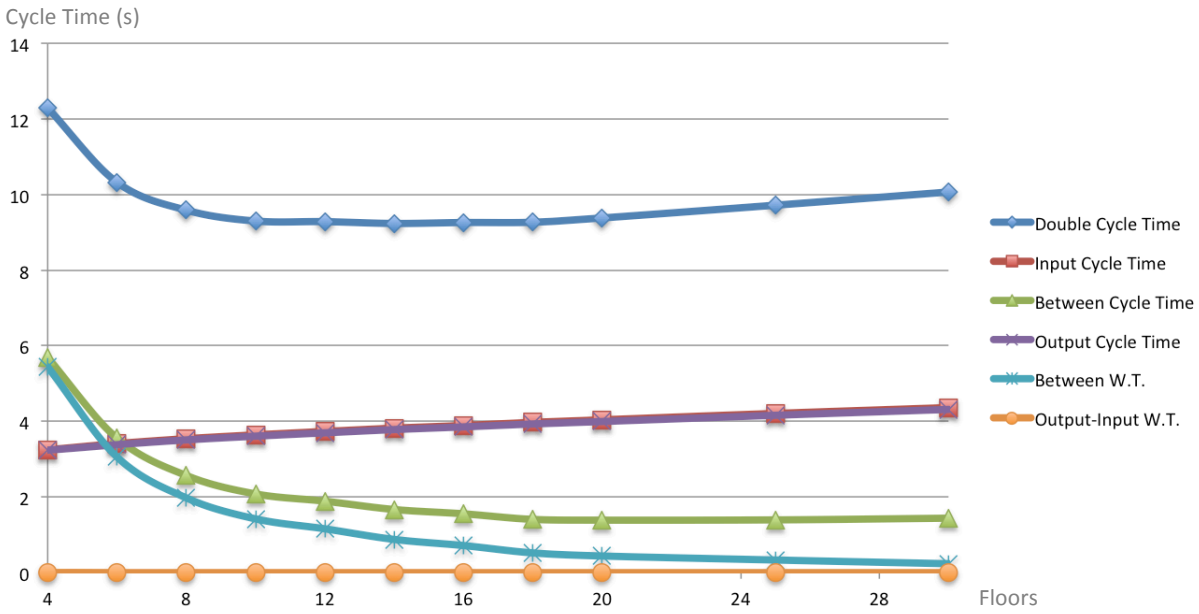

Figure 38: Double-Cycle times at a constant length of 40 positions

In first place, both input and Output Cycles follow identical paths, starting around 3 seconds due to the small number of floors and the 2,8 seconds of loading/unloading times and marked by a regular increment generated because of the increment of floors and, consequently, the increment of travelling time. Obviously, more time is needed by the lift to travel when the system is higher.

In next place, remark the curve of the Between Cycle waiting time. Notice how the curve decreases with the increment of floors. The reason is that, on constant length, the higher number of floors means less probability for the lift to go to a floor, granting the floor more time to fill the input buffer. The Between Cycle curve resembles the Between Cycle waiting time curve, but it slowly gets distanced by the increment of travel time due to the increment of floors.

The waiting time between double cycles is negligible because it is null for all floors except for the model with four floors, where its value is lower that 0,01 seconds. Therefore, the global cycle curve resembles the sum of the Input, Output and Between cycle curves. In the double cycle time curve, two different patterns can be distinguished. Until a certain number of floors, the decrease of the Between Cycle waiting time is shown. However, once the waiting time becomes smaller, the double cycle curve tends to reflect the regular line that supposes the increment of travelling times of the lift (due to the increment of floors).

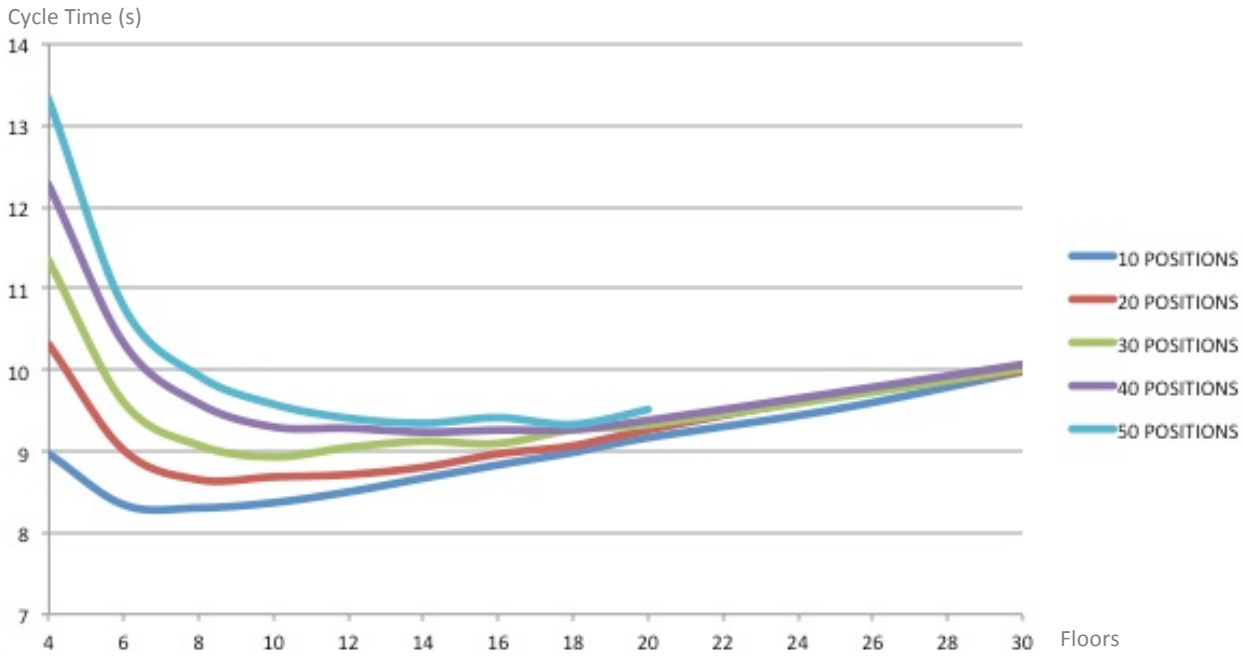

Figure 39: Double-Cycle times at constant length

Comparing the double cycle time curves at constant length with different length, the patterns are very similar and converge to a regular increasing line. This increasing line reflects the sum of the Input Cycle time, the Output Cycle time and Between Cycle time. Those three times are, with a high number of floors, only dependant on the loading/unloading times and travelling times (which obviously increase with the increment of floors).

The initial part of the curves is strongly affected by the Between Cycle time. The waiting time of the Between Cycle, as it can be seen in *Figure 39,* suffers a great decrease as the number of floors increase. The reasons, as said previously, is because if there are more floors, the frequency rate that the lift travels to an output buffer is reduced, granting in general more time to the shuttles to bring an output container to the output buffer.

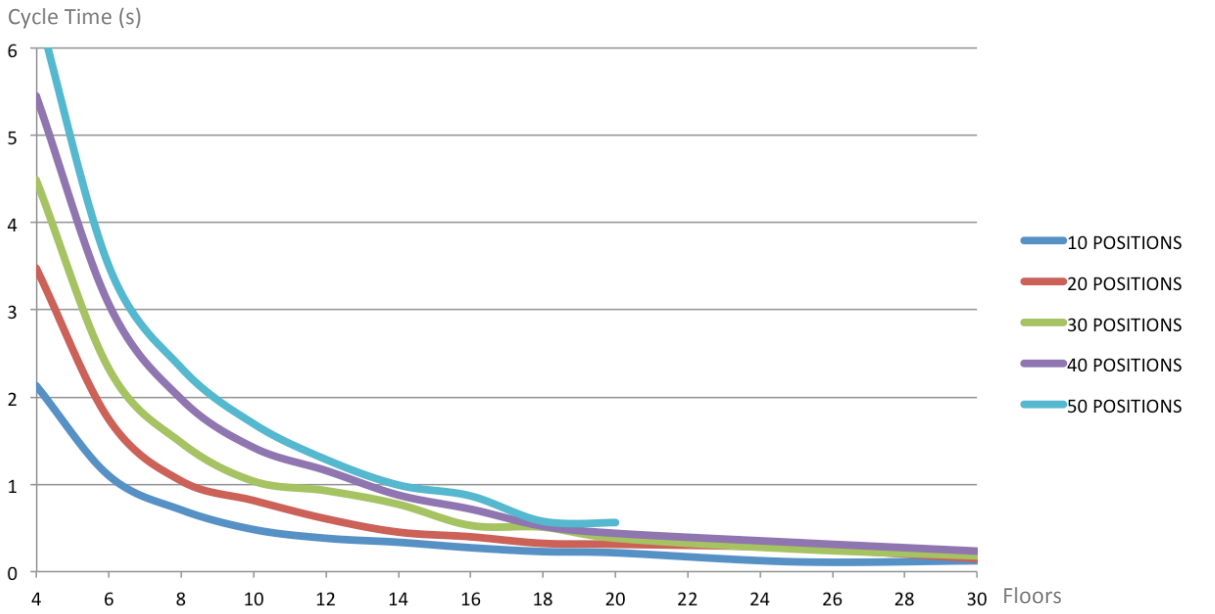

Figure 40: Between Cycle waiting time at constant length

5.2.2. Throughput

Once the cycle times of the lift have been studied, there is not much to discuss about the throughput. Because the studies are related to the average values of the whole simulation run, it is logical that, without waiting times between input and Output Cycles, the throughput will be a mere inverse parallelism to the double cycles. However, it is important to have the values of throughtput of the rack system, because it will be the comparison value to systems with other lift configurations.

