3D MICRO SIMULATION OF MILKRUNS AND PICKERS IN WAREHOUSES USING SIMIO

António Vieira[1], Luís S. Dias[2], Guilherme B. Pereira[2], José A. Oliveira[2], M. Sameiro Carvalho[2] and Paulo Martins[2]

(1) (2) University of Minho, Campus Gualtar, 4710-057, Braga, Portugal.
(1) antonio6vieira@gmail.com, (2) lsd/gui/zan/sameiro/pmartins@dps.uminho.pt

KEYWORDS
Warehouse, milkrun, picker, Micro simulation, Simio, 3D.

To help the Bosch Car Multimedia Portugal in Ferreiros, Braga to reduce its costs (both in time and space) with its warehouse, a micro simulation model is being developed in Simio. Particularly, the tool needs to be able to model pickers riding milkruns to collect containers of products, from a warehouse, to satisfy the needs of the production lines. In this sense, the storage strategy used on the warehouse, 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 milkruns and pickers needs to be adjustable. Additionally, to design the corridors of the warehouse in a configurable way, an Add-in in C#, using the API of Simio, is being developed. Thus, this paper intends to document the first part of the simulation model developed, which consists on the pickers receiving requests and riding their milkruns to collect the respective containers from the warehouse. Five different Simio models compose the main simulation model. Conclusions and future work are discussed.

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). The purpose of the BPS 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 warehouses (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 warehouse on the company Bosch Car Multimedia Portugal in Ferreiros, Braga, arose.

In this context, a micro simulation model, using Simio, is being developed. The tool needs to be able to model pickers riding milkruns to collect containers of products, from the channels of a warehouse, to satisfy the needs of the production lines. A Channel is the basic unit for storage in this warehouse. 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 warehouse 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 milkruns and pickers need to be configurable.

Additionally, the warehouse is composed by circulation corridors for milkruns 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 warehouse. 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 warehouse 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 milkruns to collect the respective containers from the warehouse. The return of the leftover containers and the restock processes are not yet modelled.

Chapter 2 presents a review over the analysed literature. In chapter 3, a description of the actual state at the case study is given. In chapter 4, the several modelling steps conducted to develop the simulation model are covered. Lastly, in chapter 5, the main conclusions of the conducted work, as well as some future work, are discussed.

2. LITERATURE REVIEW

2.1. The Case Study

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, warehouses represent a very important role on modern supply chains (Baker and Canessa, 2009).

In fact, "whilst warehouses 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 warehouses 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 warehouses 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).

2.2. Simio

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).

Simio offers 2 basic modes for executing models: the interactive and the experimental modes. In the first it is possible to watch the animated model, which is useful for building and validating the model. In the second, it is possible to define properties of the model that can be changed, in order to see the impact on the system performance (Sturrock and Pegden, 2010).

Notwithstanding the fact that this is a recent tool, it is already possible to find many studies that use this tool. Vik et al. (2010) used Simio to model a logistic system design of a cement plant. Vieira et al. also used Simio to model traffic intersections, so that they could evaluate the impact on the performance when pre-signals were introduced (2014b).

3. MODEL DEVELOPMENT

Throughout this chapter, 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. Additionally we will refer to the warehouse of the company as the supermarket.

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 Milkrun.

Figure 1: Symbol of the Picker entity

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

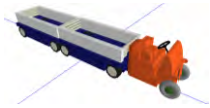

Figure 2: Symbol of the MilkRun entity

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

Figure 3: Symbol of the Request entity

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

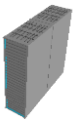

Figure 4: Symbol of the Container entity

3.2. GoToMilkRun

The only property defined for this model was a Process property named **Process**. Thus, one can use it to specify different processes to be executed at certain simulation times.

