REAL TIME MANUFACTURING SYSTEMS SIMULATOR

Luís Miguel Afonso Teixeira

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto
2012

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Luís Miguel Afonso Teixeira, Nº 1040747, 1040747@isep.ipp.pt
Orientação científica: Doutora Ana Madureira, amd@isep.ipp.pt

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto
6 de Novembro de 2012

“Nothing ever comes to one, that is worth having, except as a result of hard work.”

Booker T. Washington

Acknowledgments

I would like to thank my family for the unconditional support not just throughout this project but also, during my entire academic journey and my life. A special thanks to my sister Marlene for the help in revising the document.

I thank my supervisor Doutora Ana Madureira for her guidance, availability, interest and support shown throughout the development.

I thank Eng. Mário Felgueiras for his tireless support over the years to all the students of the *Departamento de Engenharia Electrónica* as well as his sympathy and kindness.

Finally, to all my friends who have, one way or another, tried to help me in the project and keeping me motivated and focused in my goals.

Resumo

A área da simulação computacional teve um rápido crescimento desde o seu apareciment, sendo actualmente uma das ciências de gestão e de investigação operacional mais utilizadas. O seu princípio baseia-se na replicação da operação de processos ou sistemas ao longo de períodos de tempo, tornando-se assim uma metodologia indispensável para a resolução de variados problemas do mundo real, independentemente da sua complexidade.

Das inúmeras áreas de aplicação, nos mais diversos campos, a que mais se destaca é a utilização em sistemas de produção, onde o leque de aplicações disponível é muito vasto. A sua aplicação tem vindo a ser utilizada para solucionar problemas em sistemas de produção, uma vez que permite às empresas ajustar e planear de uma maneira rápida, eficaz e ponderada as suas operações e os seus sistemas, permitindo assim uma rápida adaptação das mesmas às constantes mudanças das necessidades da economia global.

As aplicações e *packages* de simulação têm seguindo as tendências tecnológicas pelo que é notório o recurso a tecnologias orientadas a objectos para o desenvolvimento das mesmas.

Este estudo baseou-se, numa primeira fase, na recolha de informação de suporte aos conceitos de modelação e simulação, bem como a respectiva aplicação a sistemas de produção em tempo real. Posteriormente centralizou-se no desenvolvimento de um protótipo de uma aplicação de simulação de ambientes de fabrico em tempo real. O desenvolvimento desta ferramenta teve em vista eventuais fins pedagógicos e uma utilização a nível académico, sendo esta capaz de simular um modelo de um sistema de produção, estando também dotada de animação. Sem deixar de parte a possibilidade de integração de outros módulos ou, até mesmo, em outras plataformas, houve ainda a preocupação acrescida de que a sua implementação recorresse a metodologias de desenvolvimento orientadas a objectos.

Palavras Chave

Simulação, Sistemas de Produção, Modelação, Sistemas Discretos, JAVA, Aplicação.

Abstract

The area of computer simulation has experienced a rapid growth since its appearance, currently it is one of the most used management and operational research sciences. Its principle is based on the replication of operations for processes or systems, throughout periods of time, thus making it an indispensable methodology for solving several real world problems, regardless of its complexity.

Of the many applications if the various fields its use in manufacturing systems stands out the most and there is a vast array of applications available. Its application has been used to solve problems in manufacturing systems, since it allows companies to a quick, effective and weighted adjustment and planning for their operations and their systems, hence allowing a rapid adaptation to the constant changes of the global economy demands.

Simulation applications and packages have been following technology trends which is notorious due to the use of object-oriented technologies for their development.

This study was, initially, based on collecting information to support the concepts of modeling and simulation as well as its application to real time manufacturing systems. Subsequently it focused in the development of a prototype for an application for real time manufacturing systems simulation. This development was aimed at eventual academic and educational purposes, and the application is capable of simulating the model of a manufacturing system also having animation capabilities. With no disregard to the possibility of integration of other modules or its integration in other platforms, there was an increased concern in recurring to object oriented methodologies in its development.

Key Words

Simulation, Manufacturing Systems, Modeling, Discrete Systems, JAVA, Application.

Contents

List of figures

List of Tables

Acronyms

ABM - Agent-based Modeling

ADSyS - Adaptive Decision Support Systems

AGV - Automatic Guided Vehicle

API - Application Programming Interface

BI - Business Intelligence

CAD - Computer Aided Design

CAM - Computer Aided Manufacturing

CIM - Computer Integrated Manufacturing

CNC - Computer Numeric Control

CPU - Central Processing Unit

DES - Discrete-event Simulation

ERP - Enterprise Resource Planning

FIFO - First In First Out

FMS - Flexible Manufacturing Systems

FWS - Flexible Work Cells

GECAD - Grupo de Investigação em Engenharia do Conhecimento a Apoio à Decisão

GNU - General Public License

GSP – General Simulation Program

GUI	-	Graphical User Interface
IDE	-	Integrated Development Environment
IT	-	Information Technologies
JIT	-	Just in Time
JRE	-	Java Runtime Environment
JSL	-	Java Simulation Library
JVM	-	Java Virtual Machine
LIFO	-	Last In First Out
MRP	-	Material Resource Planning
OOP	-	Object Oriented Programming
OTS	-	Operator Training Simulator
RTMSS	-	Real Time Manufacturing Systems Simulator
SDK	-	Software Development Kit
SIRO	-	Service in Random Order
SLAM	-	Simulation Language for Alternative Modeling
SSJ	-	Stochastic Simulation in Java
UML	-	Unified Modeling Language
VBA	-	Visual Basic for Applications
WSC	-	Winter Simulation Conference

1. INTRODUCTION

1.1. MOTIVATION

In today's competitive business environments, careful planning and analysis of alternative strategies and procedures is essential. In the 1960's and in an effort to derive maximum benefit from available resources, engineers and business planners made mathematical and computer modeling an important part of their planning activities with simulation being the most popular modeling technique due to is broad range of applicability.

Keith Douglas Tocher was responsible for developing the first general-purpose simulator at the University of Southampton the General Simulation Program (GSP), a tool for systematically building a simulation of an industry plant [1]. Manufacturing simulation has effectively been one of the first and primary applications areas of the simulation technology being widely used to validate and design a variety of manufacturing systems.

The simulation technology has rapidly grown since its early days in the 1960's allowing many fields to rely on its use and its growth has been met by the increasing availability and ease of use of the simulation software packages. The field of simulation will continue to expand and the range of applications is moving out from the domain of the industrial, defense and gaming systems to many aspects of our lives.

With the agglomeration of the information regarding manufacturing systems, modeling and simulation present in this document students from several courses at the Institute of Engineering - Polytechnic of Porto (ISEP) might have an easier task if and when approaching these subjects. Eventually, the simulator prototype might be used for

academic purposes in lectures where simulation or other topics related to manufacturing are addressed.

As there was no previous work, the entire application was created from scratch. Due to several different subjects, the work was divided into four stages:

- Learning of Java language and OOP concepts;
- Learning and testing several Java-based simulation tools and examples;
- Select a Java simulation package of choice to further explore and develop a Graphical User Interface (GUI) to run some examples;
- Design the Simulator Prototype and implement the required functions.

1.2. OBJECTIVES

This thesis will be integrated in the R&D project ADSyS (Adaptive Decision Support System for Interactive Scheduling with MetaCognition and User Modelling Experience (PTDC/EME-GIN/109956/2009)) carried out by the GECAD (Grupo de Investigação em Engenharia do Conhecimento a Apoio à Decisão) Research Group from ISEP/IPP

Broadly, the objectives of this dissertation focus on developing a simulation tool from scratch using the object-oriented programming language JAVA that allows its users to get some perception on the concepts behind simulation and provide some animation to translate simple manufacturing models into graphical movement. In the long term it can serve as a basis for further development and can be used for academic purposes in courses that address the topic and concepts behind simulation and scheduling in manufacturing.

Thus, to summarize, the main objectives for this work are as follows:

- Elaborate a survey covering: manufacturing systems and their characteristics, modeling and simulation concepts, tools and technology behind simulation;
- Analyze and demonstrate the modeling of discrete real-world systems;
- Develop a JAVA application capable of simulating small manufacturing systems with basic animation;

1.3. DOCUMENT STRUCTURE

This document is divided into 7 chapters.

Chapter 2 describes Manufacturing systems scope, different system characteristics, different classifications and factors affecting their choice as well as a description of the different plant layouts.

Chapter 3 addresses Modeling and Simulation. An overview of the simulation concepts is introduced and the subject of simulation applied to manufacturing is detailed. Some actual modeling examples are presented to provide a better "real-world" view.

Chapter 4 presents the state of the art concerning the simulation software packages available on the market as of today for both general-purpose and manufacturing-oriented tools. The fundamental concepts supporting object oriented simulation and programming are detailed thus providing the needed explanation to support the decision of the programming language of choice which is also introduced. Some java based simulation environments and toolkits are addressed.

Chapter 5 presents all the work behind the development of the prototype. First, a brief walkthrough of the simulation models and examples used are discussed. Secondly, the design and architecture for the system is detailed. Finally the details from the prototype are presented and described, with the modules composing it being addressed, as well as all the characteristics and features of the prototype and its interface.

Chapter 6 demonstrates some simulations and runs of the prototype developed alongside the results and discussions inherent.

Finally, **Chapter 7** overviews the work developed, along with the final conclusions and provides an outlook of possible work or enhancements.

2. MANUFACTURING SYSTEMS

The use of machinery triggered with the Industrial Revolution [1] (1750 to 1850) aiming to increase the production levels and decrease the man-powered working force to try and get the economy back on track. Before that, all the manufacturing systems were handmade and crafted.

Also, after the World War II the United States of America were the promoters of the industry by providing a new manufacturing philosophy characterized by the use of interchangeable machinery and parts, work specialization and steam-based machinery (see Figure 1). This marked the breakthrough of mass production as we know it. [2]

From the changes that have occurred throughout history, one must be aware that the environmental changes have reflected a lot in the manufacturing industry. A lot of changes have occurred but the increase of the customer's sophistication and demands has been of

[1] The Industrial Revolution was a period from 1750 to 1850 where changes in agriculture, manufacturing, mining, transportation, and technology had a profound effect on the social, economic and cultural conditions of the times. It began in the United Kingdom, then subsequently spread throughout Western Europe, North America, Japan, and

great importance. They are more demanding and searching for more variety at lower costs with increased quality, which made the market change from a scale economy and, thus, mass production to a scoped economy aimed for variety. The manufacturing industry nowadays has the difficult task to balance between mass production and variety.

Figure 1 A Watt steam engine. The steam engine, fueled primarily by coal, propelled the Industrial Revolution in Great Britain and the world [3]

It is important to define manufacturing as it is, because manufacturing and production are often confused or thought to be the same when they are not. Manufacturing is the process of creating something be it an idea, a product, a service using the process of gathering the raw materials needed to make it come to life. Manufacturing can be include the production of either one item or the production of multiple items.

A system can be defined as a group of entities or objects that grouped together interact amongst them in a regular way or have some sort of dependency from one another to accomplish some purpose. Manufacturing at its terms is the process of converting raw materials to finished products by using several processes, machinery and energy. [4]

Every type of manufacturing is some sort of production though, every production may not be manufacturing. Manufacturing has all stages of producing a product so one can say that production is a subset of manufacturing [5]. As depicted in Figure 2 a production system has a determined function of converting an array of inputs into desired outputs involving one or more processes for that conversion referred as manufacturing processes or systems.

Elwood Buffa [6] defines production as “a processes by which goods and services are created.” meaning that the production process implies that the value addition will occur throughout the process stages with an implicit cost. Production is the process of converting inputs to outputs, it is a broader term.

The term production is often interchangeably used along with manufacturing although production is broader as it includes the manufacturing and the non-manufacturing processes as well. There are numerous systems or operations that fit as an example of the stages of a production system. When thinking about a dental clinic some Input resources such as the Dentists, the Nurses or Auxiliary Staff, the dental equipment and the patients themselves can be easily identified. The conversion process takes part in examining and providing the needed dental treatments as well as providing preventive orientation and awareness what will result in the outcome of the outputs being the treated patients with a healthy oral and dental condition.

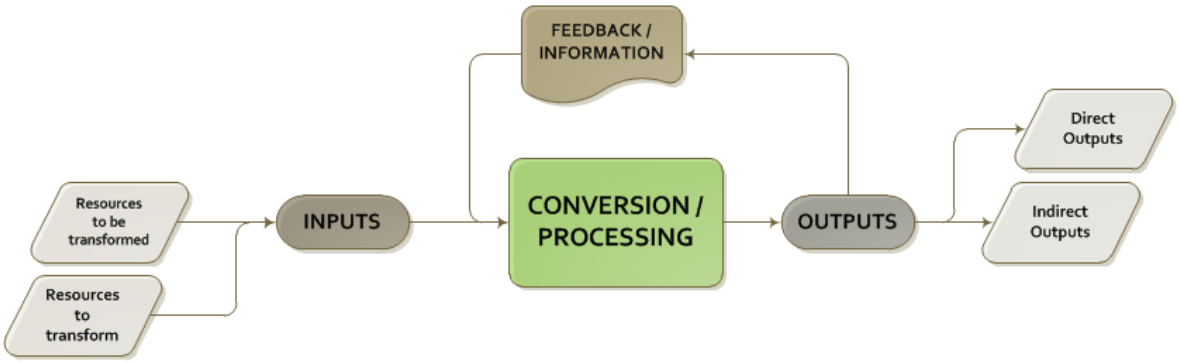


Figure 2 Diagram of a basic production system

With the introduction of automation and computer science in the production systems, the business has changed and evolved which lead to the information flow being the critical one from both of them, something unusual some decades ago.

When considering different *input-transformation-output* processes, it is clear that there is a difference between manufacturing operations producing physical goods or products and service operations producing intangible assets. Slack [7] presents a good example of a hospital where there are several inputs such as doctors, administrator, beds, medical equipment, pharmaceutical products, blood, clothing and others meant to transform ill patients in healthy patients. There are several outputs from that transformation operation not only treated patients but also medical exams results, medical research, medical procedures. From the outside of the building, factories could be thought to be similar as a hospital.

Table 1 Some examples of operations described in terms of input, output and its transformation. (Adapted from [7])

Operation	Input Resources	Transformation Process	Outputs
Airline	- Airplane - Pilots and boarding crew - Land crew - Passengers - Cargo	- Moving the passengers and cargo from one location to another	- Transported passengers and cargo
Police Force	- Policemen - Computer network - Information - Public (citizens and criminals)	- Crime prevention - Crime solving - Criminals arrest	- Protected society - Public feeling safe
Containers Port	- Ships and cargo - Employees - Cargo movement equipment	- Moving the containers from the ship to the port and vice-versa	- Loaded or unloaded ships

Table 1 describes other examples similar to the one presented previously about the hospital, where *input-transformation-output* processes can be identified.

2.1. TYPES OF MANUFACTURING SYSTEMS

Although most people think of manufacturing systems to be all the same or much alike in fact they are not. There are thousands of different products being produced and obviously, they do not require the same processes as one or another thus meaning that various systems must exist in order to meet the unique demands and characteristics of all products.

The main purpose for classification is helping to understand the object or subject being studied so the relationships between the inherent characteristics, as well as the typical problems and solutions, can be identified and, although several classifications of manufacturing systems are spread throughout the literature, it is commonly thought of accordingly to the production type under carried. Moreira [8] presents the classification of the manufacturing systems according to the production type (see Figure 3) in the so called *Traditional Classification* [8]. Several authors have their own interpretation on the definitions for the several systems so we cannot say that a consensus has been reached throughout the years. We can see manufacturing systems classified according to the flow of the products (continuous or intermittent), to the type of customers support (stock or

order oriented systems), standardization degree, operation type, product nature and many others [4].

Manufacturing systems can also be classified accordingly to both the diversity of the generated outputs and/or products and the production/operation volume as *Project*, *Job-Shop*, *Batch*, *Mass* and *Continuous* Production.

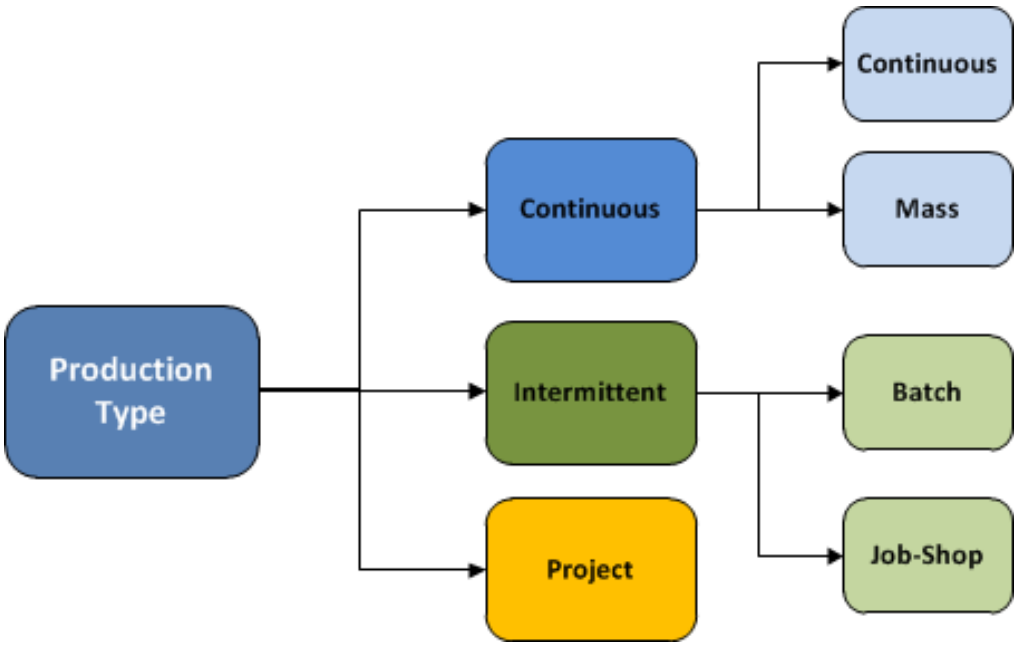


Figure 3 Diagram for the different types of manufacturing systems and their classifications according to the production type

At a higher level, manufacturing systems could be grouped accordingly to the flow of the processes being continuous, intermittent or with no flow or repetition in case of project production as is depicted on Figure 3. On continuous production systems, the operations are executed in a continuous and linear flow and the material or parts move in an uninterrupted flow whereas the intermittent production systems face frequent variations in the works and operations resultant of the diversity of the manufactured products thus resulting in an intermittent flow.

As will be mentioned further on, continuous production systems ensure less work in processing inventory and also high product quality with the disadvantage of large investments when it comes to machinery and equipment. On the opposite intermittent production systems are produced partly for stock and inventory and partly taken in consideration the current customer orders [9].

With different characteristics of the manufacturing operations, the systems can be grouped according to four particularly important output measurements [7]:

- Volume;
- Variety;
- Varying Demand;
- Customer involvement.

Figure 4 shows several manufacturing systems grouped according to their volume/variety ratio. This is one of the major considerations in selection of manufacturing process. When the volume is low and variety is high, intermittent process is most suitable and with increase in volume and reduction in variety continuous process become suitable, as depicted in Figure 4.

