AUTOMATIC SIMULATION MODELS GENERATION OF WAREHOUSES WITH MILKRUNS AND PICKERS

António Vieira[1], Luís S. Dias[1], Guilherme B. Pereira[1], José A. Oliveira[1], M. Sameiro Carvalho1 and Paulo Martins[1]

(1) (2) University of Minho, Campus Gualtar, 4710-057, Braga, Portugal.
(1, 2) (antonio.vieira, lsd, gui, zan, Sameiro, pmartins)@dps.uminho.pt

ABSTRACT

To help a company of the Bosch Group to reduce its costs (both in time and space) with its supermarket, a micro simulation model was developed in Simio. Particularly, the tool is able to model pickers riding milk runs to collect containers of products, from a supermarket, to satisfy the needs of the production lines. Practitionaer may benefit from this tool, since it is able to model different supermarket scenario, for instance different storage strategies. Additionally, the supermarket itself is automatically created, through an Add-in of Simio that was developed in C#, which implements the API of Simio. Conclusions and future work are discussed.

Keywords: Warehouse, milk run, picking, Micro simulation, Simio.

1. INTRODUCTION

In recent years, the Bosch Group has been applying concepts of the Toyota Production System (TPS) (Monden, 1998) and of the Lean Manufacturing (Womack et al., 1990, Womack and Jones, 1996), designated as Bosch Production System (BPS) (Yildiz et al., 2010, Costa et al., 2011). Its purpose is to "eliminate waste in production and all related business processes. BPS provides the basis for continuous improvements in quality, costs, and supply performance" (Bosch, 2014).

A significant part of the costs of a company are its supermarkets (Baker and Canessa, 2009). Since one of the objectives of the BPS is to reduce costs, the need, to study alternatives to the current design and picking system of the supermarket on the company Bosch Car Multimedia Portugal in Ferreiros, Braga, arose.

In this context, a micro simulation model, using Simio, was developed. The tool is able to model pickers riding milk runs to collect containers of products, from the channels of a supermarket, to satisfy the needs of the production lines. A Channel is the basic unit for storage in this supermarket. Each has the capacity to hold several containers. On the other hand, a container holds many units of one type of product.

The storage strategy used in this supermarket is the dedicated. This is the most simple that can be used, since it consists on having a channel dedicated to a single type of product (Bartholdi and Hackman, 2008). One of its great advantages, resides on the fact that, since the locations of the product don't change, the pickers can memorize them, making the picking process more efficient (Bartholdi and Hackman, 2008). Nevertheless, the problem with this strategy is that "it does not use space efficiently. In fact, it is expected that, on average, the storage capacity is about 50%" (Bartholdi and Hackman, 2008), which represents a high amount of costs associated. To overcome this problem, other strategies can be considered (e.g. random storage). Thus, the simulation model must be able to model several storage strategies. In addition to that, the quantity of requests a picker gets per shift, the time between shifts, the number of types of products, the arrival rate of requests, and the number of milk runs and pickers need to be configurable.

Additionally, the supermarket is composed by circulation corridors for milk runs that gives them access to corridors of racks. In its turn, each rack is composed by a variable number of channels, in height and in width, whereby it is necessary to create several layouts of the supermarket. To do so, the API of Simio is being used to create an add-in, in C#. The latter reads data from an excel file, where the user is able to specify several inputs, e.g. the number of corridors, their positions, their rotation angles, the number of channels on each rack (in height and in width), among others. Nevertheless, the creation of the add-in will not be covered in this paper. Regardless of that, the simulation model was built so that several layouts of the supermarket could be modelled. Thus, this paper intends to document the first part of the simulation model developed, which consists on the pickers receiving requests and riding their milk runs to collect the respective containers from the supermarket. The return of the leftover containers and the restock processes are not yet modelled.

Section 2 presents a review over the literature. In section 3 and 4, the development of the simulation model; the data collected and validation of the model will be covered. Section 5 addresses the development of the add-in to automatically create the supermarket and, in th final section the main conclusions are discussed.

2. LITERATURE REVIEW

According to Coyle et al. "Warehousing provides time and place utility for raw materials, industrial goods, and finished products, allowing firms to use customer service as a dynamic value-adding competitive tool" (1988). Thus, supermarkets represent a very important role on modern supply chains (Baker and Canessa, 2009).

In fact, "whilst supermarkets are critical to a wide range of customer service activities, they are also significant from a cost perspective. Figures for the USA indicate that the capital and operating costs of

supermarkets represent about 22% of logistics costs (Establish, 2005), whilst figures for Europe give a similar figure of 25%" (Baker and Canessa, 2009). These costs impel us to understand the problematic and to use the storage space as efficiently as possible (Bartholdi and Hackman, 2008).

Thus, the need to provide companies with methods capable of improving the performance of supermarkets arises. According to Gu et al., some of these methods include simulation, analytical methods and benchmarking. The former is the most used whether in literature or in practice (2010). One example is the simulation model developed by Costa et al. using Arena. The authors conducted experiments to identify changes that could be made on a material delivery system to improve the efficiency and precision of the logistic train functioning they were modelling (2008).

Since the number of simulation tool options can be very high, tool comparison becomes a very important task. However, most of scientific works related to this subject "analyse only a small set of tools and usually evaluating several parameters separately avoiding to make a final judgement due to the subjective nature of such task" (Dias et al., 2007).

Hlupic and Paul (1999) compared a set of simulation tools, distinguishing between users of software for educational purpose and users in industry. In his turn, Hlupic (2000) developed "a survey of academic and industrial users on the use of simulation software, which was carried out in order to discover how the users are satisfied with the simulation software they use and how this software could be further improved". Dias and Pereira et al. (2007, 2011) compared a set of tools based on popularity on the internet, scientific publications, WSC (Winter Simulation Conference), social networks and other sources. "Popularity should never be used alone otherwise new tools, better than existing ones would never get market place, and this is a generic risk, not a simulation particularity" (Dias et al., 2007). However, a positive correlation may exist between popularity and quality, since the best tools have a greater chance of being more popular. According to the authors, the most popular tool is Arena and the good classification of the Simio is noteworthy. Based on these results, Vieira et al. compared both tools taking into consideration several factors (2014a).

Simio was the chosen tool for this project. It is based on intelligent objects (Sturrock and Pegden, 2010, Pegden, 2007, Pegden and Sturrock, 2011). These "are built by modellers and then may be used in multiple modelling projects. Objects can be stored in libraries and easily shared" (Pegden, 2013). Unlike other object-oriented systems, in Simio there is no need to write any programing code, since the process of creating a new object is completely graphic (Pegden and Sturrock, 2011, Pegden, 2007, Sturrock and Pegden, 2010). The activity of building an object in Simio is identical to the activity of building a model. In fact there is no difference between an object and a model (Pegden, 2007, Pegden and Sturrock, 2011). A vehicle, a costumer or any other agent of a system are examples of possible objects and, combining several of these, one can represent the components of the system in analysis. Thus, a Simio model looks like the real system (Pegden and Sturrock, 2011, Pegden, 2007). This fact can be very useful, particularly while presenting the results to someone non-familiar to the concepts of simulation.