Since Containers and Picker travel on a MilkRun and Containers are added to the batch of the latter, there was a need to separate the Picker from the Containers, in order for the animation to become more realistic. Therefore, the model GoToMilkRun was created. Its only purpose is to transfer a Picker to the riding station of the respective MilkRun. This way, the Picker will seem to be riding the MilkRun, while the Containers will stay at the wagon of the MilkRun. Figure 5 presents the external view of the model

Figure 5: External view of the GoToMilkRun model

3.3. StopPlace

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 main goals of this model are: to model the MilkRun stopping; the Picker leaving the MilkRun to the set of Channels reachable from this StopPlace; the placement of the Container the Picker brought on the batch of the MilkRun and to evaluate whether the Picker needs to return to the Channels or not. The Facility of this model is presented on Figure 6 and its external view is presented on Figure 7. Apart from the nodes this model also has a queue to display the MilkRuns stopped on this model.

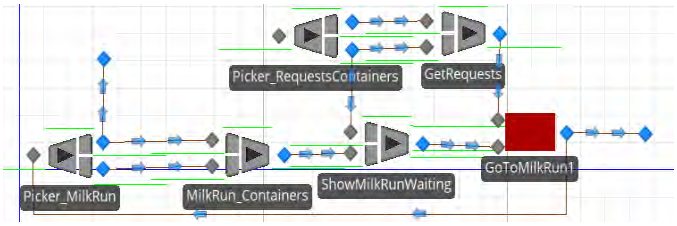

Figure 6: Facility of the StopPlace object

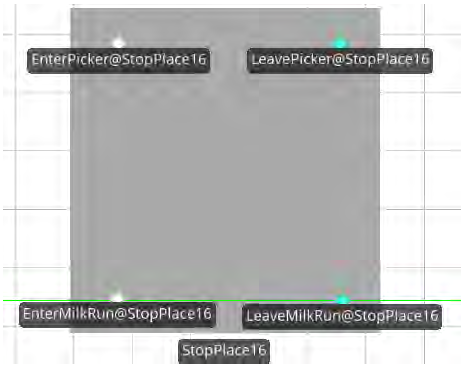

Figure 7: External view of the StopPlace model

3.4. StopPlace_Channel

No properties were defined for this model since its only purpose is to create a copy of a Picker that left the MilkRun and place it in front of the set of Channels. On the other hand, the original entity travels through the set of Channels and this model. Had this model not been created and during the animation, what would be seen, would be the Picker going completely inside the Channels, regardless of its height. Additionally, 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. As can be seen, apart from the nodes, the model also has a queue to display the copy of the Picker.

Figure 8: External view of the StopPlace_Channel model

3.5. Channel

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.

The main purpose of this model is to model the behaviour of the Pickers, when they analyse a channel of a warehouse to select the container they want. The external view of this model is presented on Figure 9. As can be seen, apart from the nodes, the model also has a queue to display the containers on this Channel.

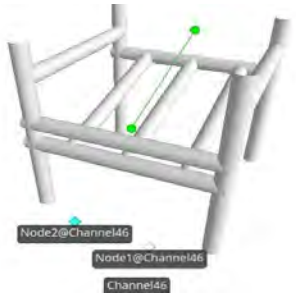

Figure 9: External view of the Channel model

3.6. Supermarket

On the Facility of this model, all the object that compose the supermarket itself will be placed. Those objects include instances of the previously presented models. As such, we will refer to those instances as objects, not models. Figure 11 shows an example of a set of Channels designed with the API.

The properties defined for this model were:

- **NumberMilkRuns:** Expression property that indicates the number of MilkRuns 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.

In order to access all the Channel and StopPlace objects placed on the Supermarket, two data tables were developed: Channels to gather all the Channel objects and StopPlaces to gather all StopPlace objects. Each object occupies an index of the data table correspondent to its Place or Position property, whether it is a Channel or a StopPlace. Additionally, the 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.

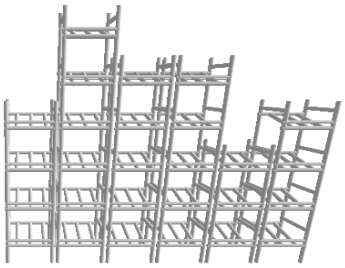
Figure 11: Example of a rack

3.6.1. Requests Creation

Since the production lines were not modelled, the arrival of requests to the system needs to be based on a distribution expression. Thus, to define its creation rate an exponential expression, with an average defined by the RequestsIntensity, was used. Lastly, to define the type of product the Request refers to, the following expression is executed, when a Request is created:

$$Math.Round(Random.Uniform(1, TotalProducts))$$

The number returned by the expression is saved on the ref state of the Request. To associate the number to a type of product, a list of products is imported from excel to the data table ProductData, at the beginning of the simulation run. Thus, the number returned by the expression corresponds to the data table row number.

