Faculdade de Engenharia da Universidade do Porto

An analysis and evaluation of Discrete Production Systems: a Simulation based approach

Joana Garcia Duarte Ferreira Rodrigues

Dissertation performed in the scope of the
Integrated Master in Electrical and Computers Engineering
Major in Automation

Supervisor: Prof. Dr. Américo Lopes de Azevedo

July 2008

Abstract

In the world that we live in today, which is pulled by customer demand and where the market dictates what the newest trends are and what paths are to follow, Production Systems play a major role. They are the pillar of all available products and services and therefore must always be at the edge of technology and evolution.

One of the arising technologies that is gaining more ground everyday that goes by is Simulation. It provides a way to analyse a system without interfering with its normal functioning and acts as a decision support tool in a great amount of varied situations.

This dissertation has as goal to analyse and evaluate Production Systems through the eyes of Simulation. Its objectives are to understand how these systems can benefit from the use of this growing technology and what is being done in this field nowadays. In order to fully grasp the importance of this subject, simulation models of some Production systems will be built and analysed, using a computer based simulation platform. This platform will also be examined to determine its potential usefulness for Production Systems' simulation. A case study which concerns the internal logistics of an assembly factory will be analysed and simulated using this technology.

Key words: *Production Systems, Modelling, Simulation, internal logistics, milk-run*

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 – Context and Relevance

After the Industrial Revolution and until the late nineteenth century, manufacturing firms were mostly a family business, specialised in a specific product or process which monopolised the industry's resources. Around that time, the industrial business started to develop towards a different direction, where the rapid growth of the population led to a boom in the demand for each and every product. Consumerism had to be satisfied no matter what, meaning that it was the time for mass production to emerge and the assembly line concept was born.

Well within the twentieth century, another trend appeared in the industrial world; diversity in enterprises began to grow and became the biggest asset for a firm that wishes to be competitive. Markets expanded even more and so did the need for new products and services, accompanied by the birth of new manufacturing activities and also management and administrative functions. This evolution is still in progress nowadays and at a faster pace each day that goes by.

In a society that is rapidly growing and is everyday more demanding, the production industry has the absolute need to keep up with it and be as responsive as possible. Production systems must be able not only to respond to the demand in terms of quantity but also of high-quality and even customisation. The key to most production management philosophies nowadays is, therefore, being able to make the right product, in the right quantity, with the necessary quality and at the right time. This implies that the production system must be designed and operated in the best possible way, in order to respond to all of these needs and also so that the profit is maximised and the company remains as competitive as possible.

Industry and technology go hand-in-hand, now more than ever. Leaving aside the industry that concentrates on technology from a manufacturing perspective, we can see the role of technology from the management point of view. A word that represents one of the biggest trends in today's industry is 'Integration', where information and knowledge sharing and exchanging are essential. Technology here comes as an enabler of such situation.

This is the point where an important question arises: what is the role of Simulation in this scenario? First of all, it must be clear what the word 'simulation' means in such a context. To simulate is to imitate, to pretend, to make something pass as something else. Here, it means to use a model to substitute reality, a model that represents our system and that is fairly easy to change and adapt to our demands in order to make it possible to test different possibilities and observe the results. Therefore, and first of all, one always has to take into account if it is possible to build a model of the system and if such model can be used for the purpose of simulating what one desires from the system. If so, then Simulation presents itself as one of the enablers of information sharing in the industrial context. It enables you not only to represent your system and its behaviour in a clear way but also to test different scenarios that in the real system would probably be impossible to try out. So, in this dissertation, one of the goals to be achieved is a deeper understanding of the role of simulation of discrete production systems. In order to do so, a computer based simulation tool[1] was used to build some models of production systems and then to simulate them.

In summary, we nowadays live in a heavily industrialised society where customers are more and more demanding and the market is populated with competitors. The need to always stay above the average and be as competitive as possible is a reality which implies the need to fully know the system we are working with and the ability to analyse what changes are possible and their results. Therefore, this piece of work was found relevant in order to explore the possibility of modelling Discrete-event Production Systems and the role of Simulation of such systems at this point in time and also, going more in depth, the potential of the chosen software for this purpose.

This dissertation is closely related to the area of Industrial Management, more specifically Production Systems. These systems can be described as Discrete Systems, in a sense that their activities, although continuous in most cases, can be considered as discrete states, without significant error in such an approach.

1.2 - Objectives and research questions

The main objectives of this work are the following:

- Analysing the role of Modelling and Simulation in the context of Production Systems;
- Analysing the 'state-of-the-art' of simulation techniques and technologies for production systems;
- Exploring the potential of the AnyLogic simulation technology regarding the modelling and simulation of production systems;
- Specifying and implementing simulation models, both theoretical and real (a case study).

These objectives can lead us to the following research questions, which represent the aim of this piece of work in a more direct and accessible way:

[1] The chosen simulation platform was AnyLogic (http://www.xjtek.com/anylogic/).

- Is it possible to model discrete production systems and use these models to simulate these systems?
- What is the relevance of modelling and simulating such systems?
- Is it adequate to use the chosen simulation software for this purpose?

1.3 - Methodology

A work methodology, which can be divided in three main parts, was adopted in order to achieve the proposed goals.

The first part consists of a literature review on the subject of Modelling and Simulation, including the simulation of production systems and existent technologies in this field.

A second part consists of analysing, grouping and classifying different issues / problems that may arise in a production system and that may benefit from the use of simulation as a way to clarify the source of the problem and/or test different solutions to it.

At a final stage, the AnyLogic software is used as a modelling and simulation tool for production systems which are affected by one or several of the issues in production systems mentioned above and the results of these simulations are to be analysed.

1.4 - Structure of the document

This dissertation is organised in six different chapters and two appendixes. The chapters can be grouped in four main sections, the first one being the introduction, the second one the literary review, the third related to the chosen simulation platform and the practical use of models and simulation, and the last section the conclusions.

This first chapter of the document consists of an introduction, where the relevance of the subject being studied is stated, as well as the objectives achieved by this work, the methodology used and the structure of the document.

The second and third chapters are essentially the result of the literary review on the subject of Modelling and Simulation in general and the Modelling and Simulation of Production Systems in particular.

Therefore, the second chapter gives an overview on Modelling and Simulation of systems as a whole. It was found important to address these subjects before going into the actual purpose of approaching the simulation of production systems itself, in order to make clear the basis that make this piece of work possible and give some insight on this issue to readers that might not be too familiar with it.

The third chapter approaches Modelling and Simulation of Production and Logistics Systems, in particular, presenting an overview on the evolution of Simulation and making reference to some of the currently available technologies for simulating these systems. The different issues that may arise in a production system which can benefit from the use of simulation are also addressed in this chapter.

The fourth chapter is related to the chosen simulation platform. It presents the AnyLogic software, including its characteristics, its potential and also limitations for this specific application of simulating a discrete production system. Some applications that were developed in the context of this piece of work are also presented here.

The fifth chapter refers to the case study. It describes the structure of the system under study, the inputs and the desired outputs. It presents the different situations that were modelled and simulated, making clear their implementation and results obtained.

The last chapter presents the overall conclusions of the work.

In the appendixes is included most of the code written for the test applications and the case study.

At the end of the document are placed the References and Bibliography.

Chapter 2

Modelling and Simulation

This chapter consists of an introduction to the subject of Modelling and Simulation, in order to make some concepts that will be used throughout this piece of work clearer to readers that might be less familiarised with them. Firstly, some definitions will be presented, followed by the reasons that make simulation a useful tool and its advantages and disadvantages. The third section of this chapter presents a more theoretical view of simulation, giving an insight on the different existent types of models and simulation. The following section addresses discrete simulation more in depth. Finally, the last section focuses on simulation technologies, making the distinction between general-purpose programming languages and simulation languages.

2.1 - Definitions

When addressing any subject, it is important to make clear from the start what we are referring to and the exact meaning of the concepts we are working with. In this particular case, there are three very important concepts that will be referred to frequently. These are *system*, *model* and *simulation*.

A *system* is "a set of connected items or devices which operate together" [1], more specifically, it is any object (or set of objects) we intend to study.

A *model* is "a representation of something, either as a physical object which is usually smaller than the real object, or as a simple description of the object which might be used in calculations"[1], so the term *model* will be used here to describe some kind of approximate representation of the object(s) we are effectively studying.

To simulate is to "imitate or reproduce the appearance, character, or conditions of [ORIGIN Latin *simulare* 'copy, represent']" [2], therefore a *simulation* is a representation, a substitution of reality with a personalised version of it. From this definition, it appears that simulating and modelling are the same and actually sometimes the terms are used interchangeably. However, in this context *to simulate* will be used as an extension of modelling, to mean that some kind of experimentation is performed over a period of time on a *model* used to represent the real *system*. It is also important to highlight that when the word *simulation* is used nowadays it usually concerns *computer-based simulation*, as it is the most common type of simulation around presently.

2.2 - Simulating: why?

It is obvious that simulation has been playing an important role in today's world but what is it actually used for? These are the main purposes of simulation:

- Gaining insight on the system's behaviour;
- Obtaining information on the system without disturbing it;
- Developing and testing new concepts and policies (or even systems) before actual implementation.

Simulation can be used to study a system's behaviour in detail, to understand how each element works and interacts with the others. It is a tool to, for example, find the source of a problem that might not be visible in the real system. This can be referred to as a "as-is" analysis of the system. Taking a practical example, assume that there is a certain product manufactured in a production line that is constantly not ready on time. Observing the real system, it might seem impossible to find where the problem is, due to the huge amount of variables and elements interacting; but using simulation, one can observe each element of the system in action, stop the simulation at any point and even change some of the variables in order to better understand how the system actually works and even find out where the sources of some problems are and/or find opportunities for improvement.

Simulating a system can help you get information on it that might not be easily available in the real system. Imagine for example a critical system in which you wish to test the introduction of a new element but whose functioning can not be disturbed, such as chemical equilibriums with unstable elements or an airport security checkpoint.

Simulation can also be used to develop new ways of improving a system's performance, like changing its operating or resource policies or configuration, and to test the introduction of new elements without testing them in the real system. This is commonly referred to as a "what-if" analysis. The advantages of using simulation for this purpose and the ones mentioned before will be highlighted in the next sub-section.

However, it is essential to point out that simulation is not an optimization technique. It is a technique to estimate performance measures of the modelled systems. It can also be viewed as a decision support tool, as it allows the analysis and evaluation of the consequences of different options in the system's operation.

2.2.1 - Advantages

The advantages of simulation will be divided in five categories as in [3]:

- Cost: although simulating a system can prove to take a significant amount of time and involve a great deal of knowledge and effort from the one(s) building the model and performing the simulation, it can still be the right alternative to a test in the actual system, as it will probably be a lot more economical, especially if the test turns out to be inconclusive or even wrong.
- Time: as it was already mentioned, simulating a system is time-consuming but once the simulator is ready, a great amount of time can be saved from using it, as (especially with computer simulation) it is usually possible to use techniques that allow time compression.
- Replication: the use of simulation allows the exact recreation of the conditions for the performance of a certain experiment, allowing its replication, unlike the real world.
- Safety: simulation enables the testing of extreme conditions in a system, which could be potentially hazardous.
- Legality: sometimes, it might be necessary to test the effects of possible changes in legislation, which may represent illegal situations at the time, therefore not performable in the real system.

2.2.2 - Disadvantages

Although simulation has the unarguable advantages mentioned above, it obviously also has disadvantages. First of all, one must not forget the essence of using a model: it is only an approximation, it is not the real system. This means that there is always some degree of error deriving from the use of assumptions and estimates. To be able to choose which system's characteristics are the more significant and discard the others is a complex task.

Another disadvantage that was already referred to was that building a simulator is not easy and takes a significant amount of time. To build a useful simulator of a system, it is necessary to know it in depth in order to build an accurate model which can then be experimented on; this can prove to be very time-consuming. This leads to another disadvantage, which can be how much a task like this would cost. As almost everything that takes a considerable amount of time, it can become too expensive. However, it was already mentioned that both these disadvantages are usually surpassed by the advantages that come after the completion of a useful simulator.

Finally, as in any system, the output data can only be, at most, as good as the input data, meaning that even if the model is as precise as possible, it will not provide good results with inaccurate input information.

2.3 - Types of models and types of simulation

Figure 2.1 below depicts the existent *types of models*. It was adapted from Figure 1.1 that can be found in [4].

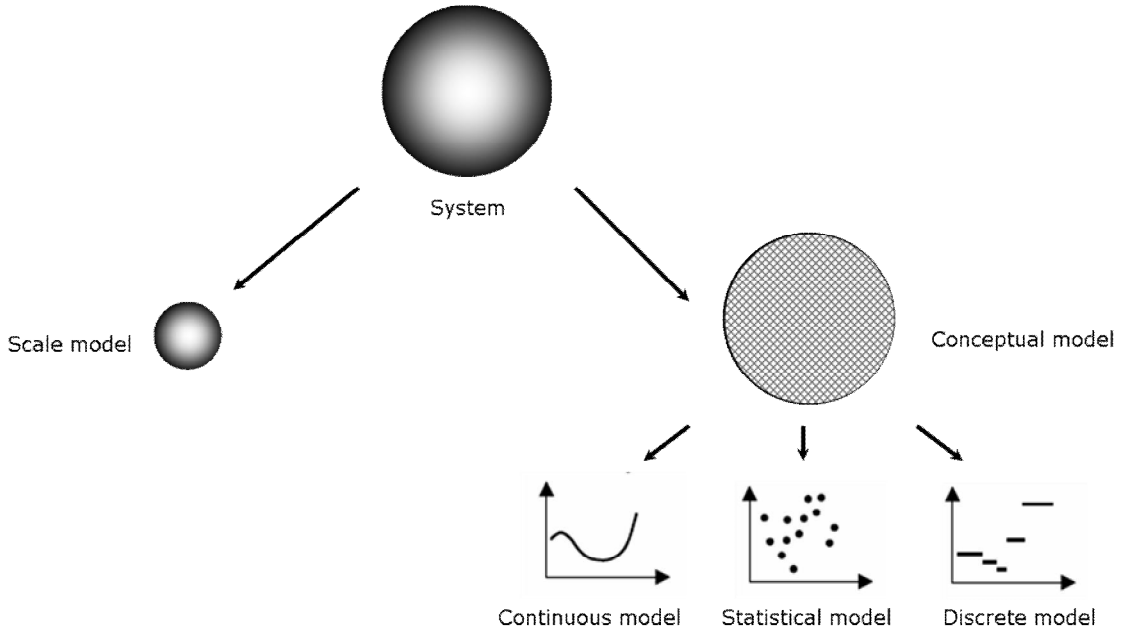


Figure 2.1 – Types of models (adapted from Fig 1.1 in [4])

Scale models, as the actual name says, are a scale version of the system itself, like a prototype or a mock-up.

Conceptual models are a conceptual version of the system. They are a set of mathematical equations or functional cause-effect relationships that describe the system. They are usually more versatile than scale models, as they are easier to handle, more portable and compact and allow changes more simply, making them dynamic. Moreover, conceptual models are usually easily implemented in a computer therefore enabling great data storage possibilities and also processing abilities.

Conceptual models can be classified according to the modelling method adopted to represent the system. Therefore, they can be *continuous*, *statistical* or *discrete*.

Continuous models are the ones where the system's states depend continuously on the independent variable. An example of such a system can be a projectile travelling in the air, where the independent variable can be time and the dependent the position. If the relationship between the dependent and independent variables is known, the analysis of these systems is relatively easy, especially if there is no feedback situation. However, most of the times such equation (or set of equations) between the variables is not known explicitly, only through a set of differential equations, which makes the simulation more complex and sometimes even impossible without recurring to more approximations, such as discretization for example, which will be explained two paragraphs ahead.

Statistical models characterise the system using probability distributions. In reality, in these cases usually there is no actual model of the system but a set of functions that characterise the system's behaviour, instead of a descriptive model of the system. The system is considered to be a black box where you do not need to know how the processes on its inside are, only its inputs and/or outputs described by statistical distributions.

Discrete models concern the systems whose states vary in a discrete way with the independent variable. These states are well defined and in a finite number. Discrete models can also describe continuous systems whose states can be considered discrete (discretization). For example, in a system where a car starts driving and then stops, we have a continuous change from the state of being stopped until the state where it reaches a constant speed and then from that speed until it stops again. However, the analysis of this system can become simpler if we consider that there are only two discrete states, stopped and moving, instead of the continuous evolution of states. Another example of continuous systems that are considered discrete is Production Systems. Discrete models (and discrete simulation) will be deeper analysed in the next sub-section.

Usually, when referring to the *types of simulation*, what is referred to is the type of conceptual model used. Therefore there are *Continuous Simulation*, *Statistical Simulation* and *Discrete Simulation*. Statistical Simulation is also known as Monte Carlo Simulation, an analogy with casinos and card games that are usually simulated using a statistical approach.

2.4 - Discrete simulation

Discrete simulation concerns the simulation of discrete systems, such as production systems. In this section, this type of simulation will be analysed more in depth, as it is the basis for this study.

2.4.1 - Elements of discrete simulation

Discrete-event simulations usually are composed by the following elements:

- Entities;
- Activities;
- Events;
- Queues.

Basically, a discrete system is described by its *entities* (that have attributes and states) which perform *activities* in the system, which generate *events* when they start or finish. The entities wait in *queues* until the conditions that enable them to perform a certain activity are met.

An *entity* in a system is any element that contributes to the set of possible states of the system (which is simply the set of all the possible states of all the entities in the system). An entity's *attributes* are properties and/or values that characterise the entity.

According to the performance or not of operations in a system, an entity can be classified as an *active* or a *passive* entity. An active entity is responsible for changing the system's state while a passive entity only changes its state due to other entities.

According to the time an entity spends in the system, it can be classified as *temporary* or *permanent*.

Activities, as the name itself says, are the operations or procedures performed by the entities in the system. These include waiting activities, interaction with other elements in the system or activities where only the entity itself takes part. Activities are responsible for the change of an entity's state. An activity can be *live* or *dead*, the later usually corresponding to a waiting activity and the first to the performance of an activity that has a defined duration and contributes to a state change.

It is important to mention that many times the expression *activity cycle* is used to describe the set of activities that an entity goes through in a system and its order. In an activity cycle, a live activity is always followed by a dead activity, even if this one has instant duration. This approach is commonly used for systems with a heavy component of waiting queues, such as production and logistical systems.

Events are generated when an activity starts or ends. They have zero-time duration, i.e. they are instant, and make a change of state occur. They can be classified as *internal* if they are originated by the system itself or *external* if they are due to the systems' inputs or outputs.

Queues are where the entities are placed while they are waiting for a live activity to occur, meaning that they are used to model dead states (states where dead activities take place).

2.4.2 - Time-handling

When describing the existent types of models, the relationship between the independent and dependent variables was always mentioned. The most common independent variable in simulated systems, such as Production Systems, is *time*, so it is necessary to consider the way it will be handled during a simulation because a system's state changes are modelled through time.

One of the advantages that were pointed out for simulation was the ability to compress time, enabling to preview a result that in a real system could take weeks or more in just a few seconds. Therefore, one must know not only the current simulation time but also how to jump to the next relevant point in time. There are two time-advance mechanisms available: *Time-slicing* or *Fixed-increment time advance* and *Next-event time advance*.

In *Time-slicing*, the system is examined and the information is updated at regular time intervals. It is easy to implement but there is no synchronism between the sampling of the system's state and the states' transitions.

Using this method obviously implies that the decision on how long the time interval will be is taken before the simulation is carried out. On one hand, if an interval that is too large is chosen, some of the system's state changes might be lost and the simulation will not be

accurate. On the other hand, if the chosen interval is too small, the model will be analysed too frequently, even when no state changes are possible, which may lead to computer runs that are longer than what would be necessary. As this is a case of sampling a system, the time interval must obey to the Nyquist law, which states that a system must be sampled at a frequency at least twice as big as its events' highest frequency. This means that if we have, for example, a system where an event occurs every minute, then our sampling time interval must be of at least 30 seconds.

Although Time-slicing is simpler to implement, usually *Next-event* is the chosen technique as most systems have idle periods of varying length. Usually there is a *schedule* in the simulation, where the future events that are supposed to occur are marked (this can be modelled in a computer simulation by an array of events, for example). The model is then examined and updated when there is an event about to happen. In this way, the system is only examined when there is a significant event and unnecessary checks are avoided, as the time interval adapts to both high and low activity periods in the system. However, the use of this method implies that more information is held and the simulation time does not flow smoothly but the advantages of being able to sample the system when it is actually necessary are usually enough to surpass these disadvantages.

It is also worth pointing out that the Next-event technique is more general than Time-slicing. If the events in the system occur at regular time intervals and a Next-event approach is adopted, this technique will act exactly in the same way as if a normal Time-slicing had been adopted at start. The reverse is not true though.

2.4.3 - Approaches towards discrete model creation

Usually, there are considered to be four different approaches towards the creation of discrete models and subsequent simulation. These concern the way the system's state diagrams are implemented. The author of this piece of work believes that generally, in the end of the modelling process, one finds out that a somewhat hybrid approach was used to some extend, as all the different approaches have their pros and cons and it is relatively easy to be able to take advantage of most of the strong points of each method. However, a brief presentation of all the approaches will be presented, as they are important in the context of the theoretical basis of discrete simulation. Still, they will not be referred to in the further explanation of the work developed.

So, these are the four more common approaches towards the creation of discrete models:

- Activity approach;
- Event approach;
- Process approach;
- Three-phase approach.

In the *activity approach*, there must be a detailed description of all the activities in the system, specifying the sequence of actions to be taken when each activity takes place. Only live activities are described, the dead activities are considered to be waiting queues. The order of occurrence is important; the analyst must define priorities between all the activities. An advantage of this method is that it focuses on the system's activities, making their

implementation faster and simpler. A disadvantage is that, at each time interval, all the activities are examined to check if they should occur or not.

In the *event approach*, there must be a detailed description of the system's events, related to state transitions, and the actions that are associated with those transitions. These actions occur whenever a given event takes place and also determine what the future events are. A simulation schedule is created, with the list of all the events due to happen, whose implementation can be somewhat complex. However, this technique is quite versatile and efficient, as the activities are not examined at every cycle, like in the activity approach; they are only run when the event they are associated with occurs.