In Simio the model logic and animation are built in a single step (Pegden and Sturrock, 2011, Pegden, 2007). This feature is very important, because it makes the modulation process very intuitive (Pegden and Sturrock, 2011). Moreover, the animation can also be useful to reflect the changing state of the object (Pegden, 2007). In addition to the usual 2D animation, Simio also supports 3D animation as a natural part of the modelling process (Sturrock and Pegden, 2010). To switch between 2D and 3D views the user only needs to press the 2 and 3 keys of the keyboard (Sturrock and Pegden, 2010). Moreover, Simio provides a direct link to Google Warehouse, a library of graphic symbols for animating 3D objects (Sturrock and Pegden, 2010, Pegden and Sturrock, 2011).

3. MODEL DEVELOPMENT

Throughout this section, some terms will be used that may be unknown for a user not familiar with Simio. For those, a reading of the paper written by Vieira et al. (2014a) would be advisable.

For this simulation project, 4 types of entities and 5 models (4 sub-models and a main one) were created. In the first section of this chapter, the former will be presented, while the models will be analysed on the following sections. Particularly, the main goals, the properties and the external view of the sub-models will be presented, so that it becomes easier to understand their use on the main model, which will be addressed in the last section. The 4 created types of entities were:

3.1. Types of Entities

- **Picker**: Represents the pickers of the system. Their functions are to collect Requests at the beginning of a shift and take Containers from Channels of the Supermarket to place them on the milk run.

Figure 1: Symbol of the Picker entity

- **milk run**: Represents the milk runs of the system. Its only purpose is to transport the Picker and the selected Containers between the Supermarket.

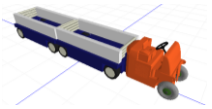

Figure 2: Symbol of the milk run entity

- **Request**: Represents the request of the system

Figure 3: Symbol of the Request entity

- **Container**: Represents the containers of the system.

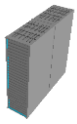

Figure 4: Symbol of the Container entity

3.2. Models

In this subsection, the developed models will be explained. The first developed model is the GoTomilk run. Its only purpose is to transfer a Picker to the riding station of the respective milk run. This way, the Picker will seem to be riding the milk run, while the Containers will stay at the wagon of the milk run. Figure 5 presents the external view of the model

Figure 5: External view of the GoTomilk run model

To model the milk run stopping; the Picker leaving the milk run to collect containers, as well as the return of th ePicker and to evaluate if the Picker needs to return to the Channels or not - the picker collects 1 container at a time – the StopPlace model was developed. The properties defined for this model were:

- **Place:** Numeric property that works as an identifier number of the instances of this model placed on the Supermarket.
- **Rack:** String property that identifies the Rack that this model belongs to.
- **LastOfCorridor:** Boolean property to indicate whether this model is the last of a corridor or not.
- **ConnectTo:** Object property to specify instances of this model. Used when a corridor has sets of channels on both sides.

The Facility of this model is presented on Figure 6 and its external view is presented on Figure 7.

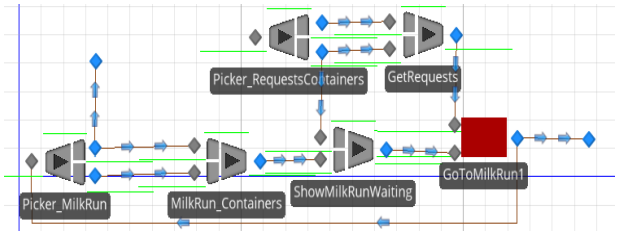

Figure 6: Facility of the StopPlace object

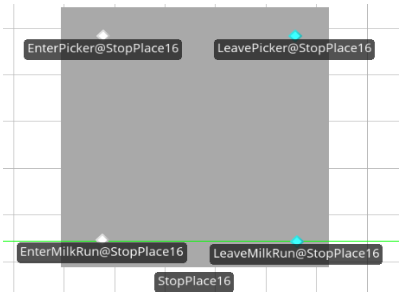

Figure 7: External view of the StopPlace model

For animation purposes, a new model – whose only purpose is to show a copy of the picker in front of the rack of channels – while the original travels inside the channel to collect the intended containers, was developed. Moreover, had this model not been developed and, after entering the Channel, the Picker would disappear for some time, before returning with the selected Container. Figure 8 displays the external view of this model.

Figure 8: External view of the StopPlace_Channel model

To store containers and model the behaviour of the Pickers, when they analyse a channel to select the container they want, the channel model was developed. The external view of this model is presented on Figure 9. The properties defined for this model were:

- **Position:** Numeric property that works as an identifier number of the instances of this model placed on the Supermarket.
- **TotalProducts:** Expression property that indicates the number of types of products to be modelled.
- **StockPolicy:** Expression property. This property indicates the stock strategy to be modelled. Since the restock process is not yet modelled, the containers are being created inside each Channel. Thus, this property indicates if the type of each container being created should be in accordance to the channel (dedicated storage) or not.
- **StopPlace:** Numeric property. The value of this property must be equal to the Place property of the StopPlace that allowed the Picker to reach this model.

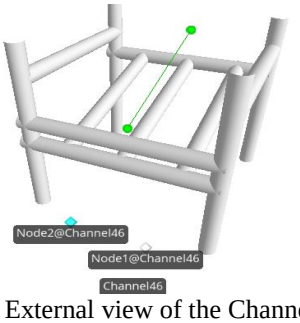

Figure 9: External view of the Channel model

All the presented models compose the supermarket itself. The properties of the supermarket model are:

- **Numbermilk runs:** Expression property that indicates the number of milk runs and Pickers to be modelled.
- **StockPolicy:** Expression property. It indicates, to all instances of the Channel model, the storage strategy being used.
- **NumberRequests:** Expression property that defines the way the Pickers add Requests to their batches.

- **TotalProducts:** Expression property that indicates, to all instances of the Channel model, the number of types of products to be modelled.
- **RequestsIntensity:** Expression property that defines the average Interarrival time of Requests to the system.

To access all the Channel and StopPlace objects placed on the Supermarket – among other reasons, to be able to consult the containers stored in each channel - two data tables were developed: one to gather all the channels and another to gather all the StopPlace models. Each object occupies an index of the data table correspondent to its Place or Position property, respectively. Additionally, a Corridors data table was created to gather all the information of all the corridors. Figure 10 shows an example of a Corridors data table.

Corridors

	Corridor ID	Number Of Ways	Image Index_Rotation Angle
1	1	2	1
2	2	2	0
3	3	1	0
4	4	1	0

Figure 10: Corridors data table

As can be seen, the data table holds information relative to the rotation angle of the corridors, the number of ways and the identifier numbers. The former will not be analysed on this paper.

After being created, Pickers and milk runs will travel through the supermarket in a serious of picking shifts, by executing a set of processes. Figure 11 shows one of such processes.