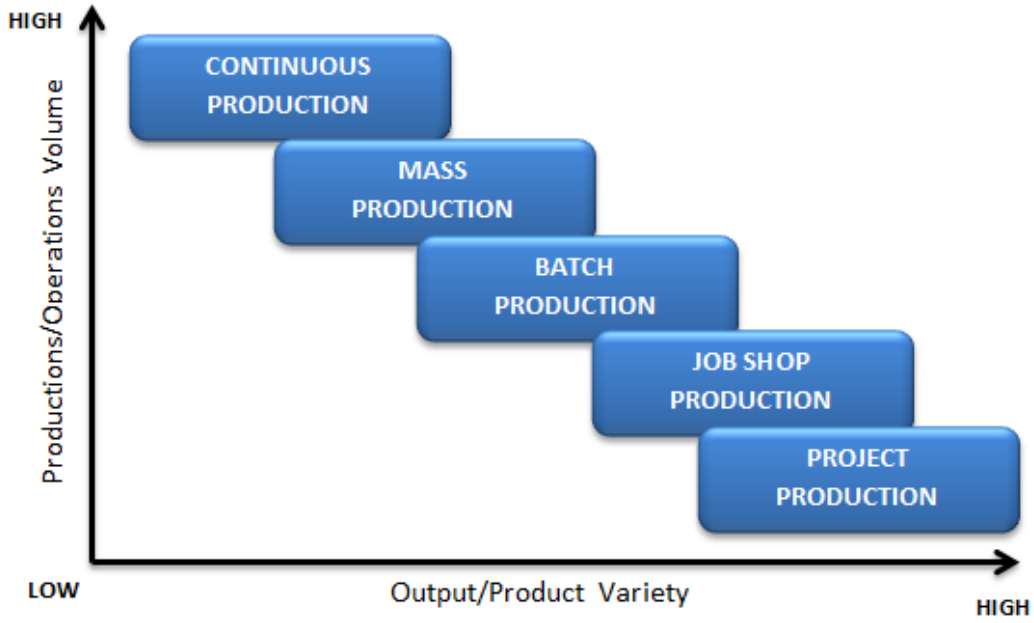


Figure 4 The various manufacturing systems and their volume/variety ratio

2.1.1. PROJECT PRODUCTION

The client usually presents its own project for the product or service and the needed specifications for its manufacture. This means that the resources and the activities needed for each project are most likely to differ. The products are then highly customizable and the time span is fairly large on most occasions. The constructions of a bridge, a ship, a wind turbine are good examples of project production systems [9].

Table 2 Characteristics, advantages and limitations for the Project production manufacturing systems

Characteristics	Advantages	Limitations
- *Flow* – No flow - *Flexibility* – Very high - *Products* – Unique - *Capital Investment* – Very Low - *Variable Cost* – Very High - *Labor Content and Skill* – Very high - *Volume* - Low	- The inputs are brought to the project only when needed reducing the necessity for stock meaning a very low capital investment; - As the requirement of resources is not uniform they can be re-deployed elsewhere in other projects;	- Frequent changes in the manufacturing activities due to uncertainty in their definition; - A project usually involves many tasks and therefore different work teams demanding good coordination amongst them; - Network planning techniques like PERT and others are usually adopted to overcome scheduling and control problems.

The essence of the project production is that each project has a determined beginning and conclusion date and the time between the start of different projects is usually big as the workers and the resources that operate on it are organized specially for each one of them.

2.1.2. JOB-SHOP PRODUCTION

A Job-Shop production system describe a manufacturing environment in which small batches of products are produced, meaning that there is a diversity of jobs to be handled where each job is different [9]. This means that the products are design and produced as per the specifications and demands of costumers. The environment comprises of general purpose machinery commonly arranged into different sections.

Furthermore, the complexity and diversity in the manufacturing process means that the control of the flows is extremely complex creating a complex system where a job exits from a machine to wait on a new machine because of other jobs are taking place. Planning a Job-Shop production is a process of prioritizing the jobs on each machine.

The whole operation is considered as one and the work is completed on each product or batch of products before passing to the next work until the production of a single complete unit by one operator or more is then finished. Examples of Job-Shop production systems are the automobile building industry or a machine shop creating parts for industrial machinery.

Table 3 Characteristics, advantages and limitations for Job-Shop manufacturing systems

Characteristics	Advantages	Limitations
- Flow – No flow - Flexibility – Very high - Products – Unique - Capital Investment – Very Low - Variable Cost – Very Low - Labor Content and Skill – Very high - Volume - Low - Inventory – Large	- Variety of products can be produced due to the existence of general purpose machinery; - Operators can easily develop their skills as their full potential is required; - Opportunity for creative methods and innovative ideas.	- Frequent Set-Up changes means higher costs; - Higher level of inventory required; - Complicated production planning;

As Table 3 shows job-shop production processes deal with a high variety and low volume. The resources are shared amongst the different products. Examples of Job-Shop production are furniture restorers, tailors, and a Graphical shop producing flyers for an event. The processes produce more and usually smaller products when compared to project production but, likewise, the degree of repetition is also low meaning that most of the jobs will also be unique.

2.1.3. BATCH PRODUCTION

The batch production system is somewhat an extension of the Job-Shop production as the whole project is considered as one operation and the work must be completed on each product before passing to the next, however, the main difference is that the variety in batch production is not the same as in the Job-Shop production. As the name implies in a batch production system means similar items are produced together in batches [9].

Table 4 Characteristics, advantages and limitations for the Batch production manufacturing systems

Characteristics	Advantages	Limitations
- Flow – Disconnected - Flexibility – Moderate - Products – Several - Capital Investment – Moderate - Variable Cost – Moderate - Labor Content and Skill – Moderate - Volume - Moderate	- Larger possibility for sales as a broader variety of products can be possible; - Risks are reduced as the variety of products reduces the risk of concentrating efforts in only one product; - Useful when referring to products for which it is difficult to forecast demand;	- There is the need for large storage spaces as the ingoing progress inventory is high; - Frequent need of changes in machines an tools; - The need for down times; - Longer production times; - Higher skills required due to the diversity of the jobs.

The jobs pass through the functional departments in small lots or batches and different flows or routes are possible. This is a step by step process meaning that one job cannot begin until the previous one has been finished.

Batch production is often adopted whenever there is the need for introducing variants in the finished products. That creates the need for down times, referring to the setup time needed to change or prepare the machines that allow variants in the manufacturing process but also generates high cycle times. Examples of batch production systems are bakeries, printing presses and the production of electronic instruments and clothing.

2.1.4. MASS PRODUCTION

The main characteristic of the mass production systems is its standardization as the products are produced in large quantities disregarding the existing costumer order so that production is to stock. The volume of the production is fairly large whereas the variety is low, making it possible to minimize any changes necessary which saves time and therefore reduces costs [9].

The mass production system creates a uniform and also uninterrupted flow of material or parts which is maintained through a predetermined sequence of operations meaning the system can only produce one product at a time. Since most of the operations in mass production are fairly repetitive and therefore tedious, many companies have gathered efforts throughout time and invested their time and money not only by using job rotations to try and enhance the work life quality of their employees but also by introducing robots in the manufacturing operations.

Table 5 Characteristics, advantages and limitations for Mass production manufacturing systems

Characteristics	Advantages	Limitations
- Flow – Connected assembly line - Flexibility – Low - Products – Few - Capital Investment – High - Variable Cost – Low - Labor Content and Skill – Low - Volume - High	- Non-productive efforts are reduced; - Little to no time preparation for materials and tools; - The increased use of machinery reduces human error, variations and labor costs and increases production rates;	- Inflexible as result of the difficulty to alter the production process after the system has been implemented; - Little variety to the products.

The most common example of mass production is the assembly lines for automotive manufacturing (made famous by Henry Ford due to the introduction and development of assembly lines for the production of the Ford Model T [10]) but other systems can be mentioned such as the production of airplanes, the processes of food products and a beer factory.

2.1.5. CONTINUOUS PRODUCTION

Continuous production is placed one step ahead of mass production due to the fact that they operate bigger volumes at an even lower variety and, usually, for a very long period of time. The system may run twenty four hours a day, for weeks or even month stopping only when maintenance is needed or unexpected breakdowns occur. It is often mentioned to as Flow Production as the product flow is, at times, literally continuous this creating and interrupted flow [9].

Like in mass production, there is a continuous flow with a pre-determined and fixed pace and sequences. Examples of continuous productions systems are the oil and petrol refineries, the plastic industry, some medicine productions and heavy chemical industries.

Table 6 Characteristics, advantages and limitations for Continuous production manufacturing systems

Characteristics	Advantages	Limitations
- Flow – Continuous - Flexibility – Very Low - Products – One - Capital Investment – Very High - Variable Cost – Very Low - Labor Content and Skill – Low/High (the direct labor content and its associated skill is low but there might be times when overview of the equipment may require higher skills) - Volume – Very High	- The ongoing production makes it possible to achieve higher production volumes in short amount of time thus eliminating the need for start-ups and shutdowns; - The use of automation can reduces unnecessary labor and also lower unit costs and better energy savings; - Higher product quality standards by reducing opportunities for human errors.	- Although the low unit cost per product, the capital cost of some equipment can be fairly expensive resulting in high initial costs; - Reduces the human work force;

2.2. COMPONENTS OF MANUFACTURING SYSTEMS

Although, the existence of several different manufacturing systems for a variety of products with different characteristics some, of the components present are common

amongst them. As stated previously, manufacturing is the process of converting raw materials to finished products by using several processes, machinery and energy thus a collection of integrated equipment (be it machines and tools, material handling and work positioning devices or computer systems) and human resources is present.

Table 7 provides a list of examples regarding the major common components in the manufacturing systems.

Table 7 Manufacturing components and some examples [11]

Manufacturing Components	Examples
Product	- Parts/pieces - Routings - Process Times - Setup Times - Bill of materials - Rework
Resources	- Equipment layout - Number of machines - Downtime - Maintenance - Storage areas - Tools - Labor - Shifts
Demand	- Customer orders - Start date - Due date - Inventory
Control	- Warehouse management - Inventory control - Shop floor control - Station rules

A more meticulous description of such components is presented in [11].

2.3. SYSTEM LAYOUTS

The layout of the manufacturing systems is one of the strategic areas important to determine the production operations' efficiency [12]. The objective of the layout strategy is the development of a layout arrangement capable of fulfilling the requisites:

- Product strategy – product and volume projects;
- Process strategy – process and capacity equipment;
- Human Resources strategy – quality work space;

- Location strategy – facility requirements and constraints.

The layout specifies the arrangement of the processes, the required equipment and the work areas, therefore an efficient layout will facilitate the flow of material and personal within the confined work spaces. Layout decisions include obtaining the best positioning of the machinery, offices, secretaries, service centers or such always with the ultimate goal to operate the system at its maximum efficiency and effectiveness. Four types of system layouts can be identified as follows:

- Facility layouts (sometimes mentioned as continuous flow layouts);
- Product layouts (sometimes mentioned as production line layouts);
- Process layouts;
- Fixed-position layouts;
- Cellular manufacturing layouts.

2.3.1. FACILITY LAYOUTS

In the facility layouts strategy, the configuration of all machines, employee workstations, storage areas that constitute the facility used to create a firm's product or service. The facility in this case represents a high capital investment as they are usually highly automated and projected to work as a whole unit. Figure 5 illustrates a dam which is a known type of facility layout.

Figure 5 Dam in Castelo do Bode. Picture available at the CNPGB [2] website

[2] CNPGB stands for *Comissão Nacional Portuguesa das Grandes Barragens*. The picture presented as well as many others are available at http://cnpgb.inag.pt/.

2.3.2. PRODUCT LAYOUT

In a product layout all items to be produced follow the same sequence of operations from beginning to end, such as an assembly line (see Figure 6).

Figure 6 Icom[3] factory assembly line for radio and electronic products

These layouts are obtained alongside the working force and the machinery in accordance to a pre-defined sequence of operations to perform in a product. These are very common layouts for automobile production and assembly, dairy products, electronic components production and so forth. Although they produce very efficient results and allow for cheaper and less qualified workers these layouts allow little flexibility and create high dependency between operations meaning that, for instance, if a machine has a breakdown the whole production may be compromised.

2.3.3. PROCESS LAYOUT

In process layouts the system is designed so that similar machines or functions are grouped together in so called *departments*. It is usually applied to situations where the production of different products with different demands is required. A product or a batch is produced by moving from one department to another in a pre-defined sequence for the task.

It allows for a good flexibility either in the equipment or in the tasks attributed to the workers and guarantees the availability of the system as even with a faulty machine the work can be moved on to another machine or even to another department. However it

[3] Icom is a Japanese held corporation Its roots are in designing, engineering, and manufacturing highly advanced, compact solid-state radio equipment for use in the amateur (ham) radio industry. The organization's product line has since expanded to include communications equipment and products based in the marine, avionics, land mobile and wide-band receiver industries.

demands for greater management capabilities of the system regarding the routings, machine preparation and such which most of the times means higher costs.

2.3.4. FIXED-POSITION LAYOUT

A fixed-position layout designs a production system arrangement in which the product being built or produced stays at one location and the machines, workers, and tools required to build the product are brought to that location as needed, as for the building of ships or other bulky products (see Figure 7).

Figure 7 Ship building facility in Gdansk, Poland [13]

With this type of layout it is possible to obtain a greater planning and control of the task at hands as everything is directed towards a single objective, however, the costs of associated material and specialized workers travel are usually fairly large.

2.3.5. CELLULAR MANUFACTURING LAYOUT

Cellular manufacturing layouts are a combination of process and product layouts, in which machines and personnel are grouped into cells containing all the tools and operations required to produce a particular product or family of products (see Figure 8).

Figure 8 Manufacturing cell area from NSPI's nuclear sensors manufacturing plant

Cellular Manufacturing is an approach that helps build a variety of products with as little waste as possible. A cell is a group of workstations, machine tools, or equipment arranged to create a smooth flow, so families of parts can be processed progressively from one workstation to another without waiting for a batch to be completed or requiring additional handling between operations. A part family can be parts similar in size or parts created using similar manufacturing steps and typically, a cell is dedicated to a single part family [12].

Cellular layouts commonly adopt two configurations either *U-shaped* or *parallel* layout as is depicted in Figure 9.

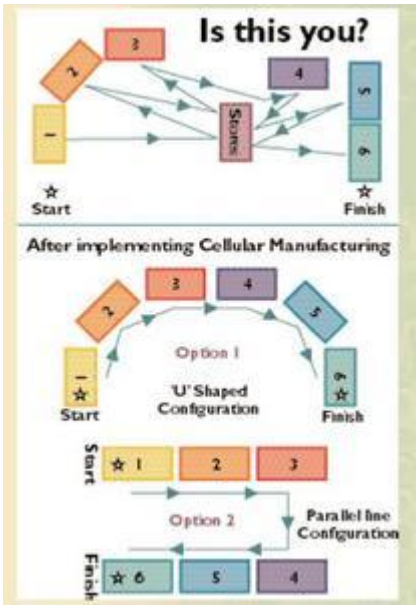


Figure 9 Difference between a typical unorganized manufacturing layout and two different cellular manufacturing configurations, *U-shaped* and *parallel* line

Although the implementation process of shedding the traditional manufacturing processes and embracing the drastically different cellular manufacturing techniques can be a daunting task it ensures a number of benefits such as reducing material handling and transit times meaning the product spends more time on the machinery and less time routing between them, reducing production times thus allowing shorter delivery dates and increased worker productivity, just to name a few.

2.3.6. COMPUTER INTEGRATED MANUFACTURING

With the increased rate in production levels and the typically highly mechanized assembly lines, the role of *automation* has gained prominence. This is clearly noticeable due to the replacement of people by machines to make their tasks. Over a period of time factory

owners have integrated computer systems in order to streamline the production process. CAD, or computer aided design, has been able to aid the operators in the formulation and blueprints of more sophisticated products and technology.

By working alongside with CAD, the computer aided manufacturing (CAM) has allowed manufacturing systems and companies to reduce their cycle times, enhance productivity and create benefits throughout the overall production process.

The manufacturing sector has moved towards the computer integrated manufacturing (CIM) which differs in the addition of the exchange of data and information between computers, e.g. an arm robot exchanging information with a computer numeric control (CNC) machine.

To better understand the philosophy behind CIM, the implemented technologies such as *robots* and *computers* should be distinguished.

Robots are reprogrammable machines capable of performing numerous programmed tasks by manipulation materials and tools. Figure 10 shows a fully automated one-armed robot production line where robots have been programmed to solder all the chassis components. With the decrease in technology costs and the increase in flexibility robots have become increasingly common in the industry now moving outside the fields of automotive and electronics industries, where their usage was triggered. Robots have gained our trust and proved they are worth the investment mostly due to the fact that they are not exposed to the hazards and are not dangerous to humans.

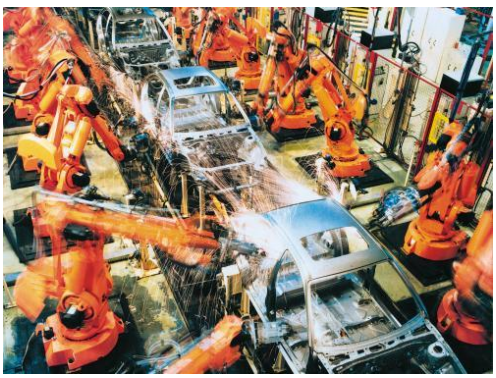


Figure 10 Manufacturing line on a car manufacturing plant

It is also remarkable the advances in robot technology in the past decades that are making them especially useful in the service industries such as the health care industry where robots now assist surgeons in performing delicate operation.

The growth in the usage and implementation of robots has been closely followed by the integration of new computer technologies, the already mentioned CAD, CAM and CIM. Other than that, the *Flexible Manufacturing Systems* (FMS) have peaked to currently be the state of the art in industry. These systems are composed by different machining centers, could be depicted in Figure 11, and are characterized by their ability to quickly be modified and adapt to manufacture different products whilst being entirely controlled by computers.

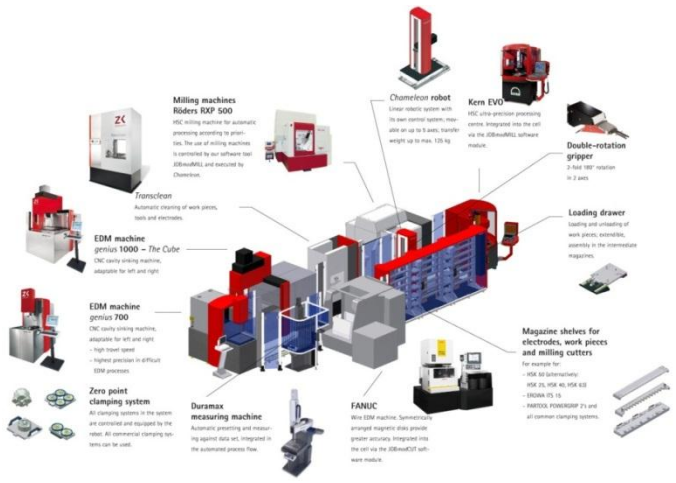


Figure 11 Example of a FMS system implementation by Zimmer&Kreim [4]

2.3.7. MEASURES OF PERFORMANCE

To measure the performance of a manufacturing system and its model, several statistics could be typically collected. Bear in mind that when collecting these factors the average and the variability should be collected, if possible. The following factors are typically collected and should thus be provided also by the models of such systems:

- Throughput;
- Cycle time;
- Queuing;
- Response time for material handling equipment;
- Work in progress (WIP);
- Resources utilization (both equipment and labor);
- System specific performance measurements.

[4] Zimmer&Kreim is a german based enterprise which supplies its cutting-edge technology products to an international portfolio of customers in the business areas of EDM machines, handling systems and software solutions.

2.4. MANUFACTURING SYSTEM CHOICE – AFFECTING FACTORS

The classification of the manufacturing systems makes it easier to plan and decide which management strategies apply to the system but also rationalizes the choice of what system adopt on different circumstances. The choice must be able to meet two main objectives:

- Meet the final product needed specifications;
- Be cost effective.

There are several factors to be considered when determining the choice of the production system [14]:

- Effect of Volume/variety;
- Capacity of the plant;
- Availability of necessary equipment or work force;
- Type of product;
- Flexibility;
- Efficiency;
- Environment.

2.5. CONCLUSION

The aforesaid is only an introduction to the aspects regarding manufacturing systems. The various type of manufacturing systems, their designations and characteristics were detailed as well as its advantages and disadvantages.

Also, some common components of such systems were presented. The various types of existing layout configurations where also detailed.

Finally, the factors which may affect the choice of the manufacturing system to implement are presented but, for further detail, please refer to [14].

3. MODELING AND SIMULATION

Simon [15] defines modeling as "*a principal – perhaps the primary –tool for studying the behavior of large complex systems"*.

Models are descriptions of the systems (see Figure 12), developed based on theoretical laws and principles. They may be scaled physical objects (*iconic models*), mathematical equations and relations (*abstract models*) or Graphical representations (*visual models*).