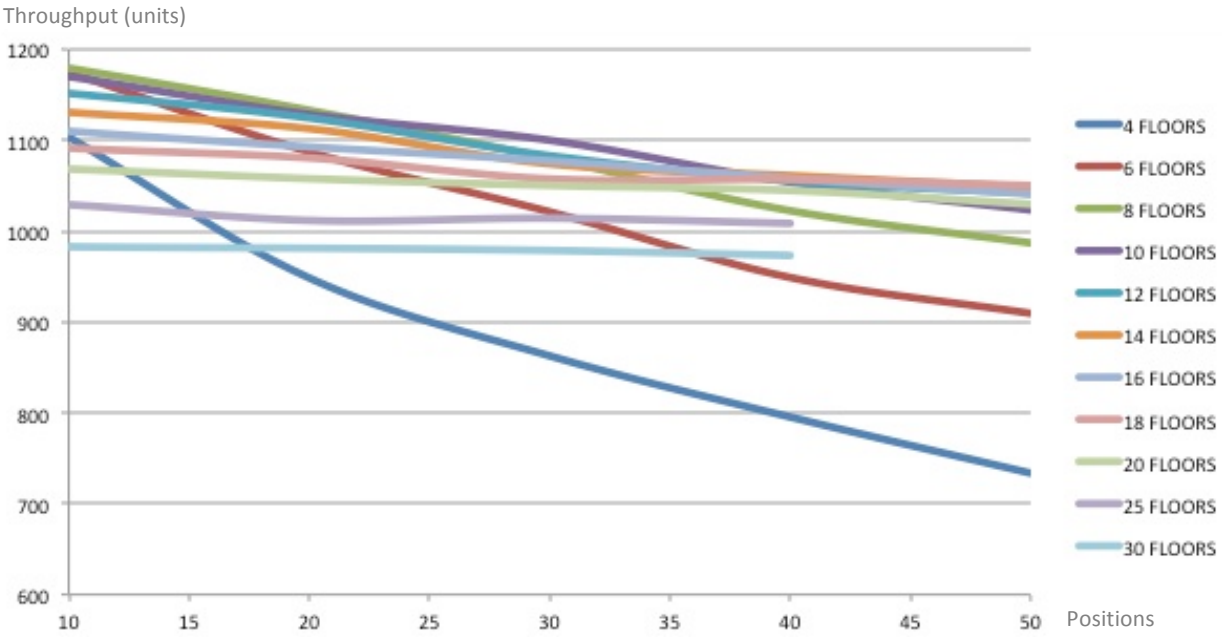

Figure 41: Throughput at constant height

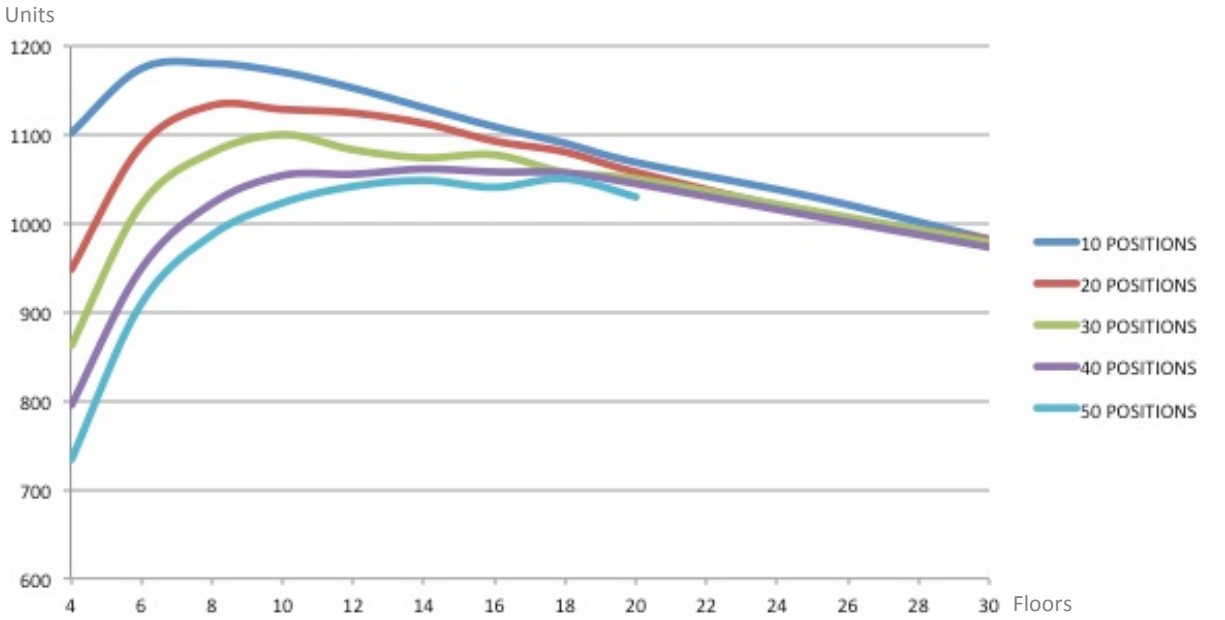

Figure 42: Throughput at constant length

5.3 Comparison with MATLAB

Thanks to the project developed by Herr Lukas Fleischer, in which the same Shuttle Storage System has been modelled with MATLAB, a comparison between results from a numerical simulation (MATLAB) and a graphical simulation (ARENA Simulation Software) can be done.

MATLAB and ARENA Simulation Software are two totally different softwares and are designed for different purposes. Whereas ARENA Simulation Software is a discrete process simulator, MATLAB is intended for numerical computing (matrix manipulation, plotting functions and data, implementation of algorithms…).

Regarding the adaptation of those softwares to modelling a Shuttle Storage System, ARENA Simulation Software, as a process simulator, provides an environment prepared for its proper development. However, MATLAB requires a total user-defined development of such an environment, which adapts the powerful numerical tools of the software to be turned into a conglomerate of functions and algorithms that allow the simulation. In return, if done successfully, all the powerful tools of MATLAB can be used to upgrade, complement and present the model simulator, granting a much more flexible and sophisticated simulation configuration. This allows, for example, running many simulations varying different parameters, while saving and plotting its results, all under one single simulation configuration. With ARENA, however, simulations are normally done one-by-one, slowing down the process of experimenting with different parameter values.

Although both models have been developed concurrently under the KNAPP company parameters, there are some differences on the functioning, especially regarding the logic of the transporters, that need to be pointed out. For example, the double cycle division of times. As explained in this project, the Between Cycle includes the travelling to the corresponding floor and the waiting time. In the model developed using MATLAB, however, the Between Cycle also includes the loading time of the output container. Anyway, this is just to point out the fact that models developed by different students may have undergone different developing steps, because this particular change does not affect the final results in terms of double cycle time and throughput.

There are, however, some major differences regarding the logic that do affect the final results and need to be clarified. In first place, in this project and following the strict functioning of the real KNAPP Shuttle Storage Systems, the lift is not allowed to travel to a floor with a full input buffer. However, in the model developed with MATLAB, this restriction is not implemented. The lift is allowed to travel to any floor, independently of the input buffer's state. Therefore, a new waiting time is defined, which reflects the time where the lift waits with an input container until the input buffer is emptied and ready for the deployment. Also, note that in this case special considerations are also made to avoid the deadlock situation, but this is done by scanning the status of the system (specifically of the desired floor) before start travelling. The shuttle logic then is set to prioritize emptying the input buffer first, which means prioritize input.

Also, there is a feature included in the MATLAB model not developed in the ARENA Simulation Software model. The MATLAB system allows a pre-defined number of input containers to activate and prepare their corresponding floor shuttles so they are ready when the input containers arrive at the input buffer. This is a time saving measure, because it looks into the future of the system's global state. In ARENA's model, only that box that is going to start an Input Cycle is allowed to gain control and priority on its floor shuttle.