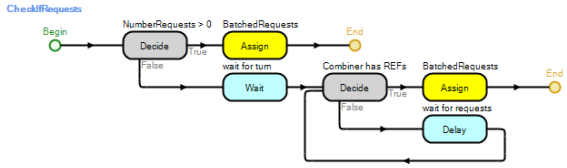

Figure 11: Process CheckIfRequests

The goals of this process are to make the picker wait for its turn and to specify the quantity of Requests to be allocated. If the property has a negative value, the Picker waits an amount of time (in minutes) equal to the module of that value. In this case, the number of Requests that are on the Combiner object, is saved to the BatchedRequests state. This way, when the Picker enters the Combiner object itself, that number of Requests are added to its batch. On the other hand, if the value is positive, the associated token will save that number to the BatchedRequests state of the Picker. Once on the Combiner, it will wait the time needed for that amount of Requests to be added to its batch. After the batch is formed, an associated will execute the process displayed on Figure 12.

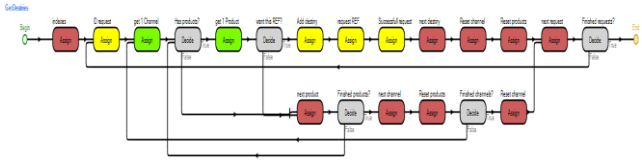

Figure 12: Process GetDestinies

The goal of this process is to save all the Channels that have the Containers correspondent to the Requests added, on an object array of the Picker. This way, each Picker has its own array of destinies. When analysing the Requests, the token saves the number identifier of the Picker on the state Requested of each Request. By doing so, it is ensured that there will be no exchanges of Requests during a picking shift. Lastly, when analysing each Container, the token also saves the ID of the Picker on their Requested state. This way, since the Containers are requested, it is ensured that the destinies of the Picker are the right ones and that no other Picker will take the Container requested.

Once the process ends, the Picker enters the GoTomilk run object, where the corresponding milk run is. In this object the Picker will be transferred to the riding station of the milk run. Additionally, on the Process property of this object the value GetStopPlaces is inserted, i.e., the milk run will have an associated token execute that process. Figure 13 shows the process GetStopPlaces. Similarly, to the process GetDestinies, this intends to save the StopPlaces where the milk run needs to enter, to an array of objects of each milk run. To that end, the StopPlace, with a value on the Place property equal to the value of the StopPlace property of the Channel on the array of destinies of the Picker, will be added. It should be noted that no repeated objects are added. Once the process ends, the milk run leaves the object and initiates its picking shift.

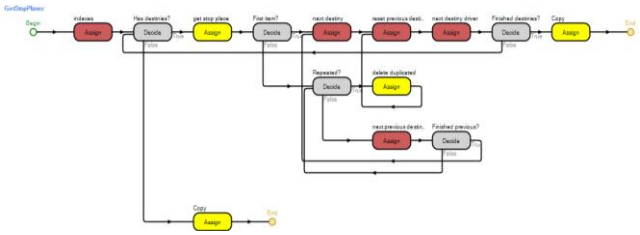

Figure 13: Process GetStopPlaces

The milk runs can travel through two types of corridors. In the first, they only have access to corridors of racks on one side of the corridor. In the second type, they have access to corridors of racks on both sides. Figure 14 and Figure 15 display examples of corridors of type 1 and 2, respectively. It should be noted that the placement of the objects this way only intends to make it simpler to understand the way the corridors work.

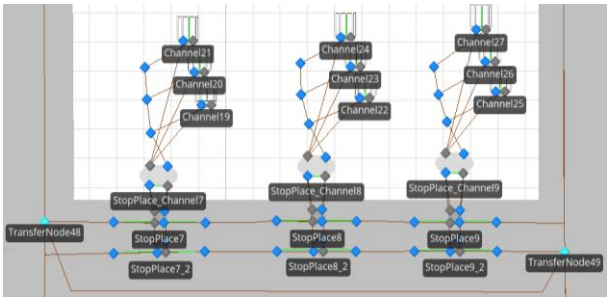

Figure 14: Example of a corridor of type 1

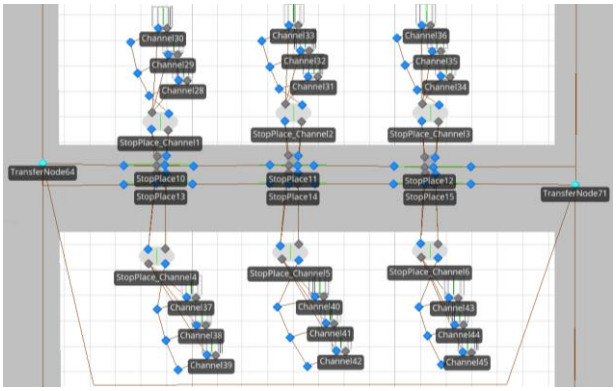

Figure 15: Example of a corridor of type 2

As can be seen, regardless of the type of corridor, both have two entry nodes. When the milk run enters one of the entry nodes of a corridor, it executes the process represented on Figure 16.

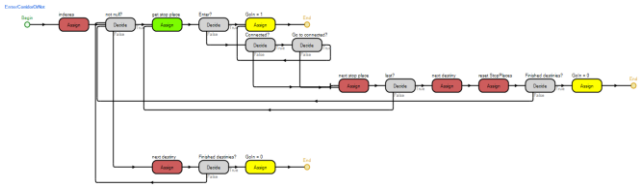

Figure 16: Process EnterCorridorOrNot

This process intends to evaluate if a milk run enter a corridor, or not, by evaluating if a StopPlace of that corridor belongs to the array of destinies of the milk run, or not. Since the token needs to know if the StopPlace being evaluated is the last of the corridor, the property LastOfCorridor needs to be checked. Lastly, in the possibility of being a type 2 corridor the ConnectTo property of all StopPlaces needs to be analysed. If there is an object specified on that property, the token also needs to evaluate if it exists on the array of destinies of the associated milk run.

Once inside a corridor, the milk runs enter a succession of StopPlaces, where each one accesses a different set of Channels. To better understand the objects that need to be used on a corridor of type 1, Figure 18 was created. Once again, the placement of the objects the way the figure displays only intends to make it simpler to understand the way they work and thus, it is not the final result of the animation of the model.

As can be seen, for any circulation direction, there is a TransferNode before and another after a StopPlace. Thus, on the first node, the process illustrated on Figure 17 is executed.

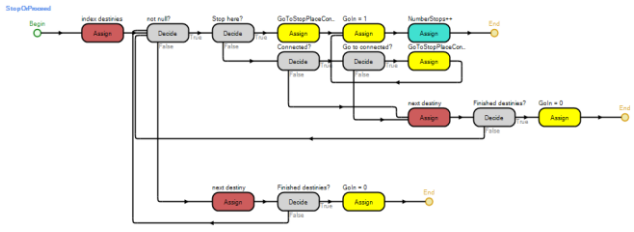

Figure 17: Process StopOrProceed

In this process, the milk run checks if the StopPlace belongs to the array of destinies of the milk run. Once the process ends, the milk run will select the Path based on the value on a state of the picker. For instance, considering that a milk run enters the TransferNode42 (from Figure 18) and executes the StopOrProceed process, if its GoIn state has the value 1, the milk run will select the Path that takes it to StopPlace8. Otherwise, it will choose the Path that takes it directly to the TransferNode43.