In the *process approach*, the processes or paths that an active entity goes through in the system are analysed. All the entity's possible paths along its activity cycle must be described. For a given entity, all the others are considered to be *resources* that can be available or unavailable. The entities are organised in *classes* and the processes are associated with the different entity classes.

The *three-phase approach* separates the model in three parts, hence its name. One part is the simulation executive, which controls the entities and resources' activities end events. It maintains the simulation clock and schedule. The other two parts correspond to the application logic, the Bs and the Cs. The Bs are the activities and events that have a predictable time of occurrence and can be scheduled in advance. B is an abbreviation of Book-keeping activities or Bound (to happen) activities. The Cs are the activities that cannot be classified as B, i.e., Conditional or Co-operative activities, that only occur in certain circumstances and cannot, thus, be scheduled in advance.

2.5 - General purpose languages vs. Simulation languages

In computer based simulation, the first step to take before implementation is obviously choosing the tool we are going to use. Firstly, it is necessary to decide between using a package based on a specific simulation language or a simple general purpose language. It is also worth mentioning that are still some simulations that are done based on spreadsheets and their analysis, although these are becoming less and less used due to the evolution of other available options.

A *general purpose language* is any programming language that can be used to produce any desired application based solely in code writing, subject to the language's own limitations, of course. Examples of such languages are C, C++, Java, Pascal, FORTRAN, Basic, amongst others.

A *simulation language* (or a simulation environment based on such languages, commonly called a *simulator*) is a language that was developed already for modelling and simulation exclusively. Although usually general in nature, it comes dedicated to a certain type of systems. It has features such as pre-programmed building blocks of certain typical functions present in the system or automatic code completion. Examples of simulation languages

and/or simulation environments are Arena, AnyLogic, Simula, GPSS or ModSim, amongst many others.

Now it is time to analyse the pros and cons of both options, to make clear in which situation each can come as the best choice. This choice between the two options depends on several factors, like:

- Type of system being modelled and simulated;
- Purpose of the simulation;
- Experience of the developer;
- Potential support needed;
- Project time;
- Price.

A *general purpose language*'s most striking advantage is that typically the person who is going to develop the model already has some background in programming and, so, already knows one or more programming languages and has some ease in using them. Moreover, the use of a general purpose language is usually available on every computer, whereas a simulation language probably is not. This also means that the software cost will likely be lower (but not necessarily project cost).

Being general purpose, programming languages are adaptable to virtually any situation. However, all the model building and subsequent simulation is dependent on code writing alone, which can prove to be a very time-consuming task and also potentially hard to correct eventual errors. It can also mean that certain features are somewhat limited or are too demanding to be achieved in the purposed time for the development of the model. An example of such features can be the graphical part of the model.

An efficient program written in a general purpose language might need less execution time than one written in a simulation language. This is due to the fact that simulation languages' pre-programmed components are made in order to suit a relatively large number of different systems, meaning that many times they possess features that are not useful in the particular model being built and that could be discarded.

As a final note, it is worth mentioning that there might be pre-written libraries for a general purpose language which can have some of the features present in the pre-programmed components of simulation languages, saving some of the developer's time.

Simulation languages are usually very specific to certain systems, which can be an advantage as they are oriented to the problem in hands, making it easier to analyse, but it can also mean that each type of system must have its dedicated simulation language.

These languages present options as the automatic code completion or the use of building blocks already mentioned before, and also random number generators, probability distributions, determination of the next event due, etc. that are there to facilitate the model building task. These features make such tools easier to use to people that might not have a strong background in programming. However, this can mean that the language itself limits the developer, depending on the level of freedom one has to alter the pre-programmed features of the language / software, which does not happen when using a general purpose language.

In spite of this, the fact that the developer does not have to write all the code dramatically reduces the time it takes to build the model, providing that it does not take too

long to learn how to use the tool and that it has good manuals or supporting literature. Also the fact that most of the code is not written by hand makes error detection easier, as most potential errors are already accounted for.

Finally, simulation models built using simulation languages are usually more easily changed.

Chapter 3

Production and Logistics' Systems

In this chapter, Production and also Logistics' systems will be analysed. More than ever, these two types of system are becoming impossible to disassociate. The definitions of Production and Logistics' system and their relationship will be exposed in the next section. Then, an analysis of these systems through the eyes of modelling and simulation will be made, starting by an overview on the evolution of simulation and an analysis of the state-of-the-art. Section three presents the major applications of simulation in manufacturing systems, followed by a proposal for a classification for potential problems that can arise in these systems having in perspective a possible use of simulation techniques. Finally, there is a reference to distributed simulation in the context of Production systems.

3.1 - Definitions

A possible definition for *Production* is "any of the methods used in industry to create goods and services from various resources" [5], presenting production as a transformation process and, therefore, associated with *manufacturing*.

Logistics can be viewed as "the process of planning, implementing, and controlling procedures for the efficient and effective transportation and storage of goods including services, and related information from the point of origin to the point of consumption for the purpose of conforming to customer requirements. This definition includes inbound, outbound, internal, and external movements." [6]. This more simply means that logistics manages the flows of both materials and information in a supply chain. A *Supply Chain* consists of a group of partners (like the vendors, manufacturing facilities, logistics service providers, distribution centres, distributors, wholesalers, other intermediaries, etc., who are the links in the chain) who convert a basic commodity (such as raw materials and/or resources; upstream) into a finished merchandise (a good or a service; downstream) which has some value to the end-

customer. Each link in the supply chain contributes to the added-value of this final product throughout the process. Logistics is many times associated with *distribution*. However, distribution is only a part of all logistical movements; it corresponds to external movements of products, usually the delivery either to intermediaries like wholesalers or the end-customer him/herself.

As it was mentioned earlier, the evolution of industry since the late 18th century has led us from a focused, familiar, one-product based industry to a more flexible, larger and integrated industry, part of a larger chain of facilities and services. This means that an industry cannot be a silo, an isolated element but has the need to be in direct contact with other industries, collaborators and competitors alike, and obviously customers. This kind of information and material flow is not only external but also internal, within the different departments of the individual industry we are considering. Hence, in today's world, it is impossible to think of a Production system without considering the Logistics' system associated with it and, so being, from now on when referring to Production systems it is implicit that we are also referring to all the logistical movements, both internal and external, inbound and outbound, of information and materials associated with such systems.

3.2 - Modelling and Simulation of Production systems

It is now time to address Production systems through the eyes of Simulation, starting by an overview on the evolution of simulation in general and of these systems in particular. It is important to remember now that Production systems are viewed, as it was mentioned before, as Discrete Systems, which means that they pass through different, defined and countable states over time.

Simulation of Production Systems is not at all a recent development. Since the beginning of the development of simulation techniques, discrete simulation has been the type of simulation that has had the biggest attention because it is considered to be the easiest one to implement. This is due to the characteristics of a discrete system, which enable it to be modelled with relative ease using computer based techniques as they behave in a more similar way to how computers actually work. For example, computers are based on binary code, which is obviously discrete (it has only two possible states, 0 or 1). Alongside with the development of discrete simulation, simulation of production systems also grew because of its importance in society and also its need for fast development and innovation, as it was referred already in chapter 1.

Being considered discrete systems, production systems were mainly seen as queuing systems, where the entities would flow, performing activities and waiting in queues when they were waiting for e.g. a resource to be available. Notice that it was said that these systems 'were' seen as queuing systems because this view is changing nowadays. With the birth of new production theories based on lean and agile thinking, these systems are now being viewed not only as a combination of queues that obey a 'push' policy but more like a combination of integrated parts that have some intelligence and that are able to avoid queues (considered as superfluous inventory, not value-adding) by only producing what is need when its need, in a 'pull' policy. This evolution in production theories has led to an obvious consequent development of simulation techniques and technologies. An overview of

the available technologies will be given in the following subsection but not before a brief look through the evolution of simulation since the late 1950s, in order to put things in perspective. It is mostly but not exclusively based on the analysis in [7].

3.2.1 – A walk from the past to the state-of-the-art

Simulation has always been related to the developments in the computer industry, so it is normal to find the beginning of actual simulation technologies in the same time frame as the development of computers. Probably the first milestone in the history of simulation is the first symposium on "System Simulation" that took place at the American Institute of Industrial Engineering, as early as 1958. A year after, the first version of DYNAMO (DYNAmic MOdels) was written by Fox and Pugh, an improvement of SIMPLE (Simulation of Industrial Management Problems with Lots of Equations). This became one of the industry's standards for many years to come. Actually it is still used presently (as it is visible on figure 3.1 a few pages ahead, for example). It is important to stand out that also in this decade, more specifically in 1957, the first programming language, FORTRAN, is born.

It is in the 1960s that the approaches towards the creation of discrete models (explained previously in subsection 2.4.3) are born. In 1961, there is the creation of GPSS (General Purpose Simulation System) by Geoffrey Gordon at IBM Corporation. GPSS was a process oriented simulation language that adapted very well to queuing systems and its many versions were, therefore, used from then on to simulate a great deal of Production systems. There were other languages that also emerged based on the other approaches towards model creation, such as CSL, SIMSCRIPT (a general purpose simulation language, 1963) or SIMULA (SIMulation LANguage, 1964) which was the start of the Object-Oriented Paradigm. In 1962, Petri nets were developed. It was also in the mid-60s that CAD (Computer Aided Design) emerged. This would be the beginning of a more graphical type of simulation.

In the 1970s, VIS (Visual and Interactive Simulation) is born. In 1973, Ritchie wrote the C programming language, which is considered to be the "mother" of most programming languages. It was also around this time that the microprocessor and, thus, the microcomputer and the personal computer were born, as well as the first version of the Internet (ARPANET). It is also important to point out that, in the history of Industrial Management, we can find an important milestone in 1975: the birth of the Materials Requirements Planning (MRP).

The 1980s were an extremely rich period in the history of simulation, essentially due to the development of personal computers with a graphical interface. CAD was now being used in the manufacturing industry. Also in the history of Industrial Management, this was a period of major changes, with the appearance of MRP II (Manufacturing Resource Planning) and, most of all, the disclosure to the western world of Toyota's Just-In-Time (JIT) philosophy.

In 1981, the first let us call it 'global' operating system is created, the MS-DOS from Microsoft Corp. The C++ object oriented programming language was born at the Bell Labs. In 1983, the first event graphs were presented, whose objective was to represent the dynamics of discrete systems based on event diagrams and that became a method still in great use today.

Still in the 1980s, one must not forget that VIS was still evolving, with software such as WITNESS, SIMAN/CINEMA or ProModel available with a graphical interface with the user. Although simulation was not exactly widespread in the manufacturing industry, there were already some more tools available, like SIMFACTORY, ECSL, SIMON, GASP, SLAM II, SIMAN, SIMSCRIPT II.5, GPSS/H or SIMULA-P.

In 1990, the first multi-window operating system Windows 3.0 is released and the World Wide Web (WWW) has started to connect more and more ordinary enterprises. In the simulation field, the beginning of the 90s saw the revelation of concepts like distributed simulation (a network of computers where different tasks are run in each one) and agents (elements that contain not only attributes and methods but also the ability to learn and make decisions). With the appearance of more developed, and yet a lot smaller, computers with better graphical user interfaces (GUIs), there was also a major change in simulation technologies that were, until then, most of them only pure lines of code, with little or no graphical presentation. In 1992, Pidd makes a distinction between VIS and VIMS (Visual Interactive Modelling System) [3]. For him, VIMS consisted of an interactive interface were the model could be built and simulation could be run always with a visual support, whereas VIS had only the visual interaction part during simulation, not in the model building. With VIMS, there was a blank canvas where the user could drag and drop components in the shape of icons and link them together. This is what is now commonly called a simulator.

During this decade, there was great evolution in Object-Oriented Programming (OOP) and graphic compilers for most programming languages. Consequently, simulation technologies also followed the trend and new high graphics VIMS were born and/or improved, such as ARENA, ProModel, Taylor II or Micro Saint.

In the 21st century, the evolution of simulation technologies continues to keep up with the evolution of computer systems, which means that it is growing more and at a faster pace than ever. New simulation tools have appeared, such as AnyLogic or eM-Plant, but others have continued to evolve, like ARENA (now in its 11th version), AutoMod, WITNESS or SIMUL8.

The agent-based approach to simulation has also gained ground and will soon be probably almost at same level as the traditional simulation approaches (discrete event, system dynamics and dynamic systems). The development of OOP and also the ability to communicate with other applications, in the same physical device or not, enabled the development of distributed simulation, more important than ever for global supply chains, for example. An important achievement in distributed simulation is the creation of the High Level Architecture (HLA) by the US Department of Defence, used as a starting point towards the modelling of many different types of systems, including Production and Logistics. Finally, web-based simulation is now a reality, where remote simulators can be accessed using common WWW interfaces.

In summary, there was a breakthrough in simulation technologies, as the computer industry is one of the fastest growing industries today and both of them have been walking hand-in-hand since as early as the 1950s. It is actually very difficult to keep up with the evolution of simulation technologies of Production systems especially because not only it evolves very rapidly but also it is a very vast area, with the ever growing nature of the Production industry and the more than fast growth of the Information Technology industry.

There are some surveys on simulation technologies used nowadays but they are not as abundant as one might think. On a personal note, I would say that this is probably due to the fast ever-growing and competitive nature of this area nowadays, which makes the task of surveying the existent trends very complicated.

The following paragraphs present a brief analysis of the state-of-the-art of simulation technologies for manufacturing and supply chain systems based essentially in three surveys, [8], [9] and [10]. The latter presents an analysis of 65 different general simulation packages, while the first two are the findings of the study of different articles concerning the application of simulation in the supply chain context and in manufacturing logistics, respectively.

From [8], we have in figure 3.1 the following distribution of simulation technologies used in 24 systems that were analysed by the authors, in a local simulation paradigm (thus not distributed simulation).

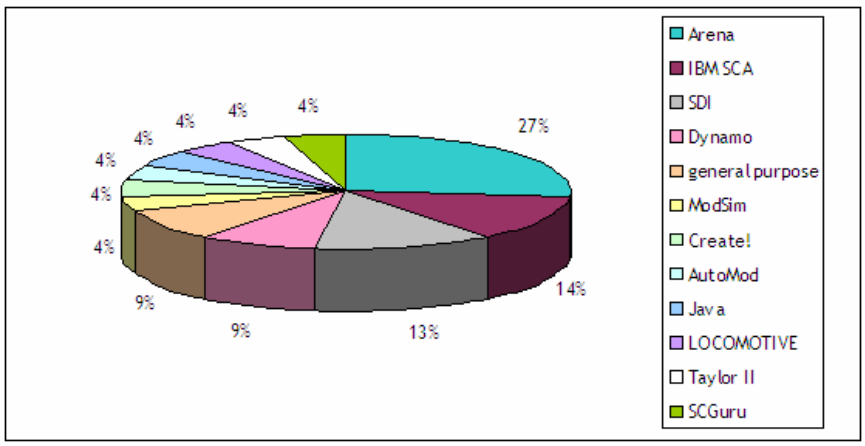


Figure 3.1 – Distribution of the simulation technologies used in 24 different Supply Chains, in a local simulation paradigm. These findings are the product of the analysis in [8].

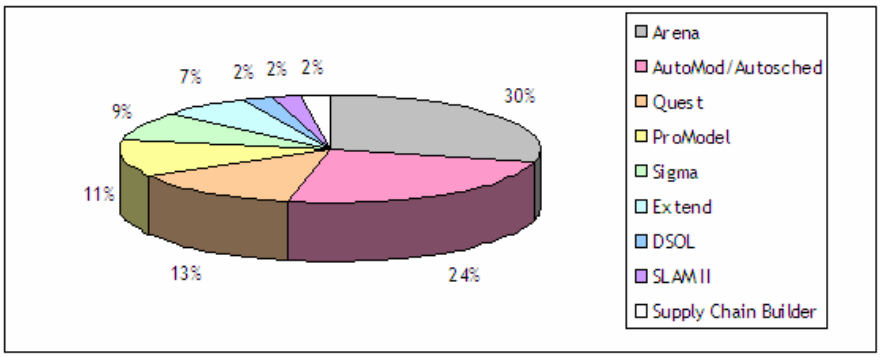


Figure 3.2 – Distribution of the simulation technologies used in 45 different manufacturing systems to aid in a decision making process. The findings are the product of the work in [9].

From [9], we can obtain the graph in figure 3.2. on the previous page which represents the distribution of Discrete-event simulation technologies used as a support decision tool in the context of manufacturing logistics of 45 systems that were analysed by the authors.

Combining the findings from both surveys [8][9], it is easy to see that the preferred Discrete-Event simulation (DES) tool is Arena (6 out of 24 systems in the 1st survey and 13 out of 45 in the 2nd), followed by AutoMod (although only 1 out of 24 in the 1st, 11 out of 45 in the 2nd). In the next paragraphs there will be a brief analysis of both these DES tools, in order to make clear not only what the general characteristics of this type of software are but also to lay some ground for comparison with the chosen technology for this work, AnyLogic, which will be presented in the next chapter. This analysis is mostly based on the conclusions of the survey [10] which presents 65 different contemporary simulation technologies.

Arena, by Rockwell Automation, is a simulation package that can be used to create applications in the manufacturing context to model e.g. facility design/configuration, scheduling, or dispatching strategies but also in other contexts like the healthcare industry, airports, the military or call centres. Regarding the model building itself, the software allows graphical model building (drag-and-drop or icons), access to programmed modules and also programming. It has runtime debugging, optimization algorithms, code reusing features, model packaging (i.e. a complete model can be shared with others that might lack the software so that they can build their own model) and mixed discrete / continuous modelling. Regarding animation, Arena allows both 2D and 3D real-time animation, exporting animations and importing CAD drawings.

AutoMod, by Applied Materials Inc., is a simulation package that is used to develop applications with the purpose of improving the design, configuration and optimization of material handling processes and its primary markets are distribution centres, warehouses, automotive, airports, equipment, shipping, semiconductor, and manufacturing companies, therefore it can be considered to be a rather dedicated software for simulating Production systems. It does not have the graphical model building feature (no drag-and-drop of components), only building using programming and programmed modules. It has runtime debugging, optimization algorithms, code reusing features and model packaging. It is essentially a discrete simulation software, so it does not have any special continuous simulation features. Regarding animation, it also allows both 2D and 3D real-time animation, exporting animations and importing CAD drawings. Actually it is considered to be one of the most powerful 3D animation packages in the market.

3.3 – Major applications of simulation in Production systems

Simulation in the Production systems' context is used mainly as a Decision Support tool but also as a project design tool.

A typical classification for manufacturing issues addressed by simulation is to group them according to the performance indicators that are under analyses and over which the taking of a decision is being analysed. One such classification is present in [11] and it is the following:

- The need for and quantity of equipment and personnel;
- Performance evaluation;
- Evaluation of operational procedures.

In spite of being dated from over 15 years ago, this classification is still relatively accurate, as the basic nature of manufacturing has remained unchanged and will be for a long time to come as it is, very simply, to create something using some kind of resources, adding value to it. To support this statement, the following paragraphs will clarify which manufacturing issues fall under each one of the three categories above and present an example of a survey of simulation in manufacturing systems that supports this classification's up-to-date quality.

Under the first category, it is possible to find issues such as the number and type of machines for a certain task or introduction of a new piece of equipment; number, type and physical arrangement (routes) of material-handling and transportation resources (e.g. forklift trucks, conveyors, carts); location and size of work-in-progress buffers; evaluation of the impact of a change in product volume or mix (e.g. introduction of a new product); labour-requirements planning.

On the second category, one can find throughput, bottleneck and makespan (time-in-system) analysis.

Under the last category, there are issues like production scheduling (including batch sizes and sequencing); inventory level policies; control strategies; quality-control policies; reliability analysis (e.g. maintenance).

The graph in figure 3.3, taken from one of the surveys [9] mentioned in the last section, shows the decision areas supported by the simulation applications observed. Analysing it, one concludes that, in a study made two years ago, 30 out of the 52 systems under study used simulation in the area of production system design, 21 in the area of production policies (including lot sizes and work-in-process levels), 8 in short term production plans / schedules, 4 in inventory policies and 1 in physical plant location. It is easy to see that all of these issues are contemplated in the classification above.

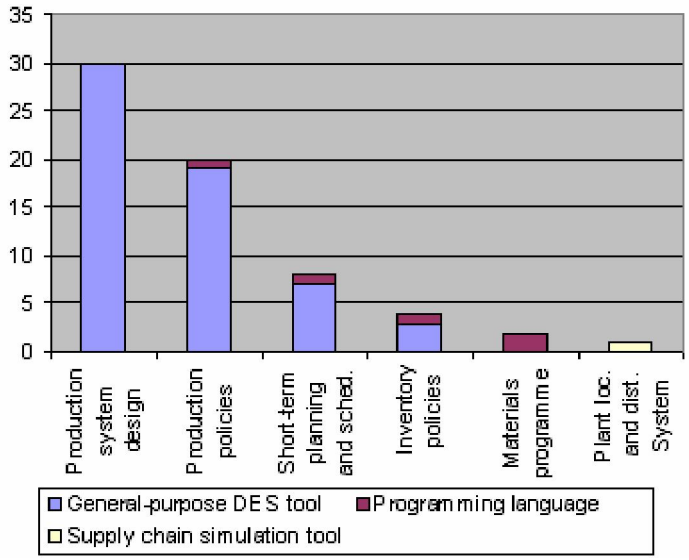


Figure 3.3 – Decision areas supported by the software analysed in [9].

3.4 - Simulation in different areas of Production and Logistics' Systems

One of the proposed objectives of this dissertation was to analyse the role of Simulation in Production systems. Part of this analysis includes a review of the issues/problems in Production systems that could benefit from the use of simulation and a suggestion on a way of grouping and classifying them. A typical classification for manufacturing issues addressed by simulation was already presented in the last section. However, another classification is proposed here. It groups the issues in the Production system according to the different areas identifiable in the system. In order to organise this classification, it was necessary to start by examining the different operations in a Production system and all the actions and decisions associated with each area that was identified. The classification that followed this analysis is organised similarly to the way materials flow in a manufacturing process, i.e., it starts to classify the issues from raw material reception all the way through the transformation process until the final product, without leaving out management functions.

The proposed classification for the potentially problematic areas that could, thus, be subject to simulation is the following:

- Reception of materials;
- Warehousing;
- Internal Logistics;
- Transformation processes;
- Inspection;
- Production and order planning;
- Distribution.