Those measures affect differently the waiting times of the lift. As previously seen, the waiting times affect greatly the double cycle of the lift when racks have small dimensions (basically a low number of floors), but tend to stabilize and converge when the system has a fairly high height and it is balanced.

Based on an initial result comparison between the MATLAB model and the ARENA model, the graphs of the results follow similar patterns. There is a relative difference of time at the initial part of the graph, with a low value of the height, but **results from both models tend to converge to the same values at normal rack heights**. This is demonstrated at *Figure 43* and *Figure 44*.

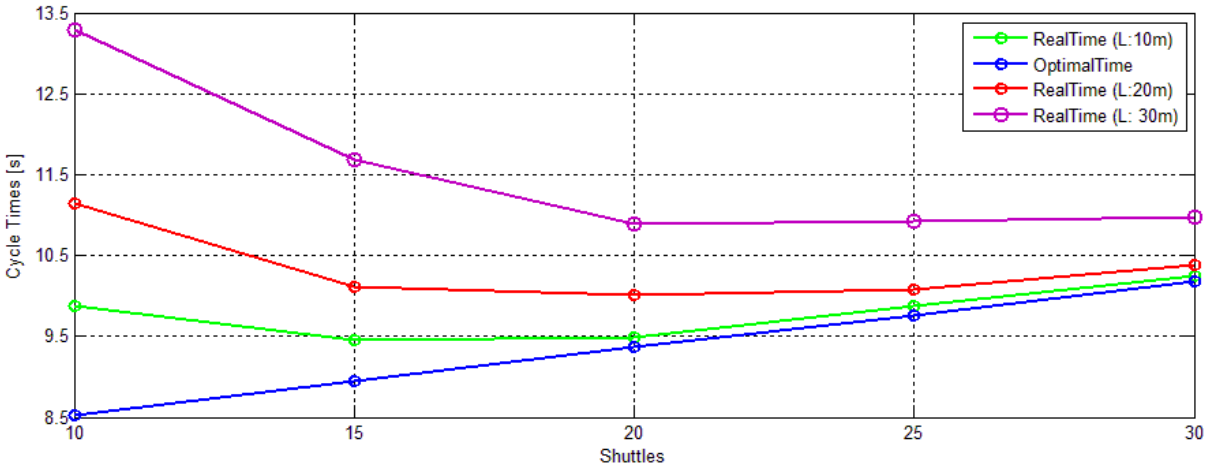

Figure 43: MATLAB simulation results. Double-Cycle times at constant length

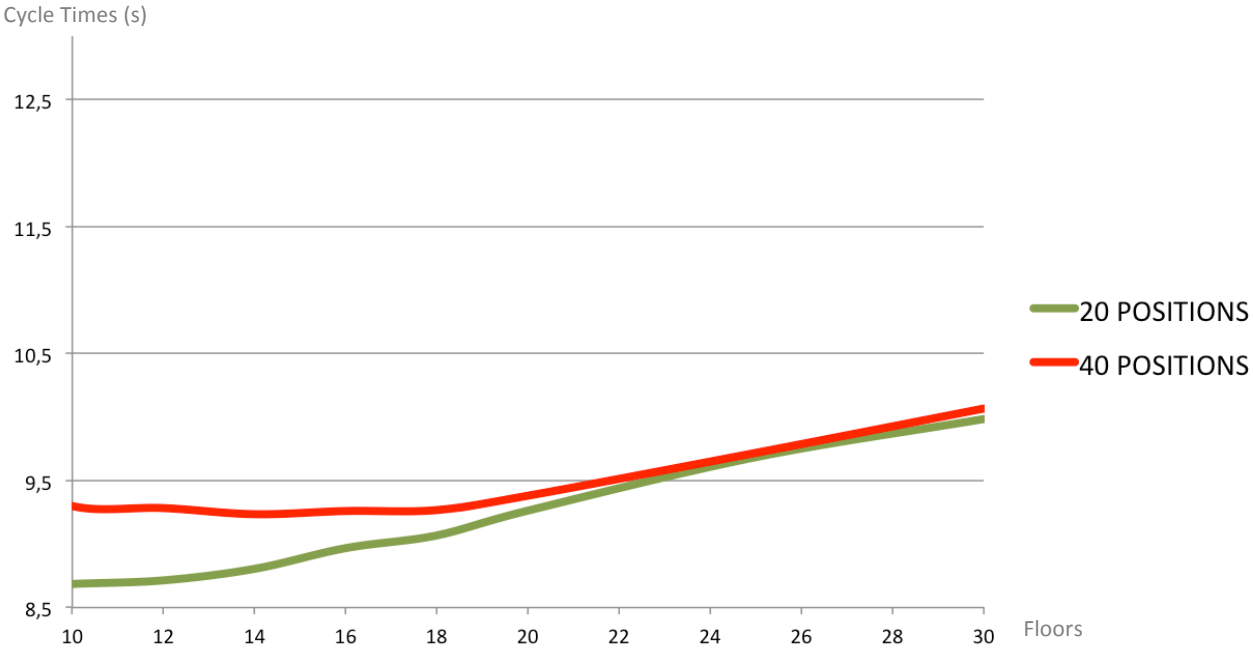

Figure 44: ARENA simulation results. Double-Cycle times at constant length

In the figures, the plots of a system of 20 positions and 40 positions with variable height can be compared. Note that the initial values are relatively lower at the model with ARENA, but both graphs tend to converge to a regular increasing line with similar slope. At this line, waiting times have lost its importance and the double cycle times are mainly only affected by the increase of travelling times.

The fact that the initial values of both models are different is because the different logical configurations. The fact that the values compared are not referred to a small sized or an unbalanced rack grant that in ARENA there will not be waiting times between cycles of the lift (as demonstrated

in the previous results). This is a major asset in comparison to the MATLAB model, because the ARENA model will only be affected, then, by one waiting time. On MATLAB, however, there will also be the waiting time of the Input Cycle, which is what makes the time difference.

Leaving aside the differences between initial values, **the most important conclusion is that both models resemble the same operating of the system when the system is fairly well sized and balanced**. Cycle times, and consequently throughput, behave in very similar ways.

6. Optimization and Improvements

Regarding the Shuttle Storage System, there are many possible different alternatives based on size, transporter's features and configuration. On this specific project, many of those choices are pre-defined, either by the KNAPP Company's features or by the nature of the study. Sizes of the rack and, very specially, transporters are defined following the parameters of the real transporters of KNAPP. Also, regarding the configuration, the main goal of the study is focused on the lift functioning logic, which is based on double cycling.

However, there is a very crucial factor that has been a priori defined for this project, without any initial restriction. The shuttle logic configuration, as explained in *Section 2.2.4.4*, has been chosen to prioritize output. Technically, the shuttle will scan, in first place, the status of the output buffer and, if empty, it will proceed to transport an output container to the output buffer. If the output buffer is, otherwise, full, the shuttle will then scan the input buffer for emptying it (if input buffer is full) or just remain idle (if input buffer is empty).

6.1. Input – Output prioritization

In first place, it was considered as a possible case of study the fact of changing the initial prioritization of the shuttle logic. In other words, configure the shuttle to scan the status of the input buffer and, if full, empty it automatically. If the input buffer was empty, then the shuttle would scan the output buffer and transport a container for output (if empty) or just remain idle (if already full).

Prioritizing either input or output on the shuttle logic will directly influence the waiting times of the double cycles. Recall that there are two waiting times on the double cycle of the lift: on one hand, the waiting time of the Between Cycle, which is the time that the lift waits for the output container at its floors; on the other hand, the waiting time between double cycles, which is the time where the lift has to start an Input Cycle but all input buffers are full, so the lift cannot have a floors assigned.

The shuttle priority strategies are, consequently, focused on reducing those times, but affect then inversely. The default strategy, which is prioritizing output, intends to reduce greatly the Between Cycle waiting time, because it brings output containers to the output buffer before doing any other activity. However, it might then increase the waiting time between double cycles, because input buffers will remain longer full. Otherwise, if the shuttle logic strategy where input is prioritized is implemented, the waiting time between double cycles will be logically reduced. However, taking containers to the output buffer will be the second action and therefore, the Between Cycle waiting time will be probably increased.

As it was explained while analysing the results on *Section 5*, with the default output prioritization, the waiting time between double cycles is in all cases were null, and only with a rack with four floors and fifty positions a non-sensible average value of less than 0,01s has been reached. Leaving aside the fact that normally racks with so fewer floors are not configured, high numbers of floors grant the system the ability of having many choice options for the inputting lift and giving enough time for shuttles to carry out Output Cycles and empty the lift on time. This is true unless there is a very high number of positions per floor, meaning that the shuttle might require very long travelling times.