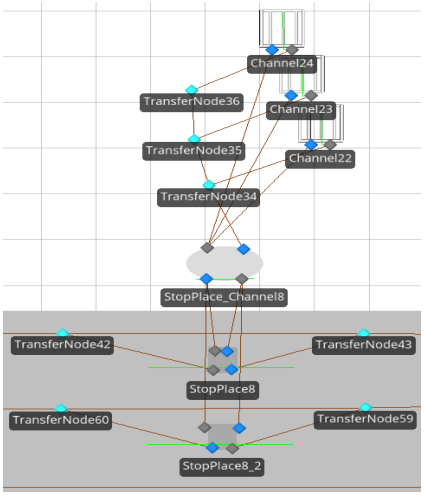

Figure 18: Objects used alongside a StopPlace and the corresponding set of Channels on a corridor of type 1

When a milk run enters a StopPlace, it will wait for the respective Picker to return from the set of Channels. In this context, if the corridor is of type 1 (e.g. Figure 18), the Picker will chose the Path that takes it to StopPlace_Channel8. However, if the corridor is of type 2, the Picker will choose its destiny based on the value on its GoToStopPlaceConnected. To help clarify this situation, Figure 19 was created.

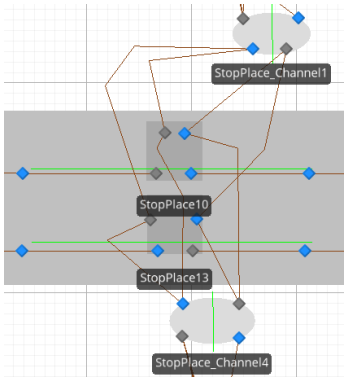

Figure 19: Pair of StopPlaces of a corridor of type 2

The Picker always returns to the StopPlace where its milk run is waiting. As soon as the Picker enters a StopPlace_Channel object, the remaining logic until it returns to it, is the same for both types of corridors.

Considering Figure 18 again, it is possible to see that there is only one TransferNode that gives access to a Channel (e.g. TransferNode 34 to Channel22 and TransferNode35 to Channel23 and) and, after leaving a Channel, the Picker will necessarily return to the

StopPlace_Channel object, i.e., it can only take one Container at a time. When a Picker enters a node that gives access to a channel, it executes the process presented by Figure 20.

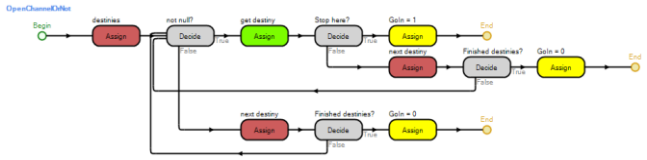

Figure 20: Process OpenChannelOrNot

The purpose of this process is to evaluate if the Channel in question belongs to the array of destinies of the Picker. To that end, the associated token consults the Channels data table on the index returned by the previously updated target state of the Picker and evaluates whether the returned object is one of the destinies of the Picker or not. If it is a destiny, the token assigns the value 1 to the GoIn state of the Picker, otherwise, the value 0. Afterwards, the Picker selects the next Path, based on the value of its GoIn state. Thus, if the value is 1, it selects the Path that takes it to the Channel. Conversely, if the value is 0, it selects the Path that takes it to the next TransferNode, updating its target state again, once on this Path. It should be noted that, since the milk run only enters a StopPlace if any of the Channels (that the StopPlace gives access to) is a destiny of the Picker, it is guaranteed that the Picker will at least enter one Channel.

Once inside a Channel object, the Picker selects the required Container and adds it to its batch. After leaving the Channel, the object is removed from its array of destinies. Then, the Picker returns to the StopPlace_Channel and, after that, to the StopPlace.

Naturally, before leaving the StopPlace object, the Picker needs to be transferred to the riding station of its milk run. Therefore, on the Facility of the StopPlace, there was the need to use a GoTomilk run object (Figure 6). On its Process property, the name of the process displayed by Figure 21 is inserted.

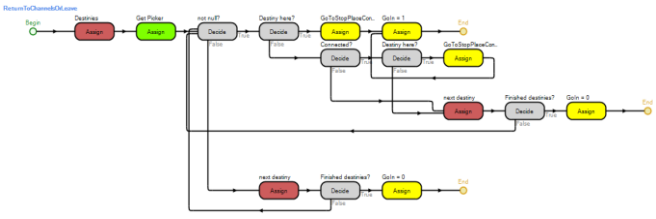

Figure 21: Process ReturnToChannelsOrLeave

The purpose of this process is to verify if the Picker needs to return to the set of Channels or not. To that end, the associated token verifies the StopPlace property of every Channel on its array of destinies. If any of those properties has a value equal to the Place property of the StopPlace where the Picker is at, the token saves the value 1 to the GoIn state of the Picker. Otherwise, it saves the value 0. Additionally, if the StopPlace has an object on its ConnectTo property, the token needs to repeat the verification to that object. This way, the GoToStopPlaceConnected state of the Picker will be updated, to ensure that the Picker chooses the Path that takes it to the correct StopPlace_Channel.

Once the Picker has placed all the required Containers on the batch of the milk run, the latter removes the StopPlace from its array of destinies and resumes its route. When all the Containers from all the corridors have been collected, the milk run returns to the start point to restart a new shift.

3.3. Automatic Creation of Simulation Models

To make it simpler for the user to introduce the data related to the warehouse he wants to create, it was established that he would only have to introduce the data on an Excel spreadsheet. Table 1 shows an example of the content of the mentioned file and in this section the cells that the user needs to fill will be covered.

Table 1: Input Excel table

New corridor?	Size Length	Width	Height	Coordinates x	y (z in Simio)	Symbol index	Directions	Rack description	Channels per column								
1	0,23	0,42	0,58	-50	-50	0	2	AP	3	3	3	3	3	3	3	3	3
0								AO	3	3	3	3	3	3	3	3	3
0								AN	3	3	3	3	3	3	3	3	3
0								AM	3	3	3	3	3	3	3	3	3
0								AL	3	3	3	3	3	3	3	3	3
0								AJ	3	3	3	3	3	3	3	3	3
0								AK	3	3	3	3	3	3	3	3	3
0								AI	3	3	3	3	3	3	3	3	3
0								AH	3	3	3	3	3	3	3	3	3
0								AG	3	3	3	3	3	3	3	3	3
0								AF	3	3	3	3	3	3	3	3	3
0								AE	3	3	3	3	3	3	3	3	3
0								AD	3	3	3	3	3	3	3	3	3
0								AC	3	3	3	3	3	3	3	3	3
0								AB	3	3	3	3	3	3	3	3	3
0								AA	3	3	3	3	3	3	3	3	3
2								BE	3	3	3	3	3	3	3	3	3
0								BD	3	3	3	3	3	3	3	3	3
0								BC	3	3	3	3	3	3	3	3	3
0								BB	3	3	3	3	3	3	3	3	3
0								BA	3	3	3	3	3	3	3	3	3
0								AZ	3	3	3	3	3	3	3	3	3
0								AY	3	3	3	3	3	3	3	3	3
0								AX	3	3	3	3	3	3	3	3	3
0								AW	3	3	3	3	3	3	3	3	3
0								AV	3	3	3	3	3	3	3	3	3
0								AU	3	3	3	3	3	3	3	3	3
0								AT	3	3	3	3	3	3	3	3	3
0								AS	3	3	3	3	3	3	3	3	3
0								AR	3	3	3	3	3	3	3	3	3
0								AQ	3	3	3	3	3	3	3	3	3

In order to allow the user to specify any number of racks per corridor, it was established that on each line of the excel file, the user inserts data related to a single rack. Therefore, to start a new corridor, the user has to enter the value “1” on the column “New corridor?”. Conversely, if the user wants to keep adding racks to a corridor, he just has to keep entering the value “0” on the corresponding rows, on the same column. Additionally, for each corridor, the user can choose one of two types: a simple corridor, which is comprised by one or more racks; and a set of two corridors that are disposed inwards, so that a milk run traveling it may collect containers from both corridors of its left and right. To make it simpler to refer to these corridors, on the remaining sections of this document, these will be referred as simple and double, respectively. In this sense, to specify a double corridor, the user needs to assign the value “2” to the row corresponding to its first rack.