Each one of these areas, and also all of them as a group, present some of the issues in manufacturing systems that were referred to in the last section, i.e., this classification does not oppose to the previous one; it is simply another option. In the following sub-sections it will be explained what are the issues addressed in each of the areas from the above classification, clarifying the performance indicators under analysis in each one.

3.4.1 - Reception of materials

This can be considered to be the starting point in a Production system. In order to manufacture a product or provide a service, one needs raw materials and/or resources. Most of the times, these are not available right on-site, which means that they must be delivered to the facility. Consequently, there must be an area dedicated to receiving them and those receptions must be planned in some way.

Simulation here comes as a testing tool, to examine different layouts for the area and control and planning policies. Simulation can be used, for example, to optimise a system's reception time windows.

3.4.2 - Warehousing

Warehousing involves everything connected with a warehouse: layout, material handling inside the warehouse, picking strategies, replenishment methods. Simulation can be used,

again, as a tool to analyse different scenarios, in an attempt to optimise the existent system and/or test different "what-if" possibilities.

3.4.3 - Internal Logistics

In Internal Logistics, one can also read Internal Movements or, more commonly, Material Handling, although these classifications are a little bit more narrowing (but yet more common), as they discard a part of what logistics concerns, the information flow (considering only the material flow).

In this area, one can find issues mostly related with the moving and storing of work-in-progress along the process. In this context, simulation can be used to test different physical transport solutions, transport strategies or even layouts.

3.4.4 - Transformation process

Transformation process can be a synonym for the manufacturing process. Simulation can contribute here to create a working model of the system that enables to analyse each part of a production cell in order to discover the sources of problems, such as bottlenecks, for example. It can be a useful tool to also test layouts, batch sizes, set-up strategies, etc..

3.4.5 - Inspection

By inspection, we mean Quality Control and Maintenance operations. Quality has become one of the most important order winning factors in today's competitive industry. Consequently, industries have invested greatly in quality control and optimizing their inspection strategies. In order to guarantee good product quality, it is necessary to ensure that all the equipment is in good shape and according to safety standards. Therefore, it is imperative to perform regular maintenance operations. Simulation is used in this area to, again, test different scenarios in order to achieve the best solution. It gives the chance to test quality control policies, as well as different maintenance schedules.

It is also worth mentioning here that simulation is used at quality control already at the product development stage too, even before the product reaches the actual manufacturing stage.

3.4.6 - Production and order planning

This is the area in this classification that concerns what can be considered as management tasks. It is probably one of the areas that has benefited from Simulation for the longest. This is most likely due to the nature of most production planning strategies, which are based in analytical expressions which could be easily manipulated by earlier simulation techniques and do not need a graphical component to be clearly understandable. So, simulation of production and order planning is obviously used to predict the effects of the use of a certain ordering or manufacturing strategy in the company's productivity.

3.4.7 - Distribution

This last area in this classification concerns the external, outbound logistics of the system, i.e., its distribution strategy. Simulation here is used for similar purposes as for the internal logistics matters.

3.5 – Simulation in a distributed environment – Supply Chains

In section 3.2, the subject of Distributed Simulation was briefly referred to. However, there is a need to explore it a little bit more in depth, as it is a growing field in the area of Production and Logistics system's simulation.

Distributed simulation consists of a group of simulation applications running in parallel in different machines but connected by a network and sharing information. When reading this having Production systems in mind, the concept of Supply Chain (SC) immediately arises. This association is, in fact, completely correct as distributed simulation strategies are starting to be developed and applied to SC simulation nowadays. In spite of these developments, this is a very recent research area and only now have SCs been analysed in a more holistic way, taken as a whole but considering the effect of each part. This means that Discrete-Event (DE) modelling concepts need to evolve and adapt to this wider perspective, as in the past DE has looked to manufacturing systems as enclosed inside a box, with little or no communication with the preceding and following links in the SC. However, it is important to mentioned that, besides the efforts to develop distributed simulation, there are already some other attempts to simulate SCs using a mixed approach, based not only in discrete event modelling but also agent-based, for example (e.g. using the AnyLogic software).

Chapter 4

Chosen simulation technology

This chapter has the purpose of presenting the chosen simulation technology. The first section focuses on the software's characteristics, its potential and limitations. The second section presents different applications in the Production System's context, implemented by the author of this piece of work. They are divided in three categories: production lines, internal logistics and warehousing.

4.1 – Characteristics

The chosen simulation platform is AnyLogic. AnyLogic is presently at its 6th version. Its 1st public presentation dates of the year 2000. The headquarters of the company responsible for it, XJ Technologies, are in Russia. It was developed by a group of people with a background not in simulation modelling but in computer science, concurrency theory and distributed systems, which explains many of the features in this software that differ from other common simulation software. It is based in Java programming language [12] and the Eclipse framework [13] and has influences from UML (Unified Modelling Language). It runs in most operating systems (e.g. Windows, Linux, MacOS). The simulations can be exported as Java applets and be run remotely in any web browser.

This software's most distinct feature is that it does not support a single modelling paradigm, as the majority of simulation software, but supports all of the four paradigms mentioned before in chapters 2 and 3: Discrete Event (DE), System Dynamics (SD), Dynamic Systems (DS) and Agent Based (AB). In this software, it is possible to develop models and run simulations using any of the four paradigms alone or combined together.

The models are built with the drag-and-drop of system elements. There are already some libraries of pre-made components for several systems, including manufacturing[2]. These components can be used to model the different elements of the system, such as queues, delays, resource networks, etc. in a Production system, for example.

Other elements available for the model building include presentation components (e.g. buttons, text boxes, geometrical shapes), connection components[3] (e.g. databases, external files, ports), flow diagram, action chart and statechart components, importing of images and CAD[4] drawings, amongst others. The user can manipulate and change parameters of most of the components. There is also the option of building customised functions, adding classes, objects and agents, all of them implemented in Java. It has features of code writing assistance, like automatic code completion.

This software allows different visualisation and animation options as it includes several data analysis components such as histograms, pie charts or bar charts. It also allows run time debugging and analysis.

Figure 4.1 shows a screenshot of the model building environment of this software. On the left there is the Project View where it is visible which models are being built and the tree of components in each one. On the right is the Component menu, where the banner that corresponds to the Enterprise Library mentioned before is selected. In the middle there is the canvas were the different model elements can be dragged onto. At the bottom there is the Problem view, were the compiling errors show up, and the Properties View, where the different properties of all the elements can be defined by the user.

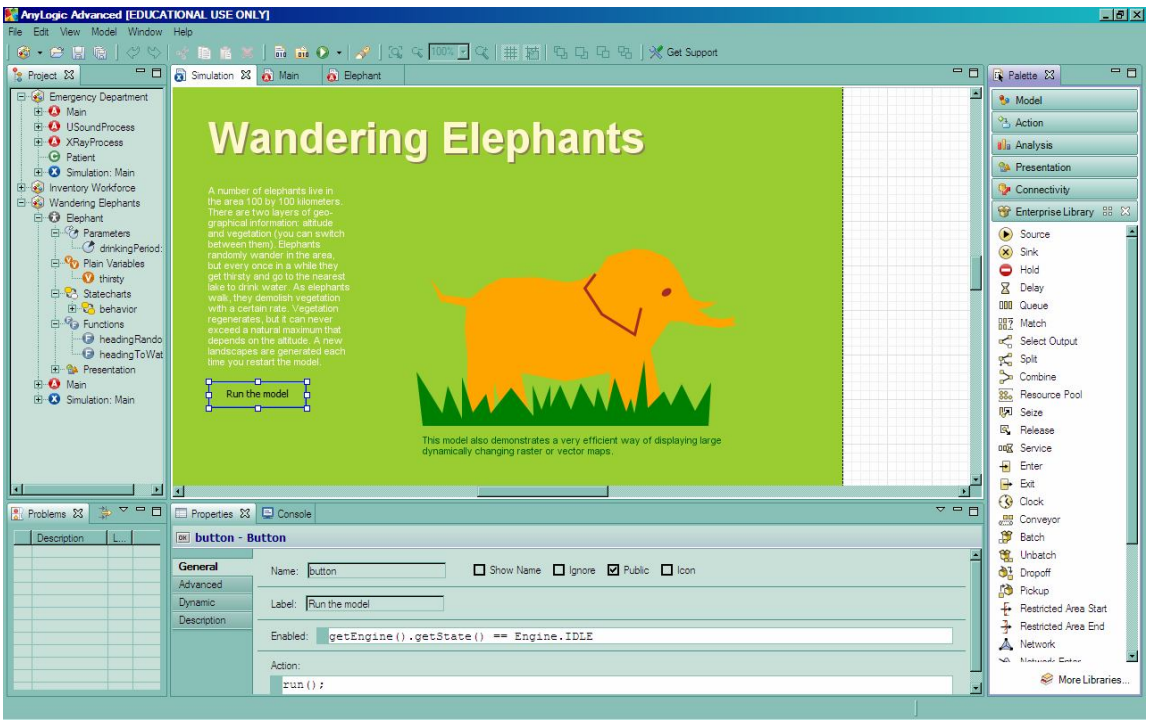


Figure 4.1 – Screenshot of the AnyLogic software.

[2] Actually, the Enterprise Library, which includes the components to model manufacturing systems, is the only library available for all the different licenses for this software, as the Educational license (the one used in this particular research). This accounts for the importance of simulation software for Production systems, as well as for the adequacy of this software for that purpose.
[3] Most of them are only available with the Professional license.
[4] Computer Aided Design drawings, only in the Professional license as well.

4.1.1 – Potential and limitations

This software's biggest advantage is probably the multi-mode modelling capability, specially the fact that it enables the building of hybrid models using the different paradigms, instead of only one exclusively.

As it is based in Java, it is very flexible and Java applets of the simulations are able to run remotely in any regular web browser, allowing easy sharing of the files.

Model building and simulation running are easy because of its drag-and-drop features, enabling virtually anyone to prepare a simple simulation using the pre-made components.

However, to build more complex models, some programming knowledge and also some Statistics background are imperative. The software is not as intuitive as it might seem at first sight with all the graphical support and simple drag-and-drop and connection of components. If a more complex model is needed, with some degree of customisation and even artificial intelligence, it is a lot more demanding for the developer.

The software has a 'Help' menu which is quite complete as it covers all the pre-made components and presents some tutorials but it is not very extensive, i.e. the explanations are not as clear and deep as they could be. There is still no supporting literature for the 6th version as well, apart from the user's manual, but there is an online forum for the user's community, where both professional and educational users share their problems and achievements and also some specialists answer to the user's questions. Another good point is the many demonstration applications of simulations in various fields, implemented using the software and that come with it or are available at the company's website. In spite of this, the developer still has to adopt a trial and error strategy a great deal of times in order to achieve some results, which can prove to be time-consuming and sometimes frustrating.

There is another down point, when comparing AnyLogic with other available simulation software, which is the price. If you take into consideration the survey on existent technologies [10] mentioned earlier, it easy to see that this software is a lot more expensive that the others. However, it is actually the only one supporting different modelling paradigms and running in different operating systems.

4.2 - Different applications in the Production Systems' context

AnyLogic is a general simulation software. It is designed to simulate a wide range of systems, including production ones. It actually includes a library of ready-to-use components for this purpose, the Enterprise Library, as it was mentioned before.

In the next sub-sections, a set of different applications that model simple parts of production systems will be presented. All of them were implemented in the context of this dissertation by its author, in order to try to find an answer for the research questions in section 1.2.

4.2.1 – Simulating Production lines

As a means of getting familiarised with the software, a first set of applications simulating simple production lines was implemented. All of them take advantage of the Enterprise

Library's components. The most relevant functions that were written for these applications can be found in the first section of the Appendixes, A.1.

4.2.1.1 – Situation 1

In the first situation, there was considered to be a generic machine that could process two types of parts, parts A and B, with different processing times for each type of part.

The parts were modelled using a Java class with the attribute 'type', of the Java type 'String'. The parts come either from source A or source B, according to their type. The inter-arrival time used was the default value in both cases, which is an exponential distribution with rate 1. The source objects are connected to a single queue where both types of parts wait for being processed, in a first-in-first-out scheme. The generated part types are saved in a list. The queue object has a graphical representation, where it is possible to see what parts are in the queue, as each type has a different colour. The machine is modelled like a simple delay, where the delay time is set by a function that checks the part type and sets the delay's value equal to the correct processing time. After being processed, the parts go through a 'selectOutput' object, which guides them to the correct 'sink' element according to their type, where they exit the system.

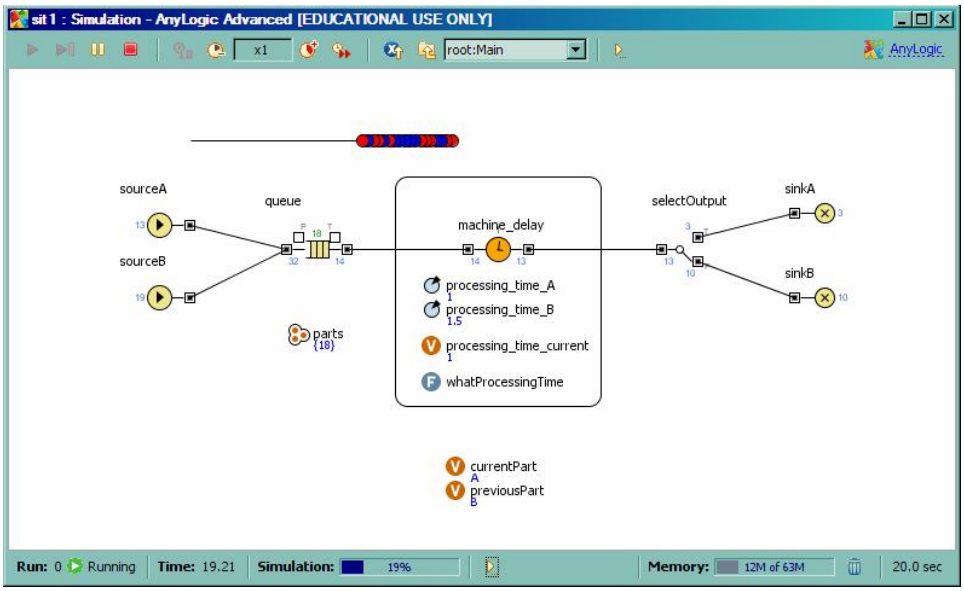


Figure 4.2 – Simulating a simple production line: situation 1.

During the simulation run, it is possible to visualise a lot of information about the system. Looking at the 'source' objects, one can see how many parts from each type have been created since the beginning of the simulation. Regarding the queue, one can know how many parts have entered the queue, how many have exited and how many are in the queue at the moment, as well as an animation of the queue itself, as mentioned before. Concerning the machine, there is information on the processing time for both types of parts as well as the current time selected, the current and previous part's type and how many parts have entered and exited the machine. At the end of the production line, one can see how many parts from each type have been processed and have exited the system through both 'sink' objects.

For example, at the moment of the screenshot in figure 4.2, 13 type A and 19 type B parts had already been created, so 32 parts overall had entered the queue since the beginning and 14 had exited. At that moment, there were 18 parts in the queue and, looking at the queue's animation, one can see that 9 of them were type A and 9 type B. The machine had processed a type B part before and was then processing a type A, with the correspondent processing time of 1. 13 parts had exited the machine, 3 of which were type A and the remaining 10 type B.

4.2.1.2 – Situation 2

The second situation is an improvement of situation 1. Here, it was considered that each part not only has different processing times but also a different set-up time for the machine. The set-up has to be performed every time a part enters the machine.

The parts are modelled and enter and exit the system similarly to situation 1. The only difference in the machine is that it is now composed by three elements: one 'hold' and two 'delays'. The 'hold' component works as a logic gate that only allows a part to go into the machine when it is not processing any other parts, i.e. when it is ready to receive a new part. The first delay models the set-up time and the second one the processing time. There is a function that analyses the part's type and selects the correct set-up time and another one to select the processing time.

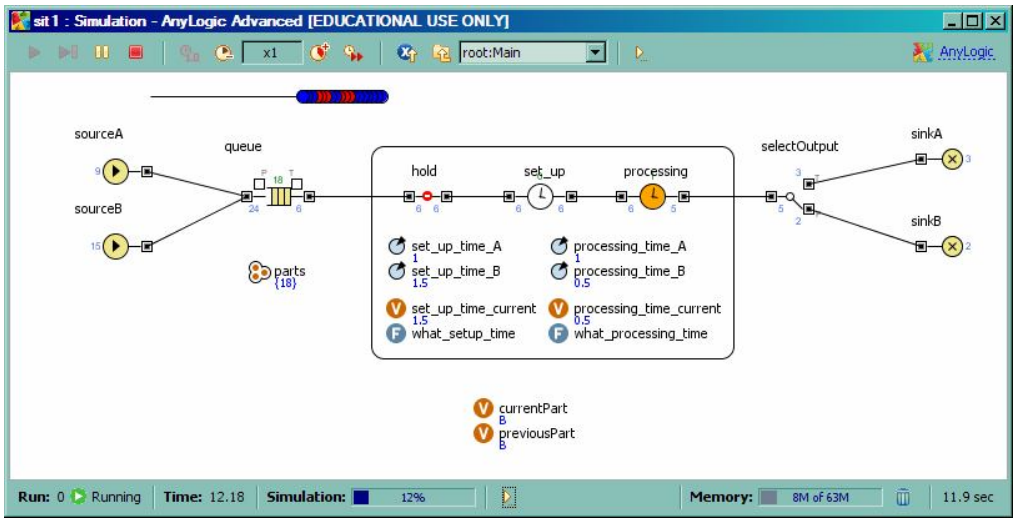


Figure 4.3 – Simulating a simple production line: situation 2.

Similarly to situation 1, one can also access a lot of information about the system during the simulation run. Besides the items mentioned in situation 1, now it is also possible to see the two set-up times available and the current set-up time selected. To know how many parts have been processed by the machine, one can now look to either the 'hold' object or one of the two 'delays'.

4.2.1.3 – Situation 3

Situation 3 is exactly the same as situation 2, except that the machine's set-up is only performed whenever the type of part entering the machine changes. The function that sets the set-up time now only sets a different set-up time for the machine in the correspondent delay element when the part type changes.

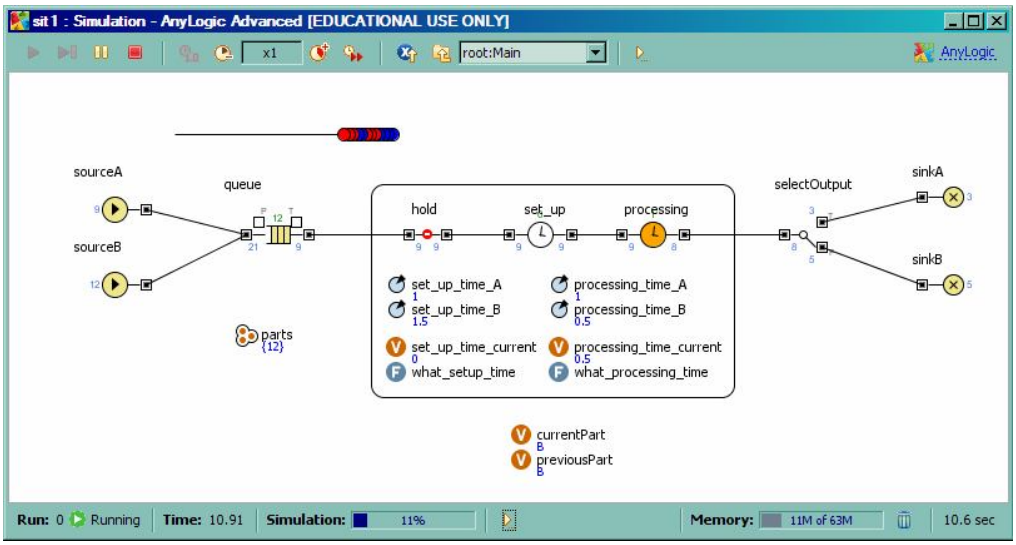


Figure 4.4 – Simulating a simple production line: situation 3.

As situation 3 has only a minor change comparing to situation 2, the information one can get about the system here is exactly the same as in the previous situation.

4.2.1.4 – Situation 4

In situation 4, all the parts in the queue that have the same type should be processed before changing to the other type of part. This means that if we start by processing type A parts, we should only change to B parts when there are no more As in the queue. The set-up is, again, only done when there is a change of type of part. Another component from the Enterprise Library was used here, 'enter'. This is because now the parts do not exit the queue in a first-in-first-out basis, they are removed manually. Therefore, the queue element could not be attached directly to the downstream part of the line. The parts have to enter through the new component. A new function was added to verify if there are still parts with the desired type in the queue, so that they can be removed and processed before changing types. The set-up and processing times are selected as before.

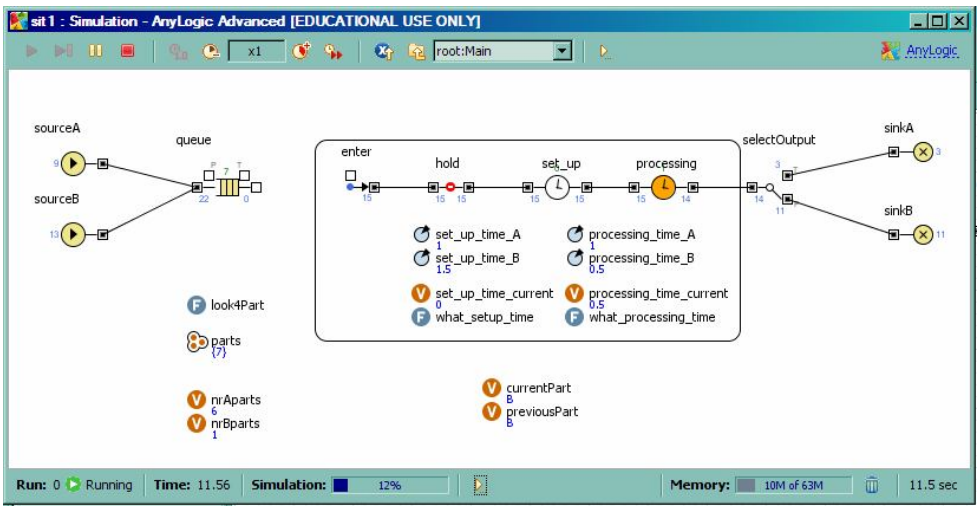


Figure 4.5 – Simulating a simple production line: situation 4.

