António Vieira
Researcher, *

Luís S. Dias
Assistant Professor, *

Guilherme A. B. Pereira
Associated Professor, *

José A. Oliveira
Assistant Professor, *

M. Sameiro Carvalho
Associated Professor, *

Paulo Martins
Assistant Professor, *

*: University of Minho
Production and Systems Department

USING SIMIO TO AUTOMATICALLY CREATE 3D WAREHOUSES AND COMPARE DIFFERENT STORAGE STRATEGIES

This paper focuses on a simulation based approach to reduce warehouse costs. At an early stage, the tool needs to be able to generate different types of warehouses. To accomplish this, a Simio add-in was built in C#, using the Simio API, where the user only needs to insert the layout data on an excel spreadsheet. Afterwards, the created warehouse is capable of modelling different storage strategies and compare them. The obtained results indicate that the proposed strategy is able to reduce the picking time in about 15% and the number of stops per milk run in 50%. Moreover, it was found that the strategy currently in use needs 35% more space than the proposed one.

Keywords: *Warehouse, milk run, picker, Simulation, Simio, 3D, API, Excel, C#.*

1. INTRODUCTION

In recent years, the Bosch Group has been applying concepts of the Toyota Production System (TPS) [1] and of the Lean Manufacturing [2, 3], designated as Bosch Production System (BPS). The purpose of the BPS is to "eliminate waste in production and all related business processes. Thus, BPS provides the basis for continuous improvements in quality, costs, and supply performance" [4].

A significant part of the costs of a company are related to its warehouses [5]. 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 a company of the Bosch Group, arose. This warehouse is comprised by corridors through which the pickers ride the milk runs to collect containers of products to satisfy the needs of the production lines. A corridor is a set of racks, which in its turn is a set of channels, where the containers, that hold several units of products of a single type, are placed. In this context, a simulation model, using Simio, is being developed. Among other parameters, the tool must allow several properties to be parameterized, such as: different storage strategies, types of products, quantity of requests a picker gets per trip, time between trips, arrival rate of requests, the number of milk runs and pickers, the layout of the warehouse, among others. The principal steps conducted to model the logic of the system have already been documented [6].

Thus, the main purpose of this work is to use a simulation tool, developed in Simio that allows to automatically create different storage layouts and to compare different storage strategies for a warehouse of the Bosch Group. In a traditional approach and due to many reasons, such as the re-adaptation to new products, new clients, etc., it may be necessary to perform some changes on a model in order to maintain it updated, or simply to accurately respond to new scenarios. This is a process that, made manually can be very time consuming. Consequently, the need to automatically design warehouses, using Simio, arose.

Simio provides an API (Application Program Interface), allowing the users to use their methods, classes and others. Thus, an add-in for Simio was created-using C#. By executing it, it is possible to automatically create and place sets of Simio objects, which collectively form the intended warehouse. Moreover, the add-in needs to be able to: create any number of corridors of channels (simple corridors and sets of two corridors faced inside out), racks per corridor, channels per rack (channels per column and number of columns) and specify the size of the channels, its position, rotation and the number of ways the milk runs are allowed to travel on. To specify all these features the user only has to enter the respective data on an excel spreadsheet.

Chapter 2 presents a review over the analysed literature. In chapter 3, the steps to create the add-in are covered. In the fourth chapter, the developed add-in will be used to create a warehouse correspondent to the one being studied and the two types of picking strategies will be compared. Finally, in the last chapter, the main conclusions of the work will be 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" [7]. Thus, warehouses represent a very important role on modern supply chains [5]. 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

Received: , Accepted:
Correspondence to: Luís Dias
Departamento de Produção e Sistemas,
Campus de Azurém, 4800-058 Guimarães, Portugal
lsd@dps.uminho.pt

capital and operating costs of warehouses represent about 22% of logistics costs, whilst figures for Europe give a similar figure of 25%" [5]. These costs impel us to understand the problematic and to use the storage space as efficiently as possible [8]. Thus, the need to provide companies with methods capable of improving the performance of warehouses arises. According to Gu et al., several methods could be used to model warehouses, such as simulation, analytical methods and benchmarking. Nonetheless, "simulation is still the most widely used technique for warehouse performance evaluation in the academic literature as well as in practice" [9]. 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 [10].