In the columns “Size” and “Coordinates”, the user can specify the size of the channels (length, width and

height) and the position on which the corridors starts to be built. These values are only read if the user entered the value "1" on the "New corridor?" column of that row, since it was assumed that this information does not vary in the same corridor. The same approach applies for the "Symbol index" and "Directions" columns. On the first, the user can specify a symbol, from an array of symbols, to be assigned to the channel. The only difference between the symbols on this array is its rotation angles. This approach had to be considered, since the API of Simio does not provide methods for rotating a fixed object and, for animation purposes, it was very important to rotate the corridors and its channels. However, this approach has a couple of flaws. Firstly, since the waiting queue of the object is not considered part of the symbol, it is not "rotated", i.e., despite the fact that a different symbol is assigned to an object, its queue remains with the rotation as the original. Lastly, the possible rotation angles have to be previously assigned. For this case, rotations of 45 degrees were considered (e.g. 1 means a rotation of 45 degrees, 2 means a rotation of 90 degrees and so on). On the "Directions" column the user can define the number of ways through which the milk runs can travel on the corridor. On the last column, "Channels per column", the user can define any number of columns per rack and any number of channels per column, depending on the number of cells that have values and the values on each of those cells, respectively. On the "Rack description" column, the user can specify a string that, as the name implies, indicates the rack description of the rack in question.

To create an object using the Simio API the user needs to call the CreateObject method. This method takes a string and a FacilityLocation as arguments. The later defines the coordinates x, y and z in Simio and the first is the name of the object that is supposed to be created on the specified location. This object can be any one of the Standard library of Simio, any other created by a user (e.g. a sub model) or even the object that represents an entity or a worker. Thus, to create the developed Simio sub-models, which have already been discussed [6], this method is used. Notwithstanding, to create a path, a conveyor, a time path or a connector between objects a different method is used, even though these are also objects in Simio. In these cases, the method CreateLink has to be used. Examples of both methods are given below:

```
createObject("Channel",new FacilityLocation (xx,yy,zz)) as IFixedObject;
CreateLink(String, INodeObject, INodeObject, IEnumerable<FacilityLocation>)
```

As can be seen, this method takes a string, two INodeObjects and a collection of FacilityLocations as arguments. The first corresponds to the object being created, while the following two arguments correspond to the two nodes the method is supposed to connect. Lastly, the collection of FacilityLocations is a list of coordinates used to create the vertexes of the object. If the user does not want to specify any vertexes, the value *null* can be passed through this argument.

Apart from creating objects, the Simio API may also be useful for other reasons, such as editing object properties. In many cases, to accomplish this, it is necessary to know the name of the property and use the following code line:

```
Object.Properties["PropertyName"].Value="NewValue";
```

However, there are some properties that require other means to edit them, like the name of the object, its size, symbol index, location, among others. Nonetheless, knowing the name of the property in question is not always a simple task, due to the lack of information concerning the Simio API available. In fact, when a user interacts with the tool and edits an object property, the name presented by Simio for that property is actually the display name. To confirm this situation Figure 2 shows the properties inherited by an object of the standard library of Simio.

As can be seen, the name of the selected property is "EnteredAddOnProcess", while its display name is "Entered". Thus, to learn the name of this property, the user would have to access the list of properties of the object and check its name, which is very troublesome.

To create different orientations for the corridors, or simply to create two corridors faced inwards, composing a single corridor, it would be necessary to use a Simio method that could rotate an object, just like it is possible to do when interacting with the tool itself. However, the API does not provide any method for this task, so other workarounds were considered. The solution adopted for this task was to assign different symbols to the objects, each one representing a different rotation angle. Nonetheless, this does not affect the queue of the objects. This fact can be seen on Figure 6 and on Figure 7 (chapter 4), where all the queues, of all the channels, of the two faced inwards corridors, are facing the same direction. Thus, the queues of the channels on the second set of channels are facing an opposite direction to where the pickers and the milk runs travel.

When the add-in starts its execution, all the data is read from the excel spreadsheet to avoid having to make multiple communications with the application. The method created to that end is given below.

```
public string[,] getData(int firstRow){
  Excel.Application app=new Excel.Application();
  Excel.Workbook workbook = app.Workbooks.Open("C:\\Path\\file.xlsx");
  Excel.Worksheet sheet = (Excel.Worksheet) workbook.Worksheets.get_Item(1);
  Excel.Range range = sheet.UsedRange;
  string[,] data = new string [range.Rows.CurrentRegion.EntireRow.Count -
  firstRow+1,range.Columns.CurrentRegion.EntireColumn.Count];
  for(int i=firstRow;i<= range.Rows.CurrentRegion.EntireRow.Count;i++){
    for(int j=1;j<=(range.Rows[i] as Excel.Range).CurrentRegion.EntireColumn.Count; j++){
      if((range.Cells[i,j] as Excel.Range).Value2!=null)
        data[i-firstRow,j-1]=(range.Cells[i,j] as Excel.Range).Value2.ToString();
      else data[i- firstRow,j-1]="NoData";
    }
  }
  workbook.Close();
  excelApp.Quit();
  return data;
}
```

As can be seen, the variable app is used to start Excel. Afterwards, the workbook variable opens the

intended excel file, by providing it with the correct path. Lastly, the sheet variable accesses the pretended worksheet (the first of the opened workbook) and the range variable gets the range currently being used. At this point, to read data from a cell of the opened sheet, it was necessary to use the following expression:

```
(range.Cells[i, j] as Excel.Range).Value2.ToString()
```

As the purpose of this method is to save the data contained on the excel sheet to a multidimensional array, the remaining code lines search through the cells with content and saves its string value to the respective position on the array to be returned. Once all the data is read, the communication with Excel can be terminated.