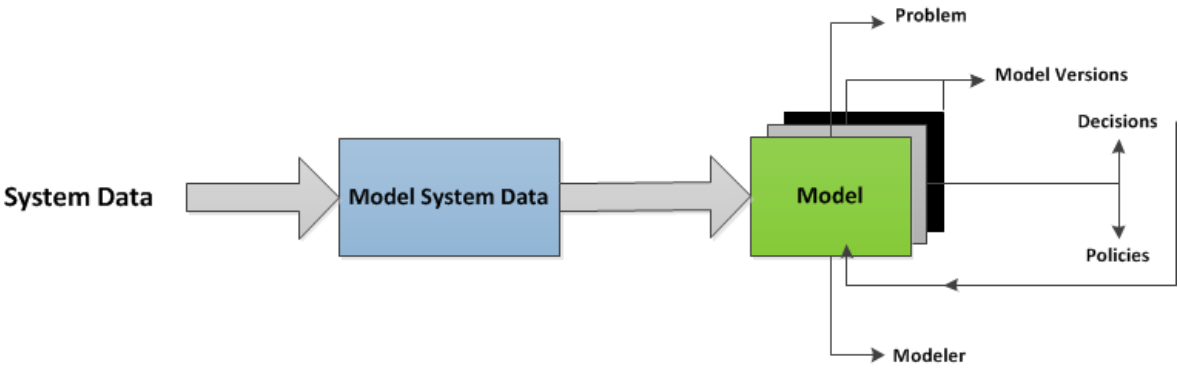


Figure 12 Model based problem solving process (adapted from [16])

The first step in model building is the clear definition of the purpose and the goals so that it is possible to decide on the elements of the system to include. These choices are crucial as a model is an abstraction of the system.

Alan Pritsker [17] attempts to provide modeling principles based on his experience:

Principle 1 – Conceptualizing a model requires system knowledge, engineering judgment and model-building tools.

Principle 2 – The secret to being a good modeler is the ability to remodel.

Principle 3 - The modeling process is evolutionary because the act of modeling reveals important information piecemeal.

Principle 4 – The problem or problem statement is the primary controlling element in model-based problem solving.

Principle 5 – In modeling, combined systems, the continuous aspects of the problem should be considered first. The discrete aspects of the model – including events, networks, algorithms, control procedures, and advanced logic capabilities – should then be developed. The interfaces between discrete and continuous variable should then be approached.

Principle 6 – A model should be evaluated according to its usefulness.

Principle 7 – The purpose of simulation modeling is knowledge and understanding, not models.

3.1. MODELING CONCEPTS (SYSTEMS, MODELS AND SIMULATIONS)

The definition of a system was presented in section 2. It is also important to understand the boundaries of a system as the system itself is often affected by changes that occur outside making it necessary to clearly decide and define the boundaries between a system and its environment. These boundaries obviously depend on the purpose of the study in hands. This also concerns the modeling as a model that is a representation of an actual system thus raising a concern about the boundaries of the system.

There are several entities which compose a system that must be addressed such as:

- **State Variables** – can be easily understood as being the information that is needed to understand and define what is happening within a system at a given time (the state of the system). These variables vary from time to time and from different situations in

the same system, this meaning that for the same system the state variables can be different regarding the purpose of the investigation at hands.

- **Events** – are the occurrences that are responsible for creating changes in the system. These can be internal and external to the system (*endogenous or exogenous* respectively);
- **Entities and attributes** – an entity is an object integrated in the system and it can be dynamic when it moves along within the system or static if it serves other entities. If we consider a bank, a customer is a dynamic entity whilst a teller is a static entity. The term attribute is pretty self-explanatory as it refers to certain characteristics of the entities. If we think of another entity within a production system, like a machine, its attributes can be its speed, capacity or availability;
- **Resources** – are entities responsible for providing services to dynamic entities travelling within the system. A resource can provide one or more entity at the same time and an entity can also request more than one unit of resource at a time. There are many possible states for a resource *idle, busy, failed, blocked* or *starved;*
- **Operation** – a step carried by an entity that is moving through the system.

3.1.1. ENTITY STATES

As entities work their way through the system, their states will change thus meaning that they will always be in one of the possible states detailed below [18]:

- **Active** – refers to the state of the current moving entity in the system. It progresses through the system until it encounters a delay at which time it changes to another alternative state so that another entity can become the active entity;
- **Ready** – defines that one or more entities are ready for processing. This means that whenever there are entities waiting to start moving through the system, they are in the ready state. Considering that although several entities may be in the ready state they can only move (thus changing to active state) one by one;
- **Condition Delay** – is the state of entities waiting until it is their turn to use a machine.

3.1.2. DISCRETE AND CONTINUOUS SYSTEMS

Systems could be categorized of two types, discrete and continuous. In a discrete system the changes occur instantaneously and at separate points in time, meaning the variable

changes are predominantly discrete. A bank is the most common and easily understood example of a discrete system as changes occur only when a customer arrives or leaves after being served (see Figure 13).

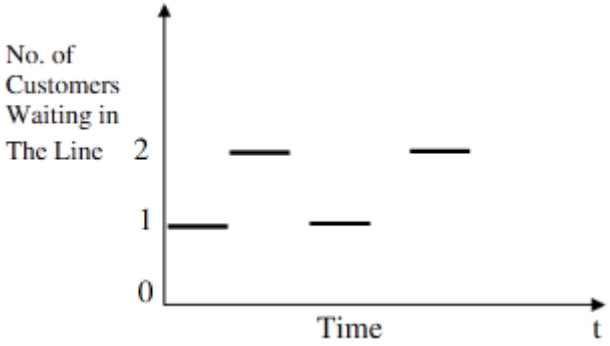


Figure 13 Example of the changes over time in a discrete system (bank) [19]

A continuous system as the name stands is a system in which the changes occur continuously throughout time and the changes are predominantly smooth. A bird flying is an example of a continuous system as its variables such as position, altitude[5], speed can change continuously (see Figure 14).

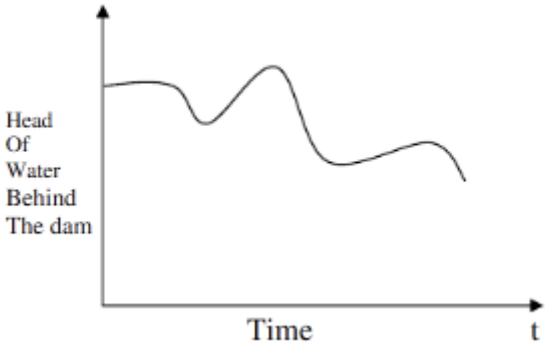


Figure 14 Example of a continuous system. A bird changes its altitude [19]

3.1.3. WAYS TO STUDY A SYSTEM

It is possible to identify several approaches and ways to study a system. Simulation could not be the appropriate tool to study a system as will be addressed further in section 3.4.2.

[5] As a general definition, altitude is a distance measurement, usually in the vertical or "up" direction, between a reference axis and a point or object.

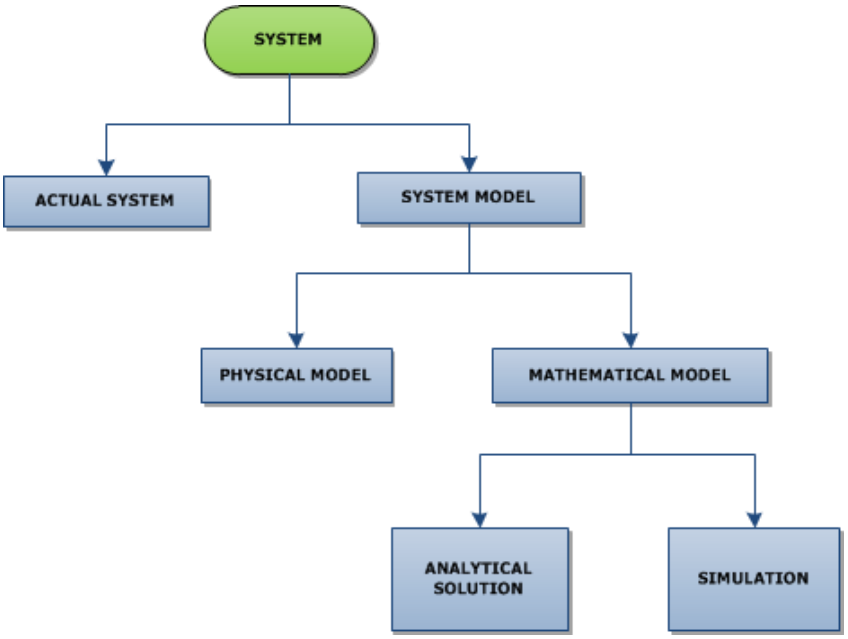


Figure 15 Diagram of the possible forms to study a system (adapted from [20])

It must be understood that if the experiments should take place using the current system or a model of the current system. It may be cost effective to adapt the system itself physically to understand its behavior under the new conditions, but this is usually true for small or simple systems. So, whenever a situation arises when stopping the system to implement changes or the implementation itself is not cost effective, modeling the current system is the feasible way to approach the problem. This carries a problem which is the question of whether the model is an accurate representation of the system or not, that will be addressed on section 3.6.

When it comes to modeling the system, there are two ways it can be done, creating a physical or a mathematical model of the system. In the past, people would associate models with physical representations of the system, plane cockpits to simulate flights, cars in wind tunnels, a solar system mini-model, a skyscraper model for earthquake simulation and many other applications usually related to real life events. With the advance of technology, it has been found useful to build physical scaled models of material-handling systems to study engineering or management systems for instance. Most of the models built for these purposes are mathematical and are used in the natural sciences and engineering disciplines (biology, physics, and electrical engineering for example) and also in social sciences (economics, sociology, political sciences). If the mathematical model is valid (section 3.6) the logical and quantitative relations expressed by the model will

accurately allow predicting and simulating on how the system would react to the changes and manipulations reflected on the model.

3.2. MODELING WORLD-VIEWS

Simulation models are conceived using one or more conceptual frameworks (that later started being referred to as world views) that provide the needed tools for defining the system of interest in enough detail to simulate its behavior. There are three distinct world views used over simulation's history (*event, process* and *object*) but a few more have appeared over the years (*agent*-based and *multi*-paradigm) although the basic ideas for the initial views have not changed.

Event modeling - In event modeling the modeler defines the system as a series of instantaneous events changing the system's state over the time. Modeling such an activity that evolves over time makes it crucial that separate events are created to define the start and end of the system activity and the simulated time cannot advance within an event. Although being very flexible and efficient it is also a relatively abstract representation of the system thus many people find this event orientation modeling to be a difficult task.

Process modeling – in process view the modeler describes the system as the movement of entities through the system as process flow described by various process steps (e.g. start, delay, release) that are responsible for modeling the state changes taking place in the system. So, unlike in event modeling, a sequence of events can be executed over time.

Object modeling – This is considered to be one of the easiest approaches in most cases. In object modeling, the system is modeled by describing the objects that build up the system (e.g. the workers, machines, conveyors, robots, stacks in a factory plant). The system behavior emerges from the interaction between the objects. Object based simulation and the definition of objects will be described further on section 4.1.2.

Agent-based modeling – software agents[6] are one of the fastest growing areas of Information Technology (IT), therefore the term agent-based modeling (ABM) has become widely used in recent years [21]. The basic idea behind the concept is to determine and

[6] An agent is a discrete entity with its own goals and behaviors (e.g. people, groups, organization, robots or systems of cooperating robots.

placing the agents in a system representation and let them interact with each other thus evolving the system state. These agents individually assess its situation and makes decisions on the basis of a set of rules that will determine the relationships between them.

Multi-paradigm modeling – this applies to tools where multiple modeling approaches can be used to best resolve the problem at hand [21]. The idea of mixing alternative modeling approaches in a single model was first promoted by SLAM (Simulation Language for Alternative Modeling) where processed and object modeling were mixed. In modern day approaches, and since when event modeling or its logic is incorporated into objects is somewhat restrictive due to the simulated time not being able to advance within the events (e.g. it is not possible to wait for a machine to become available within an event as it would require time to advance) a more powerful approach is to combine objects with processes instead. This makes it possible to extend the behavior of the objects since processes can span time, hence processes can be embedded within and object and wait for a specific time or condition (e.g. a robot being available).

3.3. COMPONENTS OF A DISCRETE-EVENT MODEL

In the remainder of this document we will mostly address discrete, dynamic and stochastic models henceforth called as *discrete-event simulation (*DES) *models*. They will be addressed further on and detailed so we will refrain from it now and will focus in the components of such models.

The following components could be found in most discrete-event simulation models [18]:

- **System State*:*** The collection of state variables necessary to describe the system at a particular time;
- **Simulation clock*:*** a variable given the current value of simulated time;
- **Event list*:*** a list containing the next time when each type of event will occur;
- **Statistical counters:** variables used for storing statistical information about system performance;
- **Initialization routine:** A subprogram to initialize the simulation model at time 0;
- **Timing routine:** A subprogram that determines the next event from the event list and the advances the simulation clock to the time when that event is to occur;
- **Event routine:** A subprogram that updates the system state when a particular type of event occurs;

- **Library routines:** A set of subprograms used to generate random observations from probability distributions that were determined as part of the simulation model;
- **Report generator:** A subprogram that computes estimates (from the statistical counters) of the desired measures of performance and produces a report when the simulation ends;
- **Main program:** a subprogram that invokes the timing routine to determine the next event and then transfers control to the corresponding event routine to update the system state appropriately. The main program may also check for termination and invoke the report generator when the simulation is over.

3.4. NATURE OF SIMULATION

Computer simulation history dates back to the early 40's while the World War II was taking place [22]. As of today, simulation has gained its importance and dimension throughout the years becoming, very important to organizations regardless of their industries. Robert E. Shannon [23] defines simulation as "*the process of designing a model of a real system and conducting experiments with its model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system*."

Considering the technological evolution, the advances reached in both hardware and software translated in to the present simulation systems being more powerful and economical. That, along with the increasingly availability and accessibility of the simulation resources and tools has largely contributed for the widespread of simulation. Simulation is theoretically a reliable and low cost analysis tool for systems, namely manufacturing systems.

Regardless of the industries, simulation has a widespread of major areas of application [24] such as:

- Manufacturing Systems;
- Public Systems (i.e. Healthcare, Military);
- Transportation Systems;
- Construction Systems;
- Process Re-engineering;
- Food Processing.

However, it should be very clear that simulation is not always guaranteed to be the best solution of tool to analyze and solve problems as there are numerous situations where simulation would not be fit for use.

3.4.1. THE SIMULATION PROCESS

When simulation is due to be applied there are several specific steps that must be followed in its correct order, regardless of what type of problem or the objective of the analysis as is depicted in Figure 16. The following steps should be applied in any simulation for its study to be successful [25]:

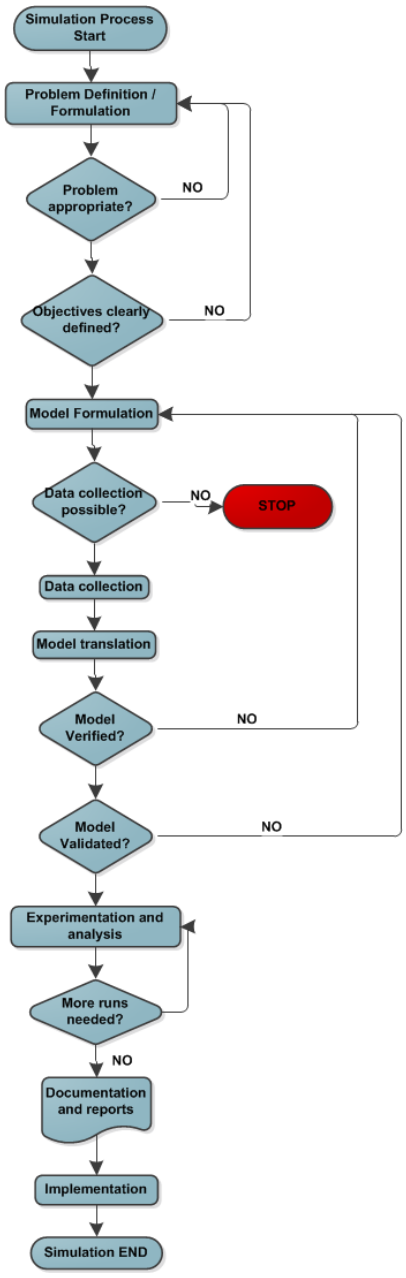


Figure 16 Set of steps and their order for the application of simulation (adapted from [16])

- **Problem Definition/Formulation** – The system should be defined and the problem identified as clearly as possible. The goals of the analysis that will take place should be clear so that it is understandable why the problem is being studied and what questions should be stated. A clear definition will also allow to verify if simulation is indeed the appropriate tool for the problem under investigation. If the problem is not appropriate to be solved by recurring to simulation or the objectives are not clearly defined, the process should go back to square one and re-define the problem.
- **Model Formulation** – the step of model formulation basically means that there should be a clear understanding on how the system works and behaves not forgetting the basic requirements of the model. The components, variables and logic interactions that constitute the system must be defined.
- **Data Collection** – the first step after the model being formulated is to collect input data for the analysis. Before any further efforts, the availability of data to collect should be verified so it is possible to stop the process at this time without any waste of time and resources if no data is available. If data collection is possible, all data needed by the model should then be identified, collected and further fitted to theoretical distributions.
- **Model Translation** – The model translation step implies that the model is translated into the appropriate programming or simulation language.
- **Experimentation and analysis** – Only after the model is verified and validated (see section 3.6) the experimentation can take place. Experimentation means that the desired information for the simulation is defined so as how the test runs will be executed (simulation length, number of replications, etc.) making it possible to execute the simulation model and thus generate the data needed for the posterior analysis. If possible, alternative models should be available so it will be possible to compare the simulation run with alternative system performance with the real system.
- **Documentation and reports** – The results of the various simulation runs should be observed, analyzed and documented as well as documentation of the model and its use for better understanding of common users and for further uses.
- **Implementation** – Put the results to use and perform the proposed changes as an outcome of the simulation.

It is of extreme importance that a simulation study completes all the required steps increasing the likelihood of its success. Also, it is of same importance to be aware that simulation may not be the best approach to the solution of each and all problems.

3.4.2. SIMULATION AS THE APPROPRIATE TOOL

Even though simulation costs have been reduced throughout the years, it is not always guaranteed that simulation is the best tool for the problem. Simulation should be used for certain when a new system is being designed and significant financial resources will be expended. Currently, most new facilities run into the millions or billions in investments, constructions and equipment costs, so if by using simulation an organization can save as little as a piece of equipment, for instance, the simulation costs will have paid for itself.

If, at times, problems can be solved analytically there might be several other occasions where simulation is the proper tool to verify those solutions. When these problems involve sudden or unexpected changes, simulation is also the tool of choice to prevent any problems by predicting different scenarios thus allowing manufacturers to be prepared beforehand for any eventual changes that might occur in the system.

But most importantly, one should be aware that simulation may not be the appropriate tool. Banks and Gibson [26] in their article, present a set of rules that should be considered for evaluating when simulation is not the appropriate tool:

1 – The problem can be solved using “common sense analysis”;
2 – The problem can be solved analytically;
3 – It is easier to change or perform direct experiments on the real system;
4 – The cost of simulation exceeds possible savings;
5 – There are not proper resources available for the project;
6 – There is not enough time for the results to be useful;
7 – There is no data (or estimates for that matter);
8 – The model cannot be verified or validated;
9 – Project expectations cannot be met;
10 – The system behavior is far too complex or cannot be correctly defined.

3.4.3. SIMULATION TAXONOMY

After a mathematical model has been developed it must be examined either analytically or recurring to simulation. If the system and thus the model are simple enough, it may be possible to achieve a solution analytically. Many systems, however, are too complex to be studied with a paper and pencil meaning the model must be studied by means of simulation to see how the changes will affect it. Simulation should not be seen as a "last resort" of any kind as simulation is used in most simulations due to the complexity of most models which tends to keep on increasing.

When it comes to simulation models there are different dimensions to classify them (see Figure 17) as they have different characteristics and purposes [27]:

- **Static vs Dynamic models:** A static simulation model is used to represent a system in which time plays no role meaning it does not account the element of time as it represents the system at a determined instant of time. On the other hand a dynamic model does take time in consideration and the way the system evolves over time.
- **Deterministic vs Stochastic models:** A deterministic model does not include any randomness. Its variable states are determined solely by the initial conditions given, meaning that the outputs will be determined whenever the inputs and relations are set. On the other hand, randomness is present in stochastic models as the variable states are not described by unique values. The outputs produced are random and described by probability distributions meaning they must be treated as estimation of the model characteristics.
- **Continuous vs Discrete models**: Differences between continuous and discrete systems have been addressed previously in sub-section 3.1.2. When referring to continuous and discrete models these can be somewhat described analogously to how the systems presented previously were described previously. As discrete event simulation is traditionally used for industrial applications, namely in the manufacturing sector, it will be addressed further in section 3.5.