Notwithstanding, due to the appearance of new products, new clients, demand changes or other reasons, it may be necessary to perform some changes on a model, in order to maintain it updated or just to accurately respond to new scenarios. This is a process that, made manually can be very time consuming. Thus, the possibility of automatically create a simulation model has already been studied [11].

Hlupic and Paul [13] compared simulation tools, distinguishing between users of software for educational and industry purposes. In his turn, Hlupic [14] 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. [12, 15] 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" [12]. However, a correlation may exist between popularity and quality, since best tools have greater chances 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 [16] taking into consideration several factors.

Simio is based on intelligent objects [17-19]. These "are built by modellers and then may be used in multiple modelling projects. Objects can be stored in libraries and easily shared" [20]. 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 [17-19]. The activity of building an object in Simio is identical to the activity of building a model. A vehicle, a customer 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 [17, 19]. This fact can be very useful, particularly while presenting the results to someone unfamiliar to the concepts of simulation.

In Simio the model logic and animation are built in a single step [17, 19]. This feature is very important, because it makes the modulation process very intuitive [19]. Moreover, the animation can also be useful to reflect the changing state of the object [17]. In addition to the usual 2D animation, Simio also supports 3D animation as a natural part of the modelling process [18]. To switch between 2D and 3D views the user only needs to press the 2 and 3 keys of the keyboard [18]. Moreover, Simio provides a direct link to Google Warehouse, a library of graphic symbols for animating 3D objects [18, 19].

Notwithstanding the fact that this is a recent tool, it is already possible to find many studies that use this tool. Vik et al. [21] 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 [22].

3. BUILDING THE WAREHOUSE

In this chapter, the several steps of the creation of the Simio add-in will be covered. Moreover, in the last section, the add-in will be used to create several different warehouses.

3.1. Data input

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.

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 chose 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 considered example, illustrated in Table 1, the user intends to create 2 corridors, one of each type.

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. Figure 1 shows the warehouse created by executing the developed Simio add-in with the input defined on Table 1.

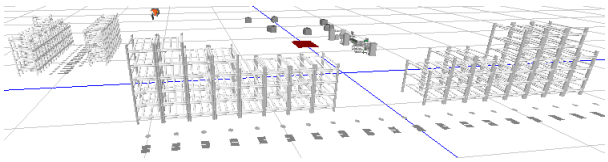

Figure 1: Warehouse created 1 (user-specified rack layout)

As can be seen, two corridors were created: a simple and a double. Moreover, they were created at different locations and with different rotation. The number of columns and channels per column created also corresponds to the data specified on Table 1.

3.2. First steps using the Simio API

The add-in was developed in Microsoft Visual Studio 2012. To start using the Simio API, it is necessary to create a class that implements the IDesignAddIn interface:

```
namespace BoschAddIn{
  public class DesignSupermarket:IDesignAddIn{ //class definition
}
```

Afterwards, it is necessary to define the methods of the implemented interface, otherwise the implemented IDesignAddIn interface cannot be used. In this sense, the methods Name, Description, Icon and Execute. The first three define the name, description and icon that will be presented in Simio, when a user wants to select an add-in to execute. Lastly, the Execute method will contain the code the add-in is supposed to execute. In this case, the code to create the warehouse. The following code lines illustrate the above mentioned:

```
public string Name{ get{return "Name of the add-in";} }
public string Description{ get{ return "Some desciption";} }
public System.Drawing.Image Icon{ get{ return null;} }
public void Execute (SimioAPI.Extensions.IDesignContext context){
  if (context.ActiveModel != null) { //Code of the add-in here }
}
```

After defining the methods of the IDesignAddIn interface, it is possible to start to create objects and edit its properties. 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.

Moreover, to create different orientations for the corridors of channels that compose the warehouses, 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 had to be considered. The solution adopted for this task was to assign different Simio symbols (Object image representation) 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.

3.3. Excel communication

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.

3.4. Algorithm

After retrieving the data, the add-in can start building the warehouse. 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.

3.5. Add-in Validation

To demonstrate that the add-in is building the intended warehouses, in this section, several inputs related to different warehouses will be considered. Time and Simio objects required for the creation will be discussed. Three warehouse sizes were considered: warehouse 1 with a total of 60 channels, warehouse 2 with a total of 300 channels and warehouse 3 totalizing 1500 channels. The numbers of corridors created, number of objects used and elapsed time to build them are displayed on Table 2. The way each type of corridor is built and the number of objects required to create them has already been explained and documented [6].