After being created, the Requests enter a Combiner where they will be added to the batch of Pickers.

3.6.2. Creation of Pickers and MilkRuns

The creation of these types of entities only occurs at the beginning of the simulation run and their quantity is equal to the value on the property NumberMilkRuns. Lastly, when they are created, each entity is assigned with an identifier number that is unique for each pair of entities Picker and MilkRun. This way, a Picker is associated to a MilkRun and vice-versa.

3.6.3. Set Up the Shifts of the Pickers

After being created, the MilkRuns enter a GoToMilkRun object, where each will wait for the correspondent Picker. In its turn, after the creation of the Pickers, they enter on the ParentInput node of a Combiner. The capacity of this node is set to 1 so that no conflicts occur when adding Requests to the batch of the Pickers. Once a Picker is inside the node, the process illustrated on Figure 12, is executed.

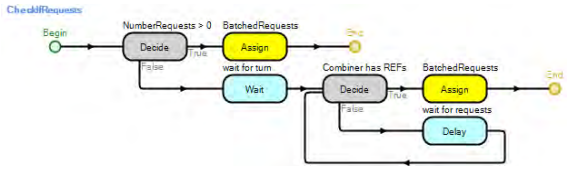

Figure 12: Process CheckIfRequests

The goals of this process are to make the picker wait for its turn and to specify the quantity of Requests (on the state BatchedRequests) to be added to its batch, taking into consideration that both depend on the NumberRequests property. In this sense, 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 13.

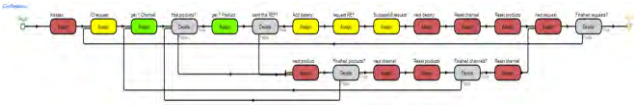

Figure 13: Process GetDestinies

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 GoToMilkRun object, where the corresponding MilkRun is. In this object the Picker will be transferred to the riding station of the MilkRun. Additionally, on the Process property of this object the value GetStopPlaces is inserted, i.e., the MilkRun will have an associated token execute that process. Figure 14 shows the process GetStopPlaces. Similarly to the process GetDestinies, this intends to save the StopPlaces where the MilkRun needs to enter, to an array of objects of each MilkRun. 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 MilkRun leaves the object and initiates its picking shift.

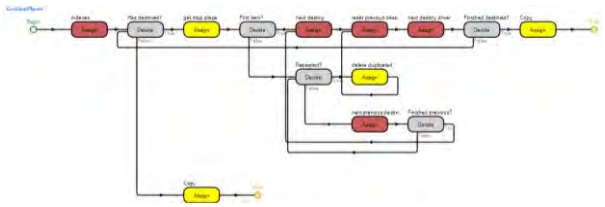

Figure 14: Process GetStopPlaces

3.6.4. Shifts of the Pickers

We prepared our simulation model to model two types of corridors. In the first, the MilkRuns only have access to corridors of racks on one side of the corridor. In the second type, they have access to corridors of rack on both sides. Figure 15 and Figure 16 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. In fact, the real model is constructed via the Simio API, even though this will not be addressed in this paper.