If, with the default prioritization implemented that is supposed to return high values of the waiting time between double cycles, those values are negligible for all the simulated cases, it is secure to state that the input priority strategy will, by far, not be as adequate as the default strategy, because it will increase the Between Cycle waiting time without supposing any advantage.

6.2. Shuttle logic optimization

As demonstrated in the previous section, the best strategy (in time saving terms) is to prioritize bringing a container to an empty output buffer rather than emptying a full input buffer. This choice is taken at the moment of starting a shuttle cycle (which normally is right after finishing the previous cycle). At this moment, the shuttle scans the output buffer and decides.

Notice that only the status of the output buffer at that specific time is taken under account for the decision. If an Output Cycle is started, the shuttle will bring an output container to the output buffer. Then, after finishing an Output Cycle, the output buffer will be obviously full. It will scan the input buffer and if it is full (or an input container is coming with lift) it will be assigned to the Input Cycle.

Each cycle of the shuttle lasts 11,5 seconds of loading and unloading of the shuttle plus all the travelling time. The average travelling time depends, logically, on the length of the rack. Remark that the shuttle's velocity and acceleration values are lower than the lift's values and normally the length of the rack is higher than the height, and the loading/unloading time values of the lift are lower than the shuttle's values. This means that normally shuttle cycles are last longer than lift cycles.

Therefore, if an Input Cycle is started, it is probable that its time is going to be higher than any lift cycle time. During this time, it is possible that the lift comes to the floor (whose output buffer is empty or has been already emptied) and has to pick up an output container (that is not on the output buffer). The lift will have to wait for the shuttle's Input Cycle to end and bring the required output container. This might suppose in some cases very high waiting times for the lift.

To optimize this, the concept of the Shuttle Output Parameter (SOP) has been implemented. Before allowing an Input Cycle of the shuttle to start (emptying the input buffer), it might be interesting to have a look at the sequence of floors that the lift will have to go for picking up output containers. If this floor in particular is more than once in the close steps in this sequence, meaning that it will empty the output buffer and then come back for an output container soon, it might be a good idea to change the prioritization and bring that output container. In other words, even if the output buffer is full, it might be adequate to start an Output Cycle of the shuttle to avoid high Between Cycles waiting times, rather than emptying the input buffer, if the lift output sequence accords to it.

To implement this, the shuttle logic has been modified. The prioritized strategy is still output, but the scan condition is more complex. In first place, just as the default logic, the status of the output buffer will be checked and, if empty, the shuttle will automatically be assigned to bring an output container to the output buffer. However, if when scanning the status of the output buffer it is full, the shuttle will consider if it is better to bring another output container, instead of starting (if possible) an Input Cycle. The fact of bringing another output container when the output buffer is full might lead to waiting times of the shuttle, because it cannot unload the container while the buffer is full. However, the point is reducing the waiting time of the lift, which is more important than the possible waiting times of the shuttle. In exchange, however, the waiting time between double cycles might suffer an increase of time, because input buffer remain longer full.

The Shuttle Output Parameter (SOP) is responsible for controlling this feature of the shuttle logic. The SOP is the number of destination floors on the sequence of Output Cycles that is going to scan to decide on the next step. If the output buffer is full and the lift has to go twice to a same floor within the SOP range, the shuttle will be assigned first to output, bringing the second container to the output buffer and unloading in when possible.

However, as previously proved, the cycles of the lift are considerably faster than the cycles of the shuttles in relatively compensated rack sizes. It is the travelling times of the shuttle what defines its cycle times, so the variation of the length of the rack will condition the Between Cycle waiting time. Consequently, to implement and study the SOP, an optimal value of the SOP has to be determined for each length of the rack. Therefore, simulations at constant lengths have been run to study the behaviour of the system depending on different SOP values.