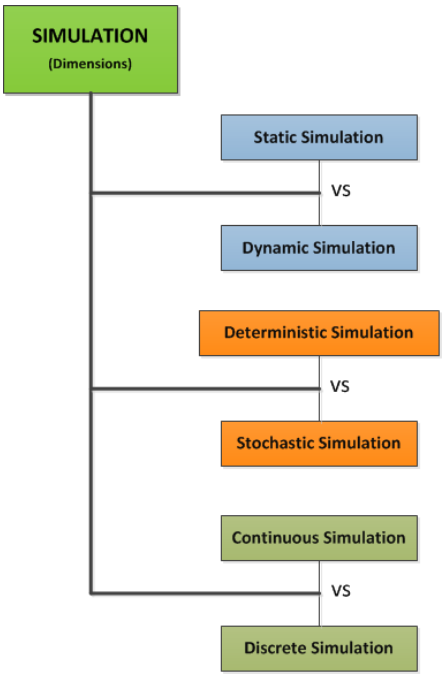


Figure 17 The different dimensions to classify simulation models

3.5. DISCRETE-EVENT SIMULATION

The lines between different types of simulation are becoming less and less distinct. In classical thinking there are three different types of simulation: discrete-event, continuous and Monte Carlo. Nance [28] has stated some clear definitions of them:

- Discrete event simulation utilizes a mathematical/logical model of a physical system that describes state changes at precise points in simulated time. Both the nature of the state change and the time at which the change occurs mandate precise description. Customers waiting for a service, the management of inventory and military combat are typical domains of discrete event simulation.
- Continuous simulation uses equation models, often of physical systems, which do not describe precise time and state relationships that result in discontinuities. The objective of studies using such models does not require the explicit representation of state and time relationships. Examples of such systems are found in ecological modeling, ballistic reentry, or large scale economic models.
- Monte Carlo simulation, the name given by Nicholas Metropolis and Stanislaw M. Ulam [29] to reflect its gambling similarity, utilizes models of uncertainty where representation of time is unnecessary. The term originally attributed to "a situation in which a difficult non-probabilistic problem is solved through the invention of a stochastic process that satisfies the relations of the deterministic problem". A more

recent characterization is that Monte Carlo is "the method of repetitive trials. Typical of Monte Carlo simulation is the approximation of a defined integral by circumscribing the region with a known geometric shape, then generating random points to estimate the area of the region through the proportion of points falling within the region boundaries.

Discrete event simulation has been extensively used by global industries throughout the years and is thought to still be one of the most effective decision support tools available.

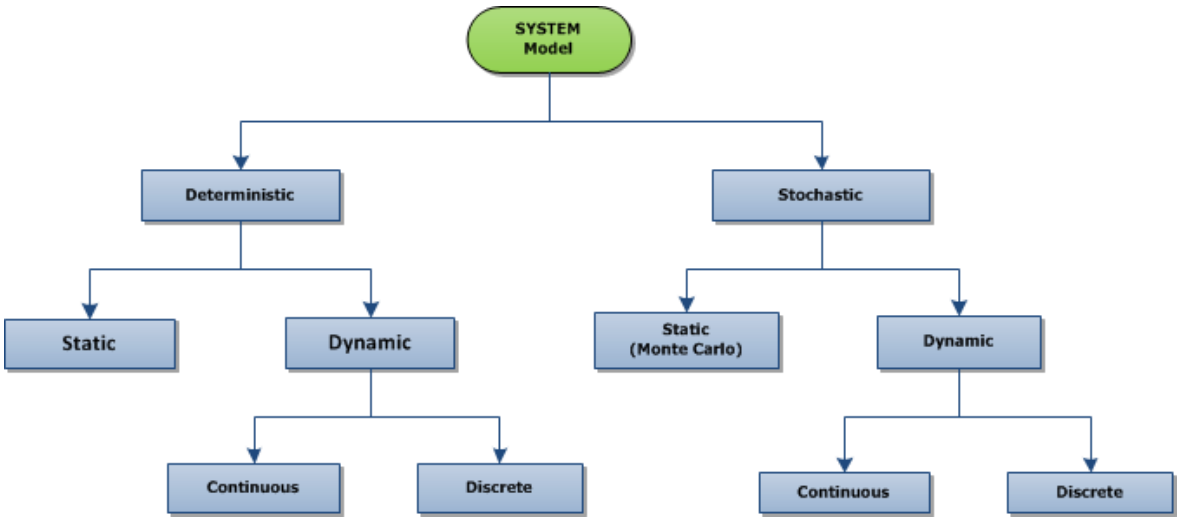


Figure 18 System model taxonomy [30]

Discrete-event simulation models are defined by three attributes: stochastic, dynamic and discrete as is depicted in Figure 18. These attributes are described in sub-section 3.4.3

3.5.1. DISCRETE EVENT SIMULATION IN MANUFACTURING

Since the late 1980s the application of discrete event simulation has become very common and some literature even refers to it as “standard”. (DES) Discrete event simulation have been supporting organization managers to cope and keep up with the technology advances and also making the right decisions. Through the decades of 1980 and 1990 there was an abrupt and rapid develop in what concerns the advanced manufacturing technology such as CAD, CAM, CIM, AGV (Automatic Guided Vehicle), FMS and others, and the use of DES allowed them to evaluate their processes and systems with different sets of conditions and proceed with analysis of the “what-if” scenarios to identify better system configurations and operational policies, thus helping evaluate the benefits of new technology available. DES has been used in the following areas [31]:

- Design and evaluation of manufacturing processes;

- Performance improvement of existing manufacturing processes;
- Optimization of the operational policies;
- Algorithms to support production planning and scheduling.

Along with the rapid development of manufacturing technology and China emerging as the world base for manufacturing, many major organizations chose to take their production facilities to third world countries due to their low-wages which resulted in the rise of newer challenges to the DES technologies. The fact that the manufacturing was now done worldwide in a global supply-chain means the existence of complete relationships among all parts involved. This globalization also created a new challenge for the DES technologies with the hybrid manufacturing systems. These systems include a mix of advanced machinery and abundant work force creating a problem of low flexibility when compared to the automated manufacturing system and, besides, it generates a bigger problem since there is a need to model the human behavior and performance under the various variables and working conditions. The DES software finds it problematic to handle the modeling of human behavior meaning this is an important challenge that must be overcome to assure the continuity of DES applications.

3.5.2. OPPORTUNITIES FOR DISCRETE EVENT SIMULATION

The changes and developments in both technology and business environments created opportunities for the DES technologies. Wang, Sun and Nooh [31] highlight two particular areas where DES has major opportunities:

- Business Intelligence (BI) systems;
- Simulation-based education.

Organizations have spent great amount or resources in implementing enterprise resource planning (ERP[7]) systems in the late 1990s which contributed to a great amount of data archived making it easier to collect when driving a simulation model. Now with solid IT infrastructures, companies focused their efforts towards business improvements with better decisions, making it crucial to consider how to effectively deploy resources. This created a

[7] ERP is business management software that allows an organization to use a system of integrated applications to manage the business. ERP software integrates all facets of an operation, including development, manufacturing, sales and marketing.

great opportunity for DES as it is and ideal platform to support those decisions with easier access to essential and important data, easily available at the ERPs.

Also, the outsourcing of production from companies to a global supply-chain faced a sustainability problem to perform in-house engineering and guarantee skilled employees. This fact created a challenge to prevent the erosion of the engineering knowledge and expertise one that could prove that DES is again an ideal platform for creating a simulated manufacturing environment for the work force to experience and learn more about the manufacturing operations.

3.6. VERIFICATION AND VALIDATION

By verifying the model it is intended to understand if it was built correctly and accordingly to the real system. It is important to investigate if the model and simulation can fulfill all the specifications intended so that both purposes (design and implementation) meet all the requirements. Practically it means to verify if the conceptual model of the system was correctly translated in to the simulation model and, thus, to the simulation software or programming language.

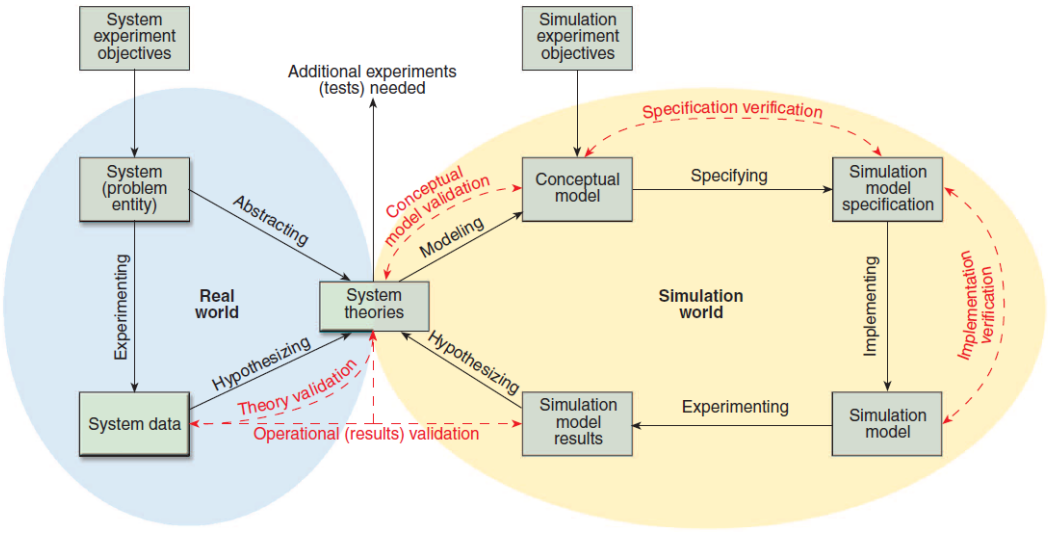


Figure 19 The process of verification and validation of a model and simulation [32]

Validation consists in investigating if the model or simulation will be able to cope with its intended use and were correctly built. In other words it is intended to determine if the model is an accurate representation of the system always minding the objectives of the study at hand, besides, the results should also be validated by being compared with

appropriate references to make it possible to prove and demonstrate that the model can actually fully support its expected use.

Figure 19 accurately describes the several stages in the verification and validation of a model and its simulation. It identifies the main activities and the relationships existing in both real-world and simulation environments. More information about verification and validation can be found in Dr. Robert Sargent's work [33].

Two other concepts should be mentioned when referring to the process of verification and validation: **accreditation** and **credibility**. Accreditation tries to evaluate if a model or its simulation results should be used by determining if it meets all the needed requirements. Credibility refers to the results of the simulation and the acceptance of these results as correct by those responsible, usually managers.

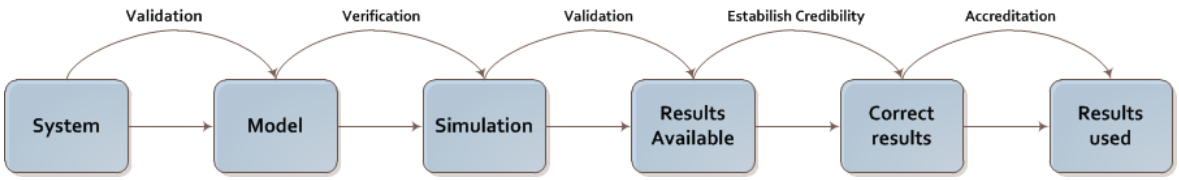


Figure 20 Steps in the process of verification, validation, establishing credibility and accreditation of a model and simulation (Adapted from [18])

The relation of model validation and verification to a simplified version of the modeling process is depicted in Figure 20. The first step consists in translating the system in to its equivalent model, thus requiring the first validation step, which is intended to determine if the theories and assumptions underlying the conceptual model are correct. After, ensuring that the implementation of the model is correct translates in to the verification of the model thus allowing for the simulation operations to take place. By determining that the model's output behavior has sufficient accuracy for the intended purposes the results become available. By accepting these results, thus establishing its credibility, the correct results will be available and plausibly used after its accreditation by evaluating if they should be used or not.

3.7. ADVANTAGES AND DISADVANTAGES OF SIMULATION

It is possible to identify advantages and disadvantages from simulation. Several authors have mentioned them in their work [11, 18] so both advantages and disadvantages from

recurring to simulation can be grouped and are explained thoroughly below. As advantages it is possible to refer the following [18]:

- **No disruption of current system –** by being able to simulate the operation procedures, decision rules, information flows and such, it is possible to explore all these aspects and their changes to the system without the need to disrupt any operations on the real system;
- **Chose careful and correctly –** again, simulation allows testing of every change or addition to the system without the need or compromise to acquire the resources needed for those changes;
- **Feasibility –** all hypotheses on how or why a certain change can affect the system can be easily tested for feasibility;
- **"What if" and "Why" –** these common answers can be easily explored answered and explained. With simulation a certain scene or aspect of the system can be careful and thoroughly examined to determine why certain phenomenon occurs thus being able to understand the "what if" of the question at hands;
- **Correct understanding of the system –** a simulation can help to understand how the system actually works instead of how everyone thinks it really operates;
- **Bottleneck analysis –** these can be easily performed so that the delays throughout the system can be calculated, indicated and possibly corrected. This is fairly important as people tend to forget that bottlenecks are an effect rather than a cause and they sure give manufacturers headaches;
- **Preparation for the future –** Future brings change. Being able to answer the so called "what if" questions is a good way to prepare for possible future changes on the system. With simulation, a model of a future system (or the actual system in the future) makes it easy to access and study alternative scenarios;
- **Reducing costs and expenses –** On several cases, changes or modifications to a system after they have been installed are costly and expensive. Simulation is a wise investment assuming it can guarantee that the correct changes will be made, if necessary;
- **Requirements specification –** by simulating different scenarios, different and specific requirements can be achieved for the machines;
- **Training –** last but not least, simulation can be a great tool for training teams of operators. Operator Training Simulator (OTS) are digital simulation systems utilizing

the powerful calculating capability of computer to imitate the operation of power system, for training operators and are a great example of its advantages.

As for the disadvantages it is possible to identify the following:

- **Special Training –** building the models requires extensive and special training;
- **Time consuming –** the modeling and analysis phase can be exhausting, time consuming and expensive;
- **Hard to comprehend –** the results form simulation can be hard to understand and interpret;
- **Misused –** simulation might be inappropriately used when other tools would have worked better or easier.

3.8. POTENTIAL PITFALLS

Many times, in the process of simulation, there is an emphasis in the programming stage with little consideration regarding the current model. This may create some validity issues with the model thus resulting in a misuse of the simulation. Also, there are other common mistakes that lead to simulation results not being accurate or used in the decision-making process as they should, mainly due to the fact that the modeling stage is neglected.

These common errors are the so called pitfalls and they can be noticed in almost every different aspect of a typical simulation study. Although simulation does make perfect sense for those who are familiar with it, the process of introducing it into an organization is somehow challenging in most occasions and will create some potential pitfalls[18]:

- Lack of management support;
- Underestimation of the model development needs;
- Inappropriate addressing of the problems.

The support from the management and the clear understanding of the need associated with simulation is critical for the simulation initiatives to work and be worth. It is therefore crucial to try and direct efforts into educating all the teams that will somehow be connected to the simulation initiatives as it will play an important role in guaranteeing that the efforts will be pursued effectively. Also, as mentioned before, the process of simulation starts with a clear definition and formulation of the problem and its objectives. Therefore, an inappropriate addressing of the problem is also a common pitfall in the simulation

initiatives so if the objectives of the models and the different scenarios for the tests in the simulations are not clearly communicated and understood by the management, the simulation initiatives should not take place. If these information are correctly assimilated it will be easier for the organizations to avoid underestimating the time and efforts needed to complete a simulation project.

Law and McComas [34] go further deep in the matter of the pitfalls in simulation and discuss ten of them in if different aspects of the simulation process. These pitfalls will be mentioned next but further reading of the original paper is advised for a better understanding:

- **Simulation Modeling**
  - Failure to have a well-defined set of objectives;
  - Treating a simulation study as a programming exercise;
  - Failure to communicate with management on a regular basis;
- **Simulation Software**
  - Software which makes simulation accessible by "anyone";
  - Misuse of animation;
- **Modeling the randomness in a manufacturing system**
  - Replacing distributions by their means;
  - Incorrect choice of input probability distributions;
  - Incorrect modeling of machine break-downs;
  - Making only one simulation run for a particular system;
  - Failure to Warm-Up a simulation model.

3.9. SIMULATION IN MANUFACTURING SYSTEMS

From all the areas of application, manufacturing systems can be identified as the most relevant. Law and Kelton [18] state some reasons for this:

- Increased competition in many industries – resulting in greater emphasis on automation to improve productivity and quality, meaning automated systems are more complex and can only be analyzed by simulation;
- The decrease of the cost of computing due to the reduction in prizes of PCs;
- Simulation software now allow more analysis as the model development time has been dramatically decreased;

- Better understanding of the simulation resulting in greater understanding of the outputs and results by non-simulator expertise's.

Besides, there are several benefits and issues that must be addressed when concerning this subject [35]:

- **The need for and the quantity of equipment and personnel**
  - Number and type of machines for a particular objective;
  - Number, type, and physical arrangement of transporters, conveyors, and other support equipment (e.g., pallets and fixtures);
  - Location and size of inventory buffers;
  - Evaluation of a change in product volume or mix;
  - Evaluation of the effect of a new piece of equipment on an existing manufacturing system;
  - Evaluation of capital investments;
  - Labor-requirements planning;
- **Performance evaluation**
  - Throughput analysis;
  - Time-in-system analysis;
  - Bottleneck analysis;
- **Evaluation of operational procedures**
  - Production scheduling;
  - Inventory policies;
  - Control strategies [e.g., for an AGV System];
  - Reliability analysis (e.g., effect of preventive maintenance);
  - Quality-control policies;
  - Throughput;
  - Time in system for parts;
  - Times parts spend in queues;
  - Queue sizes;
  - Timeliness of deliveries;
  - Utilization of equipment or personnel.

There are several indicators used as measures of performance for the study of manufacturing systems obtained as a result of simulations. These are listed below and could allow a better understanding of the benefits that urge from using simulation in these systems [11]:

- Throughput;
- Time in system for parts (cycle time);
- Times parts spend in queues;
- Times parts spend waiting for transport;
- Times parts spend in transport;
- Timeliness of deliveries (late orders);
- Sizes of in-process inventories;
- Utilization of equipment and personnel (busy time);
- Proportions of time a machine is broke, waiting for parts, blocked or under maintenance;
- Proportions of parts revoked or scrapped.

As a result of the use of simulation in manufacturing, there are potential benefits including:

- Increased throughput;
- Decreased times in system of parts;
- Reduced in-process inventories of parts;
- Increased utilizations of machines or workers;
- Increased on-time deliveries of products to customers;
- Reduced capital requirements or operating expenses;
- Insuring that a proposed system design will operate as expected;
- Better understand of the system;
- More attention to the detail and focus on potential problems that might otherwise be missed.

There has been a major increase in automation usage in manufacturing systems with robots performing many tasks and also the implementation of automated material handling systems, which translate in significant capital costs. Also, many trends in manufacturing methods appeared such as *JIT*, *KanBan*, *MRP*, *flexible work cells*. Simulation in manufacturing can be used to test new equipment before purchasing it and also, to test the implementation of changes or new methods in their operations free of risk.

3.9.1. IMPORTANCE OF SIMULATION IN MANUFACTURING

The first and probably the one benefit that matters the most for general audiences is the business and economic benefits. Although being difficult to ascertain, there are several studies from companies demonstrating the economic potential of the use of simulation in manufacturing industries inclusively estimating paybacks and payback periods. This potential is largely obtained with the ability to reduce the costs to train employees and workers (namely replacing the need to acquire OTS that are often the first choice). Also, by using simulation, organizations can, most of the times, effectively reduce damages and stoppage times to the machines and also increase the availability of the entire system.

Regarding the OTS, these simulators provide an effective method to train new operators by providing a better understanding of the process and system they are to integrate. This applies not only to new operators but also to the existing work force as today's market conditions create constant changes in the system operations that the workers must be able to understand and adapt so, being able to cope with that and reduce or remove the skill gap of the workforce is a strong benefit. Taking in consideration today's rigid and strict business demands, it is far more important than ever to guarantee a well-trained workforce.