These results can greatly vary for the same warehouse sizes, since they are very dependent of the number of channels per column of a rack, the number of columns per rack, the sets of racks, which has more columns of channels, and more. Thus, to be able to withdraw some conclusions from this test, the authors applied the same number of channels, per column, columns per rack and racks per corridor to the same warehouse (c.f. Stopplace_Channels created and Channels created rows). As Table 2 suggests, regardless of the size of the warehouse, the type of corridor that requires more time to build is the simple one. This can be explained by the superior number of objects needed to build this type of corridor, in comparison to the double one, which is capable of providing the same amount of channels in less space, since it can access two sets of channels, and thus less objects are needed to create it. Consequently, less time is also needed to create this type of corridor. In fact, even building 3 simple corridors and 3 double corridors can require more objects that building a single simple corridor, with the same total number of channels.

4. COMPARISON OF STORAGE STRATEGIES

After building a warehouse, the model is ready to run and/or perform simulation experiments. In this chapter, a warehouse built using the developed add-in will be used to compare two storage strategies. Figure 4 illustrates the complexity associated to the construction of a warehouse. In this figure, some of the paths, TransferNodes and other Simio objects needed to build the displayed warehouse can be seen, whilst Figure 5 shows the same warehouse, by only showing the important objects for animation purposes.

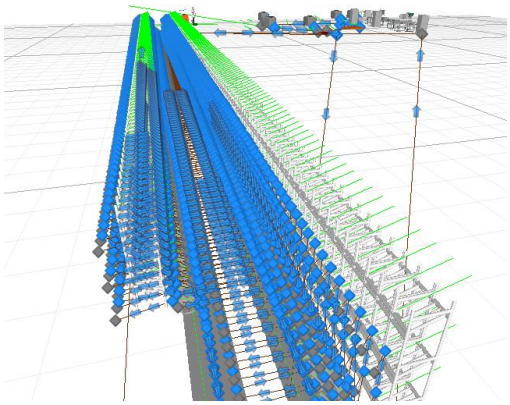

Figure 2: Warehouse created 2

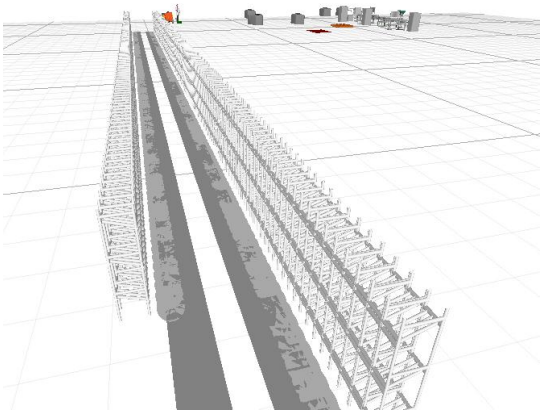

Figure 3: Warehouse created 3

The system being modelled consists on an advanced warehouse, located next to the production lines, which stores about 500 different products. Products are placed in containers and each container stores only one type of product. The products are produced and sent to the warehouse, for later being collected by the pickers and sent to the respective production lines. These lines consume the needed product units 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 warehouse with the leftover product units in it.

The storage strategy used in this warehouse is the dedicated (single-product within each container and in a fixed position - channel). This is the simplest case, since it consists on having a channel dedicated to a single type of containers [8]. One of its great advantages resides on the fact that, since the locations of the containers don't change, the pickers can memorize them, making the picking process more efficient [8]. 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%" [8], which represents a high amount of costs associated. To overcome this problem, other strategies can be considered. However, an alternative to this strategy would have to allow containers to be mixed within a same channel, 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, in order to avoid picking from the non-first position of a channel and to guide pickers to the proper channel, once they would no longer have the advantage of having memorized the location of the containers. Figure 6 displays the running of the simulation model, while modelling the single-product storage strategy.