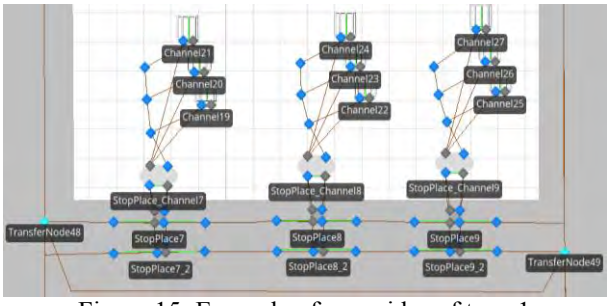

Figure 15: Example of a corridor of type 1

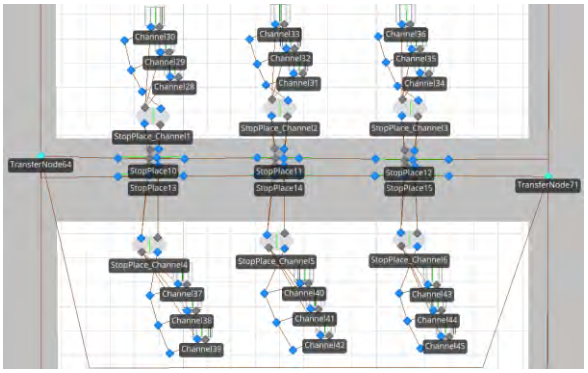

Figure 16: Example of a corridor of type 2

As can be seen, regardless of the type of the corridor, both have two circulation directions and two entry nodes. Considering Figure 15, it is possible to verify that TransferNode48 works as the entry node of one circulation direction (left to right) and TransferNode49 works as the entry node of the remaining circulation direction. The same logic is applicable to Figure 16. However, it is also possible to

model one-way corridors. To do that, it is only necessary to update the Corridors data table (Figure 10), on the NumberOfWays column and on the row correspondent to the corridor, to 1. For this situation, a Path that connects both entry nodes of the corridor was placed. This way, if it is supposed to be a one-way corridor, the MilkRun selects the Path that takes it to the right entry node. For instance, if the corridor from Figure 15 has only one way of circulation and a MilkRun enter TransferNode49, it would select the Path to the TransferNode48, in order to enter from the right.

All the Paths that take the MilkRuns to a TransferNode that works as an entry node of a corridor, update the state target of the MilkRuns to the value of the Place property of the first StopPlace of that corridor. After entering the node itself the process represented on Figure 17 is executed.

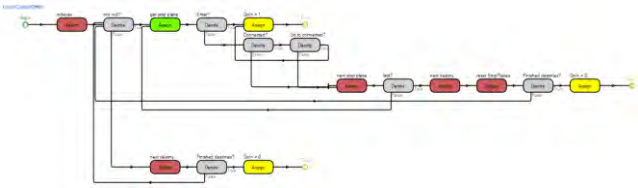

Figure 17: Process EnterCorridorOrNot

This process intends to evaluate if a MilkRun enter a corridor, or not. To do that, the previously updated target state of the MilkRun is used as an index of the StopPlaces data table. This way, the associated token can evaluate if a StopPlace of that corridor belongs to the array of destinies of the MilkRun. 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 MilkRun.

If any of the StopPlaces on the corridor belong to the array of destinies of the MilkRun, the associated token assigns the value 1 to the state GoIn of the MilkRun and it enters the corridor. Otherwise, the token assigns the value 0, the MilkRun ignores the corridor and goes to the next entry node of the next corridor where all the processes will be repeated.

Once inside a corridor, the MilkRuns 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 we can see, for any circulation direction, there is a TransferNode before and another after a StopPlace. Thus, on the Path that takes the MilkRun to the TransferNode situated before the StopPlace, the state target of the MilkRun is updated to the value of the Place property of that StopPlace. Thereafter, on the node itself, the process illustrated on Figure 19 is executed.

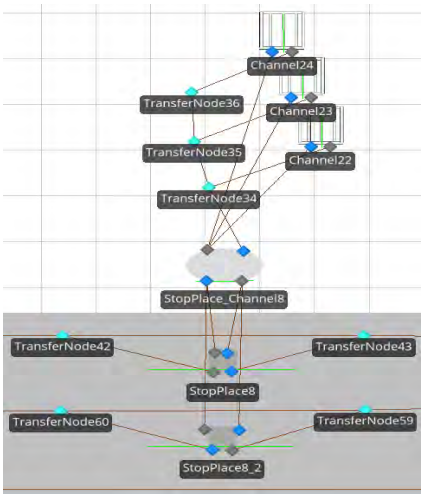

Figure 18: Objects used alongside a StopPlace and the corresponding set of Channels on a corridor of type 1