Furthermore, it is also important to refer to the role of the simulation systems in the increasingly use and implementation of automation systems. More and more industries and companies, in general, are somehow trying to automate their systems or totally focus their operations on automation. Simulation has gained a major role as the testing capability provided by it helps to identify and avoid errors or flaws early in the project cycle before it has been propagated throughout the entire system. In addition, as systems change through time, keeping the simulation models up-to-date will allow companies to effectively test their systems before any changes are made.

3.10. SIMULATION MODELING EXAMPLES

Some actual modeling examples will be presented to provide a better "real-world" view of what modeling a system could be.

Some simulation and modeling examples referred in literature will be described to illustrate simulation topics. These examples usually include Queuing Systems, Inventory Systems, Time Sharing, Banks, Call-Centers, Job-shops. Other examples can also be found

but these are the most commonly referred to. In this document we will refer to a drive-through system and a bank as examples.

3.10.1. DRIVE-THROUGH EXAMPLE

First example to be addressed is an adaptation of the example described by Ingalls of a drive-through at a fast-food restaurant [36]. This was chosen as almost any one will be able to identify and relate to such situation.

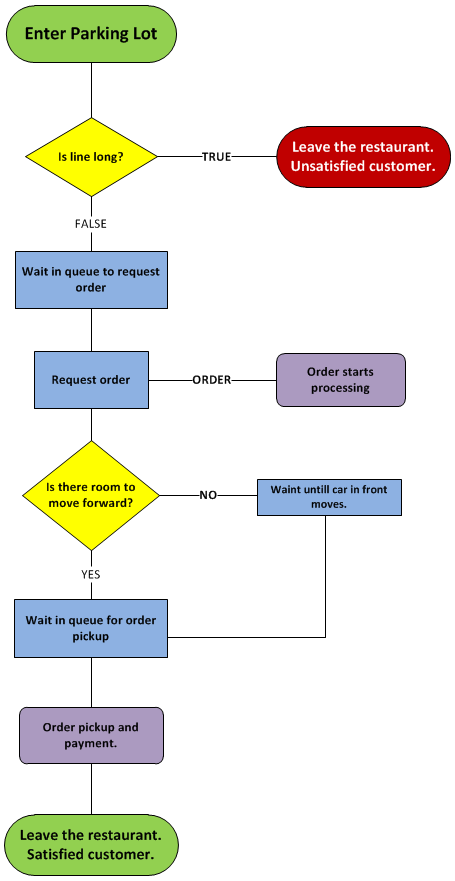


Figure 21 Drive-through example flowchart

Figure 21 describes the logic flowchart for the drive-through system. As all discrete-event simulations systems it contains the basic components referred in section 3.1:

- **Entities**: cars joining the drive-through, orders done by cars;
- **Attributes**: time of the day a car enters the parking lot of the restaurant, money values of the orders;
- **Activities and events**: delays when ordering, on order processing, on order pick up and queues when waiting for order, for order processing, for order pick up;
- **Resources:** ordering station (also referred to as *Menu Board*), kitchen where orders are processed, order pick up window;

By listing these components most of the preparation needed to start building the simulation is now done. Throughout the simulation of this model it is intended to keep track of the state of the system, the entities on the *calendar*, the values of the attributes and state-variables and the generated statistics. A *calendar* in a simulation is a list of pre-scheduled events that will occur in the future time.

As a discrete-event simulation the simulation run will move from event to event (e.g. the arrival of the car to the restaurant parking lot) instead of cycling through time intervals where possibly nothing would happen to affect the system.

A more thorough analysis of the example [36] is advised for a complete walkthrough.

3.10.2. BANKING SYSTEM EXAMPLE

Another system modeling example is a model of a one teller bank [37]. The purpose of the model is to estimate the percent of time the teller is idle as well as the average time a customer spends in the system.

The status of the system depends entirely on the status of the teller which can vary from idle to busy and the number of customers in the system. Figure 22 illustrates the activity on the model, first, a customer is arriving (1) as a new entity to the system and it will join other entities already waiting on queue (2). If the state of the system and thus, the state of the teller, changes from busy to idle, the next entity on queue is removed from it and a service activity is initiated in the system again changing the system state back to busy. The time spent by the customers in the system is collected throughout the activity.

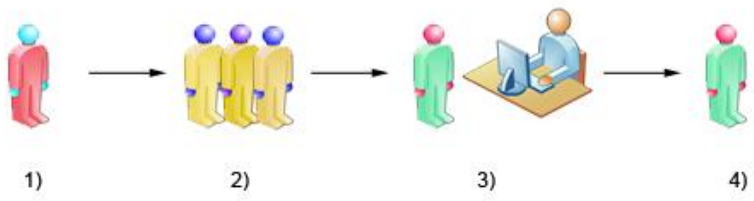


Figure 22 Diagram of the banking system (adapted from [37])

It is of crucial importance to define and determine the significant events in which a model is based. In this example the status of the system changes when two crucial events happen:

- the arrival of a customer;
- the end of the service at the teller.

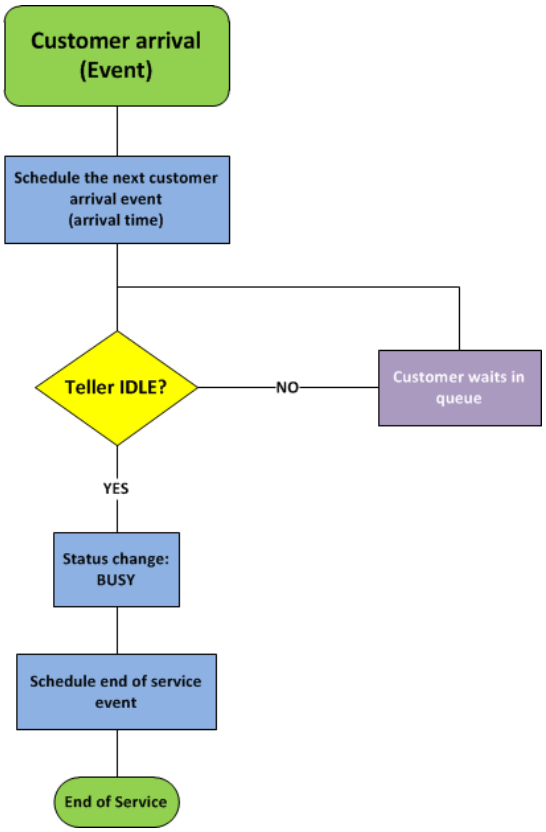


Figure 23 Bank customer arrival event logic flowchart, adapted from [37]

The logic for the customer arrival event starts with the arrival of a customer which triggers the scheduling for the next arrival event as is depicted in Figure 23. The customer then checks if the teller state is idle, if not he will join and wait in queue, if so the teller and system status is changed to busy so the service event can start and its end is also scheduled. At the end of the service the time the customer spent in the system is collected. If no customers are waiting in queue the status is set to idle whilst if there are customers in queue the first one waiting is removed and the end of service event is finally scheduled as is depicted in the flowchart in Figure 24.

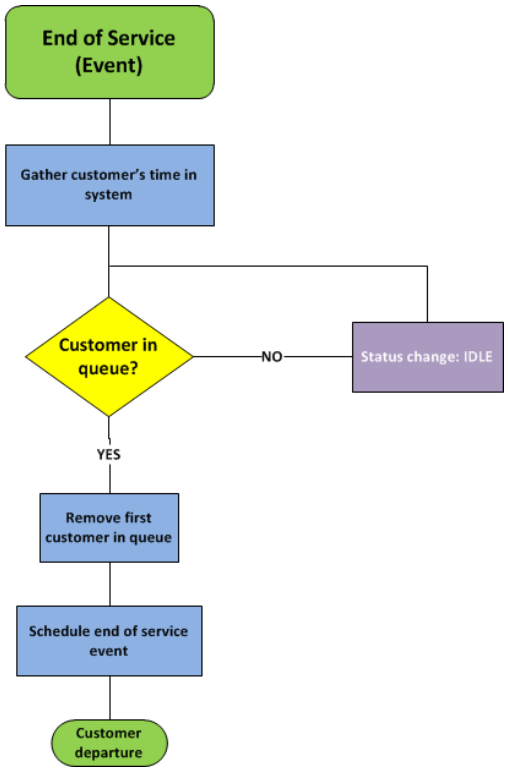


Figure 24 End of service event logic flowchart. , adapted from [37]

3.11. CONCLUSION

As it is important to understand the boundaries of a system, the various modeling concepts which compose a system were detailed.

Although discrete system will be addressed the most throughout the present document, it was important to specify the difference between discrete and continuous systems for a full understanding. A brief introduction on simulation's history and path was presented as well as the crucial steps composing the simulation process.

Regarding simulation models, there are different dimensions to classify them, these were detailed for a better understanding of their characteristics and purposes. As approach is present, concerning the advantages, disadvantages and pitfalls of simulation in general and, concerning its particular application to manufacturing systems.

Lastly, two examples of system modeling are presented for a better understanding and consolidation of the concepts addressed previously in the chapter.

4. SIMULATION: PARADIGMS, LANGUAGES AND PACKAGES

There has been a lot of work and development on simulation software and packages. This has become a very profitable area of interest for both users and developers, therefore a lot of effort has been taken upon developing better and more complete tools for simulation to fulfill all the different needs of all the areas where the use of simulation is intended.

We will focus on the concept and considerations of simulation software as well as its selection principles. Moreover an overview of some of the key features and differences between some of the mostly used simulation software will be described.

The concepts and fundamentals of object oriented simulation are presented thus supporting the decision of choosing Java as the programming language due to its understandability, easy maintenance and higher degree of reuse and portability [38].

4.1. SIMULATION PARADIGMS

Different simulation paradigms can be identified when inspecting the literature on simulation i.e. message-oriented, queue-oriented, event, activity or process-oriented [38]. As a paradigm, it is a way of representing problems and thinking about them as much as a solution method.

With object-orientation becoming very popular in the last years, as a powerful approach for system development, simulation environments based on the so-called object-oriented simulation paradigm have appeared [38].

4.1.1. DISCRETE EVENT SIMULATION

Discrete event models have a static and dynamic structure, consisting of entities interacting with each other. The static structure is responsible for defining the entities' so called physic states whereas the dynamic structure represents the changes that occur over the entities throughout time. Therefore, their behavior is normally defined in terms of the events and the subsequent reactions to such events The different approaches to describe these events are the model views previously addressed in section 3.2.

Copstein et. al [38] provide a good example on the application of different simulation paradigms to the same model.

4.1.2. OBJECT-ORIENTED SIMULATION

Computers are highly capable machines, able to execute high scale operations at very high speeds. However, they must be instructed to do so as they are not (still) capable of choosing which operations to perform.

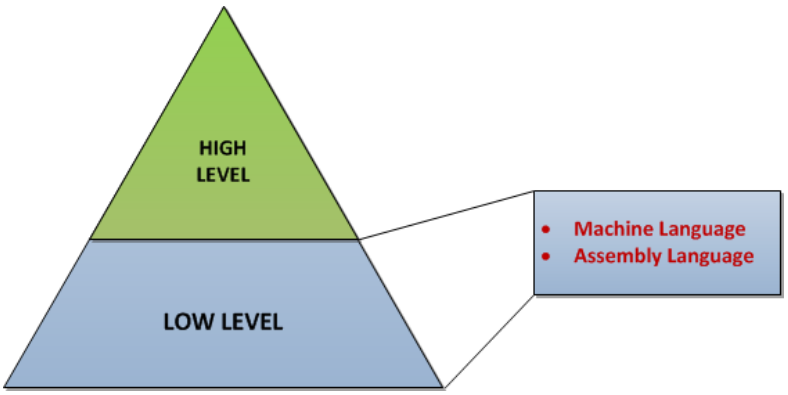


Figure 25 – Programming languages hierarchy

In order to communicate with the computer a language that can be understood by it must be used therefore different languages have been developed to perform different types of work on the computer. There are two levels of programming languages, according to their interpretation, as shown in Figure 25.

Programming languages could be categorized into high level and low level where the latter, in turn, is subdivided into machine language and assembly language [39]:

- **High level** programming languages are closer to the languages used by human beings so they are relatively easier to utilize and are machine-independent so they let the programmer concentrate on the logic of the operations rather than the intricacies of the machine architectures.
- **Low level** programming languages are machine codes or similar to it as computers can only understand binary language of 1 and 0. **Machine language** programming is the lowest and most elementary level of programming language and was the first to be developed. It sends the computer instructions in binary code consisting in sequences of 0's and 1's. **Assembly language** was developed to overcome some inconveniences of machine language and it differs from it as the messages sent to computer uses alphanumeric mnemonics instead of the binary code (e.g. ADD, SUB, START, etc.) so it is sometimes also referred as *Symbolic Programming Language*.

For the purpose of this work JAVA was the programming language selected. It is a High level, objected-oriented programming language. There are several types of high level languages which can also be categorized according to their use, objected-oriented is just one of those categories where other well know programming languages such as C++ and C# are considered.

Objected-oriented programming (OOP) is therefore an attempt to produce programs that are closer models to the way people think and deal with real world problems. Software can then be seen as system composed by a community of objects cooperating with each other by passing messages between them with the purpose of solving a problem [39]. OOP is composed by a group of fundamental features as depicted in Figure 26.

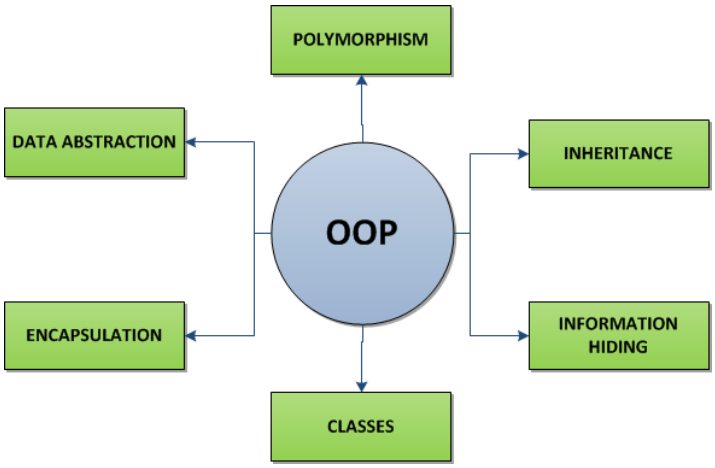


Figure 26 – The fundamental features of object-oriented programming

In a single statement, Abstraction is a technique intended as a suppression of detail. The importance of abstraction is derived from its ability to hide irrelevant details and from the use of names to reference objects. Abstraction is essential in the construction of programs. It places the emphasis on what an object is or does rather than how it is represented or how it works [39].

Encapsulation is the technique to display the information in a way as to hide what should be hidden, and show what's needed. Then, information hiding is the process of hiding all the details of an object. As for polymorphism it implies that something is capable of having various forms which, in OOP universe translates to variables and functions and hence, objects as well. Finally, Inheritance means that with OOP languages new classes can be formed by deriving previously existing ones inheriting certainty proprieties similarly to how, in real life, children inherit certain features from their parents.

4.1.3. FUNDAMENTALS OF OBJECTS AND CLASSES

So what is an Object? And what is a Class? What's the difference between them?

Objects

The idea behind Objects is quite simple. According to Sun MicroSystems [8] in "Object-oriented Programming Concepts" section of their Java Tutorial, an Object is *a "software bundle of variables and related methods"*.

[8] Sun Microsystems, Inc. was a organization that sold computers, computer components, computer software, and information technology services. On January 27, 2010, Sun was acquired by Oracle Corporation.

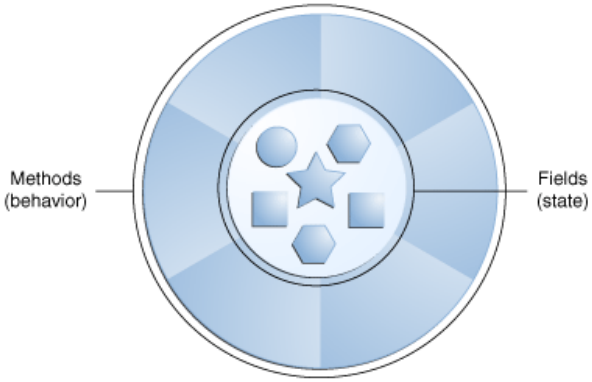


Figure 27 – Representation of an object [40]

Real-world objects have defined states and behavior (Figure 27). For instance, we all have **states** such as our name, age, height, weight and we have our **behaviors**, walking, talking, running, jumping, sleeping. Similarly software objects are modeled after real-world so they too have their state and behavior. Their states are stored in variables [9] and their behaviors are implemented with methods [10]. Objects are also referred to as instances as it refers to one object in particular [40].

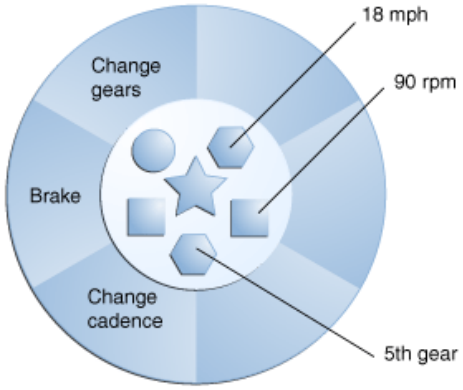


Figure 28 – A bicycle modeled as a software object [40]

By looking at Figure 28 the concept is easily understood. The figure illustrates a model of a real-world object as is a bicycle translated into a software object. We can see that three states are defined so the bicycle is running at the speed of 18 miles per hour (mph) with 90 rotations per minute (rpm) at 5th gear and the outside ring contains the methods provided that are responsible for changing those states.

---

[9] A variable is an item of data named by an identifier. The variables of an object are formally known as *instance variables* as they contain the state for a particular object.

[10] A method is a function associated with an object. In other words, it is something that will change an object by manipulating its variables.