In this situation, there is no longer an animation of the queue but one can still know how many parts from each type are in the queue by observing the variables 'nrAparts' and 'nrBparts'. As now there is an 'enter' object, this is where one should look for the information on how many parts have exited the queue and entered the machine. All the other information that was available before on the other situations is still accessible now.

4.2.1.5 – Situation 5

The last situation, situation 5, is rather more complex than the previous ones. Each part now has an expiration time, after which it is rejected without being processed, exiting the system through a specific 'sink' object. The parts that have the shortest expiration time should, consequently, be processed first. There is still different processing and set-up times according to part type and the set-up is only performed when the type changes.

The 'Part' class now has a new attribute 'time' whose value is given by a uniform distribution with minimum value 5 and maximum 30, chosen randomly when the part is created. The part's type is saved in one ArrayList and the expiration time in another one. Every 0.1 time units all the expiration times of the parts in the system are updated. There is one function that goes through all the elements in the list and chooses the one with the shortest expiration time. The correspondent part is then taken out of the queue and sent to the machine. As before, there are two functions responsible for selecting the correct set-up and processing times.

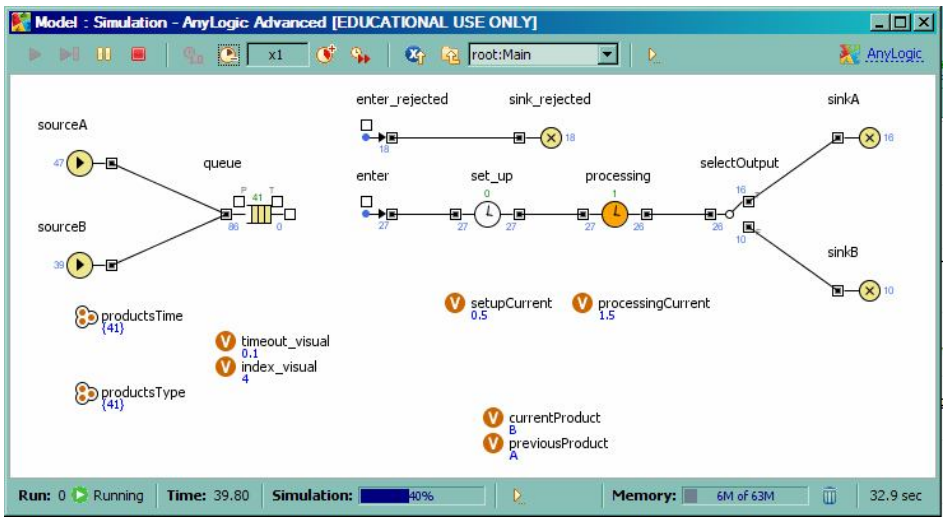


Figure 4.6 – Simulating a simple production line: situation 5.

During situation 5's simulation run, one can see how many parts from each type have been created at both 'source' objects and have entered the queue since the beginning, as well as how many parts are in the queue at the moment. One can also see how many parts have entered the machine and how many have exited, sorted by type. There is also the information on the previous and current part's type and the correspondent current set-up and processing times. Finally, one can see how many parts have been rejected because their expiration time had passed, the position of the part with the shortest expiration time in the queue and the correspondent time value.

For example, at the moment the screenshot in figure 4.6 was taken, there had been created 47 type A parts and 39 type B, so 86 parts had entered the queue. There were 41 parts in the queue and the shortest expiration time was 0.1, belonging to the part at position 4. 18 parts had already been rejected because they had expired. 27 parts had entered the machine and 26 had exited, 16 of which were type A and 10 type B. The previous part was of type A and the current one was type B, with a set-up time of 0.5 and a processing time of 1.5.

4.2.1.6 – Conclusions

In all the five situations described above, the production systems were viewed as perfectly discrete systems, were a countable set of possible states existed and could be modelled by simple queues and delays. Using this type of modelling, it is very easy to understand how the system works and it becomes very easy to model and simulate it, as it was the case here. The Enterprise Library objects used adapted perfectly to these situations, as they were basically the queues and delays mentioned. The visualisation of the information on the system during the simulation was also very accessible and could be easily customised.

4.2.2 – Simulating internal logistics

This next application concerns the internal logistics within a production facility. There was considered to be a production cell that makes components to be processed in another

cell. 'Cell A' produces the components obeying a cyclic event that follows a normal distribution with mean value 0.2 and standard deviation 0.02. The components enter a queue and when they achieve a certain quantity, they are made into a batch and the full boxes of components are sent to wait in a queue until they are taken to the other cell, 'Cell Final'. When a box arrives at this cell, the components are unbatched and put into another queue, waiting to be processed. The components are consumed cyclically, again following the same normal distribution. After being processed, they go to 'queue_final'. The empty boxes of components are put into their own queue to wait to be taken to 'cell A' again, to be refilled. Cell A only produces components when there is an empty box to be filled, following a 'pull' philosophy.

The moving of the boxes is performed by a cart which is defined as 'Agent', controlled by a state diagram and symbolised by the red circle. The cart goes around the different 'stations' (the green squares in the model), picking and delivering the boxes. This is the 'milk-run'[5].

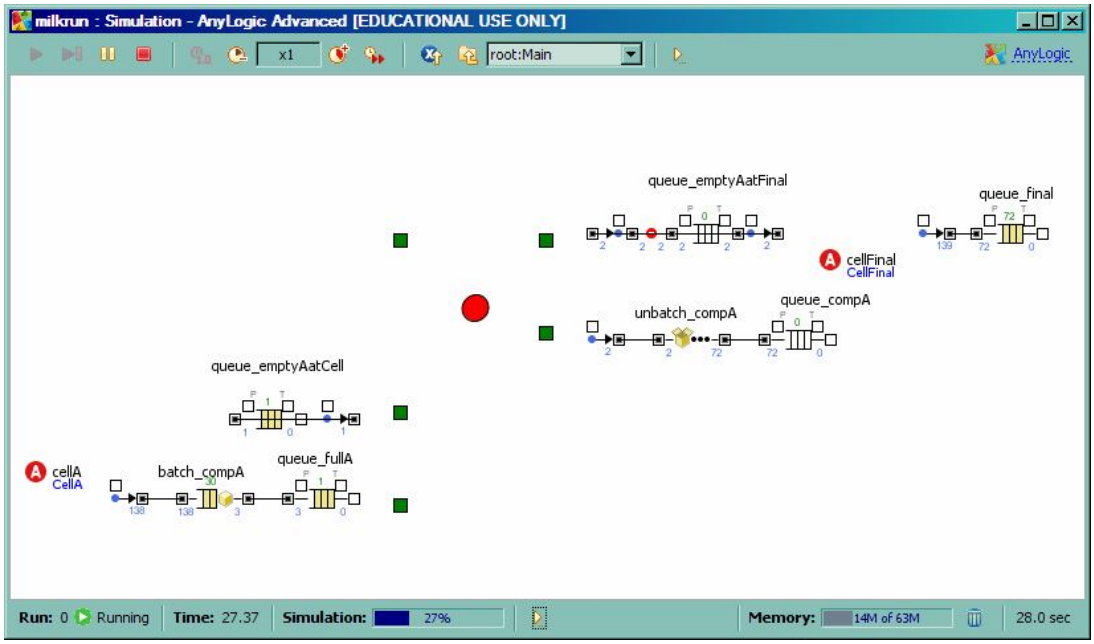


Figure 4.7 – Simulating internal logistics: milk-runs.

Similarly to situations 1 to 5 described in the previous sub-section, it also possible here to visualise a lot of information on the system during the simulation run. At 'Cell A', one can know how many parts have been created, how many have been batched and put in a box in the queue and how many boxes are currently waiting there. One can also know how many empty boxes that came from 'Cell Final' are waiting to be refilled. At 'Cell Final', one can see how many boxes have entered that cell and have been unbatched, how many parts have resulted from that unbatching process and entered the queue and how many are still waiting to be processed. One can also see how many boxes have been emptied at this cell and put in the queue, waiting for the milk-run to come and take them back to 'Cell A' to be refilled.

[5] 'Milk-run' is the name commonly given to internal logistics of materials at Lean Manufacturing systems. It is an analogy to the door-to-door system of delivering milk to people's houses.

For example, at the moment of the screenshot in figure 4.7, 'Cell A' had produced 138 parts, 30 of which were waiting to be batched and the remaining 108 had already been put in 3 boxes of 36 parts each. There was still one full box waiting to be taken to 'Cell Final' and one empty box waiting to be refilled. At 'Cell Final', 2 boxes had already been unbatched, originating 72 parts that had already been processed. 2 empty boxes had been sent to 'Cell Final' before and there were no boxes waiting to go there.

4.2.2.1 – Conclusions

In this scenario, the same option than before was used to simulate the production lines; they were viewed as discrete systems and the Enterprise Library components served perfectly to model them.

Regarding the milk-run, animating it was relatively easy thanks to use of the statechart and specially the use of the 'Agent' class' methods for animating agents. Using the statechart made it simple to define the states the object went through, how the state changes occurred and what were the activities the object was supposed to perform at each state. These activities were modelled using the referred ready-to-use methods.

4.2.3 – Simulating warehouses

Warehousing, one of the categories in section 3.4, can encompass issues like the layout, picking strategy or location. In this simple application, the goal was to create a small warehouse were a simple picking strategy is used. Basically, there is a worker that must get the parts in the warehouse one at a time and put them in the cart on the right. The parts are randomly stored in the warehouse and the simulator must have some degree of intelligence in order to send the worker only to the racks in the warehouse that are not empty. The location of the parts is stored in two lists, one containing the part's row and the other one the column.

The worker is sent to the part's location in the same order as they appear in the lists. He/she picks one part, takes it to the cart and then goes back to the warehouse to get the next one. The worker is defined as 'Agent' in order to take advantage of this class' methods to animate objects in the environment. Its behaviour is controlled by a state diagram, visible in figure 4.9.

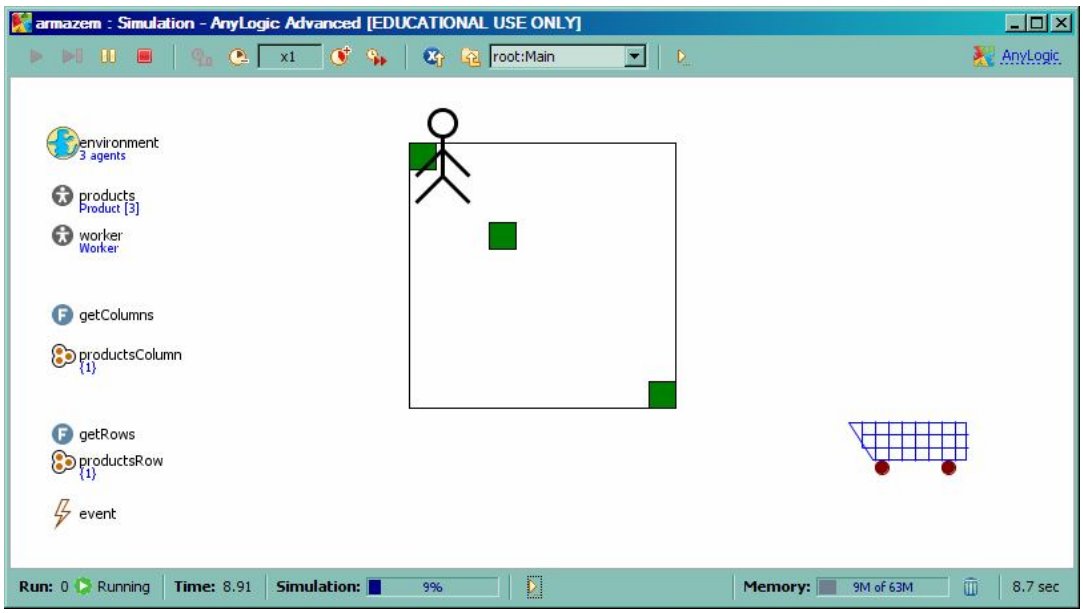


Figure 4.8 – Simulating warehouses: picking.

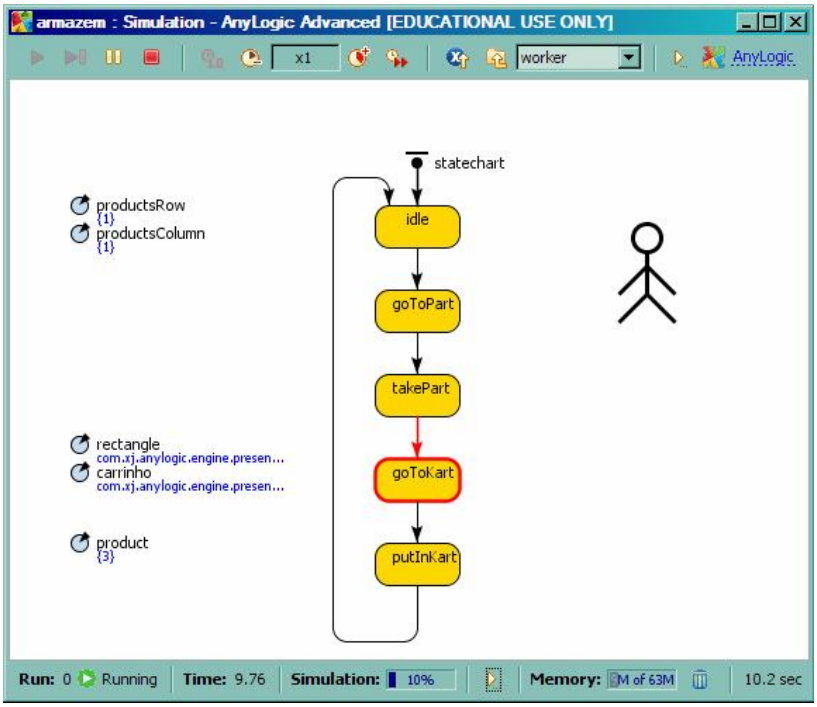


Figure 4.9 – Simulating warehouses: worker's statechart.

During the simulation run of this application, it is possible to see where the parts are located at the warehouse and also the worker's movements. To know how many parts are in the system both in the warehouse and already in the cart, one can look to the 'environment' or the 'products' icon. By clicking on the 'ArrayList's icons, one can see the row and column were the part is in the warehouse. If one changes the view to see the worker's statechart, as in figure 4.9, it is possible to get the current active state.

4.2.3.1 – Conclusions

In this application, it was possible to continue exploring the animation of agents and how they interact with other objects in the system. It was also useful to try the possibility to work with both discrete environments (the warehouse) and continuous ones (the Main object, where the worker moved around).

Chapter 5

Case Study – Internal logistics in an assembly factory

This chapter refers to the real case study that was address with the purpose of supporting the investigation on the subject of a simulation based approach of Production systems. Its first section briefly describes the company and its department under study, followed by the system requirements. After this, some examples of the system's UML class and interaction diagrams are presented. The last section refers to the implementation using the chosen software (AnyLogic), with separate sections for the "as-is", the "to-be" and the "what-if" situations.

5.1 – Characterisation of the system

The case study concerns the simulation of the internal logistics of a department in a factory belonging to one of the leading companies in the area of car-radio production. This case study was chosen because research work was already taking place there regarding this issue. Further information besides the one that will be given on the system and the problem addressed can be found in [14].

The department under study takes care of assembling the electronic components. It is composed by two distinct areas: the warehouse and the production lines. The warehouse has different rows and each row has different racks were the components are stored. There are two types of components: boards (where the components are mounted) and reels of electronic parts (e.g. resistors, capacitors, inductors, etc.). Each component's reference indicates the correspondent rack.

The parts are delivered to the production lines using a milk-run system with defined routes. There are three milk-runs operating in the department. The first one delivers to the set of the most productive lines. The other two do the same route but deliver to different lines. There are specific delivery points next to the production lines. There is only one

logistics train, belonging to the third milk-run. The other two use handcarts. Each milk-run has a cycle of 20 minutes. They depart at different times, 5 minutes apart from each other.

Each milk-run is done by a single worker that does the picking of the parts at the warehouse and then goes around the lines that belong to his/her route delivering the parts. Each line orders the parts it is running out of whenever needed. When a new cycle begins, the worker gets the list of the parts that were ordered during that period of time and that he/she is supposed to pick and to each line they are supposed to be delivered to.

Referring to the proposed classification in section 3.4, the issues in this system fall under the categories of Warehousing and Internal Movements. Simulation was used in this system with the purpose of comparing different scenarios, more specifically the current situation, a new proposed situation and another hypothesis that is not going to be implemented in the real system, for the time being. This comparison is necessary as this policy change within the department has as goal to make the system operate according to *Lean* practices. Very briefly, *Lean* theories aim at reducing all the *waste* in a system as far as possible, giving priority to all *value-adding* activities. According to this theory, there are seven types of waste; a very common example of a waste in a production system is unnecessary idle time.

5.2. - System requirements

To be able to build a model and simulate a system it is necessary, besides characterising it, to define what the system's requirements are, i.e. what is demanded from the model and the simulation, what they are supposed to do.

So, this simulation is supposed to represent both the picking and the delivery process. The model must include a representation of the department's layout, the workers and the resources they use, i.e. the carts. It must be able to generate the production lines' orders of parts. It also has to be able to represent the picking process with the workers' movements inside the warehouse and the delivery process too with the workers' and the carts' movements around the production lines. A visual simulation was chosen as it was considered to be extremely more appealing that a simple display of values with no visual support to them.

As input data, there is the system's physical layout, the information on the picking and delivery routes and also several probability distributions that represent the system's behaviour and several important parameters. The following list enumerates the parameters that should be modelled using these distributions, obtained through historical data:

- Worker's picking time per part (reels and boards);
- Worker's walking speed in the warehouse, while picking;
- Worker's delivery speed at the production line (reels and boards);
- Cart's speed (logistics train and handcarts);
- Number of orders per line per cycle;
- Parts ordered per cycle (or at least the module/rack where they belong to).

To obtain the random orders from each production line, a random number generator should be used.

As output data, the simulation must provide information on the system's characteristics that are being analysed. These measurements can be called the *key performance indicators* of the system. They are certain characteristics of the system that can measure both the *efficiency* and the *effectiveness* of the system. An efficient system is one that works well, without waste. An effective system is one that does what it is supposed to do; it produces the desired result for its client. In this case, we are studying the picking and delivery process, so there is a set of indicators that are of interest, such as:

- Picking time;
- Milk-run's cycle time;
- Idle time between cycles.

5.3 – UML diagrams

The different entities and resources in the system were organised in classes and objects, according to the Object-Oriented Programming (OOP) paradigm. In OOP, a computer program is viewed as a set of different objects that cooperate between them, sending and receiving messages and processing data, opposing to a set of subroutines, a list of tasks to be completed. A class defines the attributes (characteristics) and methods (behaviours) of the objects in the system; it is an abstraction. An object is a runtime instance of the class, an exemplar, a somewhat "physical" manifestation of the class' abstraction. For example, in this system we have the class Person and several exemplars of the class, such as the 'logistics operator' named 'Joe'.

To graphically show the structure of the different classes in this system and their interaction, UML (Unified Modelling Language) was used. UML is not a programming language; it is a general purpose modelling language, a visual specification language for object modelling.

The following figure shows a possible *class diagram* of the system, a structure diagram that includes all the classes that were defined, most of their attributes and some methods as an example. It was decided not to include a complete class diagram with all the information as it would be too extensive and somewhat repetitive; only the essential information towards the comprehension of the model was included. In the next section the implementation of the model will be discussed and the uses of all the classes will become clearer.

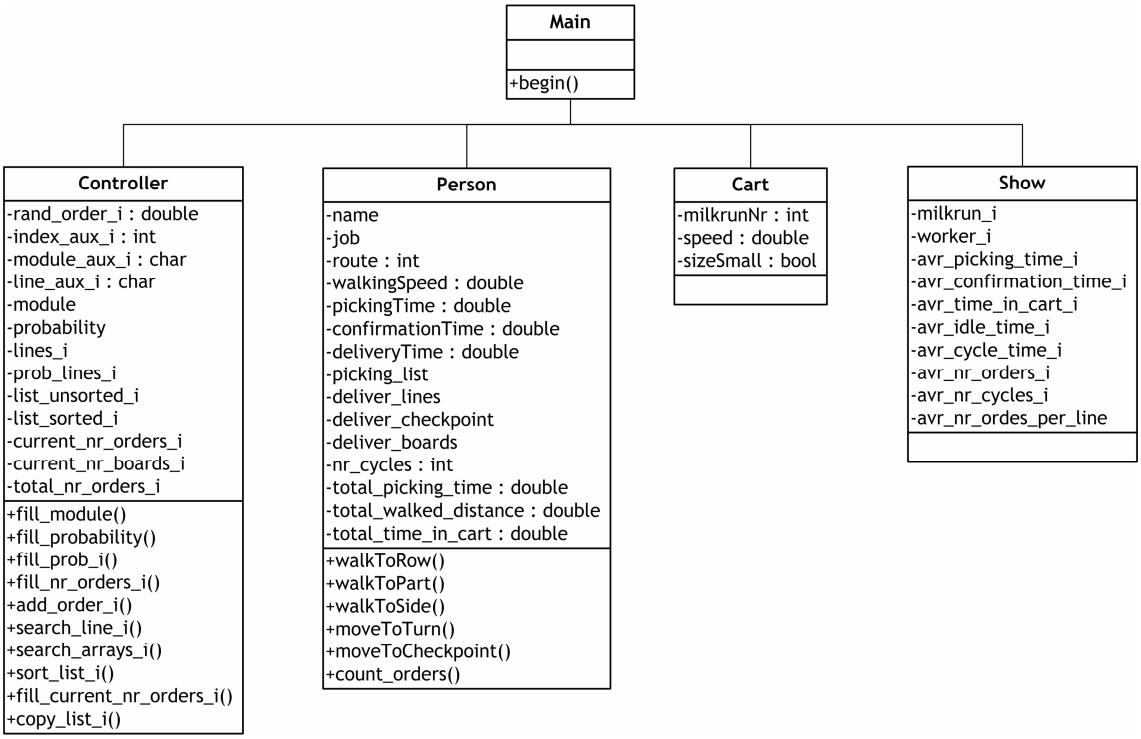


Figure 5.1 – UML Class Diagram of the system.

In a production system like the one in this case study, there are many objects interacting with each other. In order to make some of those interactions more explicit, the UML behaviour diagram on the following page was built. It is a possible *interaction diagram* of the system which shows the message sending and receiving between the several objects during one milk-run cycle. Again, it does not include all the processes in the model in order to make it simpler.