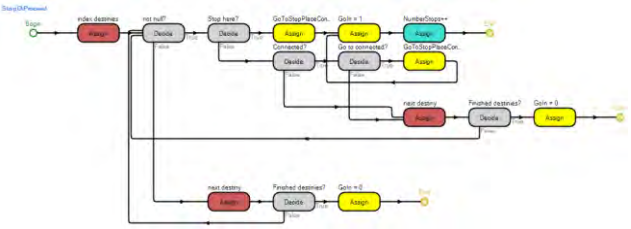

Figure 19: Process StopOrProceed

In this process, the token associated to the MilkRun, uses the previously updated state target as an index of the StopPlaces data table. This way, the token can check if the StopPlace belongs to the array of destinies of the MilkRun. If it belongs, the token assigns the value 1 to the GoIn state of the MilkRun and the value 0 to the GoToStopPlaceConnected state of the respective Picker. While analysing a StopPlace, the token also needs to verify if there is any object on the ConnectTo property. If there is and if it belongs to the array of destinies of the MilkRun, the token assigns the value 1 to the states GoIn of the MilkRun and GoToStopPlaceConnected of the respective Picker. If neither the StopPlace being analysed nor the one on its ConnectTo property belong to the array of destinies of the MilkRun, the token assigns the value 0 to the state GoIn of the MilkRun and ends the process.

Once the process ends, the MilkRun will select the Path based on the value saved on its GoIn state. For instance, considering that a MilkRun enters the TransferNode42 (from Figure 18) and executes the StopOrProceed process, if its GoIn state has the value 1, the MilkRun will select the Path that takes it to StopPlace8. Otherwise, it will choose the Path that takes it directly to the TransferNode43.

When a MilkRun 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 20 was created.

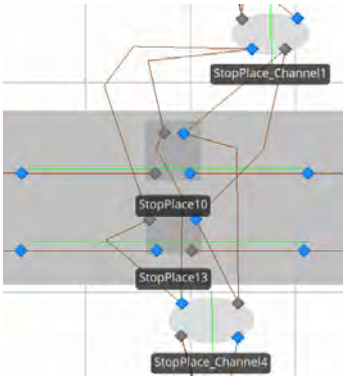

Figure 20: Pair of StopPlaces of a corridor of type 2

Considering as an example that a MilkRun is in StopPlace10, if the respective Picker has the value 0 in its GoToStopPlaceConnected state, it will go to the StopPlace_Channel1. On the other hand, if the value is 1, the Picker will go to StopPlace_Channel4. When the Picker returns with the selected Container, it will chose the Path based on the same logic. Therefore, the Picker will always return to the StopPlace where its MilkRun 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, we can see that there is only one TransferNode that gives access to a Channel (e.g. TransferNode 34 to Channel22 and TransferNode35 to Channel23…) 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 Path that takes it to a TransferNode that gives access to a Channel, its target state is updated to the value of the Position property of that Channel. Thereafter, when the Picker enters the node itself, the process presented by Figure 21 is executed.

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 MilkRun 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.

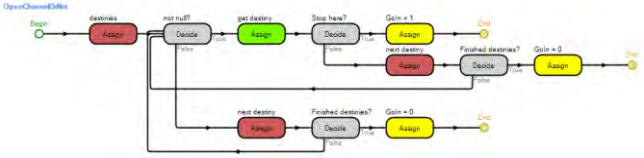

Figure 21: Process OpenChannelOrNot

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 MilkRun. Therefore, on the Facility of the StopPlace, there was the need to use a GoToMilkRun object (Figure 6). On its Process property, the name of the process displayed by Figure 22 is inserted.