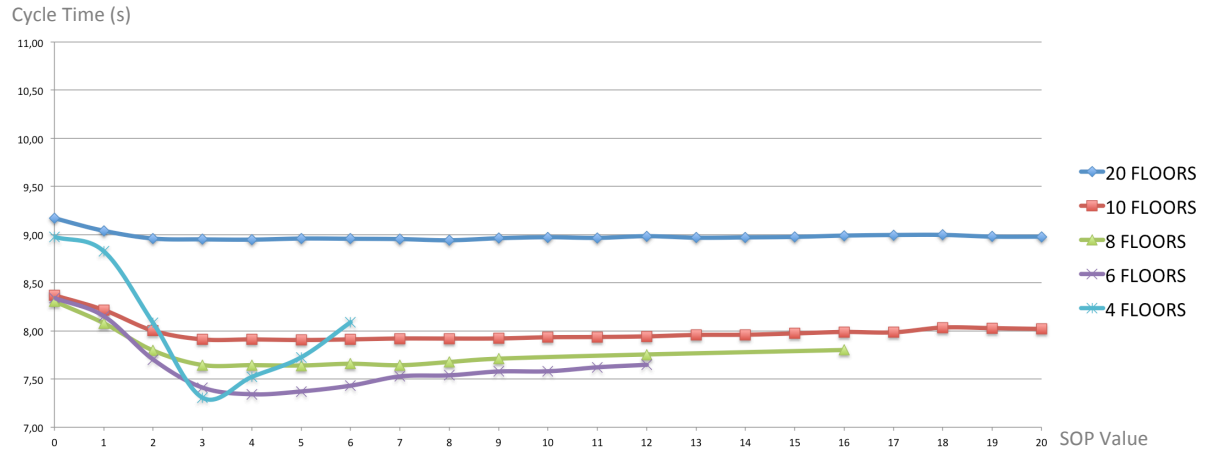

Figure 45: Double-Cycle times evolution depending on SOP at 10 positions

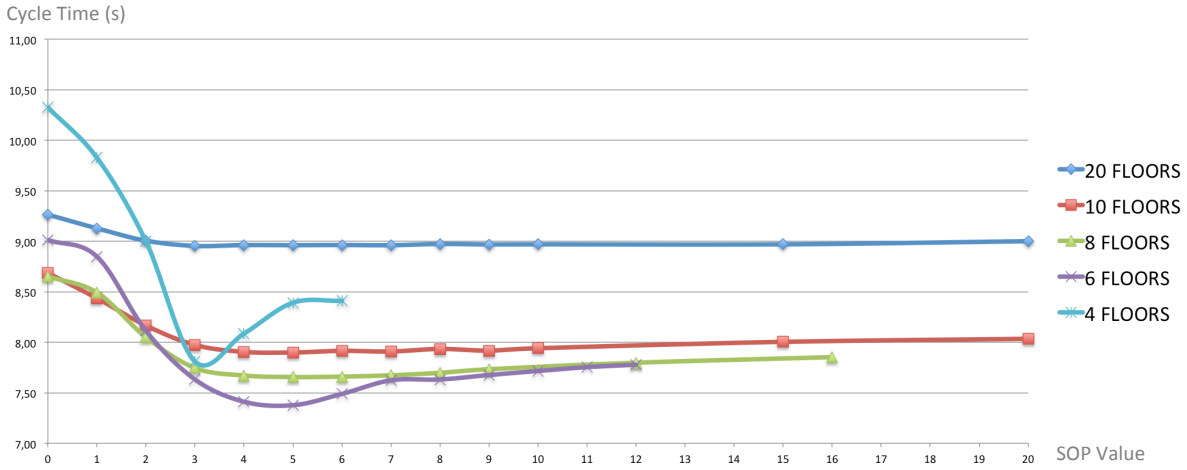

Figure 46: Double-Cycle time evolution depending on SOP at 20 positions

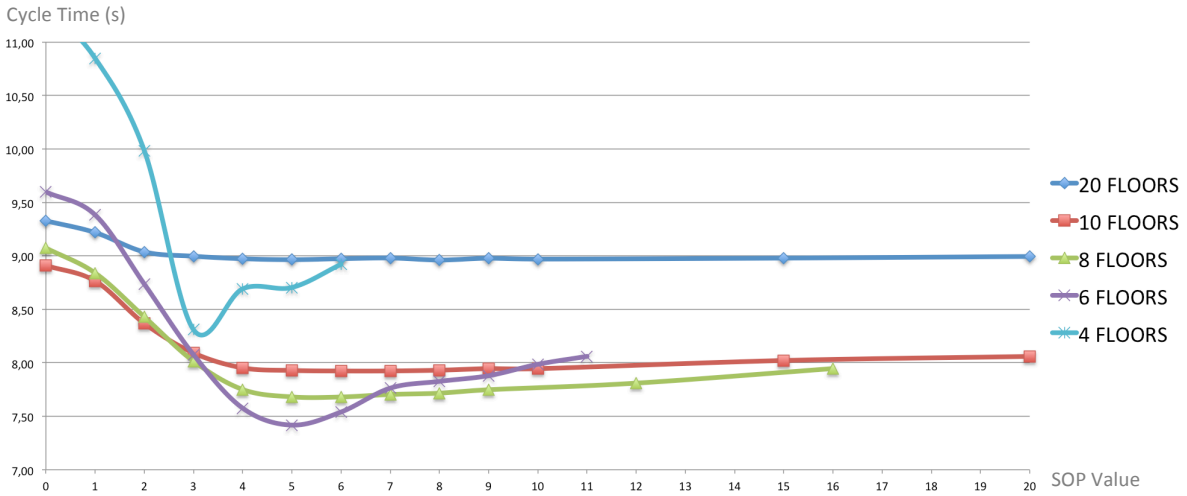

Figure 47: Double-Cycle times evolution depending on SOP at 30 positions

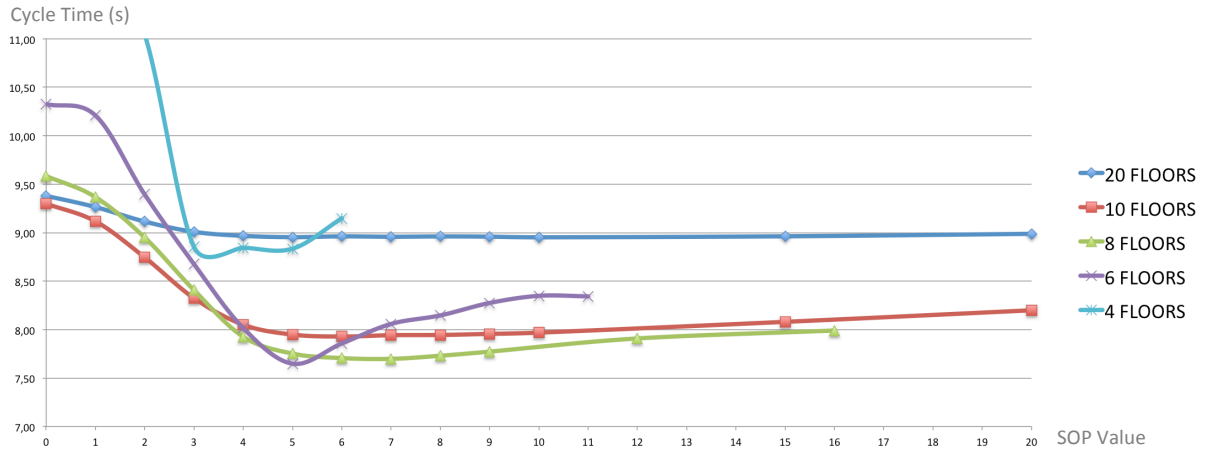

Figure 48: Double-Cycle time evolution depending on SOP at 40 positions

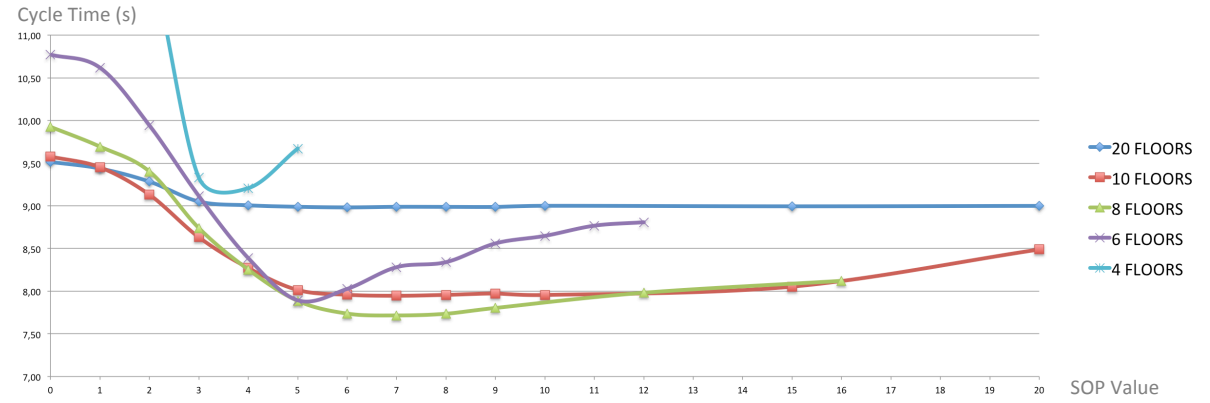

Figure 49: Double-Cycle time evolution depending on SOP at 50 positions

Notice, in first place, that values at SOP = 0 (dots on the Y axis) are the values with the non-optimized configuration studied at *Section 5*.

In next place, in the previous figures, the behaviour of the double cycle time depending on the SOP is graphed. It can be seen that the initial value at SOP can be, by far, optimized with the proper SOP value. The plots of the double cycle follow similar patterns at each different figure.

To understand the plots, two different analyses will be made. In first place and for all plots, the graph initially decreases to an optimal value (or values) of the cycle time and then starts to softly increase. That initial decrease is due to the reduction of the Between Cycle waiting time, as it can be seen in *Figure 50*. However, as the SOP value is incremented, the waiting time between double cycles of the lift is also incremented. This is reflected with the soft increase of the plot. Those two patterns are demonstrated at the *Figure 48* and *Figure 49*.

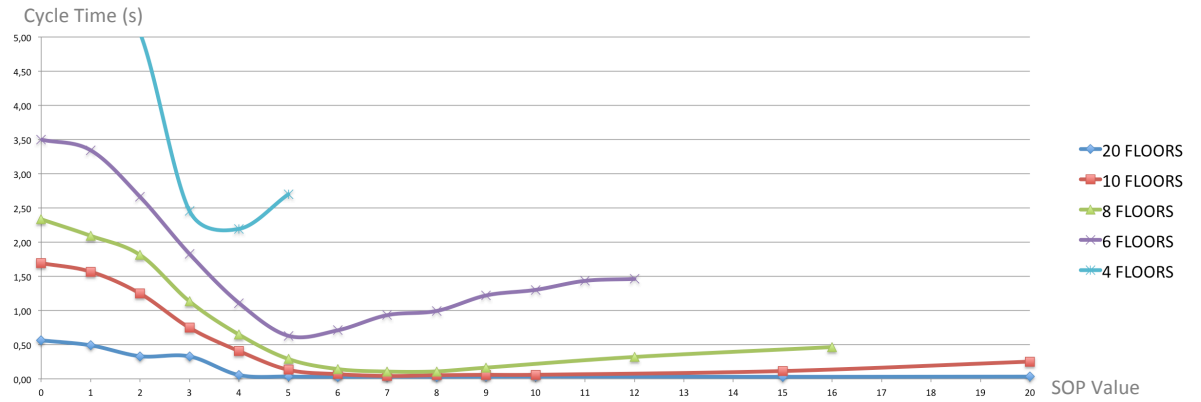

Figure 50: Between Cycle waiting time evolution depending on SOP at 50 positions

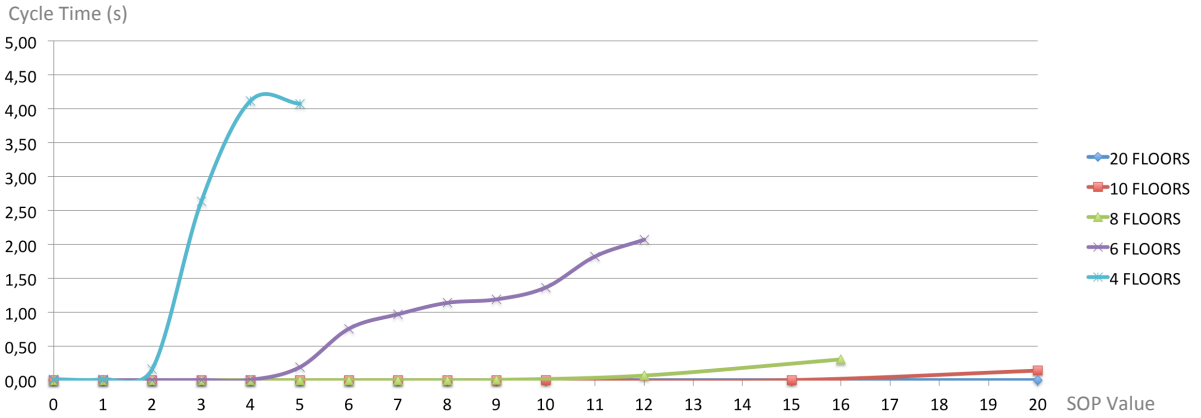

Figure 51: Waiting time evolution between Double-Cycles depending on SOP at 50 positions

The other analysis makes reference to the differences between numbers of floors. There is a great difference between the plots representing four floors and twenty floors. In first place, the decreasing slope is much more pronounced with few floors. Obviously, the Between Cycle waiting times are higher with fewer floors, because the travel rate to that floor is a lot bigger. Remark that the double cycle time also depends on the number of floors, so with higher number floors, the time values are supposed to be higher. **The optimal SOP value is achieved when the sum of Between Cycle waiting time and waiting time between double cycles is optimal (minimum)**.