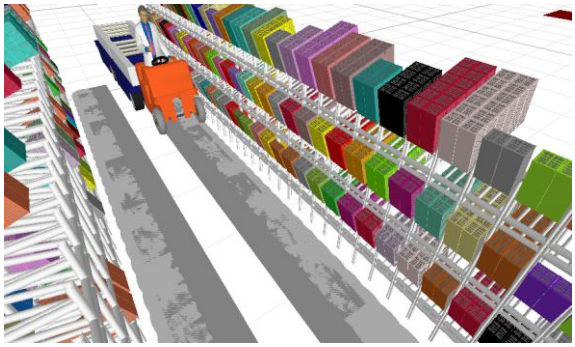

Figure 4: Modelling storage strategy 1

The remaining strategy being considered (multi-product) consists on letting the pickers know the channels they have to visit, at the beginning of their picking trips. Moreover, the containers would have to be stored, on each channel, taking into consideration their data consumption (giving priority to the channels that already have containers of the same type). Thereby, it should be ensured that pickers always know what channels they have to visit and that they always have to collect the first containers on each of those channels. To compare both storage strategies, the authors mainly considered the space gained on the warehouse (e.g. the number of unused channels), the number of stops per milk run and the time spent by the pickers while collecting containers. Figure 7 shows the simulation model execution, while modelling the multi-product storage strategy.

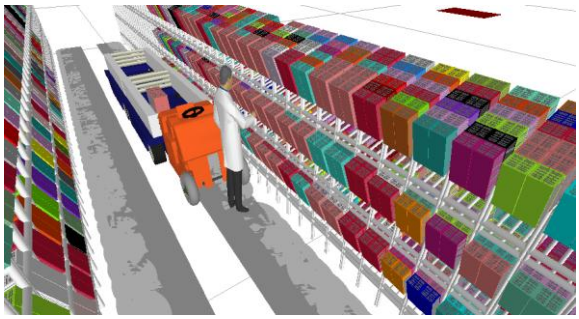

Figure 5: Modelling storage strategy 2

By comparing both Figure 6 and Figure 7, it is expected that the single-product strategy requires a higher quantity of channels to work, since it does not store different types of product on the same channel. This can be seen through the colours of the containers, wherein each colour represents a different type of product. As the figures illustrate, when the single-product strategy is modelled (Figure 6), on each channel, there are only containers of the same colour. On the other hand, when the strategy being modelled is the multi-product (Figure 7), containers of different colours are mixed within a same channel. Moreover, the containers are more concentrated and the majority of the channels are close to being full. Conversely, on the single-product strategy, the channels are divided through a higher quantity of channels.

These differences were already expected. Nonetheless, to quantify both strategies, some simulation experiments with a warehouse of approximately 900 channels were performed. These experiments were executed with 4 milk runs, a 20 minute time interval between the picking trips and a maximum capacity of 6 containers to every channel. Additionally, probabilities of 50% and 5% were considered for to the act of returning a container to the warehouse with leftover containers. These percentages can be justified by the fact that, in the multi-product strategy, the load of the warehouse is driven by the next effective production needs (electronic kanban system). Thus, the quantity of containers returned to the warehouse (leftovers) is very small. Results are summarized on Table 3.

Table 1: Comparison of the storage strategies

Storage Strategy	Multi-product	Single-product
Channel picking position (Depth)	**1**	**1**
Picking time (seconds)	**100,73**	**115,70**
Number of stops	**1,66**	**3,48**
Unused channels	**500**	**358**
Empty channels	607,04	503,12
Occupied positions	1607,63	1678,94
Round trips/milkrun/week	264	264
Accesses/channel	7,34	7,68
Late pickers	9	9
Reference changes	971	971
Full containers collected	6809	6809
Returned containers collected	17	335

Table 3 illustrates the several Key Performance Indicators (KPI) considered for this comparison. Nonetheless, some KPI were considered in order to validate the simulation model, such as: the average trips per milk run, the late pickers (pickers who were not ready to start a new trip at the respective time), the reference changes of the production lines and the full containers collected. By examining the obtained values for these KPI, it is possible to verify that all of them present the same values, regardless of the storage strategy being simulated, indicating that both strategies are based on the same data. Moreover, it increases the confidence in the simulation model. Nonetheless, for the KPI average number of accesses per channel, the average number of occupied channel positions and total number of returned containers to the warehouse that were collected once again, some differences were obtained. However, these differences can be explained by the fact that, on the multi-product strategy, the production is driven by the next effective production needs, which results in less containers being returned when a reference change occurs and, in its turn, less returned containers being collected once again and slightly less channel positions being occupied.