After retrieving the data, the add-in can start building the supermarket. In this section, the code for this task will be explained as pseudo-code, given below.

```
Executes getData() method
For each row read from the excel file{
  If data[countn, 0] ← 1{
    Read size of the Channels
    Read coordinates, symbol index (rotation based on this) and number of ways of corridor
    CorridorData ← Execute GetCorridorData() mathod
    do for i = 0 to 1 times{//One for each Position of the CorridorData array
      do for each list RACK in in CorridorData[i]{
        do for each node in RACK{
          if first node of RACK then save the rack Description
          else {
            creates StopPlaces and edits their properties
            create other needed objects and edits their properties
            if last StopPlace then draw Paths between corridors and edits their properties
            ChannelsToDraw ← read the number of Channels accessible by the drawn StopPlaces
            do for j = 0 to ChannelsToDraw{
              if first loop then{
                Create StopPlace_Channel
                create other needed objects and edits their properties
              }
              else{
                create Channel and edit the properties
                create other needed objects and edits their properties
              }
            }
          }
        }
      }
    }
  }
  Add objects to their respective object tables
}
```

As can be seen, the algorithm runs through the retrieved multidimensional array of strings, with the contents retrieved from the excel spreadsheet, and searches for the value “1” on the first column of every row it searches. Once it finds it, executes the GetCorridorData method, which is displayed below.

```
public List<List<string>>[] GetCorridorData(int n, string[,] data){
  List<List<string>>[] corridor = new List<List<string>>[2];
  int j = 0, flag = 1, foundDouble = 0;
  List<string> channels = new List<string>();
  List<List<string>> racks0 = new List<List<string>>(), racks1 = new List<List<string>>();
  while (((n + 1) <= data.GetLength(0)) && ((data[n, 0] != "1") || (flag == 1))) {
    flag = 0;
    j = 5;
    while ((j < data.GetLength(1)) && (data[n, j] != "NoData")){
      channels.Add(data[n, j]);
      j++;
    }
    if(data[n,0]=="2")
      foundDouble = 1;
    if (foundDouble == 0)
      racks0.Add(channels);
    else
      racks1.Add(channels);
    channels = new List<string>();
    n++;
  }
  corridor[0] = racks0;
  corridor[1] = racks1;
  return corridor;
}
```

The purpose of this method is to get all the information related to a corridor and store it on a single data structure. This method had to be used, since the way defined to build a simple corridor is different from the way defined to build a double one. Moreover, to make it simple for the user to introduce data on the excel spreadsheet, he only needs to assign the value “2” on the first rack of the second corridor of the double corridor. Thus, to know if the corridor in question is a simple one or a double one, it is necessary to read all the rows belonging to the same corridor.

To store the data related to a corridor, the authors defined an array with only two positions of lists of lists of strings. The strings are the data retrieved from the excel spreadsheet, while the list of strings (channels in the code given above) stores the data related to the number of channels to create, per rack (values of the column “Channels per column” of Table 1). All the information related to racks belonging to the same corridor is stored on the remaining list (racks0 on the code above). Nonetheless, if the value “2” is found, the values are saved on a different list (racks1 in the code above.). After running through all the rows of a corridor, the two lists are saved on the respective array positions, and the final data structure is returned. Once again, considering the data on Table 1, the data structures resulted from executing the GetCorridorData for the first corridor is illustrated on Figure 3.

4. DATA INPUT AND VALIDATION

The system being modelled consists on an advanced supermarket, **supermarket**, which is located near the production lines and stores a number of containers in each **channel** (shelf-like structure that stores containers in depth, usually in FIFO order). In its turn, each container can store several product units of a single type. Containers are sent to the supermarket, for later being collected by pickers that travel through the supermarket, driving milk runs. After collecting the intended containers, the pickers deliver them to the respective production lines. These consume the required material and, when it is necessary to start consuming a different type of product, a reference change occurs. In some cases, this phenomenon can result on a container being returned to the supermarket with the leftover product units inside of it.

Through many meetings, the authors were able to obtain the data required for the simulation model to efficiently model the system in analysis. Among others, values for the speed of the milk run, the picker, devolution rates, production times, number of shifts per day, number of production lines, time to remove containers from their channels and others were collected. This process is important, since it increases the confidence level in the developed model.

To build a supermarket with a layout corresponding to the real one, the authors used the developed Simio add-in that automatically creates simulation models ready to perform simulation experiments (Vieira et al., 2015b).

This add-in receives data input from excel spreadsheets, in which the user can specify any physical layout.

The created supermarket corresponds to a single corridor of two sets of channels that can be accessed by a picker who travels in between them. This supermarket corresponds to a size of 930 channels. Each channel has the capacity to hold 6 containers. More examples of automatically created simulation models, using the Simio add-in developed for this project, can be found online (Vieira et al., 2015b).

After analysing the raw data provided by the company, the authors were able to produce, using VBA, the required excel files that would "feed" the developed simulation model with real data. To do so, this data was imported into Simio. Figure 22 shows a diagram that represents the mentioned data and its purpose in the system in analysis.

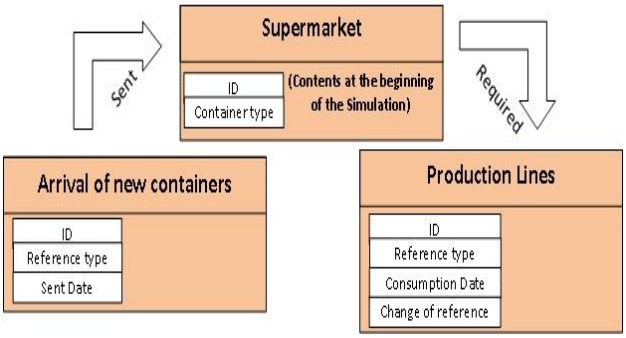

Figure 22: Representation of the system being analysed

Each of the 3 big areas, displayed in the diagram, represent a different excel spreadsheet imported into Simio. The data contained in those excel files is described inside each area. To specify containers that are located in the supermarket, at the beginning of the simulation (Supermarket area at the centre of the diagram), the excel files contain a unique identifier of each container and the correspondent type of that container. Similarly, to specify containers that are sent into the supermarket, the excel file must contain the unique identifier of each container, the type of the referred container and the date on which it is supposed to be sent into the supermarket. Lastly, the excel files that describe the containers required by the production lines must contain the unique identifier of each container, its type, the date on which it is required and a Boolean that indicates whether each container will originate, or not, a reference change. This data is important to indicate if a specific container has a chance of being returned to the supermarket, or not.

5. SIMULATION RESULTS AND ANALYSIS

In this section, the modelled storage strategies will be introduced. Many references to these strategies will be made throughout the paper. Therefore, the authors assigned short names to them that can be consulted in Table 2.

Table 2: Storage strategies definition

Short name	Storage Strategy
A	• Single-product channels;
C	• Multi-product channels; • Driven by consumption;

Strategy A corresponds to the one that is currently being used on the supermarket of the case study, on which the channels are dedicated (**single-product**). This is the most simple case, since it consists on having channels dedicated to a single type of container (Bartholdi and Hackman, 2008). One of its great advantages, resides on the fact that, since the locations of the containers do not change, pickers can memorize them, making the picking process more efficient (Bartholdi and Hackman, 2008). Moreover, it should be expected that single-product strategies require a higher quantity of channels to work, since it does not store different types of product on the same channel. In other words, the problem with this strategy is that "**it does not use space efficiently**. In fact, it is expected that, on average, the storage capacity is about 50%" (Bartholdi and Hackman, 2008), which represents a high amount of costs associated. To overcome this problem, other strategies can be considered. Figure 23 displays the simulation running, while modelling a single-product storage strategy, where different colours were assigned for each type of container. As can be seen, all containers stored within the same channel have the same colour.