Classes

In object-oriented programming as in the real-world we can have many objects of the same kind (e.g. lots of cars, lots of bicycles, etc.) that have common characteristics. Picture an automotive manufacturing plant where thousands of cars are produced from the same model, the same blueprint. The same happens in OOP where a class is used to manufacture or create objects according to its blueprint. The following *Bicycle* class is one possible implementation of a bicycle type:

```
class Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
         cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" +
             cadence + " speed:" +
             speed + " gear:" + gear);
    }
}
```

Figure 29 - Example of a bicycle class implementation in Java

Disregarding the Java syntax we can identify in the block of code above the Bicycle object's state (*cadence, speed and gear*) and the methods (*changeCadence, changeGear, speedUp, applyBreakes, printStates*) to alter those states resulting with the interaction with the world outside this class. A class is somewhat a factory responsible for building objects throughout the program simply by calling several instances of it. The class *BicycleDemo* bellow creates two separate *Bicycle* objects by instantiating the previously shown class *Bicycle* and then invoking the methods on those objects.

```
class BicycleDemo {
    public static void main(String[] args) {

        // Create two different
        // Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();
```

```
        // Invoke methods on
        // those objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```

Figure 30 - Example of the method invocation of the Bicycle objects in Java

Inheritance

Inheritance is the ability for several objects to share common features and characteristics whilst adding some additional features which make them different and unique. OOP allows classes to inherit commonly used state and behavior from other classes [41].

Looking once again at the *Bicycle* example, if we think of three different purpose bicycles they all share common characteristics (e.g. speed, current gear, current cadence) but they might also have special features to make them different from one another as is depicted in Figure 31.

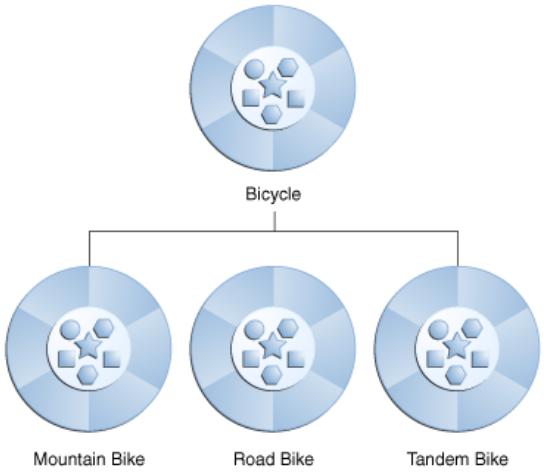


Figure 31 - A hierarchy of different bicycles inheriting similar characteristics from the superclass *Bicycle* [41]

4.2. JAVA PROGRAMMING LANGUAGE

Java was developed earlier in the 1990's in Sun Microsystem's, which has since merged into Oracle Corporation [42] research laboratories by a team led by James Gosling.

The language first started off as *Oak*, as a part a research project named the *Green Project* [43] which was aimed to create a platform that would allow electronic devices and computers to communicate with each other. As no other programming languages available was able to create programs to be executed in devices with different computer architectures Gosling and other researchers involved in the project came up with the idea of creating a new language that would allow the development of programs to be executed in any type of computer. The following five primary goals in the creation of the Java language were identified [44]:

- It should use the object-oriented programming methodology.
- It should allow the same program to be executed on multiple operating systems.
- It should contain built-in support for using computer networks.
- It should be designed to execute code from remote sources securely.
- It should be easy to use by selecting what was considered the good parts of other object-oriented languages.

Later on in 1994 Oak, was renamed to Java when the World Wide Web (WWW) became popular and Sun Microsystems realized that Java was the right programming language for the Web as it made possible to transmit programs through it and execute them locally on the user's computers. It was later publicly released in 1995, as Java, and targeted as an Internet development tool also known as applets.

Java is defined by two entities the *Java Runtime Environment* (JRE) and the *Java Software Development Kit* (SDK). The SDK is the development tool that provides the run-time environment, all the necessary libraries and classes, the java compiler and the interpreter. Even though Java is platform independent the Java platform, the SDK, is not so there are different versions for the different Operating Systems. The platforms all rely on the JRE and a *Java Virtual Machine* (JVM) which is the interpreter responsible for the platform independent.

Figure 32 shows the process of creating, compiling and executing a Java program. It all starts by writing a program in the Java language source *.java files. These files are then translated by the compiler into *bytecode* where a *byte* represents each program instruction. The fact that this code will be interpreted regardless of the computer architecture it will be run in, makes it unnecessary to compile the source code separately taking in consideration

the different architectures. The JVM is then responsible for the interpretation of such code in each different computer making it possible to execute any program on any computer.

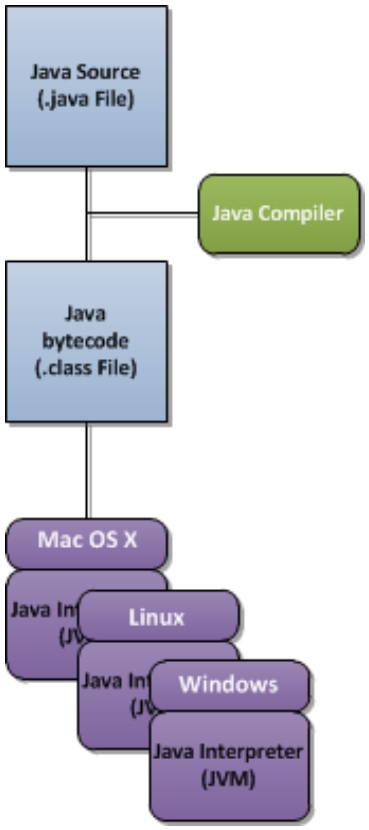


Figure 32 – Java development process

Additionally, Java is considered a good language mainly due to its *Platform Independence*, *Object Orientation* and the *Java API* with its extended library of classes that provide commonly used utility functions that most Java programmers cannot do without. However, it should be mentioned that there are also some disadvantages in using Java and they reside in having to use the JVM. Because of the need to run the Java programs on a its virtual machine it runs somewhat slower than similar programs in other programming languages (this is virtually impossible to notice in today's fast computers).

4.2.1. JAVA-BASED SIMULATION

In a sentence, Java-based simulation is a simulation programmed in Java. However, other than referring to programs executed locally on a computer Java-based simulation often refers broadly to point to simulations that can be accessed over the Internet using the WWW interface also commonly referred to as the web-based simulation.

The Java-based simulation created an opportunity to come up with revolutionary changes regarding the development of simulation models to support organizations' model development phase [45]. Using Java as a programming language for simulation development could have the following benefit:

- Being capable of producing reusable, modular and hierarchical simulation applications it has become clear that the resulting simulation provides very rich and detailed computerized real world models;
- The platform independence achieved;
- Native support for networking and Internet protocols, database, multi-threading and graphical user interfaces [46];
- It is possible to distribute and execute simulation applets over the Internet using standard browsers;

From the literature it is possible to identify the following applications for Java-based simulation:

- **Teaching and training for simulation**;
- **Web-based collaborative decision-making** – allowing for teams in different locations to use the capability of simulation over the internet to effectively use it as a decision making tool;
- **Simulation-based marketing** – manufacturers using web-based simulation to demonstrate technologies or products to potential clients over the web;
- **Simulation Research Methodology** - disseminate models, results and publications on the web allows for new approaches regarding simulation research [47].

4.2.2. JAVA-BASED SIMULATION ENVIRONMENTS AND TOOLIKTS

Several organizations and individuals have developed object-oriented simulation software toolkits [11] implemented in languages such as C, C++ and Java. Java-Based simulation started growing heavy later in the 1990's and a number of discrete event simulation tools started to be distributed. Referring to the levels of distribution from the environments they can be classified according to three different levels:

[11] Toolkits are single utility programs, a set of software routines or a complete integrated set of software utilities that are used to develop and maintain applications and databases.

- **Integrated simulation environments** – visualization and simulation are strictly bound to the runtime environment therefore they are not suited for distribution. Furthermore, they usually have a single-user, single-computer and single-threaded configuration.
- **Semi-distributed simulations** – these are usually categorized in simulation models that are distributed online so they share a web-based characteristic being executed not locally but on a server that is able to present the outputs to the client.
- **Fully distributed simulations** – these include characteristics from both environments mentioned previously as they proved integration with distributed simulation and visualization services.

Some Java-based toolkits mentioned lay within the first environment (Integrated simulation environment), such as *SimJava*, *Silk*, *JavaSim*, *SIMLIB*, *D-SOL*, *SSJ* and *JSL* while others already have characteristics which place them in the second one like *SimProd, JSIM*. Sawhney, Deshpand and Mund [48] provide a fairly good description of the most acknowledged packages:

- *Simjava*, a Java-based discrete event simulation package authored by Fred Howell and Ross McNab [49], is conceptually based on the HASE++ simulation library [50] and the *Sim++* library for C++. Based on a discrete event simulation kernel, *Simjava* includes facilities for representing simulation objects as animated icons on a screen. Also, *Simjava* simulations may be incorporated as "live diagrams" into web documents [49].
- *Silk*^{TM}, a general-purpose simulation language written using the Java programming language, combines process oriented modeling structures with powerful object-oriented language features in an intelligent design. This encourages model simplicity and reusability [46]. In *Silk*, models are developed directly in the Java programming language using an Application Programming Interface (API) composed of classes which consist of relatively few powerful process-oriented modeling features [46].
- *JavaSim* is a set of Java packages for building discrete event process-based simulation, similar to that in *Simula* and *C++SIM* [51]. *SimKit* is another Java based discrete event simulation toolkit that is currently being developed [52].
- *SimProd*, a tool for developing flexible simulation models of production systems on the WWW, is an extension of *Simjava* developed by Ross McNab and Fred Howell [53]. *SimProd* provides different objects that represent entities existing in the

production system, like machines, AGVs, conveyors and others, supporting many types of distributions, which are found useful in the manufacturing environment. Models using objects from *SimProd* can be implemented as applets and executed in a Web browser.

- *JSIM*, a simulation toolkit in which models can be built using either the event package (Event-Scheduling Paradigm) or the process package (Process-Interaction Paradigm) [54],supports a good graphical environment for displaying queues. A Java database is used for storing results.

The paper by Rosetti [55] provides a detailed look upon the *JSL* (a Java Simulation Library). Rosseti describes *JSL* as a simulation library for Java. The *JSL's* current version has packages that support random number generation, statistical collection, basic reporting, and discrete-event simulation modeling. The development of a simulation model is based on sub-classing the *ModelElement* class that provides the primary recurring actions within a simulation and event scheduling and handling. The user adds developed model elements to an instance of Model and then executes the simulation. The purpose of the *JSL* is to support education and research within simulation. Finally, *SSJ* was found to be very efficient, flexible and complete tool. It provides extensive documentation and walk-through examples. It is a framework which allows both discrete and continuous simulation and also events and processes. It has a very good support for random number generation and is implemented as a collection of classes in the Java language. It was designed to be open and very flexible and it provides what it was intended for.

4.3. SIMULATION LANGUAGES

The simulation languages progress dates back to the late 1950's as simulation moved from analog to computer simulation [27].

Computer simulation is an application domain of programming languages that permits its division in the three partitions addressed in section 3.5, discrete event, continuous and Monte Carlo simulation.

Shannon [56] provides a clear view on the advantages and characteristics of various simulation languages as well as the pertinent factors inherent to the language selection process. With so many different languages [56] developed over the years, it is a hard task to decide which language is best fitted for a particular application.

Simulation languages evolved from general purpose to the special purpose languages, developed with special attributes thus providing most, if not all, of the features needed in programming a simulation model which resulted in a major decrease in programming times and an increase in the features available.

Nance [27] provides an historical approach distinguishing the modeling differences among the special purpose languages in each major time period of development.

4.4. SIMULATION PACKAGES

The growing popularity in the use of simulation as a modeling and analysis tool resulted in a wide variety of options when it comes to simulation software (sometimes addressed as simulation packages or just packages) available in today's market. Although there are some open-source packages developed mostly in universities, as part of research projects or other academic projects, which cannot be used for commercial purposes. This way companies are forced to acquire licenses for the commercial software available which normally represent a considerable cost for them.

A proper and thorough selection of the simulation software choice is, by itself, one of the key factors for the success of the simulation projects to be developed thus making that selection increasingly based in objective criteria's that take in consideration not only the products characteristics but also what application it is intended for.

The major issue for this increasingly hard task of choosing between simulation software is the fact that the tools for modeling and simulation went from a small batch of options composed by general programming languages such as FORTRAN, Pascal, C or the simulation specific languages such as GASP, GPSS, SIMULA, SLAM to a wide variety of simulation software packages complete, efficient, and always up to date. These general purpose languages were preferred by modelers as it allowed them a greater programming flexibility when compared to the simulation languages or some simulation packages. All of that has changed and the simulation software available provides a great deal of modeling flexibility, ease to use, ease to modify and maintain, providing better error detection and moreover, offer a great amount of detailed outputs and very powerful graphical engines that provide useful animations.

Please note that Institute of Industrial Engineers (IIE) Solutions[12] offers a collection of publications which contain very useful surveys of simulation software on a regular basis.

4.5. PACKAGES DESCRIPTIONS

Historically, the simulation packages were categorized between simulation languages and application-oriented. Today, however, the software packages available offer good enough flexibility disregarding the need for programming skills at all either being it general purpose or simulation languages.

Therefore, current simulation packages are divided into two major types, general-purpose and application-oriented. The difference between them resides in the fact that a general-purpose package can be used for any application (which does not mean it might have special features oriented towards some particular applications) whilst application-oriented simulation packages are designed specifically for certain types of applications such as manufacturing, communications, defense and so on.

4.5.1. GENERAL-PURPOSE PACKAGES

General-purpose software packages are the most commonly known and mentioned of all as they can solve almost any discrete simulation problem.

Table 8 Popular general-purpose simulation packages listing.

Package Name	Year	Organization	Type	Website
ARENA	2000	Systems Modeling Corporation	Student & Commercial	[57]
AweSim	1997	Autosimulations	Commercial	[58]
Extend	1987	Imagine That	Commercial	[59]
GPSS/H	1977	Wolverine Software	Student & Commercial	[60]
SIMSCRIPT III	2007	CACI Products Organization	Commercial	[61]
SIMPLE ++	1992	AESOP Corporation	Commercial	[62]
SLX	1997	Wolverine Software	Student & Commercial	[60]

[12] IIE is the world's largest professional society dedicated solely to the support of the industrial engineering profession and individuals involved with improving quality and productivity. IIE's monthly magazine, IIE Solutions, keeps members up-to-date on issues, cutting-edge technology, IE current events, and industry newsmakers.

Some of the most popular general-purpose simulation packages are systematized in Table 8.

The different packages offer different solutions but they do share some characteristics. The one that stands out more is the search for a friendly and easy-to-use workspace and, preferably, being run as a Windows application. Virtually all the commercial packages available today also allow basic features such as debugging, simulation execution visualization, statistical analysis of the generated outputs and extended reports. What stands out the most is the eye-candy solutions these packages offer, where the animation resources go from simple graphical implementations with symbols and text, up to sophisticated 3D animations worthy of computer game display.

Arena is a product marketed by Systems Modeling Corporation and is a modular general-purpose simulation package [57]. Arena has a very complete and essentially unlimited number of random-number generation of streams capability. It offers the ability to easily model several kinds of material-handling devices used for transport and accumulation (conveyors and part-collecting platforms or buffers) and also automated guided vehicle systems (which plays a major role in today's industry).

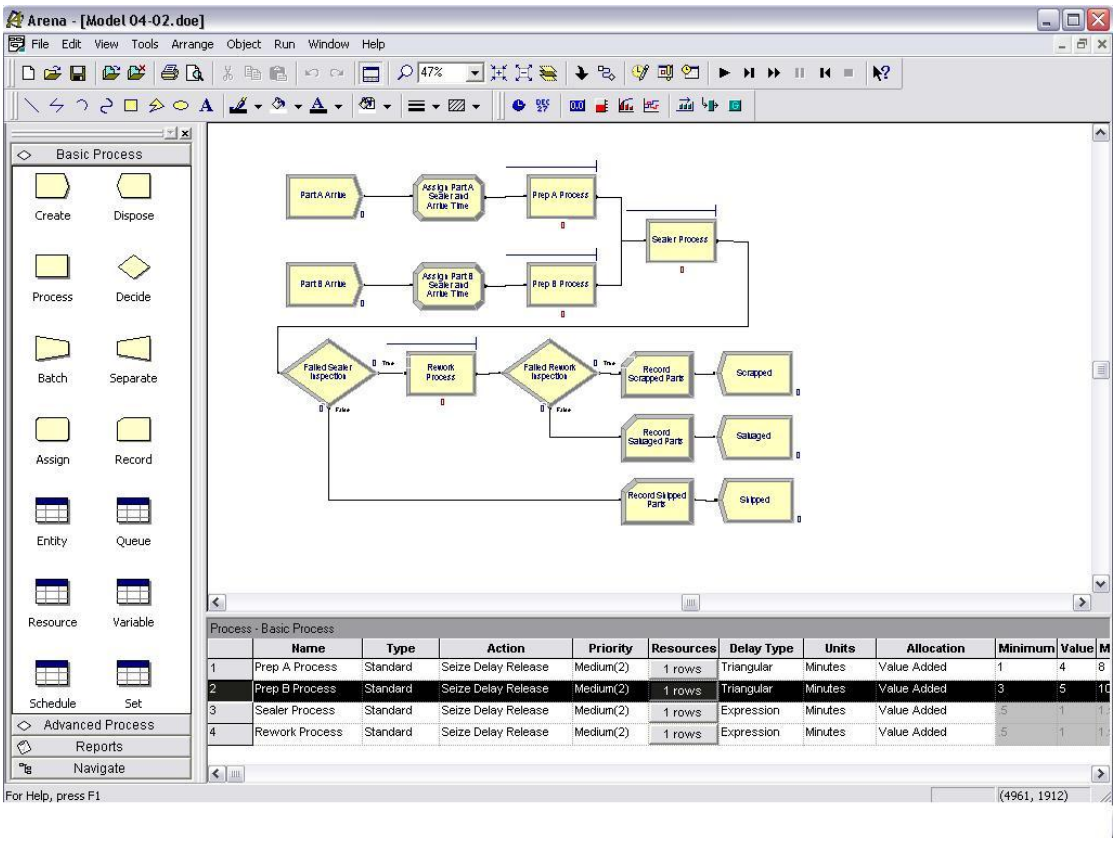


Figure 33 Overview of the package GUI with a representation of a simple model in Arena

It also allows interaction with tools from the Microsoft Office Suite such as Excel and Visio due to the use of Microsoft Visual Basic for Applications (VBA) thus permitting to exchange data with those applications. Great detail on the software and extended walk-through guides are available in Ph.D. Manuel D. Rossetti's "Simulation Modeling and Arena" publication [55].

AweSim was developed in 1997 by Autosimulations [58]. It implements the simulation language SLAM and its focus resides in creating a simulation project consisting of one or more scenarios thus representing alternative system configurations. AweSim allows threaded work meaning that tasks can be performed in parallel with the simulation run. The models in AweSim are built graphically and are very flexible. It includes a report browser that allows choosing between different outputs and comparing them side by side. Also it is important to stand out that it extends its modeling capabilities with Visual Basic or Visual C++, offering the ability to create user-defined logic to complement the models created, adding a fairly good amount of flexibility [63, 64].

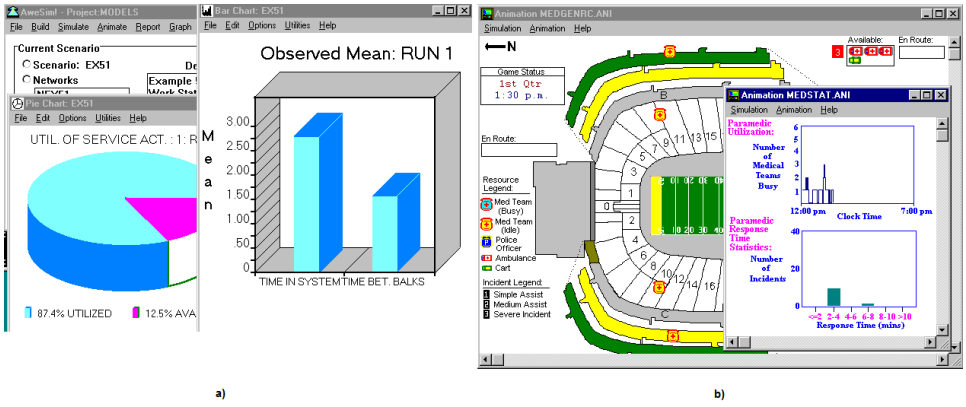


Figure 34 AveSim's feature examples. a) on the left, multiple report windows displays for the output results. b) on the right, multiple animations being displayed on several windows for a single scenario [63]

Extended from Imagine That Inc. is a fairly simple package where models are created graphically by recurring to blocks available in its libraries and dragged and dropped to the interface [59]. It provides a two-dimensional animation but additional features are available as an option. The outputs generated are also very complete in the form of plots, histograms, tables and customizable reports. It also includes the ability to use external general-purpose simulation languages such as C and FORTRAN to add custom functionalities to the models built. Finally it offers compatibility across platforms [64, 65].