According to the obtained results, the pickers of the simulated multi-product strategy could perform their picking trips in roughly 15 seconds less time, representing an improvement of about 15% of the time needed to collect the respective containers. In part, this can be explained by the different results obtained by the average number of stops per milk run, where a difference of almost 2 was registered (improvement of about 50%). In its turn, the different values registered for the KPI average number of stops per milk run can be explained by the fact that, with the implemented IS, it is possible to store the containers on the warehouse, taking into consideration the milk runs that will be collecting them and, consequently, the production lines they are destined to. Thus, the references that a single milk run has to collect are more concentrated and a single stop is enough to collect several containers of different types of product. Another aspect that influences the picking times obtained is the fact the pickers of the single-product strategy had to collect more containers (Returned containers collected column of Table 3), since in this scenario the probability of a container being returned to the warehouse, after being delivered to a production line, is higher. Lastly, the multi-product approach was able to achieve this performance and maintaining one of the advantages of the single-product approach, which consists on the fact that the pickers always collect the first container on each channel.

Focusing the analysis on the space occupied on the warehouse, it is possible to verify that the multi-product did not use 500 of the roughly 900 channels, representing a usage percentage of less than 50%. On the other hand, on the single-product strategy, only 358 channels were not used, representing a usage percentage of less than 40% and an overall better performance of the multi-product strategy of 35%, which means that the system with the single-product warehouse would need 35% more space than the multi-product warehouse. Lastly, concerning the average number of empty channels, the strategy of multi-product was able to obtain roughly 21% more of the empty channels.

5. CONCLUSIONS

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" [4]. Thus, the opportunity to develop a **simulation** model in Simio that could help a company of the Bosch Group, arose. At an early stage, the tool needs to be able to design several layouts of the warehouse. After creating the intended warehouse, the model should be capable of modelling different **storage strategies,** allowing the user to specify several properties.

Throughout chapter 3, it was explained how the user can specify the **warehouse layout** he intends to create, by inserting its data on an **Excel** spreadsheet. Additionally, since the information available regarding the **Simio API** is very scarse, some code lines needed to start using it were provided. The code used to communicate the **C#** with Excel was also provided, while the main algorithm was kept as pseudo code. On the last section of chapter 3, several inputs were used on the developed add-in, in order to test it, by building many different warehouses. As the results indicated, the add-in was able to build all the warehouses. The number of Simio objects that were created, as well as the time needed to build them was also analysed. Some Simio API gaps were also discussed at the end of chapter 3.

On the fourth chapter, the Simio add-in was used to create a new warehouse to compare two different

storage strategies. The first consisted on a dedicated warehouse (**single-product**), where each channel only stores containers of a single type of product. The second strategy consisted on allowing any number of different types of products to be stored within the same channel (**multi-product**). The comparison considered several **Key Performance Indicators** (KPI). Focusing the analysis on the KPI average **picking time**, average channel picking position (**Depth**), average **number of stops** per milk run and the total **number of unused channels**, it was possible to notice that, on the multi-product strategy, the pickers could collect the same required containers in about **15% less time** and by doing an average of **less 2 stops** per picking trip (improvement of about **50%**). Moreover, the single-product strategy needs approximately **35% more space**. Lastly, the analysed results indicate that the multi-product approach was able to achieve this performance and maintain one of the advantages of the single-product approach, which consists on the fact that the pickers always collect the first container on each channel, indicating that the company in question could benefit from this strategy, by reducing the size of their warehouse and by globally improving their picking system. Thus, the associated costs, both in time and space would be reduced.

The **good animation** results that Simio offers were an important indicator for its selection for this project. Additionally, the 3D features as well as the direct interaction between Google 3D warehouse makes the final result very similar to the system being modelled, which can be very important when trying to transmit confidence to others and also to show the results to third parties. Throughout the paper, several figures illustrate the very good animation results obtained (e.g. Figure 7).

ACKNOWLEDGMENT

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

[1] Monden Y (1998) Toyota Production System – an integrated approach to Just-In-Time. In. Institute of Industrial Engineers, Norcross, Georgia

[2] Womack JP, Jones DT, Roos D (1990) The machine that changes the world. In, Rawson Associates, NY

[3] Womack JP, Jones DT (1996) Lean Thinking. In, Siman & Schuster, New York, USA

[4] Bosch (2014) consulted online at: http://www.bosch.com/en/com/home/homepage.html

[5] Baker P, Canessa M (2009) Warehouse design: A structured approach. European Journal of Operational Research 193, pp 425-436

[6] Vieira A, Dias L, Pereira G, Oliveira J, Carvalho M, Martins P (2014) 3D Microsimulation of Milkruns and Pickers in Warehouses using SIMIO. In: ESM'2014, FEUP - University of Porto

[7] Coyle JJ, Bardi EJ, Langley CJ (1988) The management of business logistics. West Pub. Co.