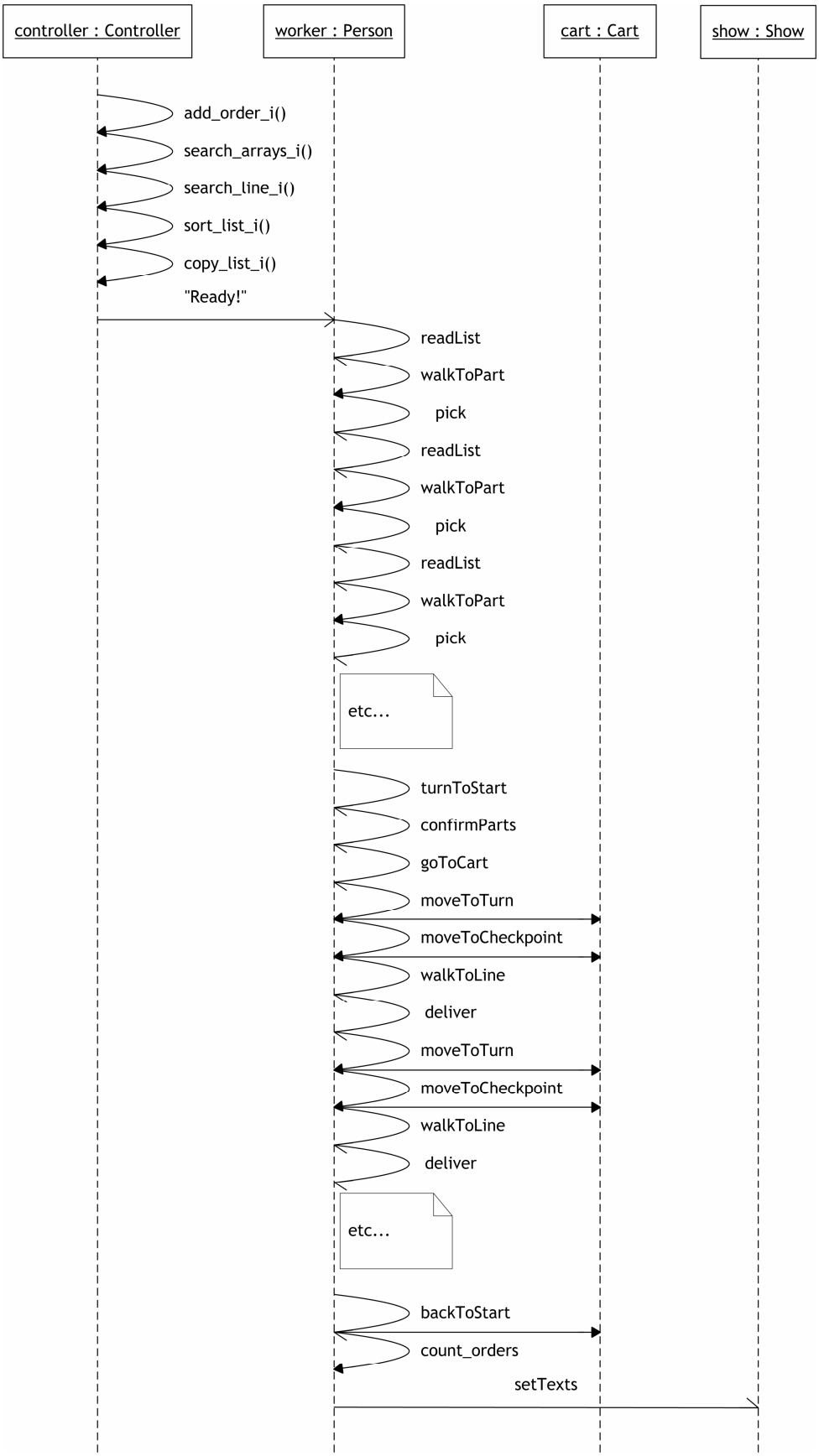


Figure 5.2 – Possible UML Interaction Diagram of the system.

5.4 – Implementation

To start the model's implementation, all the classes and their attributes and methods were defined in a new model in the AnyLogic software.

The 'Main' class was used as the environment where the objects would move and interact in and it contains the graphical representation of the layout. All the important places in the department's layout, i.e. the areas and specific places where the workers and carts should move in and go to are marked in the image of the layout, either through the use of invisible pixels (as in the marking of the different rows and racks in the warehouse, for example) or visible shapes (as in the marking of the delivery points near the production lines, for example). There is one event in the 'Main' class, which takes place at the very beginning of a simulation run. It concerns the creation of the different entities in the system, i.e. the workers and carts, and the initialisation of some of their parameters.

The 'Controller' class has only one object, also called 'controller'. This object controls the generation of random orders from the production lines. It contains all the information concerning the parts that exist in the system, the probability of each part being ordered, the name of the lines in the system, to each milk-run route they belong to and the probability of each line issuing an order during a cycle. This information is stored in lists, belonging to the Java class 'ArrayList'. The 'controller' has methods to generate a new list of orders at each cycle, obeying to the probabilities that were referred to. This process is clarified at the end of this section. This class also defines how to sort the orders' list according to the order that the parts are being picked from the warehouse and how to transmit the list onto the worker. It keeps the information about how many orders were made by each line at the given cycle and in total since the beginning of the simulation. In the 'controller' object, some events were defined. The first one, 'controller_genesis' is triggered at the start of the simulation and it initialises all the lists containing the historical data, i.e. all the probabilities that were mentioned in this paragraph. The other events are cyclic and represent the different milk-run cycles, which have a defined first occurrence time and that are repeated onward, after the specified time interval (the defined cycle time, 20 minutes).

The 'Person' class was created to model the workers of the department under study. It was defined as an 'Agent', to take advantage of certain features that agent-based modelling in AnyLogic offers. In this particular case, using agents enabled the easier implementation of the animation of the workers' movements in the system, as the 'Agent' class offers specific methods for this purpose. The shape chosen to depict an object of this class was a simple red circle.

The 'Person' class has attributes such as the person's name and job, the route he/she is assigned to, the picking and delivery time for both types of part and walking speed. The times and speed were modelled using normal distributions, with values attributed according to historical data. The 'Person's behaviour is described through a state diagram, which includes all the states the person goes through in the system and how the state changes occur. This class includes methods that control the person's movements inside the warehouse and around the production lines during the delivery. It also has several variables that store statistical

information and the methods to obtain it. This information is then passed to the class 'Show's object.

The class 'Cart' models the carts in the system, both the handcarts and the logistics train. Its attributes include the milk-run number it is affected to, its size which defines its speed (the small carts are the handcarts) and its speed modelled by a normal distribution based on historical data collected from the system. The behaviour of this class' objects is defined by a state diagram. This class has no methods, as it does not perform any activities in the system and it is controlled by the Person object. It was also defined as 'Agent', for the same reasons as the class 'Person'. The image that depicts this class' objects is a green square.

The last class that was defined is the class 'Show'. This class has the sole purpose of displaying information about the system's behaviour during the simulation. It only contains text objects which are written by other objects in the system, more specifically the Person's objects. All the key performance indicators mentioned before are shown here at runtime:

- Picking time;
- Milk-run's cycle time;
- Idle time between cycles.

Other measurements are also included, such as:

- Average walked distance inside the warehouse;
- Average time spent inside the cart delivering;
- Number of cycles performed in the current simulation run;
- Average number of orders per cycle;
- Average number of orders per line.

These other measurements have the main purpose of validating the model. They are used as comparison points with the input data, in order to confirm that the model is behaving similarly to the real system as it is desired to.

This class need not have been created if the software license allowed the writing of external files, which would mean that the data obtained from the simulation could directly be stored in an external database or spreadsheet.

To provide a better insight on how discrete-event simulation is processed, the implementation of the generation of random order lists will now be explained.

To start with, the probability distributions used were actually accumulated probability distributions, which could be used to build a histogram where each column would correspond to a possible event/state with the given probability, for example the line that issued an order.

So, at the beginning of each milk-run cycle, a random number was taken from a uniform discrete distribution that represented the possible total number of orders coming from all the lines belonging to that milk-run. Then, for each order, a pseudo-random number was obtained using a pseudo-random number generator provided by the software. The correspondent value from the accumulated probability distribution of the lines in the milk-run was then selected. This value gives the line that issued the order. It is now time to get the reference of the part that was ordered by this line. Another pseudo-random number is generated and the correspondent value in the accumulated probability distribution of the parts' references is

taken. This value gives us the part that was ordered. The reference of the part and the line that ordered it are placed in the order list and the process is restarted until there are as many part references and lines as the number of orders we got at the beginning of the cycle. The list is then sorted in order to put it in the same order as the picking is done in the warehouse.

During the implementation process, a first trial version of the application was built. This version modelled only one milk-run cycle and was used to test the different functions that were re-used in the following versions, with minor changes. In this first version, there was only one worker and one cart in the system. There was a set picking list, similar to a real one. After the picking was done, the worker would go around all the delivery points near the production lines (named 'checkpoints' in the application) and deliver two parts in each line. In the end, he/she would go back to the starting point.

It is important to point out that all the functions mentioned in this section were developed using the Java programming language. The more significant ones can be found in the Appendixes.

The next sub-sections will clarify how the implementation for each one of the three scenarios analysed was done. Table 5.1 below shows a summary of the main differences between the three situations.

Table 5.1 – Summary of the main differences between the "as-is", "to-be" and "what-if" scenarios.

scenario	nr of milk-runs	picking and delivery by the same worker?	route	nr of production lines served
as-is	3	yes	1	5
			2	10
			3	9
to-be	2	yes	1	6
			2	18
what-if	1	no	1	24

5.4.1 – The "as-is" situation

This situation corresponds to what the system was like at the beginning, before any changes were made. The system here was organised as it was described in section 5.1. There are three milk-runs. The first one delivers to three checkpoints that correspond to the most productive lines, therefore the more demanding in terms of part orders. Its route is a simple straight line. The other two milk-runs deliver to the other production lines. They do the same route around the department, but they deliver to different checkpoints. The second milk-run delivers to four checkpoints and the third to five. The milk-runs start five minutes apart from each other and they have 20 minutes cycles. The worker that does the picking of the parts at the warehouse is the same that delivers them afterwards.

The following figures show runtime screenshots of the simulation. The first one, figure 5.3, shows the Main section, were the graphical representation of the system is and where you can see the objects' animations. The worker from milk-run 1 is doing the picking at the warehouse in the top left corner, the worker from milk-run 2 is waiting to start a new cycle at the warehouse's entrance and the worker from milk-run 3 is at the top, coming back to the starting point after delivering the parts to its production lines.

Figure 5.4 shows one of the Person's objects, where it is visible the statechart that controls its behaviour. The state diagram is at the state that corresponds to driving the cart to the delivery point. On the left there are all the class parameters and some auxiliary variables.

Figure 5.5 is the Show object, were all the numerical indicators of the system's performance can be seen.

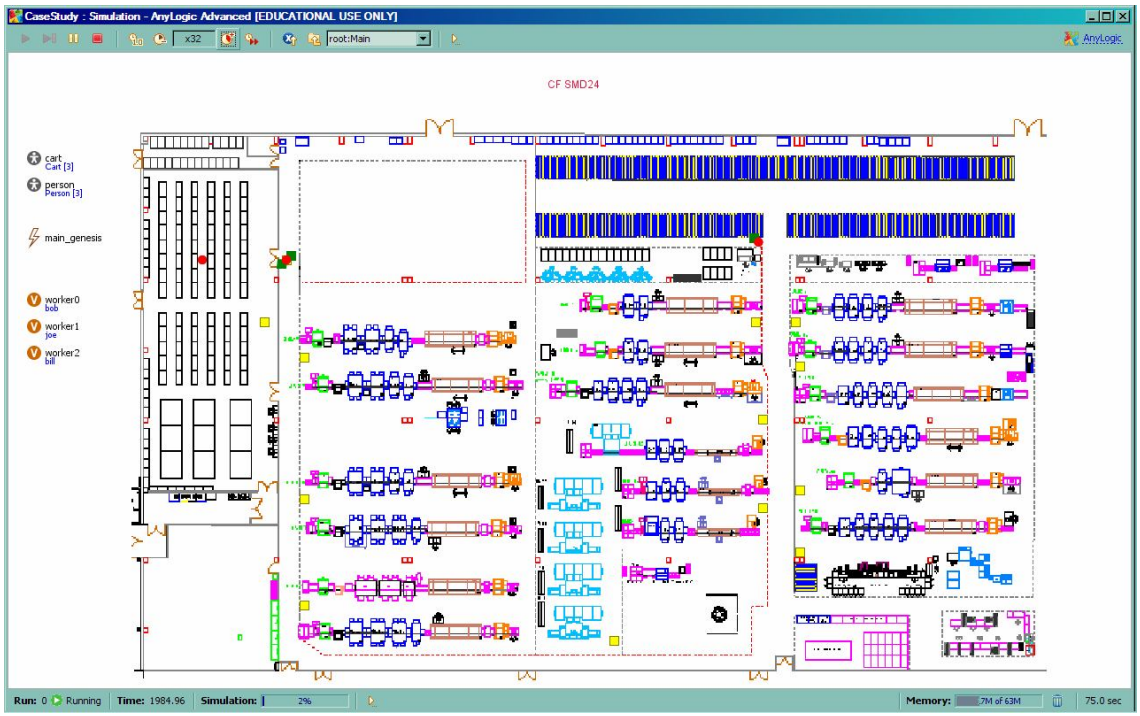


Figure 5.3 – The "as-is" situation: screenshot of the Main section.

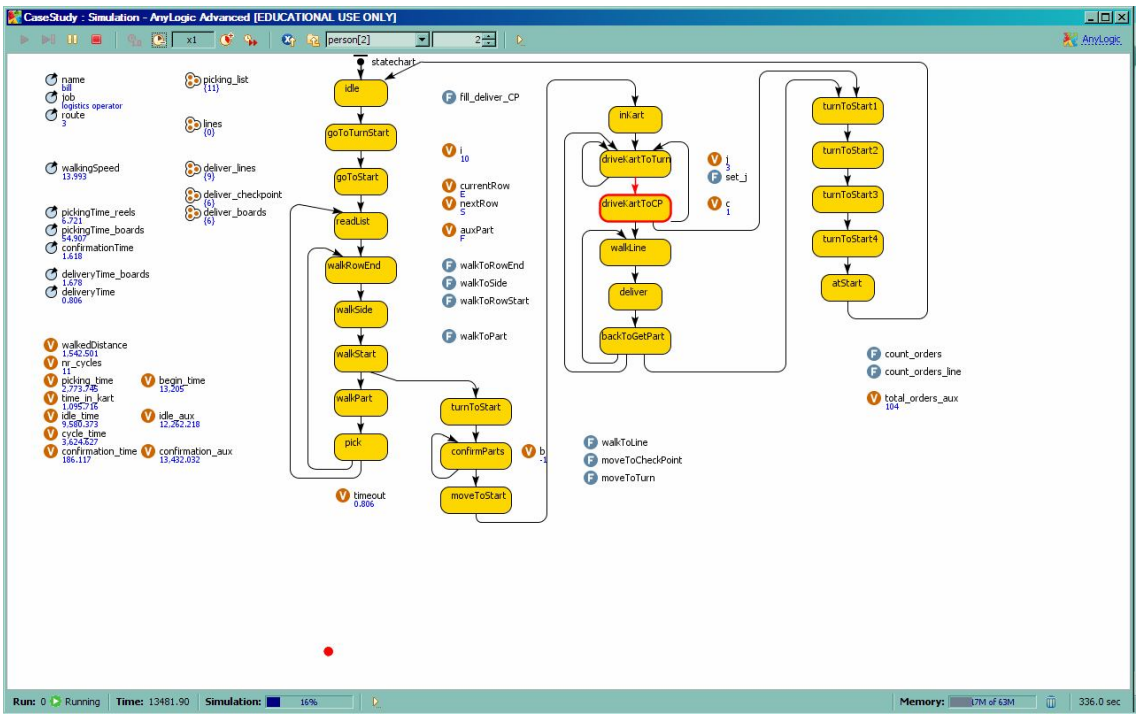
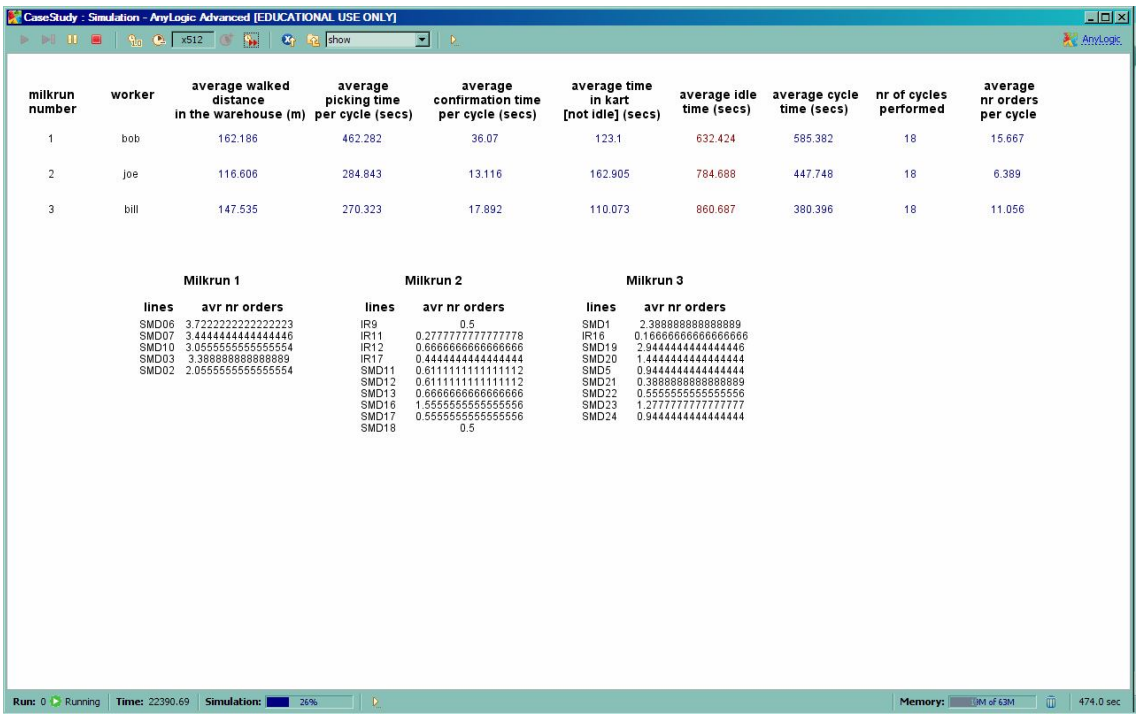


Figure 5.4 – The "as-is" situation: screenshot of one object of the class Person.



milkrun number	worker	average walked distance in the warehouse (m)	average picking time per cycle (secs)	average confirmation time per cycle (secs)	average time in kart [not idle] (secs)	average idle time (secs)	average cycle time (secs)	nr of cycles performed	average nr orders per cycle
1	bob	162.186	462.282	36.07	123.1	632.424	585.382	18	15.667
2	joe	116.606	284.843	13.116	162.905	784.688	447.748	18	6.389
3	bill	147.535	270.323	17.892	110.073	860.687	380.396	18	11.056

Milkrun 1

lines	avr nr orders
SMD06	3.7222222222222223
SMD07	3.4444444444444446
SMD10	3.0555555555555554
SMD03	3.388888888888889
SMD02	2.0555555555555554

Milkrun 2

lines	avr nr orders
IR9	0.5
IR11	0.2777777777777778
IR12	0.6666666666666666
IR17	0.4444444444444444
SMD11	0.6111111111111112
SMD12	0.6111111111111112
SMD13	0.6666666666666666
SMD16	1.5555555555555556
SMD17	0.5555555555555556
SMD18	0.5

Milkrun 3

lines	avr nr orders
SMD1	2.3888888888888889
IR16	0.16666666666666666
SMD19	2.9444444444444446
SMD20	1.4444444444444444
SMD5	0.9444444444444444
SMD21	0.3888888888888889
SMD22	0.5555555555555556
SMD23	1.2777777777777777
SMD24	0.9444444444444444

Figure 5.5 – The "as-is" situation: screenshot of the Show section.

Several simulation runs were done using different random number generators in order to achieve different results. The stop time of each simulation run was defined to coincide with the duration of a day, which translates to 68 or 69 cycles of 20 minutes. Table 5.1 presents the results for 10 random simulation runs. The figure that follows it shows the relationship between the picking time, confirmation time, time in cart delivering and idle time in the

three different routes. It is important to point out that the number of repetitions was chosen only to illustrate some results obtained with the model. If the purpose of these runs were to make a thorough analysis of the system, the number of repetitions should be higher, in order to have an also higher degree of trust in the results. With a better sampling of the system (i.e. with more simulation runs), on one hand we would have more accurate results as exceptional cases would not greatly influence the final results; analysing fewer cases/runs means that the final analysis is more sensitive to variation. On the other hand, there is also less chances of occurrence of exceptional cases with fewer samples, which means that all the possible situations might not be taken into account, resulting in an analysis that is not very precise.

Table 5.2 – The "as-is" situation: results of 10 simulation runs.