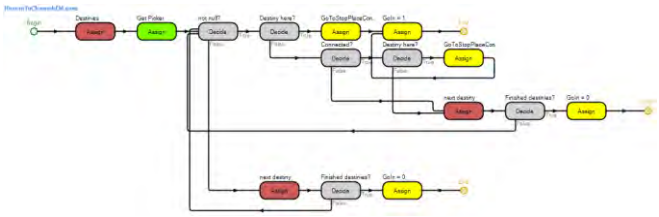

Figure 22: Process ReturnToChannelsOrLeave

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 MilkRun, 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 MilkRun returns to the start point to restart a new shift.

4. PRELIMINARY EXPERIMENTS

In this chapter, we will show some pictures that illustrate a shift of a Picker in runtime, so that it is possible to verify that the Picker collects the right Containers to satisfy the needs of the production lines. Thus, Figure 23 to Figure 27 were developed. Figure 23 shows a Picker waiting for Requests to initiate its shift.

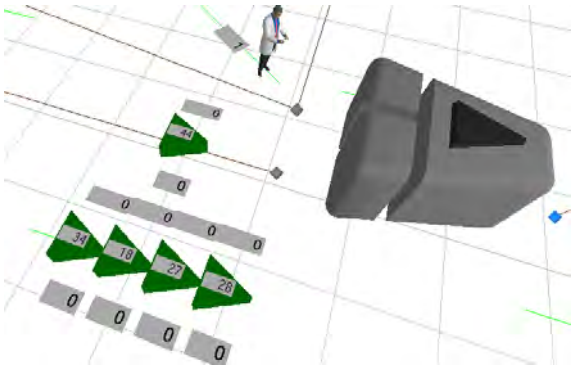

Figure 23: Requests being added to a Picker's batch

As can be seen, a label with the value of the ID of the Picker was added to that entity. To the Requests, three labels were assigned: one above it that identifies the Picker that got that Request; one inside it which identifies the type of product being requested and one below that indicates if the request has been satisfied or not (1 if it has and 0 otherwise).

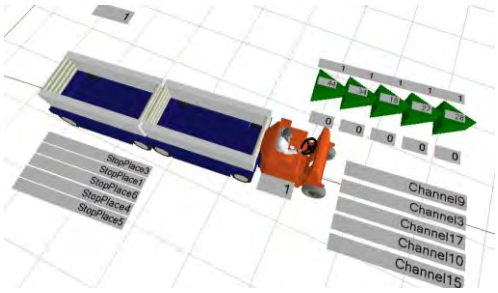

Figure 24: Destinies assigned to Picker and MilkRun

In Figure 24 we can see that the same Picker is now on the station of the MilkRun and it already has all the Requests in its batch. For that reason, the label above the Requests already has the same number as the ID of the Picker. Additionally, five new labels were assigned to the Picker to show its destinies. In its turn, the MilkRun also has a label above it, which shows its ID and five others below it that show its destinies. In this case, no repeated StopPlaces were assigned.

In Figure 25 we can see that the Picker already collected two Containers and the respective destinies were removed from the array of destinies of the Picker and of the MilkRun. Additionally, three labels were assigned to the Container entity: two above it that show the ID of the Picker and the type of products inside the Container; and one beneath it that show the Channel from which that Container was collected from. It should be noted that the Channels on these labels correspond to the objects removed from the array of destinies of the Picker. Lastly, we can also see that two labels that indicate if a Request has been collected, now have the value 1, indicating that the Container has been collected.

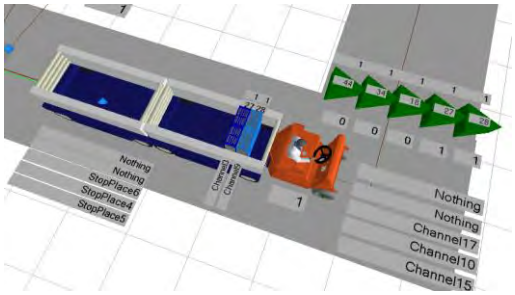

Figure 25: Half of the destinies visited

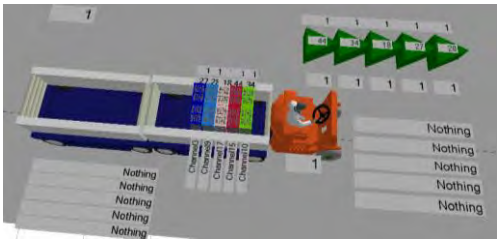

Figure 26: End of the shift

Figure 26 shows the same type of information of the previous image. However, this time the Picker has collected all Containers. By analysing all images, we can see that the Pickers always collect the right Containers from the Supermarket.