[8] Bartholdi JJ, Hackman ST (2008) Warehouse & Distribution Science: Release 0.89. The Supply Chain and Logistics Institute

[9] Gu J, Goetschalckx M, McGinnis LF (2010) Research on warehouse design and performance evaluation: A comprehensive review. European Journal of Operational Research 203, pp 539-549

[10] Costa B, Dias LS, Oliveira JA, Pereira G (2008) Simulation as a tool for planning a material delivery system to manufacturing lines. In: Engineering Management Conference, 2008 IEMC Europe 2008 IEEE International, pp 1-5

[11] Vik P, Luís D, Guilherme P, Oliveira J (2010) Automatic generation of computer models through the integration of production systems design software tools. International Journal for Simulation and Multidisciplinary Design Optimization 4, pp 141-148

[12] Dias L, Pereira G, Rodrigues G (2007) A Shortlist of the Most Popular Discrete Simulation Tools. Simulation News Europe 17, pp 33-36

[13] Hlupic V, Paul R (1999) Guidelines for selection of manufacturing simulation software. IIE Transactions 31. pp 21-29

[14] Hlupic V (2000) Simulation software: an Operational Research Society survey of academic and industrial users. In: Simulation Conference, 2000 Proceedings Winter, pp 1676-1683 vol.1672

[15] Pereira G, Dias L, Vik P, Oliveira JA (2011) Discrete simulation tools ranking: a commercial software packages comparison based on popularity

[16] Vieira A, Dias L, Pereira G, Oliveira J (2014) Comparison of Simio and Arena Simulation Tools. In: ISC, University of Skovde, Skovde, Sweden

[17] Pegden CD (2007) Simio: A new simulation system based on intelligent objects. In: Simulation Conference, 2007 Winter, pp 2293-2300

[18] Sturrock DT, Pegden CD (2010) Recent innovations in Simio. In: Proceedings - Winter Simulation Conference, Baltimore, MD, pp 21-31

[19] Pegden CD, Sturrock DT (2011) Introduction to Simio. In: Proceedings - Winter Simulation Conference, Phoenix, AZ, pp 29-38

[20] Pegden CD (2013) Intelligent objects: the future of simulation. Simio. White paper. In, Available online at:http://www.simio.com/resources/white-papers/Intelligen-objects/Intelligent-Objects-The-Future-of-Simulation-Page-1.htm

[21] Vik P, Dias L, Pereira G, Oliveira JA (2010) Using simio for the specification of an integrated automated weighing solution in a cement plant. In: Proceedings of the Winter Simulation Conference. Winter Simulation Conference, Baltimore, Maryland, pp 1534-1546

[22] Vieira A, Dias L, Pereira G, Oliveira J (2014) Micro Simulation to Evaluate the Impact of Introducing Pre-Signals in Traffic Intersections. In: ICCSA, University of Minho at Guimarães - Portugal

Table 2: Input Excel table

New corridor?	Size			Coordinates		Symbol index	Directions	Rack description	Channels per column							
	Length	Width	Height	x	y(z in Simio)											
1	1	2	1	-20	30	0	2	AAA	4	4	4					
0								AAB	4	4	4	4				
0								AAC	4							
0								AAD	4	4	4					
0								AAE	4	4						
2								ABA	3	3	3					
0								ABB	4	4	3	4	5			
0								ABC	4	3	4	3	4	3	3	4
1	2	4	1	15	20	2	1	BAA	6	6	5	5				
0								BAB	4	4	4	4	0	0	0	
0								BAC	0	3	4	4	3			
0								BAD	6	6	6	4				

Table 3: Different warehouses created using the developed Simio add-in

Warehouse size - Number of Channels created	warehouse 1			warehouse 2			warehouse 3		
	60			300			1500		
Simple corridors created	0	1	3	0	1	3	1	0	3
Double corridors created	1	0	3	1	0	3	0	1	3
Time (seconds)	40	61	53	383	481	398	6244	8470	7706
StopPlaces created	32	62	46	126	208	148	578	1140	854
StopPlace_Channels created	31	31	31	104	104	104	570	570	570
TransferNodes created	127	187	´170	555	719	614	2659	3783	3226
Paths created	469	589	555	1930	2258	2048	9624	11872	10758
Total objects created	659	869	632	2715	3289	2914	13431	17365	15408