Milk-run	simulation run	avr walked distance	avr picking time	avr confirmation time	avr time in cart	avr idle time	avr cycle time	avr nr orders
1	1	154,439	423,32	30,817	110,513	716,138	533,833	14,899
	2	161,371	426,413	32,144	105,365	716,452	531,777	14,696
	3	165,195	498,89	31,299	119,661	631,396	618,55	15,261
	4	161,802	486,096	27,556	120,13	644,818	606,226	15,826
	5	163,348	487,132	36,319	110,736	654,975	597,868	14,464
	6	164,723	361,544	38,223	108,162	778,564	469,707	15,275
	7	157,688	296,899	35,636	106,895	844,256	403,794	15,014
	8	164,168	352,091	33,681	125,577	770,836	477,668	14,986
	9	165,957	519,84	37,454	108,234	620,284	628,074	14,609
	10	164,696	509,126	32,002	114,116	628,072	623,241	15,087
	avr	162,3387	436,1351	33,5131	112,9389	700,5791	549,0738	15,0117
	%	-	33,99%	2,61%	8,80%	54,60%	-	-
2	1	119,156	223,555	12,027	143,215	885,006	366,771	5,826
	2	126,068	261,601	12,866	144,856	847,255	404,771	6,088
	3	118,98	238,548	9,51	159,622	853,987	398,17	5,971
	4	116,702	288,983	15,733	146,544	816,251	435,527	5,71
	5	122,321	229,323	14,666	140,988	881,064	370,311	5,986
	6	121,948	230,522	15,665	164,873	857,433	395,395	6,029
	7	120,534	281,394	13,477	160,551	812,09	440,452	5,824
	8	119,234	244,694	13,251	148,026	858,33	392,72	5,942
	9	120,455	196,276	12,384	135,182	919,25	331,457	5,739
	10	118,184	196,716	15,528	150,414	903,812	347,13	5,986
	avr	120,3582	239,1612	13,5107	149,4271	863,4478	388,2704	5,9101
	%	-	18,90%	1,07%	11,81%	68,23%	-	-
3	1	143,59	334,87	21,731	125,07	797,233	459,94	10,544
	2	146,68	356,276	24,249	129,18	772,087	485,456	11,074
	3	145,743	359,611	22,346	129,088	768,89	488,699	11,118
	4	149,402	303,149	25,353	126,083	827,496	429,232	10,941
	5	140,485	356,559	20,472	124,782	776,142	481,341	10,662
	6	141,813	236,788	26,156	123,435	865,505	360,223	11,029
	7	145,258	392,761	25,091	129,351	735,962	522,112	11,132
	8	147,787	423,202	30,773	129,727	705,592	552,928	10,912
	9	148,465	423,959	18,266	123,184	711,195	547,243	10,676
	10	148,798	382,303	35,049	121,79	753,72	504,093	11,338
	avr	145,8021	356,9478	24,9486	126,169	771,3822	483,1267	10,9426
	%	-	27,90%	1,95%	9,86%	60,29%	-	-

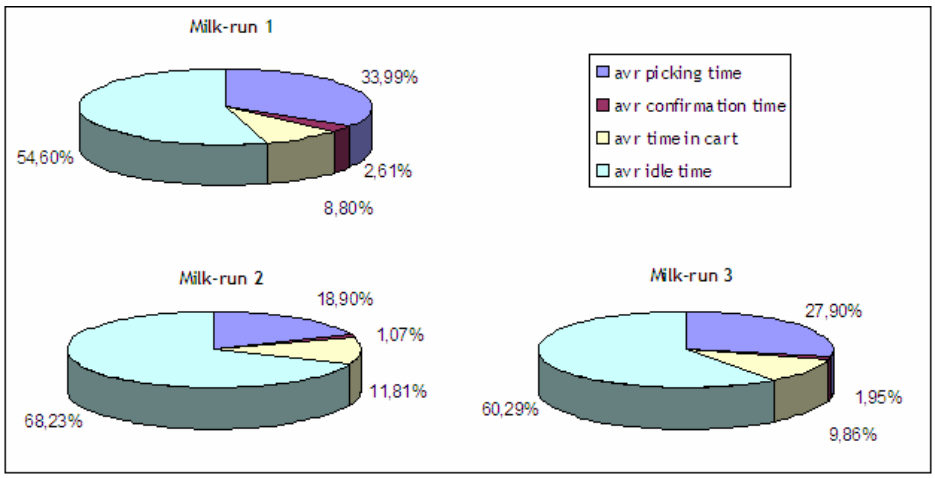


Figure 5.6 – The "as-is" situation: pie-charts of the results of 10 simulation runs.

It was possible to verify that model that was built was relatively accurate, according to the input data available. It was said before that a model can only be as good as the input data we have provide it with. This is definitely the case here, as it would be expected. The results that were obtained confirmed the initial data, i.e. the randomly generated order lists were consistent with the probability distribution adopted and, therefore, with the data available about the real system. This was also the case with the other indicators analysed, as the picking and delivery times and also the cycle and idle times during the delivery process.

However, making a deeper analysis of the real system, it would be possible to see that some situations that occur there were not taken into account in this model. For example, there is not any case of malfunctions in the system. In the real system, there are some rare situations were the parts are not available at the warehouse, the logistics train gets delayed, the delivery process takes longer for some reason, the orders are not uniformly distributed during the day (e.g. there are peaks at the beginning of a new shift), etc. These situations are not present in the model, mostly because there was no accurate information about them. There is the knowledge that they do happen in the real system but there is no registration about how often they occur or for what reason, therefore they could not be translated into a mathematical or a cause-effect situation that could then be included in the model. In spite of this fact, the model that was built and the simulation results obtained from it can still be considered useful for a good analysis of the system's behaviour, as they represent the real system with some degree of accuracy, even though they do not cover all the possible situations in the system. Actually, modelling is an approximation of reality and the assumptions made in this case discarded some facts that have a very low occurrence rate and can, therefore, be ignored without incurring in exaggerate errors.

5.4.2 – The "to-be" situation

The idea for this system, since it started to be analysed, was to change its operating scheme. From previous analysis, it was concluded that the milk-run system here could be improved. Referring to the results obtained with the simulation of the "as-is" situation, it is possible to verify that the idle time in all three routes accounts for the majority of the total cycle time, i.e. the workers spend more than 50% of the total cycle time in an idle state, without performing any value-adding activity. The solution that was selected was to reduce

the number of routes, from three to two in order to reduce the workers' idle time between cycles.

In the "to-be" situation, there are only two milk-runs. The first route remains the same but the second route is a combination of routes 2 and 3 of the "as-is" situation. The second route now goes around the production line and the delivery is made to eight checkpoints, corresponding to the less productive lines all together. The logistics train is used here. The picking and delivery are still made by the same worker.

Implementation wise, the "as-is" version of the application only had minor changes in order to represent the "to-be" version. Only two workers and two carts were created at the beginning. Two new 'ArrayLists' were created, containing the information on the route and the probability of getting an order from each line. The function that controls the auxiliary variable to indicate the next delivery checkpoint was also altered in order to contemplate the new route. The text fields in the Show class and the Main class that corresponded to the third route were suppressed.

Similarly to the previous sub-section, the following figures will show runtime screenshots of the Main section, one of the Person's objects and the Show class. Figure 5.7 shows the Main section. The worker from milk-run 1 is doing the picking at the warehouse on the top left corner and the worker from milk-run 2 is delivering to the production lines on the right. Figure 5.8 shows the screenshot of one object from the Person class. The state corresponding to the confirmation of the parts after the picking is active. Figure 5.9 corresponds to the Show section.

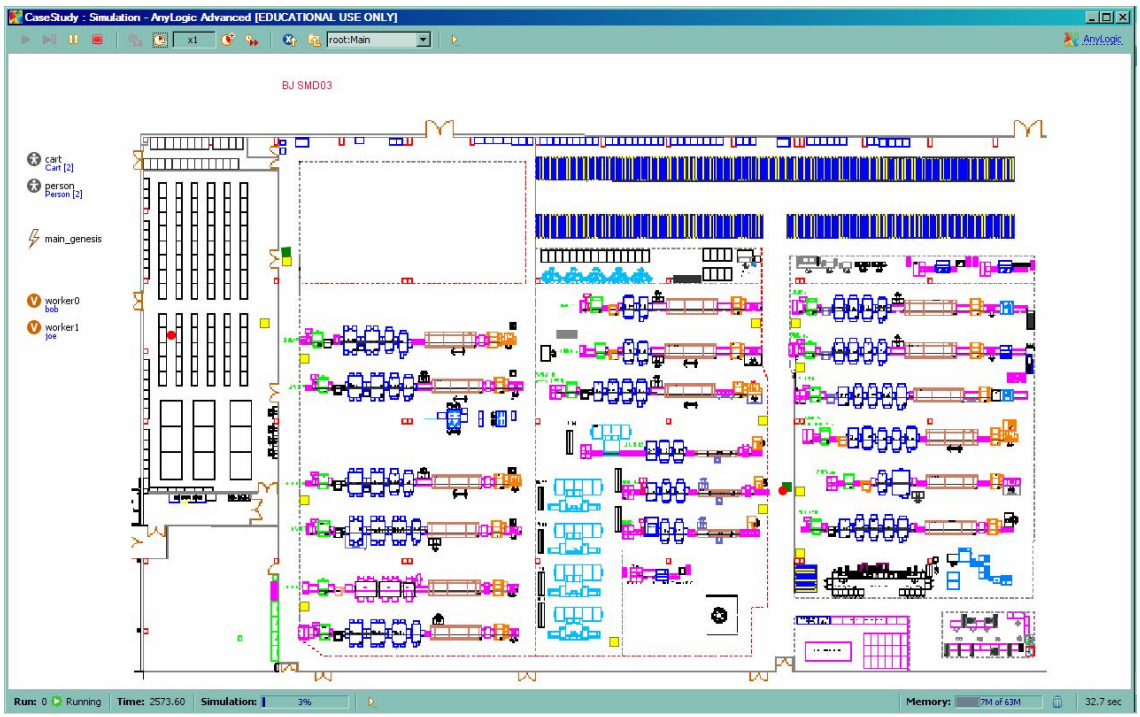


Figure 5.7 – The "to-be" situation: screenshot of the Main section.

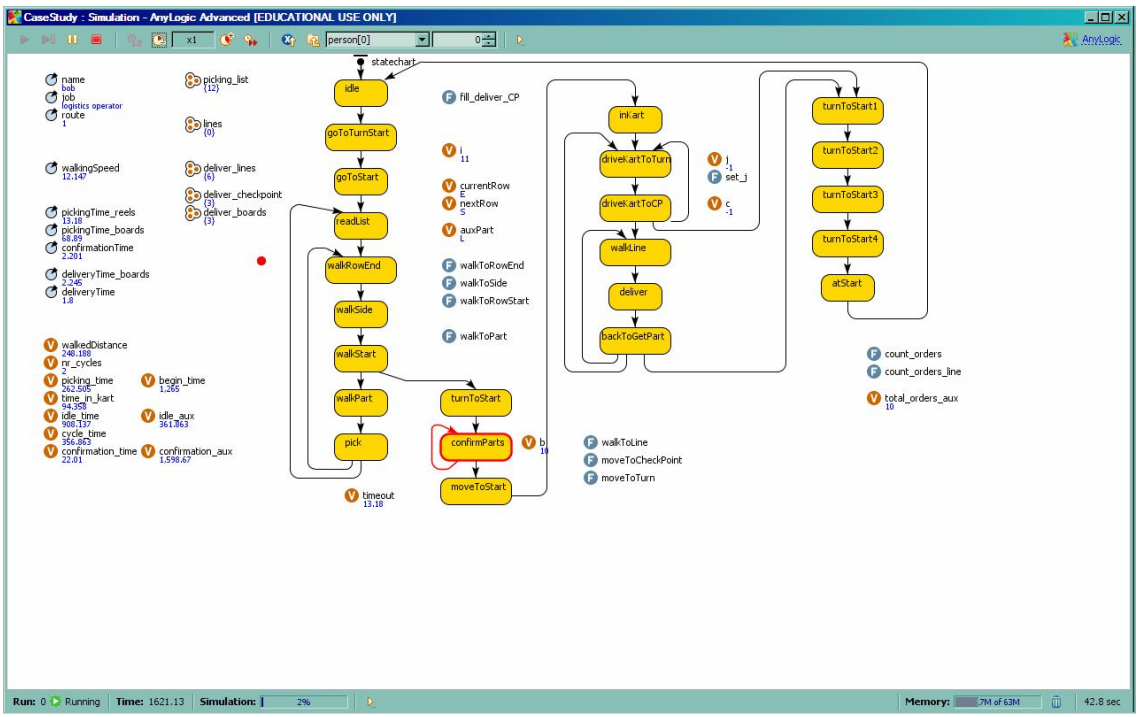
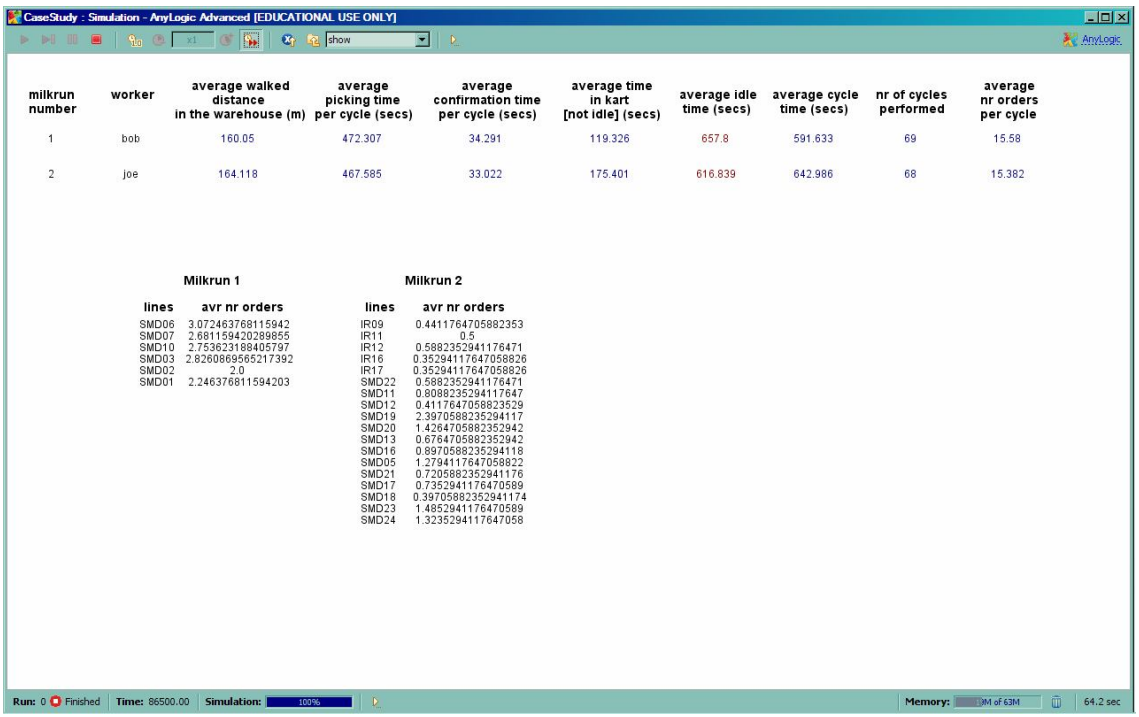


Figure 5.8 – The "to-be" situation: screenshot of one object of the class Person.



milkrun number	worker	average walked distance in the warehouse (m)	average picking time per cycle (secs)	average confirmation time per cycle (secs)	average time in kart [not idle] (secs)	average idle time (secs)	average cycle time (secs)	nr of cycles performed	average nr orders per cycle
1	bob	160.05	472.307	34.291	119.326	657.8	591.633	69	15.58
2	joe	164.118	467.585	33.022	175.401	616.839	642.986	68	15.382

Milkrun 1

lines	avr nr orders
SMD06	3.072463768115942
SMD07	2.681159420289855
SMD10	2.753623188405797
SMD03	2.826086956521739
SMD02	2.0
SMD01	2.246376811594203

Milkrun 2

lines	avr nr orders
IR09	0.4411764705882353
IR11	0.5
IR12	0.5882352941176471
IR16	0.35294117647058826
IR17	0.35294117647058826
SMD22	0.5882352941176471
SMD11	0.8088235294117647
SMD12	0.4117647058823529
SMD19	2.3970588235294117
SMD20	1.4264705882352942
SMD13	0.6764705882352942
SMD16	0.8970588235294118
SMD05	1.2794117647058822
SMD21	0.7205882352941176
SMD17	0.7352941176470589
SMD18	0.3970588235294174
SMD23	1.4852941176470589
SMD24	1.3235294117647058

Figure 5.9 – The "to-be" situation: screenshot of the Show section.

The simulation runs of this situation were made with the purpose of supporting the decision of changing the operating mode of this system, as it was already mentioned. The results of 10 random simulation runs that support this finding are presented in the following table and figure. The stop time of each simulation run was again defined to coincide with the duration of a day, which translates to 68 or 69 cycles of 20 minutes.

Table 5.3 – The "to-be" situation: results of 10 simulation runs.

Milk-run	simulation run	avr walked distance	avr picking time	avr confirmation time	avr time in cart	avr idle time	avr cycle time	avr nr orders
1	1	157,089	419,649	32,224	114,183	716,696	533,832	15,203
	2	162,456	362,559	29,315	121,358	766,228	483,917	14,768
	3	164,887	458,584	32,421	120,212	670,939	578,796	14,638
	4	168,974	435,878	27,406	114,099	701,711	549,978	14,971
	5	167,595	318,557	34,319	115,782	813,335	434,339	15,13
	6	155,901	501,166	40,639	103,278	648,926	604,444	15
	7	163,883	447,695	26,233	120,357	681,132	568,052	15,261
	8	166,496	528,6	30,337	120,93	603,042	649,53	15,145
	9	166,011	455,519	36,755	116,066	676,579	571,585	15,42
	10	157,079	489,664	26,475	112,102	646,908	601,766	14,638
	avr	163,0371	441,7871	31,6124	115,8367	692,5496	557,6239	15,0174
	%	-	34,47%	2,47%	9,04%	54,03%	-	-
2	1	160,506	473,823	34,352	157,126	628,703	630,949	14,426
	2	163,786	429,943	34,411	166,948	662,267	596,891	14,912
	3	163,576	557,274	34,524	159,969	543,659	717,423	15,324
	4	156,693	392,885	32,659	175,57	690,291	568,455	14,882
	5	159,389	371,997	32,994	167,923	718,412	539,921	14,985
	6	158,303	328,504	32,237	167,297	761,892	495,801	14,868
	7	159,689	604,371	33,621	179,839	477,663	784,21	15,029
	8	158,963	475,087	37,407	175,356	609,492	650,442	14,603
	9	155,115	460,098	39,816	176,882	622,759	636,98	14,294
	10	160,294	404,75	25,381	179,171	675,048	583,921	14,897
	avr	159,6314	449,8732	33,7402	170,6081	639,0186	620,4993	14,822
	%	-	34,79%	2,61%	13,19%	49,41%	-	-

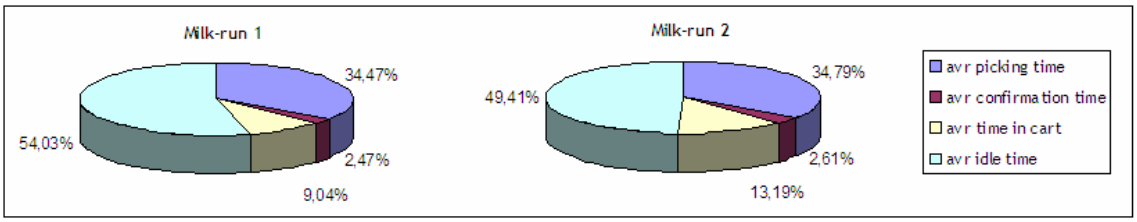


Figure 5.10 – The "to-be" situation: pie-charts of the results of 10 simulation runs.

Analysing the chosen performance indicators, it was possible to conclude that this scenario does improve the system's performance. The waste in the system is reduced, i.e. the idle time in the second milk-run is a lot smaller than it was before in milk-runs 2 and 3 together, which means that the system's resources are being better used.

Going more in depth, the results for milk-run 1 are almost the same as in the "as-is" situation, as the only difference was that another production line located at the last checkpoint in this route was added. In milk-runs 2 and 3 together in the "as-is" situation, there was an average of about 65% of idle time and only about 10% of useful time in cart, delivering. The picking time accounted for about 25% of the total time. In the "to-be" situation, there was a decrease of about 15% in the idle time, with milk-runs 2 and 3 now combined in only one route. The picking time increased to about 35% and the delivery time to 13%. In the real system, this means that with these two routes combined together, we now have one less worker who also has less idle time during the cycle. We would reduce not only

the waste correspondent to having resources which are not being fully used but also the waste of having one extra worker in an operation where he/she is not needed.

Referring to the modelling and simulation itself, there is not much to add to what was said regarding the "as-is" situation. The input data here is basically the same, so the results obtained are similar to the previous version of the application. In summary, the model of the system behaves as it was expected, according to the input data but it does not fully represent the real "to-be" situation as some facts such as malfunctions in the system are not taken into account due to lack of information on the system. For example, the new milk-run is always on time to start a new cycle, which probably would not happen in a real system as this milk-run now has to satisfy the orders of a larger number of lines, which means that the time that it takes to deliver the parts now is a lot bigger and therefore is prone to cause some delays.

5.4.3 – The "what-if" situation

This situation represents one of the proposed solutions for improving the system's performance and reducing waste which was not chosen. The idea here is to have one worker dedicated to the picking process and another one to the delivery. Every twenty minutes, a new order list comes in with orders from all the production lines. The picking worker does the picking of all the parts in the warehouse and leaves a full cart at the warehouse's entrance. The delivery worker then takes this cart and delivers the parts around all the production lines using the logistics train.

Again, the first version of the application was changed in order to obey to new functioning rules, although now the changes were a lot more substantial. The Person's statechart was changed. According to the type of worker we now have (picking or delivery) the object goes through the states on the corresponding branch of the Person's statechart. The function that controls the auxiliary variable of the delivery process was changed and made similar to the one in the first earlier version of the application, mentioned at the introduction of this section, 5.4. A new indicator was introduced at the Show class:

- Picking idle time.

The following figures will again show the screenshots for the Main section, one of the Person's objects and the Show class. Figure 5.11 is the Main section. On the top left corner, the picking worker is waiting at the picking start point to start a new cycle, while the delivery worker is doing the route around the production lines. Figure 5.12 shows the picking worker's state diagram. The 'pick' state is active. Figure 5.13 represents the Show class.