Regarding simulation experiments, of the key performance indicators (KPI) already defined, the only one that does not depend on real data is the Depth, indicated on Figure 27. This KPI indicates the position on the Channel where the selected Container was.

Name	Status	Required	Completed	NumberMilkRuns	StockPolicy	NumberRequests	TotalProducts	RequestsIntensity	Depth
Scenario1	Idle	3	3 of 3	10	1	5	45	0,05	1,47057
Scenario5	Idle	3	3 of 3	3	1	15	45	0,05	1,45038
Scenario6	Idle	3	3 of 3	1	1	1	45	0,05	1,85606
Scenario7	Idle	3	3 of 3	10	1	-5	45	0,05	1,47874
Scenario8	Idle	3	3 of 3	3	1	-15	45	0,05	1,31735
Scenario9	Idle	3	3 of 3	1	1	-1	45	0,05	1,1908
Scenario10	Idle	3	3 of 3	10	0	5	45	0,05	1,04493
Scenario11	Idle	3	3 of 3	3	0	15	45	0,05	1,00406
Scenario12	Idle	3	3 of 3	1	0	1	45	0,05	1
Scenario13	Idle	3	3 of 3	10	0	-5	45	0,05	1,07409
Scenario14	Idle	3	3 of 3	3	0	-15	45	0,05	1,05607
Scenario15	Idle	3	3 of 3	1	0	-1	45	0,05	1

Figure 27: Preliminary simulation experiments

As we see, the first six scenarios, present more random values than the last ones, that display values near 1. This represents what is expected to happen, since the first six scenarios correspond to a random storage strategy (StockPolicy = 1) and the last six scenarios correspond to a dedicated storage strategy (StockPolicy = 0). Nevertheless, for the last six scenarios, it would be expected that the values would be exactly 1. This happens because, as was explained in section 3.6.3, the Picker only collects the Container with a value on its Requested state equal to the ID of the Picker. Thus, in a few cases, even if the storage strategy is dedicated, the Picker will not take the first Container. The fact that those scenarios were executed (with considerable simulation time and replications) without errors also indicates that there are no modelling errors on the model.

5. 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 warehouse 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.

Since no real data was introduced on the model, no simulation experiments could be conducted. Despite that, we could see that the developed model is working as intended, since the **MilkRuns** stop where they should and the **Pickers collect** the **right Containers** to satisfy the needs of the production lines. Additionally, many

scenarios were executed, with considerable simulation time and replications, without errors.

Throughout chapter 3 many figures of the model were shown. Yet, the only purpose of those pictures was to make it easier to understand the way the model works and how it was developed. The final result of the animation is achieved by using the developed add-in, which was not addressed in this paper. However, the symbols of the Channel object (Figure 9) and of the entity types (Figure 1 to Figure 4) are good indicatives of the **quality** of **animation** that Simio offers.

Nonetheless, while interacting with Simio, some downsides were noted. Vieira et al. had already stated some of them (2014a). 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.

5.1. Future Work

Since this is an on-going work, there are some things that still need to be done, such as the **restock** and the **return of the leftover containers**. While the latter is not yet implemented, the former is implemented by creating Containers inside each Channel.

Another aspect that could be improved resides on the fact that the MilkRuns have a fixed route of corridors Other strategies could be pondered, such as smallest distance.

Once all the sub-systems that compose the picking system are finished, it would be mandatory to **gather real data** and **validate** the model, so that it becomes ready to perform simulation experiments.

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].

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 warehouse 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.

VIK, P., DIAS, L., PEREIRA, G., OLIVEIRA & ABREU, R. 2010. Using simio for the specification of an integrated automated weighing solution in a cement plant. *Proceedings of the Winter Simulation Conference.* Baltimore, Maryland: Winter Simulation Conference.

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.