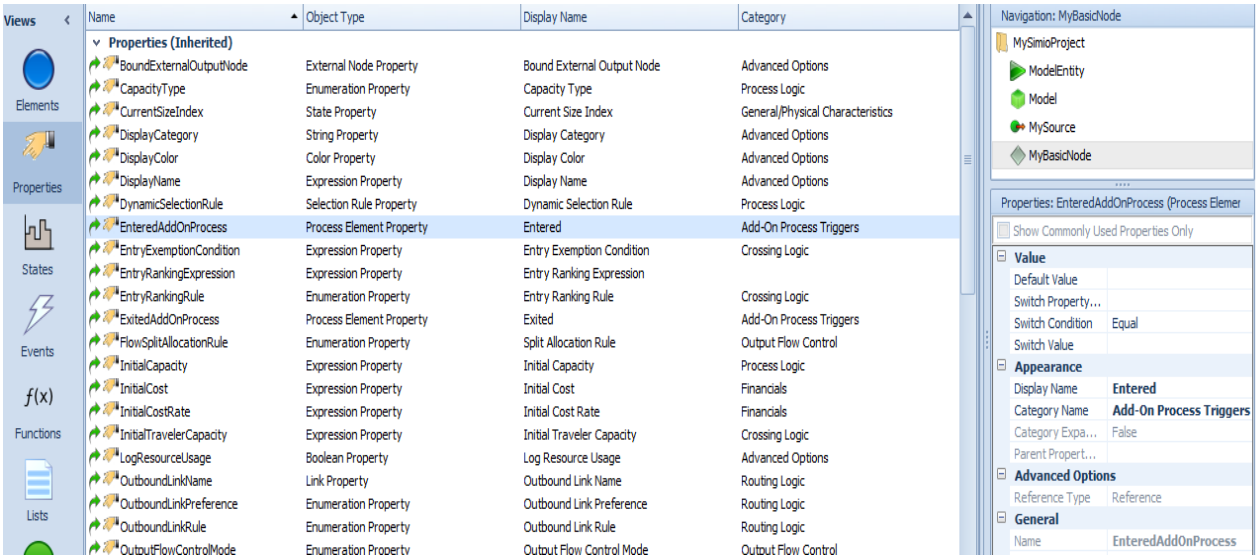

Figure 6: Verifying name and display name of a property

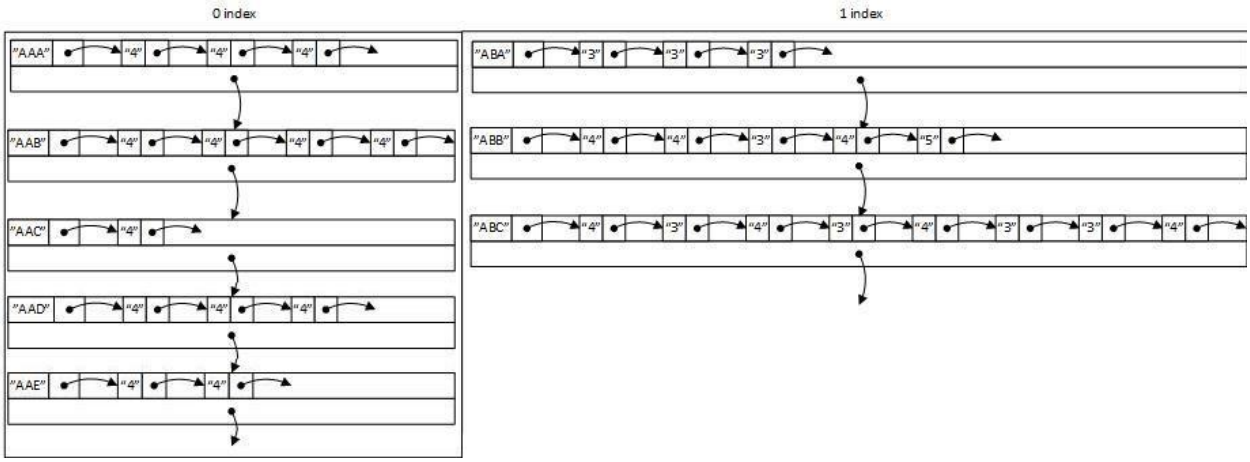

Figure 7: Data structure representation