Notice that on the previous graphs, few floor plots are simulated until a certain SOP parameter. Increasing a lot the SOP parameter in systems with few floors might lead to a new deadlock situation. If only outputs are being allowed, independently of the lift, with a SOP parameter value very high there might be the case where all shuttles are bringing output containers and not emptying the input buffers. Then, there might be the situation where the lift is stuck between double cycles (waiting for an empty input buffer) and all shuttles are busy with an output container, unable to deploy it to the

output buffer (because it is already full). A maximum value of the SOP parameter could be calculated, but its value might no be fixed due to the randomness of the system.

However, it is unnecessary to find the exact maximum value of the SOP. As it can be seen on the previous graphs, the optimal value (which is to one of interest) happens right after the decrease. As a matter of fact, with a relative normal number of floors, the range of values (after the initial scope) is very thin.

6.3. Optimized results

In order to have a clear idea of the real affectation of this optimization, the following graphs are an image of the same plots with the default shuttle logic in *Section 5*. The SOP parameter for these simulations is the most optimal value, extracted from *Section 6.2*. Note that the SOP values are different and independent from the rack sizes, so different SOP values are taken for the same length of the rack (but different height) and vice versa.

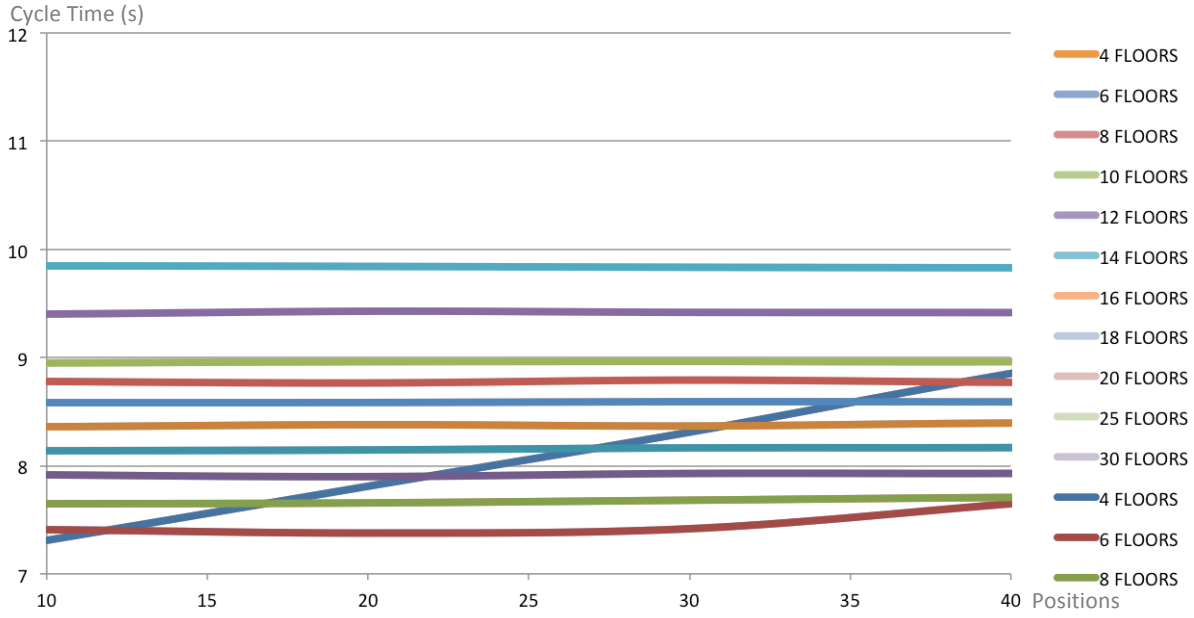

Figure 52: Double-Cycle times at constant height

In comparison with the default graphic, there is one main difference to resemble. Especially with low floor number racks, the cycle time is more constant. As explained in *Section 5*, the slope of the cycle time in this plot only depends on the scope of the Between Cycle's waiting time. As it can be seen, in this case, the slope is constant, which means that the Between Cycle's waiting time remains constant with the different rack sizes as well as the proper SOP value for each simulation.

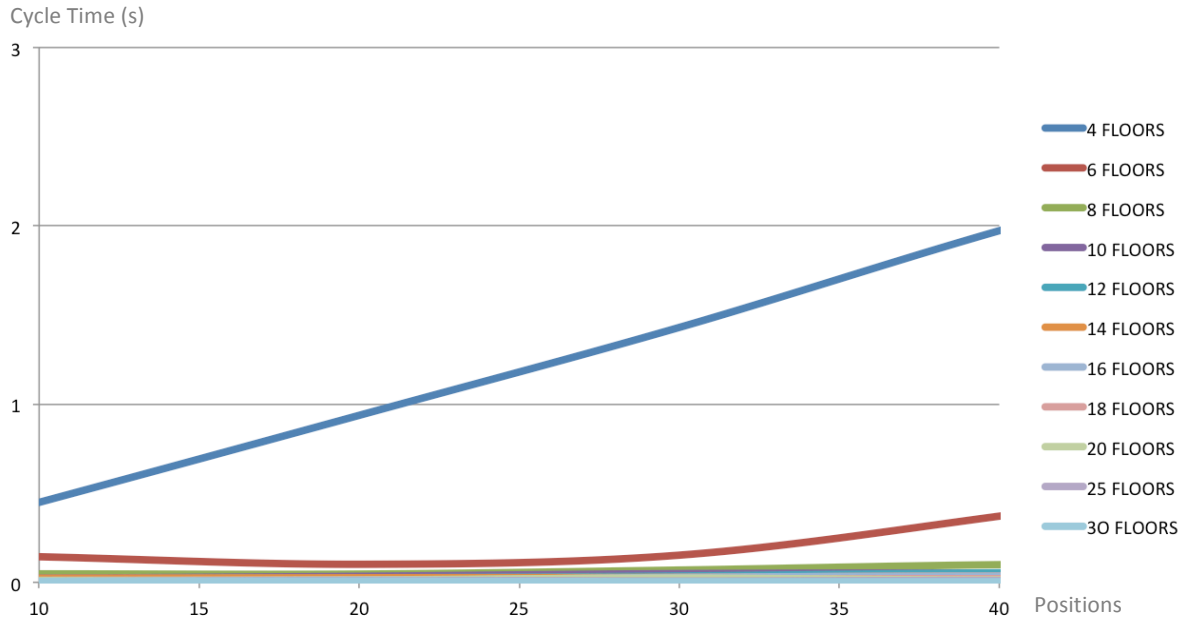

Figure 53: Between Cycle waiting times at constant height

As it can be seen, there are only sensible Between Cycle waiting times on racks with few floors. This is related to the rate in which the lift travels to a specific floor. This supposes a difference regarding the default shuttle logic, where these waiting times were non-negligible for most of the simulations. Probably, with longer lengths, there would appear more waiting times, because the travelling times are higher.

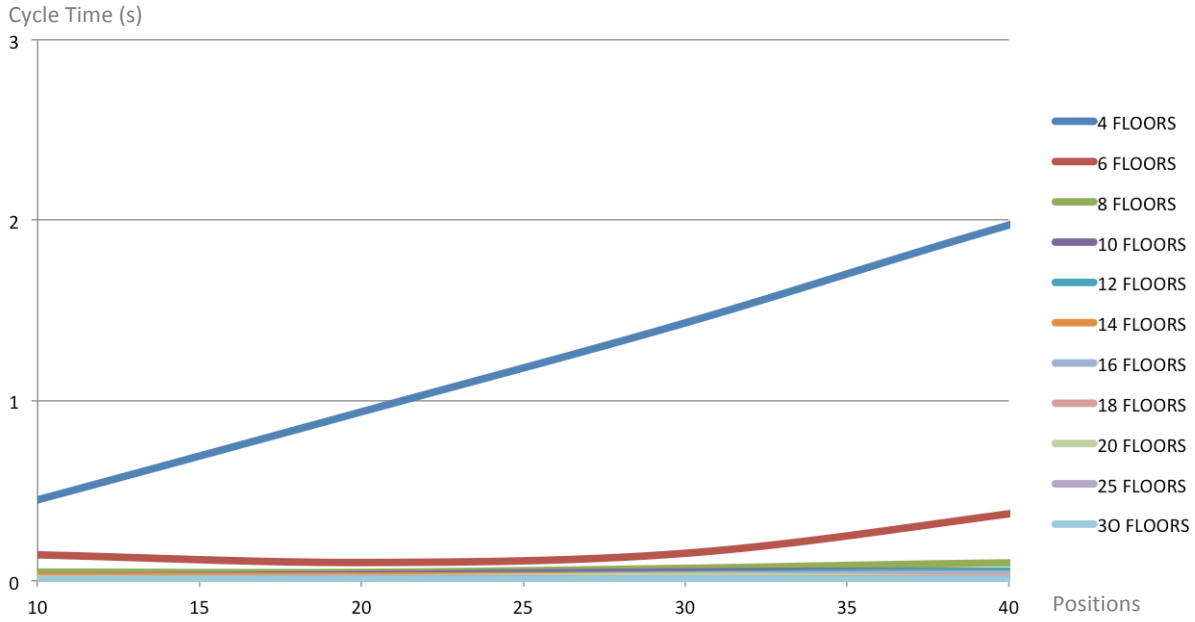

Figure 54: Waiting time between Double-Cycles at constant height

However, the waiting time between double cycles is, in this case, very sensible for the simulation with four floors. For the other sizes, however, it remains null. However, the comparison between the sum of both waiting times in the optimized model is lower than only the Between Cycle waiting time in the default model, thanks to the implementation of the SOP.