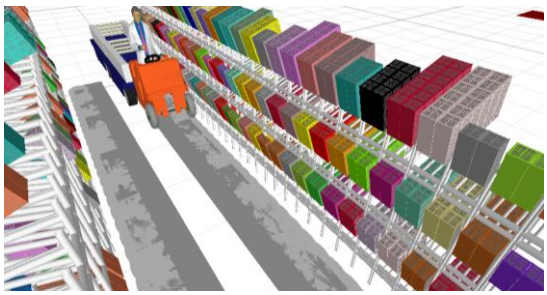

Figure 23: Single product storage strategy

Alternatives to strategy A would have to allow containers of different types to be mixed within the same channel (**multi-product**), whereby some companies oppose to its implementation. The main reason for this is that the Information System (**IS**) would have to be much more complex, to avoid picking from the non-first position of a channel and to guide pickers to the proper channel (and possibly also to the right position), once they would no longer have the advantage of having memorized the location of the containers. In a situation where the IS cannot handle this issue, the pickers would have to search for the container through all the positions of all the channels of the supermarket, which would negatively affect the picking system. Considering the afore mentioned, the proposed alternatives for the company have to analyse if, by **allowing mixes** of containers of different types on each channel, the picking of the containers will always be made in the **first position** on each channel. In this sense, **strategy C**

consists on storing containers, based on their consumption date, by giving priority to the channels that already have containers of the same type. Thus, a container can be stored in a channel with containers of other types, as long as the container to be stored has a posterior consumption date. Additionally, and similarly to the previous storage policy, an established limit of the number of containers of different types allowed per channel has to be respected. Since consumption dates are exposed to prediction errors, the impact of eventual errors has to be made. Figure 24 shows the simulation model in execution, while modelling a multi-product storage strategy. As can be seen, in this case, containers of different colour can be seen within a same channel.

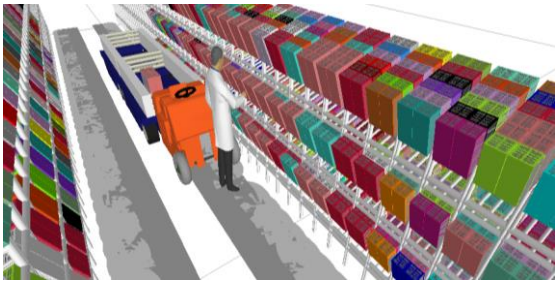

Figure 24: Multi-product storage strategy

Some simulation scenarios were defined for this problem, however only some of them will be presented in this chapter. The conducted experiments were run with a simulation time of one week. It should be noted, however, that the results presented within this chapter are directly dependent to the input data referred at chapter **Erro! A origem da referência não foi encontrada.**. As such, some of the conclusions withdrawn from this comparison should not be generalized, since it corresponds to a specific studied case.

The simulation experiments conducted in Simio considered several performance indicators: Nonetheless, the most important **KPI** (Key Performance Indicators) considered were: the average total time spent in a picking shift in seconds, the average position from which containers are removed from the channels (depth), the total amount of channels that were never used throughout the simulation and the average number of stops per milk run per picking shift. In its turn, the error assigned to the prediction errors of the consumption dates and the time interval of different types of containers in each channel were defined as the **properties** of the simulation experiments.

In an attempt to quantify the different simulation scenarios, rather than using an explicit **multi-criteria** approach, weights were assigned to the four KPI, to define a score that considers all KPI values of all scenarios. Thus, and taking into account the main objective of the company, the weights 3 and 2 were respectively assigned to the number of channels not used and depth KPI. The remaining KPI had a weight of 1.

Currently, the storage strategy being used at the company of the case study is the single-product one. This strategy is the one that has the lower number of properties that can be changed in our simulation model, since it allows a single type of container per channel. Table 3 shows the obtained results for this strategy.

As the results indicate, the pickers always collected the containers from the first position (depth), which is one of the perks of using this strategy. However, this affects the number of channels that were not used, which is lower than the same KPI on the remaining strategies, as will by shown in the next sections. Table 4 shows the obtained results for this strategy.

Scenario	Strategy	Time gap	Error	Milkruns	Different types of containers	Total time	Number of stops	Unused channels	Depth	Global Classification
4	A	0	Random.Uniform(0,0)	4	1	243,7	3,90	148	1,000	45%

Table 3: Simulation results for the modelled strategy A

Scenario	Strategy	Time gap	Error	Milkruns	Different types of containers	Total time	Number of stops	Unused channels	Depth	Global Classification
73	C	0	Random.Uniform(0,0)	4	6	214,9	1,72	473	1,000	91%
78	C	6	Random.Uniform(0,0)	4	6	215,6	1,75	428	1,000	87%
79	C	12	Random.Uniform(0,0)	4	6	215,0	1,73	403	1,000	85%
80	C	24	Random.Uniform(0,0)	4	6	215,2	1,73	401	1,000	84%
81	C	48	Random.Uniform(0,0)	4	6	215,9	1,78	357	1,000	80%
82	C	0	Random.Uniform(-01,01)	4	6	219,4	1,73	472	1,044	87%
83	C	6	Random.Uniform(-01,01)	4	6	215,3	1,74	425	1,000	87%
84	C	0	Random.Uniform(-12,12)	4	6	239,2	1,71	467	1,242	68%
85	C	6	Random.Uniform(-12,12)	4	6	222,7	1,73	422	1,075	79%
86	C	12	Random.Uniform(-12,12)	4	6	218,1	1,77	396	1,024	81%
87	C	24	Random.Uniform(-12,12)	4	6	216,3	1,81	369	1,000	81%
88	C	0	Random.Uniform(-24,24)	4	6	246,5	1,75	461	1,307	60%
89	C	12	Random.Uniform(-24,24)	4	6	221,8	1,78	395	1,058	78%
90	C	24	Random.Uniform(-24,24)	4	6	218,0	1,82	369	1,015	79%
91	C	48	Random.Uniform(-24,24)	4	6	216,1	1,80	360	1,000	80%
92	C	0	Random.Uniform(-48,48)	4	6	251,0	1,80	441	1,345	55%
93	C	24	Random.Uniform(-48,48)	4	6	225,4	1,84	370	1,084	73%
94	C	48	Random.Uniform(-48,48)	4	6	219,4	1,84	336	1,025	75%
95	C	72	Random.Uniform(-48,48)	4	6	217,0	1,81	349	1,006	78%
96	C	96	Random.Uniform(-48,48)	4	6	216,1	1,79	351	1,000	79%
97	C	0	Random.Uniform(-72,72)	4	6	253,5	1,82	424	1,365	51%
98	C	24	Random.Uniform(-72,72)	4	6	231,8	1,83	364	1,150	66%
99	C	48	Random.Uniform(-72,72)	4	6	223,1	1,83	337	1,065	71%
100	C	72	Random.Uniform(-72,72)	4	6	219,7	1,83	337	1,031	75%
101	C	96	Random.Uniform(-72,72)	4	6	218,1	1,84	346	1,013	77%
102	C	120	Random.Uniform(-72,72)	4	6	216,2	1,79	357	1,002	80%
103	C	144	Random.Uniform(-72,72)	4	6	215,7	1,77	356	1,000	80%

Table 4: Simulation results for the modelled strategy C

When analysing these results, the first thing to consider is that, similarly to the previous strategy, to significantly affect the system, the gap should be in the order of the days, rather than hours. Another aspect that should be noted is that the gap between containers of different types stored in the same channel – only for the scenarios without prediction errors - mainly affects the number of unused channels. The consequence of this fact was already addressed in the previous storage policy. When analysing the impact of the property that defines the prediction errors, the data showed that, when the errors were lower than the interval gaps, the depth values were always equal to 1 and the average picking time decreased

Assuming that a company can accurately predict the consumption date of their containers, scenario 73 (global score of 91%) can be considered the best solution. In

comparison to scenario 4 (the policy current being used at the case study), scenario 73 corresponds to a reduction of roughly **30 seconds** per trip (12% of reduction) on the average time per picking shift, **2 stops** per picking shift and per milk run (reduction of roughly 55%) and a reduction of around **69%** in the supermarket size (average difference of about 325 channels). All these gains were achieved by maintaining the rule stating that containers should be collected from the first position of any channel.