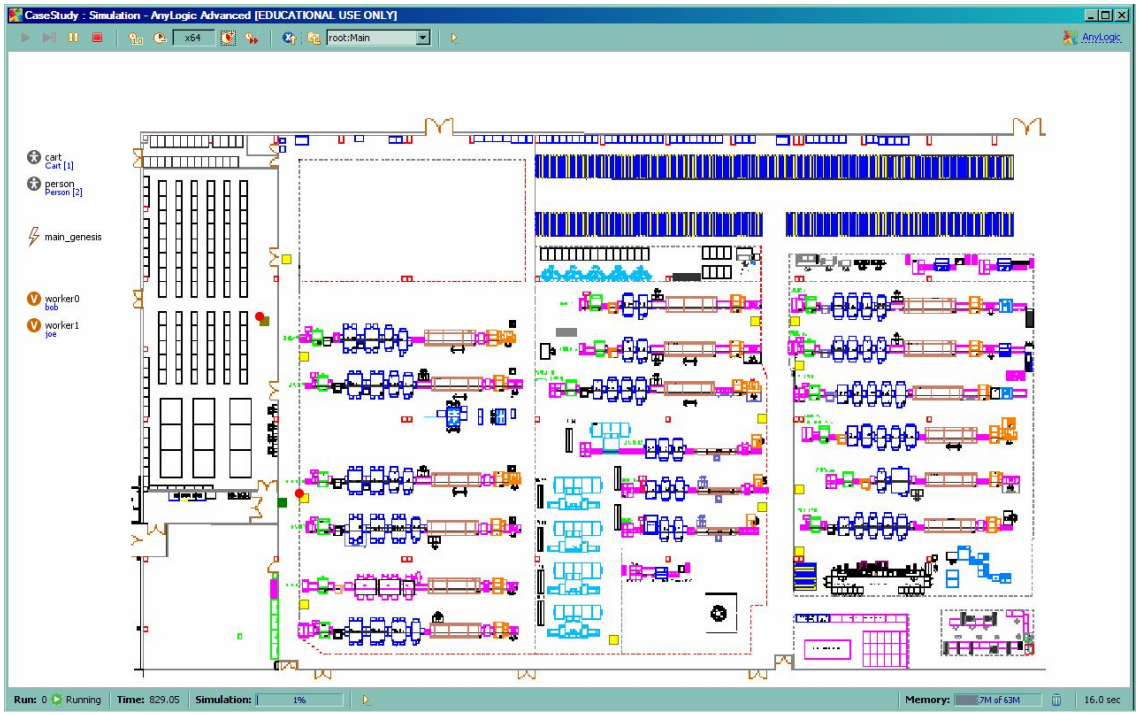


Figure 5.11 – The "what-if" situation: screenshot of the Main section.

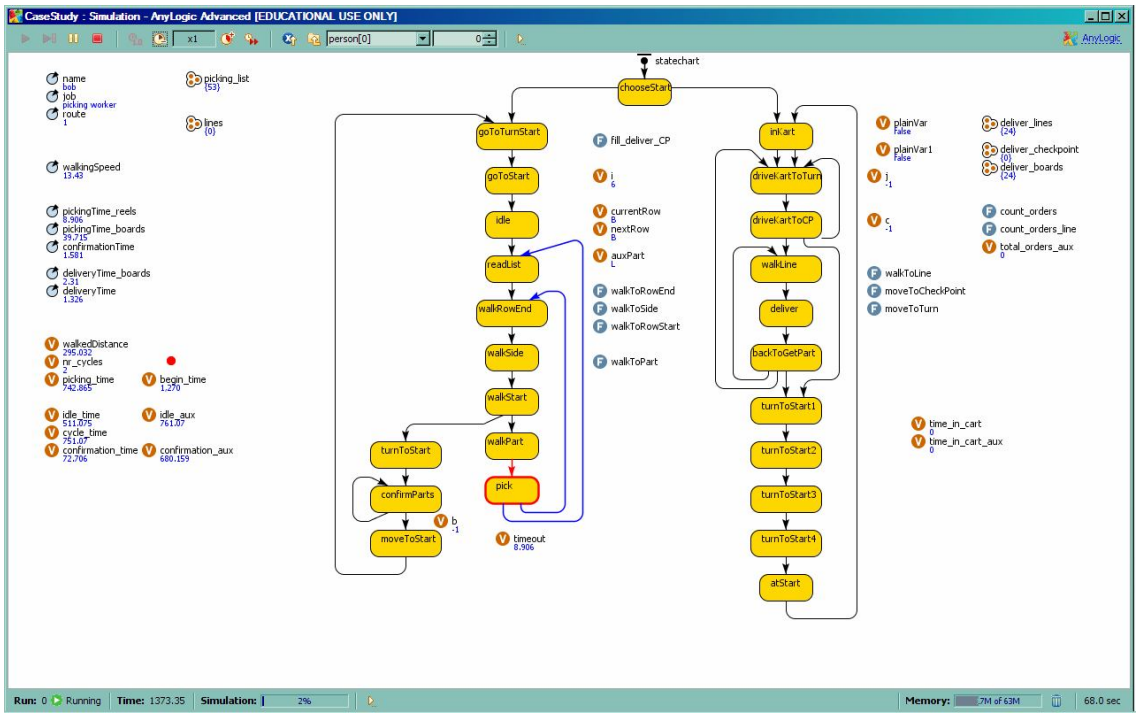


Figure 5.12 – The "what-if" situation: screenshot of one object of the class Person.

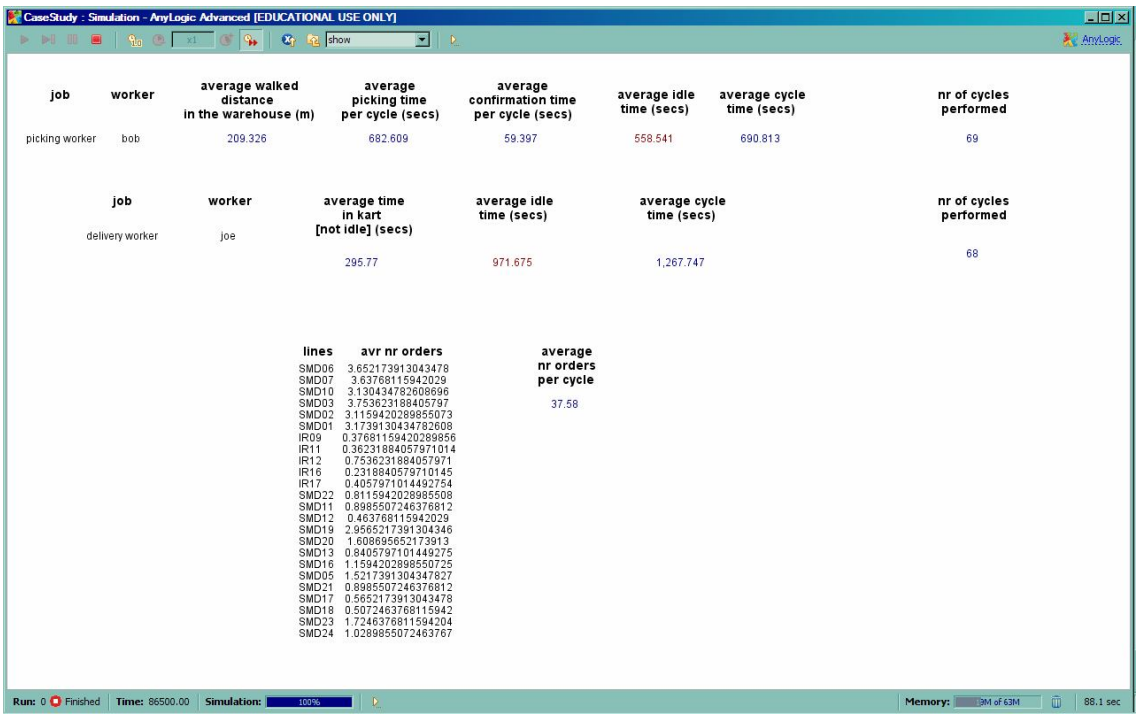


Figure 5.13 – The "what-if" situation: screenshot of the Show section.

Again an analysis of 10 simulation runs was made for this situation, to complete the examination and comparison of all three situations addressed. The stop time was once more correspondent to a day. The results can be found in the following table.

Table 5.4 – The "what-if" situation: results of 10 simulation runs.

worker	run	avr walked distance	avr picking time	avr confirm. time	picking + confirm. time	avr idle time	avr cycle time	nr cycles	avr nr orders
picking	1	212,81	973,216	133,97	1107,186	433,081	981,616	61	38,082
	2	253,405	1208,294	107,704	1315,998	422,609	1216,99	52	34,962
	3	343,374	2057,13	172,923	2230,053	514,311	2066,503	33	40,97
	4	214,715	996,575	90,264	1086,839	489,434	1005,17	57	37,649
	5	280,898	1535,702	177,134	1712,836	502,331	1544,482	41	40,463
	6	215,335	830,919	70,422	901,341	412,541	839,82	69	38
	7	203,504	808,408	87,695	896,103	472,684	818,346	66	36,318
	8	247,512	1193,125	103,985	1297,11	396,605	1202,126	54	39,352
	9	228,083	1006,084	101,741	1107,825	377,369	1014,491	62	39,919
	10	216,051	651,693	63,585	715,278	588,36	661,379	69	38,928
	avr	241,5687	1126,1146	110,9423		460,9325	1135,092	56,4	38,464
	%	-	39,75%	3,92%		16,27%		-	-

worker	run	avr time in cart	avr idle time	avr cycle time	nr cycles	avr nr orders	nr times late
delivery	1	279,48	1157,454	1438,297	60	38,082	8
	2	270,342	1411,681	1680,028	51	34,962	17
	3	281,765	2372,918	2652,504	32	40,97	36
	4	282,863	1228,892	1510,353	56	37,649	12
	5	265,051	1843,82	2108,454	40	40,463	28
	6	268,275	1002,523	1270,798	68	38	0
	7	274,161	1032,586	1307,715	65	36,318	3
	8	278,29	1350,631	1628,921	53	39,352	15
	9	259,355	1154,223	1414,699	61	39,919	7
	10	266,644	1001,29	1268,138	68	38,928	0
	avr	272,6226	1355,6018	1627,990	55,4	38,4643	
	%	16,74%	83,26%	-	-	-	

It is now very easy to observe that there is no longer a constant number of cycles; it actually varies a great deal. Taking the picking process as example, there is now an average of only 56.4 cycles per day with a standard deviation of 11.9 cycles, which is 21% of the total. In the way this model was built, if the Person object is not ready to start the cycle on time, it is simply not performed. In the first two situations, the Person would never be late (again due to the way the model was built; delays are still possible in these situations in the real system, as it was mentioned before and will be addressed again in the next section). This means that in one day's work with 20 minute cycles, there would be an average of 68 or 69 cycles per day, as it was seen before. In these 10 simulation runs, there is an average of 12.9 delays per day. There are some runs where there were no delays, such as in runs 6 and 10 but there are also runs like number 3 were there were 36 delays, which means that the delivery was not performed more than half of times it was supposed to! The runs where a bigger number of delays occurred are generally the ones with longer picking cycles, as it was expected.

Although this is not at all a thorough analysis of this "what-if" situation, it is clear to see that it would not be efficient if it was implemented just as it was described here. This, however, does not mean that this hypothesis should be discarded right away based solely on these findings; a more accurate analysis with more samples and maybe a different strategy

towards the implementation of this situation should be tested in order to better support such decision.

5.5 – General conclusions

Analysing a real case study in an analysis and evaluation such as this one, instead of analysing hypothetical situations as the ones presented in chapter 4, is extremely useful to put things into perspective. When doing the analysis of a real system, a great deal of variables come under study and the process of choosing what to consider and what to discard is what determines if the model being built is actually useful or not.

Building a model of a real Production system is not an easy task and it is time-consuming. The first step is to get to know the system and what the purposes of the simulation are. Then, it is time to identify the variables that are of interest and discard whatever else which does not come into scope. All the assumptions made must be well though through and weighted, in order not to incur in inadmissible errors.

In this particular situation, a great deal of assumptions were made, which translated in a model that, although it was useful for this particular purpose, it was not, let us say, as perfect as it could be. To support this rather controversial affirmation, as it questions the quality of this particular part of this work, there is a very illustrative example. During a day's production in this section, there are not a constant number of orders per cycle. There are peaks at certain times of the day, followed by periods of very low part consumption. In the real system, this translates into longer picking cycles, meaning that the delivery process starts later and also takes longer as there are more parts, which can mean that the cycle will be overall longer than the pre-established 20 minutes. These more demanding cycles can be followed by others that are the exact opposite; most of the cycle time is spent in an inactive state. The assumption made in this model regarding this was that the number of orders in each cycle was fairly constant. This is actually a very common approximation in such situations, as it is a way of aggregating the data and adapting it to more common probability distributions which are easier to obtain and simulate as well. However, one must not forget that these cases actually occur in the real system and must be taken into account in an analysis that has accuracy as a goal.

So, the case here was to study a real production system in order to build a model that would represent it rather fairly and then simulate it, to analyse such a system through the eyes of simulation, to understand how it is done, what the demands and challenges are when simulating such systems. To do so, the accuracy of the model did not have to be put at very high standards as this would translate into a more time-consuming process and would take the focus out of the analysis of the system itself and turn it towards the concepts behind model building.

From the decision support tool point of view, the simulation runs performed with the models built for all three situations acted as a plus to other studies done by the company, supporting the decision that had already been taken and probably acting as enablers for further simulations of different systems there.

Chapter 6

Conclusions

This chapter will present an overview on what was accomplished in the context of this dissertation, as well as some recommendations on possible future developments.

Modelling and Simulation of Production systems has been growing almost since the birth of computers. Production systems have always been one of the pillars of the world's economy and are constantly adapting and evolving alongside the society and the market. The simulation of these systems has proven to be an important step in their development and optimisation. Building a simulation model and testing it provides the people involved in the process with a system-wide view on how it works. One can not only examine each part that comprises the production system under study but also the greater view on what effects that specific part has on the whole system. Simulation has also proven to be a widely accepted decision support tool. It has been used to test different possible scenarios, building an "as-is" model and then different "what-if" cases and comparing results. This method can be used to evaluate several types of situations, such as reconfiguration of layouts, introduction of new products or machinery or adoption of different production policies.

In spite of all the advantages of using simulation, there are still inhibitors and disadvantages that do not enable it to be as widely used as one might think. The first factor against simulation that comes to mind is the fact that building a model is building an approximation of reality, it is never the same as reality itself. This means that one must be very careful when choosing which assumptions and approximations to make when modelling, in order not to incur in inadmissible errors that may compromise the results. Then, even if the model is as close to reality as possible, the input data must also be accurate and adequate to the situation under study. Finally, time and costs of using simulation are also critical. Simulating a system can be a time-consuming task that can turn out to be too lasting and therefore not responsive enough to the market's constant changes. Also such a task of building a model and simulating a system with as many variables as a production system can become very expensive. Besides being a task which takes some amount of time, it also may

involve a team of people with some degree of expertise in the subject, which means higher expenses. Apart from the team's salaries, there might also be other costs such as the purchase of specific hardware and software, training sessions for the company's staff, maintenance and updating of the simulator, just to mention a few. However, these disadvantages must always be weighed against the costs and risks of not using simulation in order to make a supported decision, which will have the desired short and also long-term effect.

In order to make simulation more appealing and more easily available in the context of production systems and other systems as well, some thoughts arise. Definitely the first step to take would be to educate people on the benefits of using simulation but also presenting the disadvantages clearly. In order to reduce costs of simulating technologies, a joint effort between the production companies interested in using them and the companies that develop them is needed. This situation can be seen as a simple supply-and-demand law kind of scenario. If there is a growing number of manufacturing companies interested in using these technologies, further development will be done in this area and the prices will become consequently lower. In order to magnify the interest in these kind of technologies, they must be made more appealing, which basically means that they must be as simple to use as possible but still producing good results in somewhat complex situations.

Moving on to another objective of this dissertation, let us analyse the conclusions reached regarding the chosen simulation technology. The AnyLogic simulation platform has proven to be a tool that has tremendous potential in the area of simulating different kinds of systems and, more specifically, production systems, the ones being addressed in the context of this dissertation. It is based in an object oriented programming language, Java, which translates in a logical and therefore easier model conception and building. It is appealing not only to experienced users but also beginners, because it is a very visual tool and uses a drag-and-drop method to aid the model building. It comprises different modelling paradigms which can be used on their own or combined in hybrid models. The technology has some limitations, however. The documentation, although easy to obtain and not scarce, it sometimes is not very clear and extensive. It is also an expensive technology and some of the licenses available, such as the educational one that was used, lack some features which are very important in model building and might prove to be necessary, like step-by-step debugging or reading and writing of external files.

Even so, this tool has proven to be very fit for the purpose of simulating the production systems analysed and was very useful in answering another objective of this dissertation, which was the actual building and simulation of these systems. Production systems are a good example of systems that benefit a great deal with simulation and the cases that were analysed during this work were no exception. Modelling and simulation of these systems provided a system wide view on how they operate and/or served as an important decision support tool.

Concerning the case study, it was possible to see that the models built using the chosen simulation platform behaved in a similar way than the actual system, according to the input data used in the model. The assumptions made did not affect the behaviour of the system greatly and the results obtained supported the studies made so far on the real system. The

analysis of a case study enabled a more realistic view on how a model is built and how a simulation should be performed in order to achieve the desired results.

6.1 – Recommendations for further research

This piece of work is only the "tip of the iceberg" in a very wide, complex and up-to-date subject, which is Simulation as a whole and Simulation of Production Systems in particular. Analysing what was done along the months that comprised this work, many possibilities are left for further work. Some of them will be presented in the following paragraphs.

It was said that the development of simulation technologies could benefit from a presentation on the pros and cons of using Simulation. So, a possible way to go from here would be to make a more thorough analysis of the benefits of simulation and its limitations and organise them in a complete and extensive manner but keeping it simple enough in order to make it accessible to anyone interested in the subject, even if they are not specialists.

Another interesting research would be to take as a starting point the other steps that could be taken to make simulation available to a wider group of people, such as reducing costs and making simulation software more appealing, and develop them. A study on the more widely used simulation technologies could be done in order to list each technology's stronger and weaker points, to create a basis for improvement in this specific area.

Regarding an analysis of more recent lines of study in this area mentioned in this piece of work, there are three paths that could benefit from further research: the use of distributed simulation in Supply Chains and what strategies are being used in this field; the information exchange strategies and techniques between simulation programs and other information technologies in the system; and how the human factor is being taken into account in simulation of Production systems, regarding the use of paradigms such as Agent-based modelling for example.

Finally, concerning the adopted simulation platform AnyLogic, more work is still left to be done. As it was mentioned throughout this work, this tool has a great potential for simulating different kinds of systems, including the ones that were of our interest, production systems. Although the Enterprise Library that is included in the package is quite complete, there are other components that could be created. It would be interesting to implement building blocks that could be used to model, for example, more complex conveyors, a basic warehouse, a standard milk-run cart, a simple configurable machine, etc. Also other features in the software could be further explored, like the use of resource networks or the several different presentation options for the output data, amongst many others.

Appendixes

A.1 – Simulating Production lines

Here are the functions that were developed in Java for the applications that simulated simple production lines in AnyLogic, described in section 4.2.1. The first function `whichProcessing()` was used in all the situations, the second one `whichSetup()` was used in all except the first situation and the last two `decrease()` and `nextProduct()` were used only in situation 5.

```
/***whichProcessing()***/
if (currentProduct=="A")
	processingCurrent=processingA;
if (currentProduct=="B")
	processingCurrent=processingB;


/***whichSetup()***/
if (currentProduct==previousProduct)
	setupCurrent=0;
else
	{
	if (currentProduct=="A")
		setupCurrent=setupA;
	if (currentProduct=="B")
		setupCurrent=setupB;

	}


/***decrease()***/
for (int i=0; i<productsTime.size(); i++)
	{
	productsTime.add(i,productsTime.remove(i)-0.1);

	}


/***nextProduct()***/
double timeout=30; /* the auxiliar variable timeout equals the maximum expiration time
possible*/
int index=0;

for (int i=0; i<productsTime.size(); i++)
	{
	if (productsTime.get(i)<timeout) /*if this part's expiration time is smaller than
the smallest one found until now*/
		{
```

```
                if (productsTime.get(i)<0)
                        {
                        enter_rejected.take(queue.remove(queue.get(i))); /*reject the part
if it has already expired*/
                        productsTime.remove(i); productsType.remove(i);
                        }
                else
                        {
                        timeout=productsTime.get(i); timeout_visual=timeout; /*if it's not
expired, make it the next part to be processed*/
                        index=i; index_visual=index;
                        }
                }
        }

productsTime.remove(index); currentProduct=productsType.remove(index);

    enter.take(queue.remove(queue.get(index))); /*take the next part to be processed at
the machine*/
```

A.2 – Case Study

This section includes the functions written in Java in the AnyLogic models for all three situations considered in the case study, the “as-is”, “to-be” and “what-if” situations. The code is preceded by a small explanation about what the function does and it is commented when appropriate.

The following function is named `begin()` and it was called by the Main object at the very beginning of the simulation of the “as-is” situation. It creates the workers and carts. In the “to-be” and “what-if” scenarios, this function was similar, the only difference being the number of objects from each type that were being created.

```
/***begin()***/
//1st worker
add_person();
        person.get(0).name="bob";
        person.get(0).route=1;
        if (person.get(0).walkingSpeed<0) person.get(0).walkingSpeed=0.07*14;
        if (person.get(0).pickingTime_reels<0) person.get(0).pickingTime_reels=5.58;
        if (person.get(0).pickingTime_boards<0) person.get(0).pickingTime_boards=19.35;
        if (person.get(0).deliveryTime<0) person.get(0).deliveryTime=0.46;
        if (person.get(0).deliveryTime_boards<0) person.get(0).deliveryTime_boards=0.88;
        worker0=person.get(0).name;
        person.get(0).jumpTo(start.getX()+5,start.getY()+5);
//2nd worker
add_person();
        person.get(1).name="joe";
        person.get(1).route=2;
        if (person.get(1).walkingSpeed<0) person.get(1).walkingSpeed=0.07*14;
        if (person.get(1).pickingTime_reels<0) person.get(1).pickingTime_reels=5.58;
        if (person.get(1).pickingTime_boards<0) person.get(1).pickingTime_boards=19.35;
        if (person.get(1).deliveryTime<0) person.get(1).deliveryTime=0.46;
        if (person.get(1).deliveryTime_boards<0) person.get(1).deliveryTime_boards=0.88;
        worker1=person.get(1).name;
        person.get(1).jumpTo(start.getX()+5,start.getY()+5);
//3rd worker
add_person();
        person.get(2).name="bill";
        person.get(2).route=3;
        if (person.get(2).walkingSpeed<0) person.get(2).walkingSpeed=0.07*14;
        if (person.get(2).pickingTime_reels<0) person.get(2).pickingTime_reels=5.58;
        if (person.get(2).pickingTime_boards<0) person.get(2).pickingTime_boards=19.35;
        if (person.get(2).deliveryTime<0) person.get(2).deliveryTime=0.46;
```

```
        if (person.get(2).deliveryTime_boards<0) person.get(2).deliveryTime_boards=0.88;
        worker2=person.get(2).name;
        person.get(2).jumpTo(start.getX()+5,start.getY()+5);

//1st kart
add_cart();
        cart.get(0).milkrunNr=1;
        cart.get(0).setVelocity(cart.get(0).speed12);
        cart.get(0).sizeSmall=true;
        cart.get(0).jumpTo(start.getX(),start.getY());
//2nd kart
add_cart();
        cart.get(1).milkrunNr=2;
        cart.get(1).setVelocity(cart.get(1).speed12);
        cart.get(1).sizeSmall=true;
        cart.get(1).jumpTo(start.getX(),start.getY());
//3rd kart
add_cart();
        cart.get(2).milkrunNr=3;
        cart.get(2).setVelocity(cart.get(2).speed3);
        cart.get(2).sizeSmall=false;

        cart.get(2).jumpTo(start.getX(),start.getY());
```

The following sequence of functions was called by the Controller object also at the beginning of the simulation of the "as-is" situation: `fill_module(); fill_probability(); fill_prob1(); fill_nr_orders1(); fill_prob2(); fill_nr_orders2(); fill_prob3(); fill_nr_orders3();`.