In the following figures, from the point of view of constant length of the rack, the evolution of waiting times is clearer.

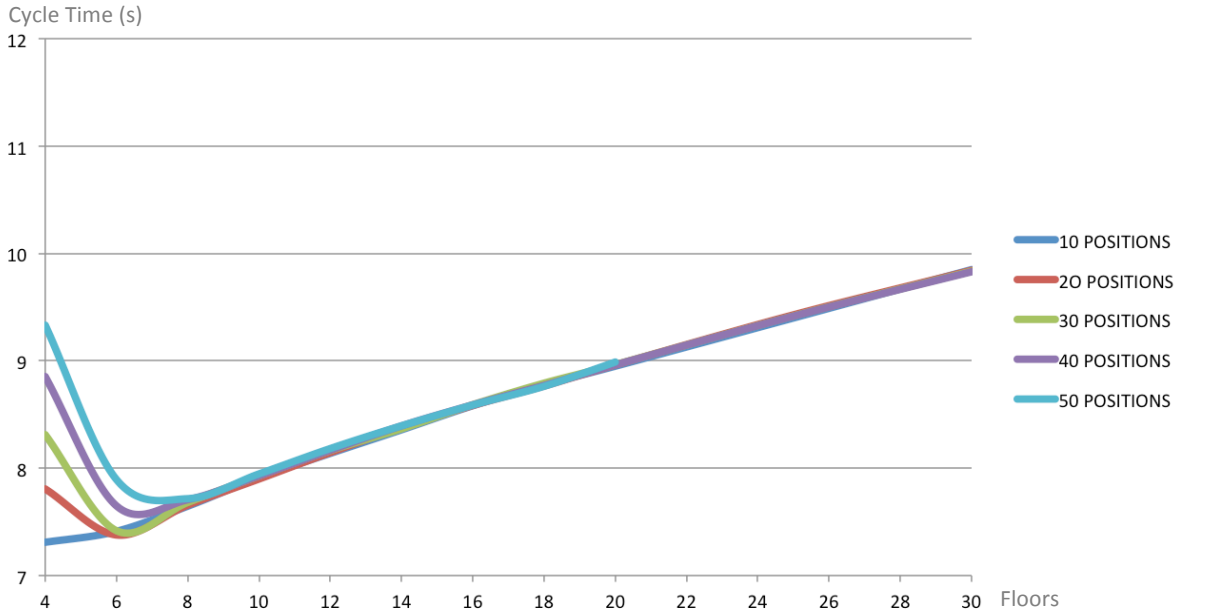

Figure 55: Double-Cycle times at constant length

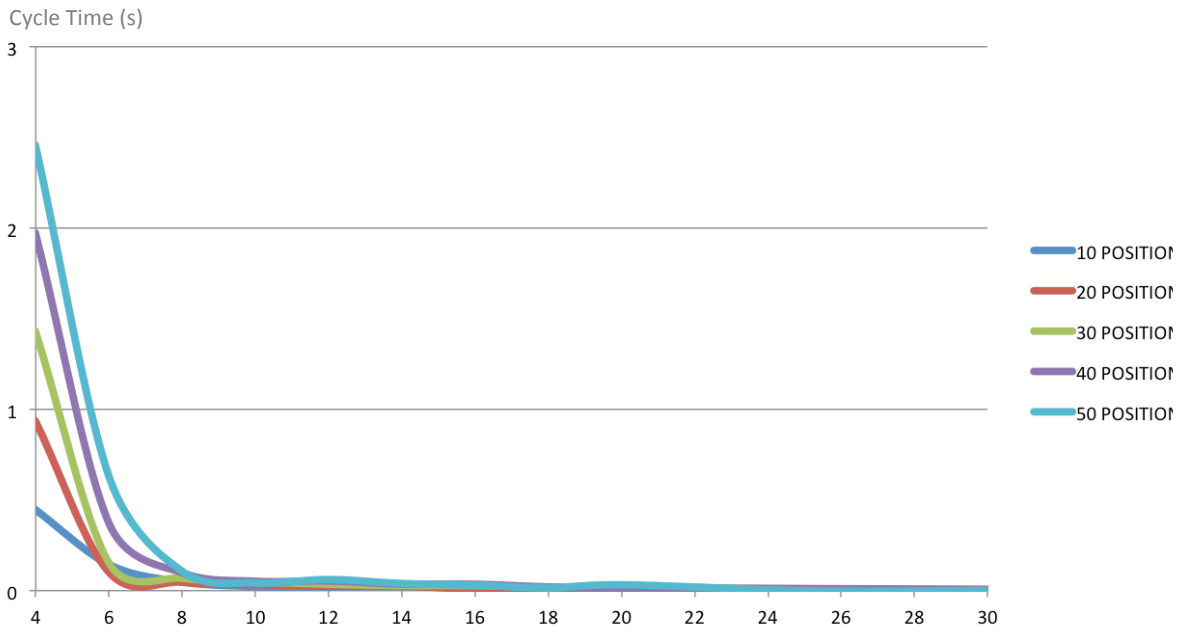

Figure 56: Between Cycle waiting times at constant length

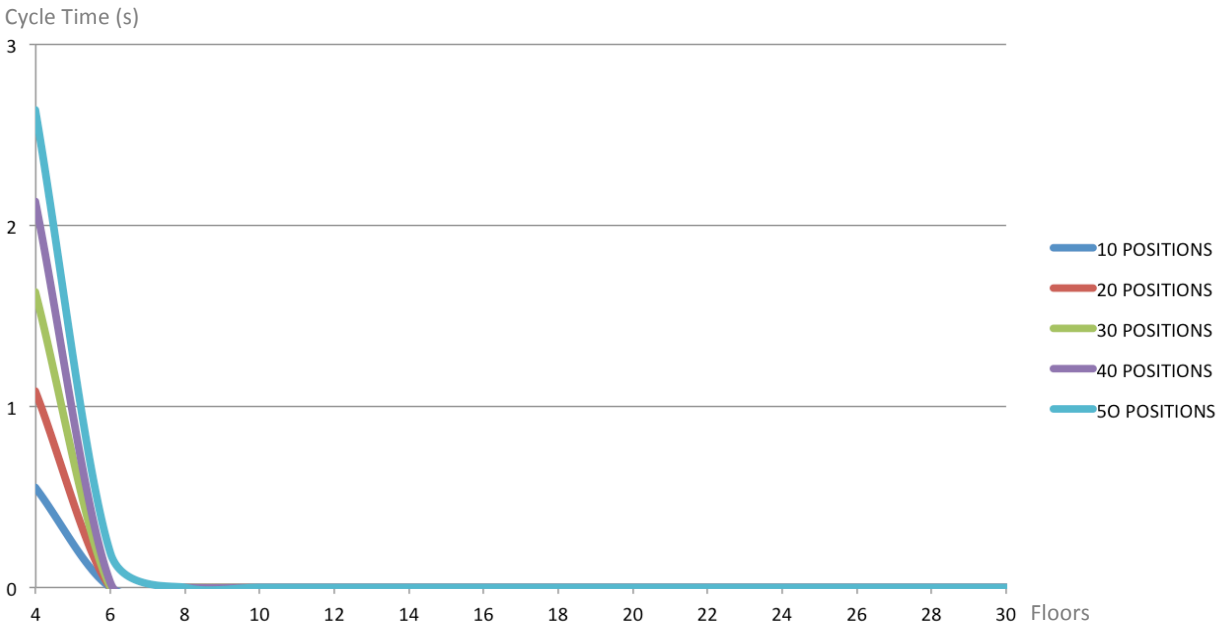

Figure 57: Waiting time between Double-Cycles at constant length

From racks with eight floors, both waiting times are negligible. Therefore, the double cycle time is a regular line that represents the sum of the constant values and the travelling times (that increase with the increment of height). Probably, as it can be seen in the figures, plots with higher number of positions would require more floors for neglecting the waiting times.