6. CONCLUSIONS

Warehouses are critical to a wide range of customer service activities and yet, they are also quite significant from a cost perspective. One of the goals of the Bosch Production System (BPS), implemented at Bosch, is to provide "the basis for continuous improvements in quality, costs, and supply performance" (Bosch, 2014). Thus, the opportunity to develop a **micro simulation** model in **Simio** that could help the Bosch Car Multimedia Portugal in Ferreiros, Braga arose. Particularly, this tool needs to be able to design several layouts of the supermarket and use them to test different scenarios of their **picking system**. In this on-going work, the present paper documents what was done to model the picking system observed at the Bosch Car Multimedia Portugal.

With the developed model, practitioners may benefit by using it to model different types of **warehouses**, not only supermarkets. Since the simulation model can be automatically created, the user only needs to insert the data correspondent to the layout and generate the intended simulation model. Afterwards, the model can be used to test different scenarios for the warehouse. Researchers may also benefit from the tool by using it to simulate different types of warehouses. The quality of the **animation** is quite perceptive, as the several figures illustrated throughout the document suggest.

Nonetheless, while interacting with Simio, some downsides were noted. Vieira et al. had already stated some of them (Vieira et al., 2014b). Moreover, the very useful expression editor feature that Simio offers, is not always enabled. For instance, on an Assign step, to define the StateVariableName property, the user can only select the state from a limited list of options. While it is true that it keeps it simpler for new users, it is also troublesome to have to use the expression editor where it is enabled to write a complex expression and then copy it to the actual place we want to use it. This is also true for other properties such as the StationName property of a Transfer step.

ACKNOWLEDGMENTS

This work has been co-supported by SI I&DT project in joint-promotion nº 36265/2013 (HMIEXCEL - 2013-2015 Project) and by FCT – *Fundação para a Ciência e Tecnologia* in the scope of the project: PEst-OE/EEI/UI0319/2014.

REFERENCES

BAKER, P. & CANESSA, M. 2009. Warehouse design: A structured approach. *European Journal of Operational Research,* 193**,** 425-436.

BARTHOLDI, J. J. & HACKMAN, S. T. 2008. *Warehouse & Distribution Science: Release 0.89*, The Supply Chain and Logistics Institute.

BOSCH. 2014. *consulted online at: http://www.bosch.com/en/com/home/homepage.html* [Online]. [Accessed].

COSTA, B., DIAS, L. S., OLIVEIRA, J. A. & PEREIRA, G. Simulation as a tool for planning a material delivery system to manufacturing lines. Engineering Management Conference, 2008. IEMC Europe 2008. IEEE International, 28-30 June 2008 2008. 1-5.

COSTA, P., ALVES, A. C. & SOUSA, R. M. 2011. Implementação da metodologia Quick ChangeOver numa linha de montagem final de auto-rádios: para além da técnica SMED.

COYLE, J. J., BARDI, E. J. & LANGLEY, C. J. 1988. *The management of business logistics*, West Pub. Co.

DIAS, L., PEREIRA, G. & RODRIGUES, G. 2007. A Shortlist of the Most Popular Discrete Simulation Tools. *Simulation News Europe,* 17**,** 33-36.

GU, J., GOETSCHALCKX, M. & MCGINNIS, L. F. 2010. Research on supermarket design and performance evaluation: A comprehensive review. *European Journal of Operational Research,* 203**,** 539-549.

HLUPIC, V. Simulation software: an Operational Research Society survey of academic and industrial users. Simulation Conference, 2000. Proceedings. Winter, 2000 2000. 1676-1683 vol.2.

HLUPIC, V. & PAUL, R. 1999. Guidelines for selection of manufacturing simulation software. *IIE Transactions,* 31**,** 21-29.

MONDEN, Y. 1998. Toyota Production System – an integrated approach to Just-In-Time. Institute of Industrial Engineers, Norcross, Georgia.

PEGDEN, C. D. Simio: A new simulation system based on intelligent objects. Simulation Conference, 2007 Winter, 9-12 Dec. 2007 2007. 2293-2300.

PEGDEN, C. D. 2013. Intelligent objects: the future of simulation.

PEGDEN, C. D. & STURROCK, D. T. Introduction to Simio. Proceedings - Winter Simulation Conference, 2011 Phoenix, AZ. 29-38.

PEREIRA, G., DIAS, L., VIK, P. & OLIVEIRA, J. A. 2011. Discrete simulation tools ranking: a commercial software packages comparison based on popularity.

STURROCK, D. T. & PEGDEN, C. D. Recent innovations in Simio. Proceedings - Winter Simulation Conference, 2010 Baltimore, MD. 21-31.

VIEIRA, A., DIAS, L., PEREIRA, G. & OLIVEIRA, J. 2014a. COMPARISON OF SIMIO AND ARENA SIMULATION TOOLS. *ISC.* University of Skovde, Skovde, Sweden.

VIEIRA, A., DIAS, L., PEREIRA, G. & OLIVEIRA, J. 2014b. Micro Simulation to Evaluate the Impact of Introducing Pre-Signals in Traffic Intersections. *ICCSA.* University of Minho at Guimarães - Portugal.

VIEIRA, A., DIAS, L. S., PEREIRA, G. A. B., OLIVEIRA, J. A., CARVALHO, M. S. & MARTINS, P. 2015a. Automatic Generation of 3D Simulation Models: Warehouses Performance Boosting. *business sustainability.* **Póvoa de Varzim, Portugal**.

VIEIRA, A., DIAS, L. S., PEREIRA, G. A. B., OLIVEIRA, J. A., CARVALHO, M. S. & P., M. 2015b. Using Simio to Automatically Create 3D Warehouses and Compare Different Storage Strategies. *Faculty of Mechanical Engineering Transactions,* 43**,** 335-343.

WOMACK, J. P. & JONES, D. T. 1996. Lean Thinking. Siman & Schuster, New York, USA.

WOMACK, J. P., JONES, D. T. & ROOS, D. 1990. The machine that changes the world. Rawson Associates, NY

YILDIZ, H., RAVI, R. & FAIREY, W. 2010. Integrated optimization of customer and supplier logistics at Robert Bosch LLC. *European Journal of Operational Research,* 207**,** 456-464.