The `fill_module()` function fills an ArrayList with the references for the parts that are going to be ordered and which indicate the row and rack they are placed in the warehouse. Here is an extract from that function.

```
/***fill_module()***/
module.add(0,"AA");
module.add(1,"AB");
module.add(2,"AC");
module.add(3,"AD");
module.add(4,"AE");
module.add(5,"AF");
module.add(6,"AG");
module.add(7,"AH");
module.add(8,"AI");
/***etc***/
module.add(42,"DD");
module.add(43,"DE");
module.add(44,"DF");
/***etc***/
module.add(111,"LD");
module.add(112,"LE");
module.add(113,"LF");
module.add(114,"LH");
```

`fill_probability()` filled the ArrayList with the accumulated probability of obtaining an order from each one of the modules in the warehouse. There is a direct correspondence between this function and the previous one `fill_module()`, i.e. the module reference at the first position of the ArrayList 'module' corresponds to the probability at the first position of the ArrayList 'probability'. Here is an extract from the `fill_probability()`function.

```
/***fill_probability()***/
probability.add(      0      ,      1923    )      ;
probability.add(      1      ,      3632    )      ;
probability.add(      2      ,      5341    )      ;
probability.add(      3      ,      7477    )      ;
probability.add(      4      ,      16129   )      ;
probability.add(      5      ,      51485   )      ;
probability.add(      6      ,      56719   )      ;
probability.add(      7      ,      59923   )      ;
probability.add(      8      ,      62487   )      ;
/***etc***/
probability.add(      42     ,      403015  )      ;
probability.add(      43     ,      427796  )      ;
```

```
probability.add(      44     ,      470201  )      ;
/***etc***/
probability.add(      111    ,      999150  )      ;
probability.add(      112    ,      999364  )      ;
probability.add(      113    ,      999364  )      ;
probability.add(      114    ,      1000005 )      ;
```

As the Educational license of AnyLogic used for this study does not allow reading of external files, a rather creative approach was used to build the necessary lists shown above. Using Microsoft Office Excel, a table was built with the all the data to be inserted in the function that was being implemented. Then the contents of this table were copied to a text file and then from this file to the AnyLogic function writing window, to avoid that all the 115 lines of code exemplified above (in each function) were written one by one.

The functions `fill_prob1()`, `fill_prob2()` and `fill_prob3()` filled one ArrayList with the probability of obtaining an order from each line in each one of the routes in the "as-is" situation, routes 1, 2 and 3 and another list with the name of the correspondent lines. In the other two situations, these functions were modified in order to fill the lists with the new information. The `fill_prob1()` function is presented next, as an example. The other two functions are quite similar to this one.

```
/***fill_prob1()***/
prob_lines1.add(0, 0.225955204); //SMD6
prob_lines1.add(1, 0.438462033); //SMD7
prob_lines1.add(2, 0.628731643); //SMD10
prob_lines1.add(3, 0.838523089); //SMD3
prob_lines1.add(4, 1.0); //SMD2

lines1.add(0, "SMD06");
lines1.add(1, "SMD07");
lines1.add(2, "SMD10");
lines1.add(3, "SMD03");
lines1.add(4, "SMD02");
```

At the beginning of each milk-run cycle, there was an event that called a set of functions. As an example, this is the sequence of functions called for milk-run 1: `fill_current_nr_orders1(); add_order1(); sort_list1(); copy_list1();`. The first function initialises the list that saves the current number of orders from each line at this milk-run cycle. The second one adds the orders from this cycle to the correspondent list. The third function sorts the list according to the order that the parts are being picked from the warehouse. The last function copies this list from the controller to the correspondent worker, an object from the class Person. The sequence of the last three functions generates the random order lists in each cycle that was described in section 5.4. This is the function `add_order1()`:

```
/***add_order1()***/
rand_order1=uniform_discr(10,20); /*random nr of orders generated, between 10 and 20*/

for (int j=0; j<rand_order1; j++)
{
	/*to look for the line that issues the order*/
	search_line1();
	search_arrays1();

	/*adds this order to the list*/
	list1_unsorted.add(module_aux1 + " " + line_aux1);

	//updates the count of total and this run's nr of parts
	for(int k=0; k<lines1.size(); k++) //looks for the right line
	{
		if (line_aux1==lines1.get(k))
		{
			nr_orders1.add(k,nr_orders1.remove(k)+1);
			current_nr_orders1.add(k,current_nr_orders1.remove(k)+1);
					//updates the nr of boards ordered per line
```

```
                        if (module_aux1.charAt(0)=='H'|| module_aux1.charAt(0)=='I'||
module_aux1.charAt(0)=='J')
                                current_nr_boards1.add(k,current_nr_boards1.remove(k)+1);
                }
        };
    }

/***search_line1()***/
double rand_line=uniform(); /*random nr generated*/

for (int i=0; i<lines1.size();i++)
{
        if (rand_line<=prob_lines1.get(i)) /*looks for the interval where that nr is in
the list of probabilities*/
        {
                index_aux1=i;
                break;
        }
}
line_aux1=lines1.get(index_aux1);


/***search_arrays1()***/
int rand_aux1=uniform_discr(1000005); /*random nr generated*/

for (int i=0; i<115;i++)
{
        if (rand_aux1<=probability.get(i)) /*looks for the place where that nr is*/
        {
                index_aux1=i;
                break;
        }
}
module_aux1=module.get(index_aux1); /*add the correspondent module to the list*/
```

This is the function sort_list1():

```
/***sort_list1()***/
int size=list1_unsorted.size();

for (int i=0; i<size; i++)
{
        String aux=list1_unsorted.get(i);
        if (!list1_sorted.isEmpty())
        {
                if (aux.compareTo(list1_sorted.get(list1_sorted.size()-1)) < 0) /*if the
current part is alphabetically before the one already in the sorted list*/
                {
                        for (int j=0; j<list1_sorted.size(); j++) /*looks for the correct
place to insert it in the sorted list*/
                        {
                                if(aux.compareTo(list1_sorted.get(j))<0)
                                {
                                        list1_sorted.add(j,aux);
                                        break;
                                }
                        }
                }
                else list1_sorted.add(aux);
        }
        else list1_sorted.add(aux);
}
```

The more important functions in the Person class are the ones that control the next place the object should move to. During the picking process, there are three main functions: `walkToRowEnd()`, `walkToside()` and `walkToPart()`. The first one concerns the movement of the worker towards the end of a certain row when the picking there has ended. The second one concerns the movement of the worker in the warehouse's corridors which are not rows that contain the parts' racks. The last function concerns the movements within each row. They all use at least one of two auxiliary variables, `currentRow` and `nextRow`, which

contain the row the worker is in currently and the next part's correspondent row, respectively. Here are the three functions mentioned:

```
/***walkToRowEnd()***/
switch (currentRow)
{
case 'A':
case 'B': get_Main().person.get(route-
1).moveTo(get_Main().turnABend.getX(),get_Main().turnABend.getY()); break;
case 'C':
case 'D': get_Main().person.get(route-
1).moveTo(get_Main().turnCDend.getX(),get_Main().turnCDend.getY()); break;
case 'E': get_Main().person.get(route-
1).moveTo(get_Main().turnEend.getX(),get_Main().turnEend.getY()); break;
case 'F': get_Main().person.get(route-
1).moveTo(get_Main().turnFend.getX(),get_Main().turnFend.getY()); break;
/*case 'G':*/
case 'H':
case 'I': get_Main().person.get(route-
1).moveTo(get_Main().turnHIend.getX(),get_Main().turnHIend.getY()); break;
case 'J': get_Main().person.get(route-
1).moveTo(get_Main().turnJend.getX(),get_Main().turnJend.getY()); break;
case 'K': get_Main().person.get(route-
1).moveTo(get_Main().turnKend.getX(),get_Main().turnKend.getY()); break;
case 'L': get_Main().person.get(route-
1).moveTo(get_Main().turnLend.getX(),get_Main().turnLend.getY()); break;
}

/***walkToSide()***/
switch (nextRow)
{
case 'A':
case 'B': break;
case 'C':
case 'D': get_Main().person.get(route-
1).moveTo(get_Main().turnCDend.getX(),get_Main().turnCDend.getY()); break;
case 'E': get_Main().person.get(route-
1).moveTo(get_Main().turnEend.getX(),get_Main().turnEend.getY()); break;
case 'F': get_Main().person.get(route-
1).moveTo(get_Main().turnFend.getX(),get_Main().turnFend.getY()); break;
/*case 'G':*/
case 'H':
case 'I': get_Main().person.get(route-
1).moveTo(get_Main().turnHI.getX(),get_Main().turnHI.getY()); break;
case 'J': if (currentRow=='H'||currentRow=='I') {get_Main().person.get(route-
1).moveTo(get_Main().turnJend.getX(),get_Main().turnJend.getY()); break;}
                    else get_Main().person.get(route-
1).moveTo(get_Main().turnJ.getX(),get_Main().turnJ.getY()); break;
case 'K': if (currentRow=='H' || currentRow=='I' ||currentRow=='J')
{get_Main().person.get(route-
1).moveTo(get_Main().turnKend.getX(),get_Main().turnKend.getY()); break;}
                    else get_Main().person.get(route-
1).moveTo(get_Main().turnKmid.getX(),get_Main().turnKmid.getY()); break;
case 'L': break;
case 'S': if (currentRow=='H' || currentRow=='I' ||currentRow=='K')
                    {get_Main().person.get(route-
1).moveTo(get_Main().turnLend.getX(),get_Main().turnLend.getY()); break;}
                    else get_Main().person.get(route-
1).moveTo(get_Main().turnJ.getX(),get_Main().turnJ.getY()); break;
}

/***walkToPart()***/
if (currentRow=='K')
       switch (auxPart)
       {
       case 'A': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().A3.getY());break;
       case 'B': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().B3.getY());break;
       case 'C': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().C3.getY());break;
       case 'D': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().D3.getY());break;
       case 'E': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().E3.getY());break;
```

```
        case 'F': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().F3.getY());break;
        case 'G': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().G3.getY());break;
        case 'H': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().H3.getY());break;
        case 'I': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().I3.getY());break;
        case 'J': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().J3.getY());break;
        case 'K': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().K3.getY());break;
        case 'L': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().L3.getY());break;
        case 'M': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().M3.getY());break;
        case 'N': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().N3.getY());break;
        case 'O': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().O3.getY());break;
        case 'P': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().P3.getY());break;
        case 'Q': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().Q3.getY());break;
        case 'R': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().R3.getY());break;
        case 'S': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().S3.getY());break;
        case 'T': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().T3.getY());break;
        case 'U': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().U3.getY());break;
        };

if (currentRow=='L')
        switch (auxPart)
        {
        case 'A': get_Main().person.get(route-
1).moveTo(get_Main().A4.getX(),get_Main().person.get(route-1).getY());break;
        case 'B': get_Main().person.get(route-
1).moveTo(get_Main().B4.getX(),get_Main().person.get(route-1).getY());break;
        case 'C': get_Main().person.get(route-
1).moveTo(get_Main().C4.getX(),get_Main().person.get(route-1).getY());break;
        case 'D': get_Main().person.get(route-
1).moveTo(get_Main().D4.getX(),get_Main().person.get(route-1).getY());break;
        case 'E': get_Main().person.get(route-
1).moveTo(get_Main().E4.getX(),get_Main().person.get(route-1).getY());break;
        case 'F': get_Main().person.get(route-
1).moveTo(get_Main().F4.getX(),get_Main().person.get(route-1).getY());break;
        //case 'G': get_Main().person.get(route-
1).moveTo(get_Main().G4.getX(),get_Main().person.get(route-1).getY());break;
        case 'H': get_Main().person.get(route-
1).moveTo(get_Main().H4.getX(),get_Main().person.get(route-1).getY());break;
        };

}

else
        switch (auxPart)
        {
        case 'A': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().A.getY());break;
        case 'B': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().B.getY());break;
        case 'C': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().C.getY());break;
        case 'D': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().D.getY());break;
        case 'E': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().E.getY());break;
        case 'F': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().F.getY());break;
        case 'G': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().G.getY());break;
        case 'H': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().H.getY());break;
```

```
        case 'I': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().I.getY());break;
        case 'J': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().J.getY());break;
        case 'K': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().K.getY());break;
        case 'L': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().L.getY());break;
        case 'M': get_Main().person.get(route-1).moveTo(get_Main().person.get(route-
1).getX(),get_Main().M.getY());break;
        }
```

Still concerning the workers' movements, there are also the functions that control their behaviour and also the correspondent carts during the delivery process. They are `moveToTurn()` and `moveToCheckpoint()`. Here is the Java code that corresponds to them:

```
/***moveToTurn()***/
switch (j+1)
{
case 1:
case 2:
case 3: break;
case 4: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP34.getX(),get_Main().turnCP34.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP34.getX(),get_Main().turnCP34.getY()); break;
case 5: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP45.getX(),get_Main().turnCP45.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP45.getX(),get_Main().turnCP45.getY()); break;
case 6:
case 7:
case 8:
case 9:
case 10:
case 11: break;
}

/***moveToCheckpoint()***/
switch (j+1)
{
case 1: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP1.getX(),get_Main().turnCP1.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP1.getX(),get_Main().turnCP1.getY()); break;
case 2: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP2.getX(),get_Main().turnCP2.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP2.getX(),get_Main().turnCP2.getY()); break;
case 3: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP3.getX(),get_Main().turnCP3.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP3.getX(),get_Main().turnCP3.getY()); break;
case 4: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP4.getX(),get_Main().turnCP4.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP4.getX(),get_Main().turnCP4.getY()); break;
case 5: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP5.getX(),get_Main().turnCP5.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP5.getX(),get_Main().turnCP5.getY()); break;
case 6: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP6.getX(),get_Main().turnCP6.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP6.getX(),get_Main().turnCP6.getY()); break;
case 7: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP7.getX(),get_Main().turnCP7.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP7.getX(),get_Main().turnCP7.getY()); break;
case 8: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP8.getX(),get_Main().turnCP8.getY());
                get_Main().person.get(route-
1).moveTo(get_Main().turnCP8.getX(),get_Main().turnCP8.getY()); break;
```

```
case 9: get_Main().cart.get(route-
1).moveTo(get_Main().turnCP9.getX(),get_Main().turnCP9.getY());
              get_Main().person.get(route-
1).moveTo(get_Main().turnCP9.getX(),get_Main().turnCP9.getY()); break;
case 10:get_Main().cart.get(route-
1).moveTo(get_Main().turnCP10.getX(),get_Main().turnCP10.getY());
              get_Main().person.get(route-
1).moveTo(get_Main().turnCP10.getX(),get_Main().turnCP10.getY()); break;
case 11:get_Main().cart.get(route-
1).moveTo(get_Main().turnCP11.getX(),get_Main().turnCP11.getY());
              get_Main().person.get(route-
1).moveTo(get_Main().turnCP11.getX(),get_Main().turnCP11.getY()); break;
}
```

Both these functions use the variable 'j', whose value is dictated by the function `set_j()`, which chooses this variable's value according to the precedent value and also the route. This function was changed in each one of the modelling situations considered, in order to adapt to the routes that were being taken into account. Here is the `set_j()` function for the "as-is" situation:

```
/***set_j()***/
if (route==1)
{
       switch (j)
       {
       case -1:
       case 0:
       case 1: j++; break;
       }
}
if (route==2)
{
       switch (j)
       {
       case -1: j=3; break; //so that when moveTurn() is called it goes to turnCP34
       case 3: j=4; break;   //moveTurn() will put it at turnCP45
       case 4: j=5; break;
       case 5: j=7; break;
       case 7: j=9; break;
       }
}
if (route==3)
{
       switch (j)
       {
       case -1: j=2; break; //moveTurn() does nothing, moveToCheckPoint puts it at CP3
       case 2: j=3; break; //moveTurn() puts it at turnCP34,moveToCheckPoint puts it at
CP4
       case 3: j=4; break; //moveTurn() puts it at turnCP45, moveToCheckPoint puts it at
CP5
       case 4: j=6; break; //moveToCheckPoint puts it at CP7
       case 6: j=8; break; //moveToCheckPoint puts it at CP9
       case 8: j=10; break; //moveToCheckPoint puts it at CP11
       }
}
```

References

[1] Cambridge Advanced Learner's Dictionary online, http://dictionary.cambridge.org/, last accessed on June 2008

[2] *Compact Oxford English Dictionary online*, http://www.askoxford.com/dictionaries/?view=uk, last accessed on June 2008

[3] Pidd, M. (1998). *Computer simulation in management science.* 4th edition, John Wiley & Sons.

[4] Brito, E.S.C. and Feliz-Teixeira, J.M. (2001). *Simulação por computador : fundamentos e implementação de código em C e C++.* Pulindústria Edições Técnicas.

[5] *Encyclopædia Britannica online "Production System"* article, http://www.britannica.com/eb/article-9106303/production-system, last accessed on June 2008

[6] Supply chain and logistics terms and glossary, http://cscmp.org/Downloads/Public/Resources/glossary03.pdf, last accessed on June 2008

[7] Feliz-Teixeira, J.M. (2006). *Flexible Supply Chain Simulation*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, Portugal.

[8] Terzi, S. and Cavalieri, S. (2003). Simulation in the supply chain context: a survey, Computers in Industry, 53 (2004), pp 3–16.

[9] Semini, M., Fauske, H. and Strandhagen, J.O. (2006). Applications of Discrete-Event Simulation to support Manufacturing Logistics decision-making: a survey, 2006 Winter Simulation Conference, pp 1946-1953.

[10] Swain, J. (2007). Simulation Software Survey, OR/MS Today, available at http://www.lionhrtpub.com/orms/surveys/Simulation/Simulation.html, last accessed on June 2008

[11] Law, A.M. and Kelton, W.D. (1991). *Simulation modeling and analysis*. 2nd edition, McGraw-Hill.

[12] Java programming language homepage, http://java.sun.com, last accessed on June 2008

[13] Eclipse framework home page, http://www.eclipse.org/, last accessed on June 2008

[14] Amorim, B.M.C. (2008). *Optimização de abastecimentos de materiais no âmbito dos processos de logística interna*. Masters thesis, Faculdade de Engenharia da Universidade do Porto, Portugal.

Bibliography

- Abu-Taieh, E. and El Sheikh, A. (2007). Commercial Simulation Packages, I.J. of Simulation, Vol. 8, No 2, pp 66-76
- AnyLogic homepage, www.anylogic.com, last accessed on June 2008
- Booch, G., Rumbaugh, J. and Jacobson, I. (1999). *The unified modeling language: user guide*, Addison Wesley.
- Borshchev, A. and Filippov, A. (2004). From System Dynamics and Discrete Event to practical Agent based modelling: reasons, techniques, tools, The 22nd international conference of the System Dynamics Society.
- Chung, C.A. (2004). *Simulation Modeling Handbook – A practical approach*, CRC Press.
- Committee on Visionary Manufacturing Challenges, Board on Manufacturing and Engineering Design, Commission on Engineering and Technical Systems, National Research Council (1998). *Visionary Manufacturing Challenges for 2020*, National Academy Press.
- DHL Glossary of Logistics Terms, http://www.dhl.com/publish/g0/en/information/customer/glossary.low.html#aPar0066, last accessed on June 2008
- Eckel, B. (2000). *Thinking in Java*, 2nd edition, Prentice Hall.
- Glebovsky, A.Y., Ivanov, V.M. and Karpov Y.G. (2006). Computer Modeling and Simulation in Higher Education, APRU Distance Learning and the Internet 2006 Conference, pp 201-210
- Java language class documentation page, http://java.sun.com/j2se/1.4.2/docs/api/index.html, last accessed on June 2008
- Marcelino, M.J. and Mendes, T. (). Estratégias e ferramentas para a construção de programas educativos de simulação, Universidade de Coimbra, Portugal.

- McLean, C. and Leong, S. (2001). The expanding role of simulation in future manufacturing, 2001 Winter Simulation Conference, pp 1478-1486
- Narayanan, S., Bodner, D.A., Sreekanth, U., Govindaraj, T., McGinnis, L.F. and Mitchell, C.M. (1998). Research in Object-Oriented Manufacturing Simulations: An Assessment of the State of the Art, IIE Transactions, Vol 30, No 9, pp 795 - 810
- Oliveira, J.F. (2001). *Simulação - Transparências de apoio à leccionação de aulas teóricas de Investigação Operacional*, Faculdade de Engenharia da Universidade do Porto.
- Pooley, R. and Stevens, P. (1999). *Using UML: software engineering with objects and components*, Addison Wesley.
- Russell, J.P. (2002). *Learn Java in a weekend*, Premier Press.
- Teixeira, J.M.D.F. and Brito, A.E.S.C. (1997). *Simulação visual de armazéns automáticos*, Masters thesis, Faculdade de Engenharia da Universidade do Porto, Portugal.
- Tham, M. (2001). Writing research theses or dissertations, http://lorien.ncl.ac.uk/ming/dept/Tips/writing/thesis/thesis-intro.htm, last accessed on June 2008