7. Conclusions and outlook

To finish this project, the most important conclusions will be pointed out. Those conclusions will be divided in three groups: conclusions in reference to the results of the project, conclusions regarding the simulated system and general conclusions experimented during the realization of the project.

Regarding the results, the behaviour of the system and values obtain seem, a priori, very satisfactory. Not only the simulations with ARENA Simulation Software and MATLAB have reported similar results of a fairly similar system, but also results converge into a very consistent scope.

When talking about results, reference to the cycle times of the lift is basically being made. Those cycle times are conditioned by the waiting times that hold the lift from working. Focus on the behaviour of those waiting times has been made during the entire project. The main conclusion regarding the double cycle times of the lift is that the waiting times are reduced to their lowest expression when the system meets three conditions.

In first place, **the rack has to be balanced**: this means that the height and the length keep relative values and one value is not too big in comparison. With an unbalanced system, waiting times increase. If the length of the rack is too high, the travelling times of the shuttle will be too long and the lift will have big Between Cycle waiting times, as well as needing more time to empty the input buffer so waiting times between double cycles also increase. To balance the system, high lengths of the rack need to go together wit high heights of the rack.

In second place, **the rack has to be fairly well sized**: this means, basically on terms of height, that the waiting times will only be optimal it the number of floors is not to low. The number of floors conditions the frequency in which the lift will travel to the floor. With high frequencies, the shuttles might not be able to perform the cycles in time (either input or output) and waiting times for the lift will appear.

Finally, **the shuttle logic must be the best**. As it has been tested, there are some alternatives for the logic of the shuttles (that is not restricted by any initial condition). The optimal parameters of the logic, as demonstrated, depend on the size of the rack. The priorities of the shuttle to which cycle to start also determine greatly the waiting times. The shuttle logic, as a matter of fact, optimizes the sum of the two waiting times in the system. This means that, although it might increase one of the waiting times, the sum of both will be lower at the end.

If those three conditions are met and therefore the waiting times are reduced to minimum, it might be even possible to extract an analytical expression or formula to calculate the cycle times of the lift.

All values depend on constant values and on the height of the rack. The waiting times, however, would be much more complicated to define when are not negligible.

Regarding the system that has been simulated, remark that the world of Shuttle Storage Systems offers a wide range of possible configurations, either physical or logical. With different configurations, different results might be obtained.

On one hand, the physical configuration might offer the best improvement of results. Leaving aside the variation of parameters such as transporter velocity, rack inner distances or container size (that would suppose not a great improvement), the best way to improve results is to modify the physical existence of the lift. Alternatives like changing the position of the arrivals/departure station to a different height (not at the ground Z = 0) or enabling inputs on one side of the rack and output at the opposite side might cause very different results, because travelling times of the transporters would be greatly affected.

However, there is one alternative that would suppose a major improvement on the system. Instead of working with only one lift for input and output, implementing (on the same side of the rack) a separate lift only for input and a separate lift only for output would probably make the difference. The disadvantages associated to the double cycling of one lift (such as the Between Cycle waiting time) would be erased. As a matter of fact, this is the configuration implemented and manufactured by the company KNAPP.

Therefore, a subsequent study is going to be carried away by Herr Mario Kvasnicka. This study will be similar to this project, with the same goals and similar model development with ARENA Simulation Software, but implementing the new lift physical configuration. Comparing the results from this project and from the subsequent project will resemble the differences of throughput of the Shuttle Storage System, probably demonstrating that the optimal solution is the implementation of two lifts, as in the company KNAPP.

Finally, critical thoughts regarding the implementation of ARENA Simulation Software for this project need to be pointed out. In first place, remark that the user and developer of this model and project did not have a high previous ARENA Simulation Software knowledge prior the project. Therefore, the successful construction of the model has been a slow obstacle-solving run with an unpredictable ending. At the end, although the model runs perfectly fine for its purpose, the model presents limitations of simulation due to the template structure development. As explained in *Section 4*, the Final Model has been developed using templates. What seemed during the construction as the only feasible way of developing the model (under time of construction terms), it turned in return a great

disadvantage. When ARENA Simulation Software is set to start simulation, the software prepares (or checks) the entire model, template by template. When working with big sized racks, the number of templates is very great. As a reference, for the biggest racks simulated in this project, the “Checking Times” were up to 7-8 hours, for only one simulation.

Also, notice that if any variation of values wants to be simulated (such as the SOP or different sizes of the rack), this requires each time a new run of simulations (at consequently the “Checking Times”). This fact makes the use of this template structure very rudimentary and not flexible, in comparison to the MATLAB model, for example. In close future, a new template structure will be implemented and tested to try to reduce the “Checking Times”.

It is due to the high “Checking Times” reported by ARENA Simulation Software that it not able of simulating higher values of the rack size, as intended initially. Although the simulations done allow to guess the behaviour of the system with higher rack sizes, it is stated that more simulations would have been necessary to confirm the tendency of the values reported (cycle times and throughput).

8. Acknowledgements

In first place, I would like to thank Univ.Prof. Dipl.-Ing. Dr.techn.Georg Kartnig for granting me the possibility of developing this Master Thesis at the Techinsche Universität Wien and for all the help, guidance and understanding during the last months.

Secondly, I would also like to thank Univ.Lektor Dipl.-Ing. Dr.techn.Karl Heinz Weigl for its external collaboration, flexibility and interaction regarding the ARENA Simulation Software.

This project would not have been feasible without the help of the other student working on this topic Herr Lukas Fleischer and Herr Mario Kvasnicka.

Last but not least, it would like to thank to collaboration of Aida Tort, Josep Tomàs, Eduard Baeza, Sandy Bradbury, Artur Boix, Marc Sagrera i Josep Torra.

9. References

ARENA SIMULATION SOFTWARE BY ROCKWELL AUTOMATION (2013). *Professional Edition: Overview*. Retrieved on the 28th August, from http://www.arenasimulation.com/Products_Professional_Edition.aspx

CHARLIE (2012). *Pallet Racks: The history and times of warehousing*. Retrieved on the 27th August, from http://es.scribd.com/doc/86131750/Pallet-Racks-The-History-And-Times-Of-Warehousing

DAIFUKU WEBB. *Automated Storage & Retrieval System (AS/RS)*. Retrieved on the 28th August, from http://www.daifukuwebb.com/products/automated-storageretrieval-system-asrs

KELTON, DAVID M; SADOWSKY, RANDALL P; SADOWSKI, DEBORAH A. *Simulation with Arena*. Retrieved on the 31st August, from http://es.scribd.com/doc/81857763/Kelton-Simulation-With-Arena-Copy-Copy-Reduced

KELTON, DAVID M (2007). *Chapter 7: Intermediate Modeling and Steady-State Statistical Analysis* (p. 293-294). Retrieved on the 31st August, from http://es.scribd.com/doc/50006909/Kelton-2007-Chapter-7

KNAPP AG BIBLIOTECA. *OSR Shuttle*. Retrieved on the 28th August, from http://www.knapp.com/cms/cms.php?pageName=glossary&iD=15&sid=vihmb3smdd227op4vv780oan00

MARTIN, MURRAY. *Warehouse Best Practices*. Retrieved on the 26th August, from http://logistics.about.com/od/tacticalsupplychain/a/wms_best_prac.htm

MILWAUKEE (April 27, 2000). Rockwell Automation Acquires Systems Modeling Corporation. Retrieved on the 28th August, from http://phx.corporate-ir.net/phoenix.zhtml?c=196186&p=irol-newsArticle&ID=825787&highlight

ROCKWELL AUTOMATION (2013). *Rockwell Automation. Listen, Think, Solve*. Retrieved on the 31st August, from http://www.rockwellautomation.com/

THE EDITORS OF THE ENCYCLOPAEDIA BRITANNICA. *Storage*. Retrieved on the 26th August, from http://www.britannica.com/EBchecked/topic/567498/storage

WYKE, TERRY. *Manchester Warehouses*. Retrieved on the 26th August, from http://www.revealinghistories.org.uk/why-was-cotton-so-important-in-north-west-england/places/manchester-warehouses.html

[Image 1: Tradicional Warehouse]. Retrieved from http://www.marktisdalephotography.com/2009/07/king-cotton/

[Image 2: Rack storing]. Retrieved from http://www.performancepeople.com

[Image 3: Forklift transporter]. Retrieved from http://shutterstock.com/

[Image 4: Automated racking systems]. Retrieved from http://top-tiger.en.made-in-china.com/product/gowEKfsrXbWQ/China-Automated-and-Retrieval-Racking-System.html