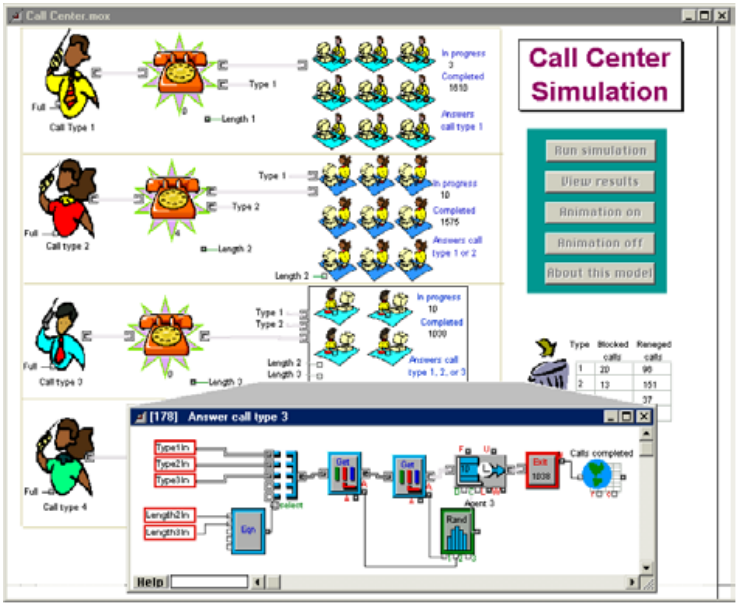


Figure 35 GUI interface of Call Center simulation modeled on Extend [64, 65]

GPSS/H is a very powerful improvement over GPSS V from Wolverine Software [60], offering a flexible package without the need for external FORTRAN routines (as in the previous version GPSS V from IBM). A system is modeled as transactions enter it and are passed from one service (represented by blocs) to another making it particularly well-suited for problems such as a manufacturing system. It was a very popular tool in the late 1960's and 1970's but has failed to keep up with the more recent packages [64].

SIMPLE++ is a fully object-oriented simulation package where objects created in the libraries represent classes whose instances can be inserted into the models [62]. The concept behind object-oriented languages and simulation will be addressed further in section 4.1.2. This structure allows it to be a very flexible tool as it has the ability to reuse the objects created. Due to its open architecture it can communicate and exchange data with other software such as databases, spreadsheets, virtual reality packages and other. In addition, it offers a dynamic analysis possible due to it being embedded within software destined for optimization, re-engineering [64].

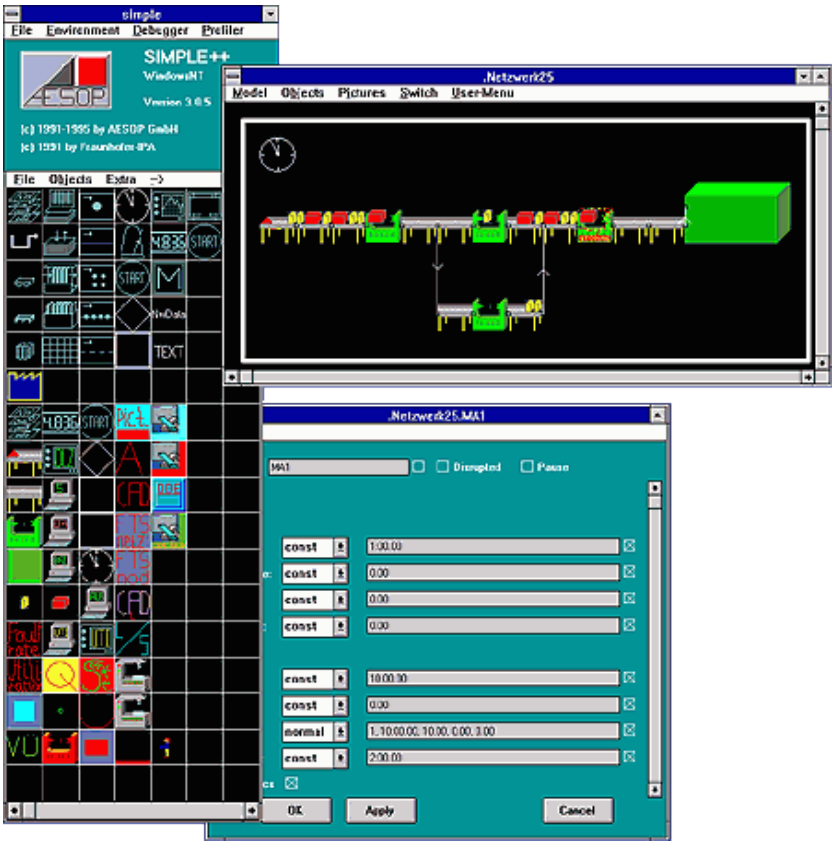


Figure 36 SIMPLE++ - Screen shot of a manufacturing line [66]

SIMSCRIPT III was developed in 2007 from CACI Products Organization [62]. SIMSCRIPT III is the second release from CACI of a modular object-oriented system for designing and building discrete and combined discrete/continuous simulations. It contains a very good object-oriented graphical front end facilitating model use by those that are not programmers, as is depicted in Figure 36 with the application template located on the left-hand side, a new object is added interactively to the model in the right hand window, and model dialog window with object parameters at the bottom. This version now contains both 2-D and 3D animation and visual presentation of the generated results and outputs. SIMSCRIPT III is a superset of the existing and well known SIMSCRIPT II.5, which means that all existing SIMSCRIPT II.5 models will work without changes under the new compiler and development. The new SIMSCRIPT compiler supports all existing SIMSCRIPT II.5 features as well as the execution speed for very large simulation models has been improved by 30 to 40% [64].

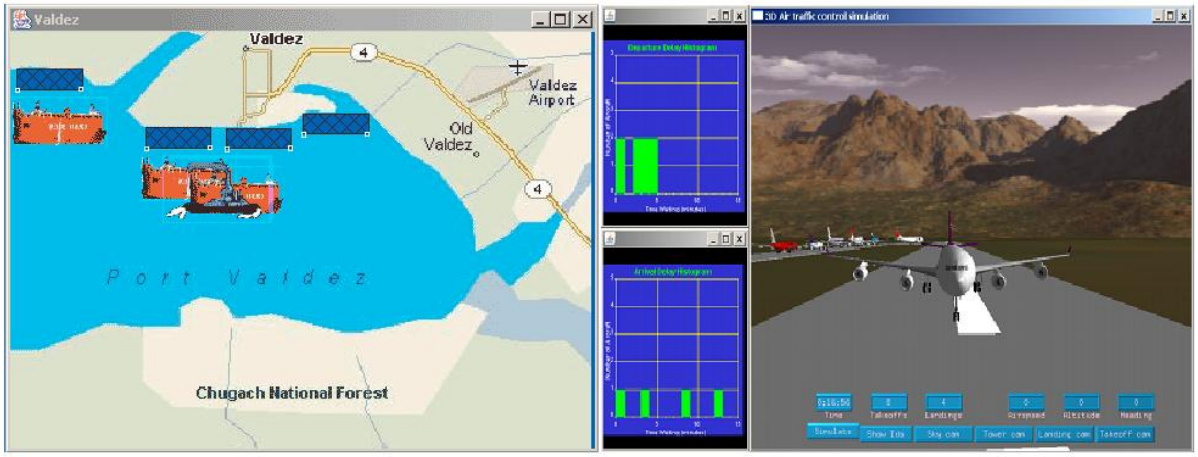


Figure 37 SIMSCRIPT III different simulation animation capabilities. 2D simulation displayed on a) and 3D simulation on b)

SLX, is another product from Wolverine Software Corporation [60] proposed in 1997. The most important characteristic of SLX is its layered architecture. Traditional language-based simulation tools fall in the middle of the SLX pyramidal hierarchy. SLX's layers extend both below and above the focus of traditional simulation languages. SLX claims that after working for a long time alongside modelers they have found that these seek for flexibility and extensibility, provided by SLX meaning that it can extend beyond its built-in feature set so it is designed to be customized. On Wolverine Software website detailed information on the package along with an explanation to experienced GPSS/H or C user's regarding the differences between those and SLX is available [64].

4.5.2. APPLICATION-ORIENTED PACKAGES

Application oriented software is specially designed for all sorts of applications such as Manufacturing, Communications, Reengineering, Schedule, Optimization, Defense, and other.

Table 9 – Popular Manufacturing-oriented simulation packages listing.

Package Name	Year	Organization	Type	Website
AutoMod	2000	AutoSimulations, Inc.	Commercial	[67]
Arena	2000	Systems Modeling Corporation	Student & Commercial	[57]
Extend+Manufacturing	2008	Imagine That Inc.	Student & Commercial	[59]
FACTOR/AIM	n/a	Pritsker Corporation	Commercial	[68]
ProModel	n/a	PROMODEL	Commercial	[69]
Taylor II	1986	F&H Simulations	Commercial	[70]
WITNESS	2002	Lanner Group	Commercial	[71]

Concerning the present work we will focus on Manufacturing-oriented software packages (Table 9) and briefly introduce some of the most relevant

AutoMod is leading graphical simulation software that provides true-to-scale 3D simulation of manufacturing and distribution operations from any angle. AutoMod automatically generates statistical output reports and charts [58]. This output provides information on all aspects of the system, such as equipment utilization, inventory levels and the total time parts are in a facility. Statistical reports can be viewed in tables or with built-in business Graphicals.

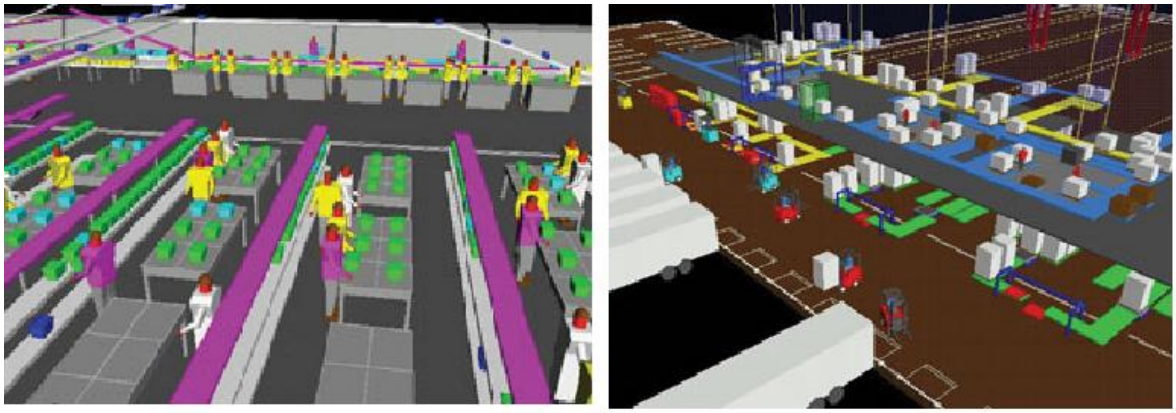


Figure 38 AutoMod working 3D models. a) a job-shop simulation were workers are filling orders to evaluate the completion rate. b) simulation of a truck loading-bay [72]

One particularly interesting aspect about AutoMod is its spreadsheet interface which allows users to add complex logic to their models without the need for programming languages knowledge. To reveal how extended and complete this software is, there is a separate utility provided, AutoView, which provides the ability for post-processing three-dimensional walkthrough capability to create high quality animations [64].

FACTOR/AIM (AIM) is a simulation system designed specifically for use in manufacturing decision support [68]. AIM has been successfully applied to engineering design, scheduling, and planning problems within numerous manufacturing organizations. AIM operates on the Windows platform and all model data is stored in a Microsoft Access database. AIM is therefore database-oriented [64].

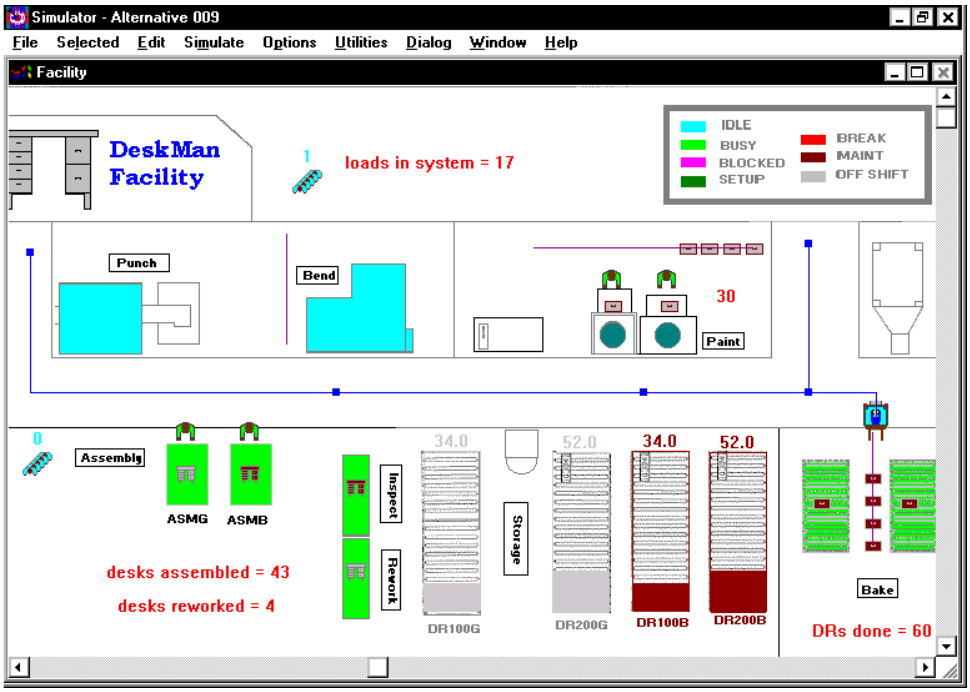
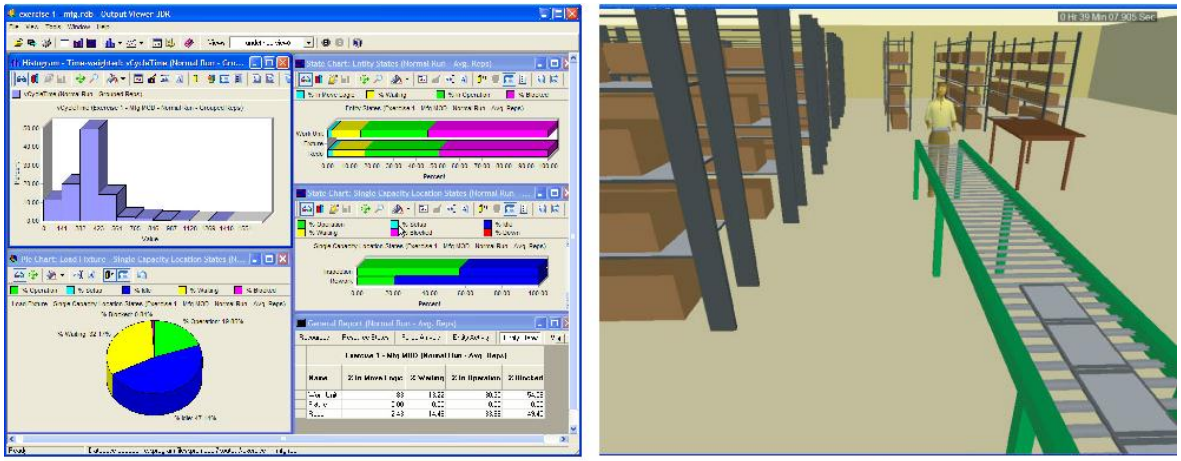


Figure 39 Factor/AIM - An overview of a Facility plant model simulation running

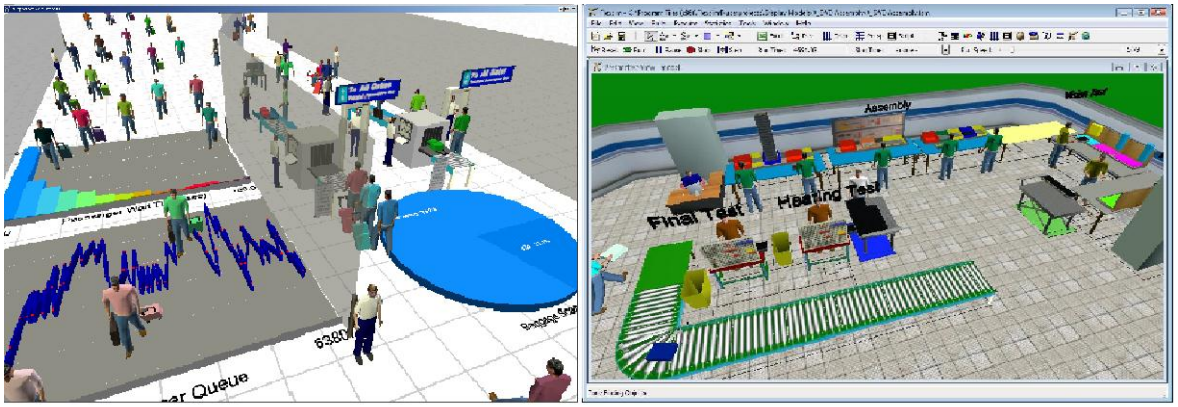
The ProModel suite from PROMODEL Corporation is, at a first glance probably one of the most complete solutions available on the market as of today [69]. It is used to plan, design and improve new or existing manufacturing, logistics and other tactical and operational systems. It allows to accurately replicate complex real-world processes with their inherent variability and interdependencies, to conduct predictive performance analysis on potential changes, and then optimize the system based on the key performance indicators. It can create dynamic, animated computer model of the business environment from existing CAD files and it can also import process information and schedules. It takes optimization in serious consideration so besides running the simulation it is possible to test the impact of changes on current and future operations, with predictive scenario comparisons. PROMODEL state that the U.S.A. Army and Air Force are long time users of their simulation package which reveals an accurate and trustworthy tool [64].



a) b)

Figure 40 a) ProModel's output viewer displaying an histogram and several charts. b) a running 3D animation of an inventory system's simulation

F&H Simulations sold, supported, and conducted training courses for their Taylor II simulation software [70]. After being purchased, F&H Simulations took that opportunity to become independent. Dr. Eamonn Lavery and Anthony Johnson joined to oversee product architecture and begin the development of a new, next generation, 3D and object oriented simulation software called Flexsim.



a) b)

Figure 41 a) Flexsim's powerful 3D analysis. Charts and Graphicals can be placed anywhere on the model simulation animation for a live feed. b) Production plant 3D simulation

F&H Simulations, Inc. changed its name to FlexSim Software Products, Inc and became the substitute for the Taylor II simulation package. Flexsim was conceived to be the most sophisticated 3D discrete event package ever created. Like ProModel it has the capability to creative immersive presentations from the actual model itself. Flexsim's powerful 3D graphics allow in-model charts and charts to dynamically display output statistics whereas users can also interface with common spreadsheet and database applications to import and

export data. Plus, the experimenter allows the users to quickly change multiple sets of variables and see the results [64].

WITNESS from Lanner Group (version 12) is a package that is able to replicate and study any manufacturing business process [71]. As a Microsoft Gold Certified Partner, Lanner has developed WITNESS 12 using the latest Microsoft development environment so it has been built to take advantage of the latest technologies featured in Windows 7. Other than that, it goes along the line with all the features presented in the previous packages being present in it as well [64].

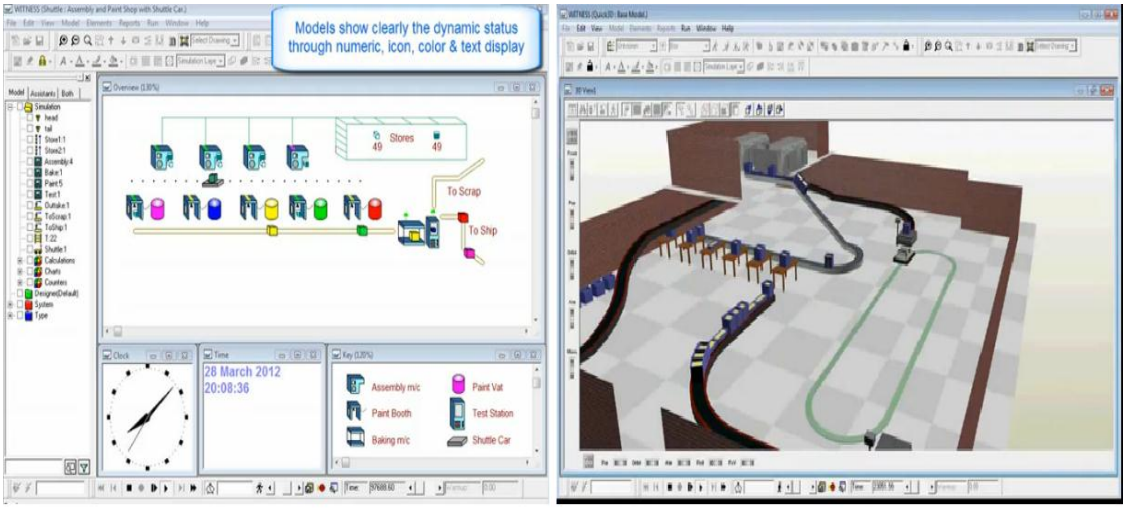


Figure 42 a) a 2D representation and build of a manufacturing system model. b) a live animation of a manufacturing system simulation run on WITNESS

4.6. SOFTWARE SELECTION

The process of selecting the right software requires a degree of knowledge that goes beyond the knowledge of the basic modeling and programming blocs. Also, with the amount of packages available today and all features, the decision process must be more careful and could be time consuming. In 1998, the simulations Buyers' Guide in IIE Solutions [73] showed 35 different packages available, 42 of which were considered useful in manufacturing which shows how complex the selection of the software can be.

Jerry Banks is one of the authors that have written more on this subject. He provided a list of features to consider in the software selection based on [74]. These will be addressed next.

4.6.1. DESIRED SOFTWARE CHARACTERISTICS

The requirements strongly advised to consider when selecting a simulation software package are presented in Table 10.

Table 10 - Possible features to consider in the software selection process.

Features regarding	Considerations
INPUT	- Point-and-click capability; - CAD translation from existing models; - Import data files; - Export data files (to use in spreadsheets for - instance); - Easy to understand syntax; - Debugging utilizes; - Multi-Language Interface; - Input data analysis.
PROCESSING	- Powerful overall processing capability; - Speed; - Flexibility; - Random Variable Generators; - Replications and multiple runs; - Custom logic representation; - Portability.
OUTPUT	- Standard reports; - Customized reports; - Graphics; - Database maintenance; - Custom performance measures; - Write to file.
ENVIRONMENT	- Ease of use; - Ease of learning; - Documentation available and its quality; - Animation capability; - Stability; - Background history; - Support.

Also, notice that the cost factor is no less important and it should be balanced with the rest of the resources needed to run the simulations. This considerations presented in Table 10 are not equally distributed on importance scale and they obviously vary from organization to organization and from system to system. A call-center organization will not probably require software with heavy animation capability or 3D features when compared to an automotive manufacturer but will most likely appreciate a fast simulation package with detailed reports. Overall, the basic needs for virtually any problem solving either be it a small or a large one in a small or large organization include:

- Generating random numbers;
- Generate random variables from probability distributions;
- Advancing and resetting simulation runs;
- Collecting output statistics and generating reports with fair amount of detail;
- Error Detection;
- Animation.

So, when selecting the software package if any doubts still remain it is a good practice to:

- ask simulation vendors to solve a smaller version of one of your problems;
- Seek references regarding the software's capabilities and know limitations;
- Seek the opinion of consultants;
- Seek the opinions of know companies with similar applications;
- Seek for the surveys of simulation software provided on IIE Solutions;
- Visit the Winter Simulation Conference (WSC) or similar events to personally meet several software vendors.

4.7. CONCLUSIONS

There is a great difference between the simulation paradigms, the ideas and tools to build simulation models, and the implementation paradigms when applied to the development of simulation systems.

The main focus was on the concept and considerations of simulation software as well as its selection principles. A selection of both general-purpose and application oriented software was presented with a brief description of the key features from each of them

The desired characteristics in a simulation software package where detailed, somewhat providing a starting ground or basis for the developed application prototype.

5. REAL TIME MANUFACTURING SYSTEMS SIMULATOR PROTOTYPE

In this chapter are described all the stages present in the development of the application software. Moreover, the system's requirements are reviewed as well as the design for all the interfaces present in the application.

The SSJ (Stochastic Simulation in JAVA) library will be analyzed more in depth and we end up with the details regarding the development of the simulator prototype as an objective of this thesis.

Do notice that the name attributed for the simulation prototype software is RTMSS that stands for Real Time Manufacturing Systems Simulator and from now on it will be address using such acronym.

5.1. RTMSS SIMULATOR PROTOTYPE

In this chapter are described all the stages present in the development of the application software. Moreover, the system's requirements are reviewed as well as the design for all the interfaces present in the application.

The concept and basis of OOP had to be studied and practiced as there was no previously background on that subject. Therefore, the first task for this project was to learn and practice programming in Java so for that matter some basic applications were created to simply allow gaining knowledge of the basics of the programming language as well as the IDE (*Integrated Development Environment*) used, *NetBeans* in this case.

It was found very useful to investigate other similar solutions to what was proposed to create for the current application. *JavaSim*, *JSIM*, *JSL* and *SSJ* (already detailed previously), still available for download as of today, where chosen to investigate. After a thorough examination of all the packages and the documentation available SSJ was chosen to explore further on mainly due to a great bundle of examples and a very complete library.

After, the idea of creating a GUI to run some selected SSJ examples occurred (see sub-section 5.1.1), which would allow for a further contact with both simulation modeling and programming as well as graphical interface development in JAVA.

Finally, the interface for the simulator prototype was planned and later implemented using the Java SWING utilities which will be detailed further on.

5.1.1. GRAPHICAL INTERFACE FOR CASE STUDY SIMULATION MODELS

SSJ is as mentioned a Java library package for stochastic simulation, developed in the *Département d'Informatique et de Recherche Opérationnelle* (DIRO), at the *Université de Montréal*. It allows the programming of discrete-event simulations with both events and processes modeling simulations.

SSJ was developed under the terms of the General Public License (GNU) as free software. As mentioned it provides a fairly broad list of examples that go from some elementary examples such as a discrete-time inventory system or a single-server queue to discrete-event simulation both event-oriented and process-oriented. For this purpose, three examples are looked further upon this document:

- Single-server queue with Lindley's recurrence;
- Simplified bank;
- Job-Shop Process-Oriented model;

Before moving further on analyzing these examples, the GUI created will be presented. It is integrated in the Simulator Prototype application. When a user executes the RTMSS it is presented with a welcome interface where it is possible to select from running either the simulator prototype or the interface to execute some selected SSJ examples (see Figure 43).

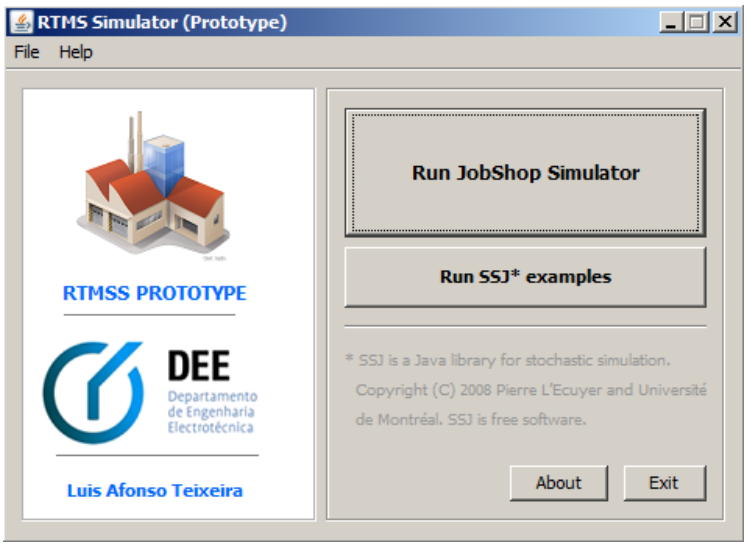


Figure 43 RTMSS initial screen

As for the SSJ examples interface (see Figure 44), the user is presented with the interface's starting tab where a brief description of the framework is described.

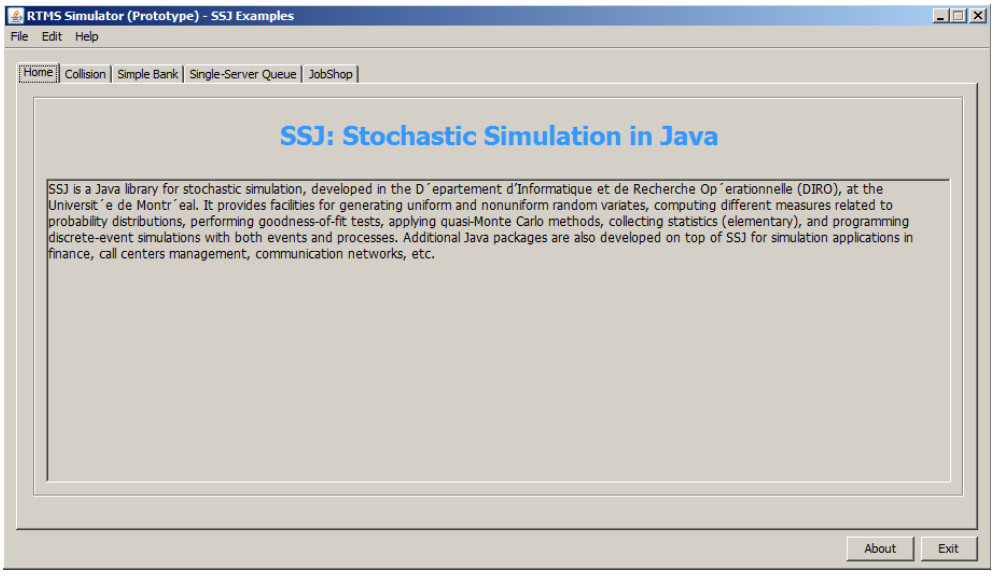


Figure 44 SSJ examples initial screen interface

The user can then alter between examples by clicking the top tabs to run the different examples possible. There is no animation available when running these examples but they are customizable so that users can understand them better by experimenting rather than just observing the simulation runs and its plain text output.

The source code was remodeled so that some parameters can be defined and a final output of the simulation is generated and available to the user.

5.1.2. SINGLE-SERVER QUEUE WITH LINDLEY'S RECURRENCE

Queuing models are used to mathematically analyze the queuing behavior of a real queuing system. They allow a number of useful steady state performance measures to be determined, including:

- the average number in the queue, or the system,
- the average time spent in the queue, or the system,
- the statistical distribution of those numbers or times,
- the probability the queue is full, or empty, and
- the probability of finding the system in a particular state.

These models are usually represented using Kendall's notation, a standard used to describe the characteristics of the queuing systems. Kendall's notation may appear in one of three forms:

A/B/S/K/N/D
A/B/C/K/N/D
A/B/n/p/k
A/S/s/c/p/D

1)

Where:

- **A** – description of the arrival process (most common describing codes are **M** - a Poisson or random arrival process, **D** – deterministic or fixed inter-arrival time, **Ek** – Erlang distribution with k as the shape parameter or **G** – independent arrivals);
- **B/S** – service times (common codes as on A);
- **S/C/n/s** – number of service channels also known as servers;
- **K/n/c** – the capacity of the queue that can also incorporate the buffer for the system;
- **N/p** – size of the costumer population;

- **D/k** – queue's discipline that is, the priority order for the jobs in queue on our waiting line (more commonly **FIFO**, *Last In First Out* – **LIFO**, *Service In Random Order* – **SIRO**)

Despite different characters being used, their meaning is alike. Many times as well, the last members are omitted thus resulting in a three variable notation such as A/B/C, A/B/n or A/S/s assuming that K/n/c, N/p = ∞ and D/k = FIFO [13].

Some examples of Kendall's notation used for queuing models:

- M/M/1
- M/D/1
- M/G/1
- G/G/1
- M/M/n
- M/M/n/n+m
- M/M/¥ (Poisson model)
- M/M/n/n (Erlang model)
- M/M/k/k/k (Binomial model)

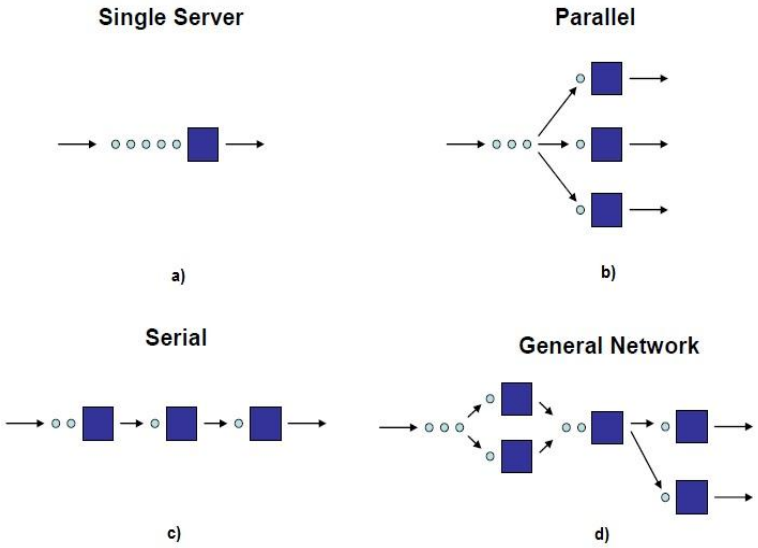


Figure 45 Queuing network structures

[13] FIFO stands for First In First Out. FIFO is a method of processing and retrieving data. In a FIFO system, the first items entered are the first ones to be removed. In other words, the items are removed in the same order they are entered.

In the present example we are using a M/M/1 model. This is a single server queue (see Figure 45.a) where costumers arrive randomly and are served one by one in their order of arrival (FIFO). The system starts off empty and whenever a customer arrives and the server is busy it must join the queue. The goal is to simulate *m* customers joining the system to obtain the mean waiting time per customer. Both the times between arrivals and the service times are exponential random variables generated by a class in the package.

This queue's waiting times follow the recurrence known as the Lindley's equation [75]:

$$W_{i+1} = \max(0, W_i + S_i - A_i) \qquad 2)$$

Where W_i and S_i are the waiting and service time of the *ith* customer in line and A_i is the time between arrivals.

Some explaining of the Java code will follow as it will fit the other examples below as well as part of the RTMSS prototype.

First, it is necessary to import all the files containing classes and methods to be called inside the present code so we will need to use all the methods inside the *stat* and *rng* packages as well as the *ExponentialDist* and *Chrono* utilities:

```
import umontreal.iro.lecuyer.stat.*;
import umontreal.iro.lecuyer.rng.*;
import umontreal.iro.lecuyer.probdist.ExponentialDist;
import umontreal.iro.lecuyer.util.Chrono;
```

SSJ offers the following packages:

- **stat** - provides elementary tools for collecting statistics and computing confidence intervals;
- **rng** - provides facilities for generating uniform random numbers over the interval (0, 1), or over a given range of integer values, and other types of simple random objects such as random permutations;
- **probdist** - contains a set of Java classes providing methods to compute mass, density, distribution, complementary distribution, and inverse distribution functions for many discrete and continuous probability distributions, as well as estimating the parameters of these distributions;
- **util** - contains utility classes used in the implementation of SSJ, and which are often useful elsewhere. For example, there are timers (for CPU usage), utilities to read or format numbers and arrays from/to text, operations on binary vectors and matrices,

some mathematical functions and constants, root-finding tools, methods for SQL database interface, and so on;

It is then time to initialize the class for the program which will be called *QueueLindley* and initialize the variables for the *Arrival (streamArr)* and *Service (streamServ)* streams (see Figure 46). The counter for the *averageWaits* is initialized by calling the constructor *Tally* from the *stat* package which is a statistical tool responsible for generating the desired output result.

```
public class QueueLindley {

   RandomStream streamArr = new MRG32k3a();
   RandomStream streamServ = new MRG32k3a();
   Tally averageWaits = new Tally ("Average waits");
```

Figure 46 QueueLindley Class

Both *streamArr* and *streamServ* are initialized as *RandomStream* objects, which is an interface that defines the basic structures to handle multiple streams of random number. For each stream, one can advance by one step and generate one value, or go ahead to the beginning of the next sub stream within this stream, or go back to the beginning of the current sub stream, or to the beginning of the stream, or jump ahead or back by an arbitrary number of steps. The generators provided with SSJ have various speeds and period lengths.

Table 11 Timings provided in SSJ's user guides

RNG	Period	Gen. time	Jump time
LFSR113	2133	51	0.08
WELL512	2512	55	372
WELL1024	21024	55	1450
MT19937	219937	56	60
WELL607	2607	61	523
GenF2w32	2800	62	937
MRG31k3p	2185	66	108
MRG32k3a	2191	109	2.3
F2NL607	2637	125	523
RandRijndael	2130	260	0.9

For each generator, the previous table gives the approximate period length (period), the CPU time (in seconds) to generate 10^9 $U(0; 1)$ random numbers (generation time), and the CPU time to jump ahead 10^6 times to the next sub stream (jump time). Table 11 provides a list of the available timings on a 2100 MHz 32-bit AMD Athlon XP 2800+ computer running Linux, with the JDK 1.4.2 [76].

In many programming languages, so as in Java, the main function is where a program starts its execution. It is responsible for the high-level organization of the program's functionality, and typically has access to the command arguments given to the program when it was executed.

```
public static void main (String[] args) {

   Chrono timer = new Chrono();
   QueueLindley queue = new QueueLindley();
   queue.simulateRuns (100, 10000, 1.0, 2.0);
   System.out.println (queue.averageWaits.report());
   System.out.println ("Total CPU time: " + timer.format());
}
```

Figure 47 Code description of a main Java function

In our main function (see Figure 47) a timer for the thread is created using the *Chrono()* class and also a queue is started by calling the default class constructor. The *Chrono()* class is responsible for computing the CPU time for the current thread only, which will allow to output the required time to run the simulation. Furthermore the outputs are generated, first, the auto-generated report for the average waiting time in the queue and second the amount of time the simulation took.

The method to execute the simulation was called and the parameters were passed on:

```
   queue.simulateRuns (100, 10000, 1.0, 2.0);
```

This will pass the arguments to the method responsible for running the simulations, in this case, 100 runs with 10000 customers on the queue. This function creates a cycle to run *n* numbers of runs and on each run it calls for another constructor where the mathematical calculations are executed. The arguments 1.0 and 2.0 refer to the λ and μ arguments to apply on the Lindsey equation since the exponential random variables already mention for the arrival and service times have mean 1/ λ and 1/ μ.

```
public void simulateRuns (int n, int numCust, double lambda,
double mu) {

   averageWaits.init();
   for (int i=0; i<n; i++)
      averageWaits.add (simulateOneRun (numCust, lambda,
mu));

}
```

Finally, as mentioned, the *simulateOneRun* is the method where the calculations for the waiting time for each customer are executed using the Lindley's equation (see Figure 48).

```
public double simulateOneRun (int numCust, double lambda,
double mu) {
```

```
   double Wi = 0.0;
   double sumWi = 0.0;
   for (int i = 2; i <= numCust; i++) {
      Wi += ExponentialDist.inverseF (mu,
streamServ.nextDouble()) -
      ExponentialDist.inverseF (lambda,
streamArr.nextDouble());
      if (Wi < 0.0) Wi = 0.0;
      sumWi += Wi;
   }
   return sumWi / numCust;

}
```

Figure 48 simulateOneRun method

The full code is listed on Appendix A. Finally, after the program is run with the parameters stated before the output generated is depicted in Figure 49.

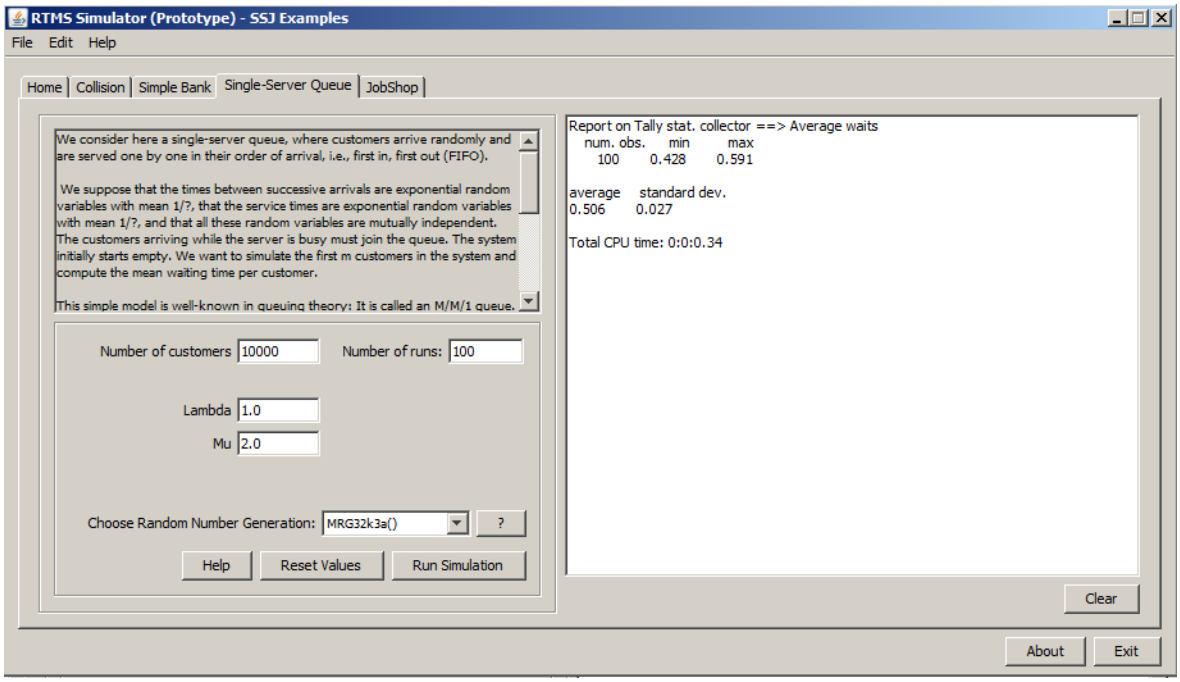


Figure 49 SSJ Single-Server Queue example interface with output

5.1.3. SIMPLIFIED BANK

The bank example is a classic found in the literature. The Simplified Bank [77] and it represents a bank with one to three tellers working from 10:00AM to 15:30PM. The model is represented in Figure 50 assuming a situation where 3 tellers are presently working.

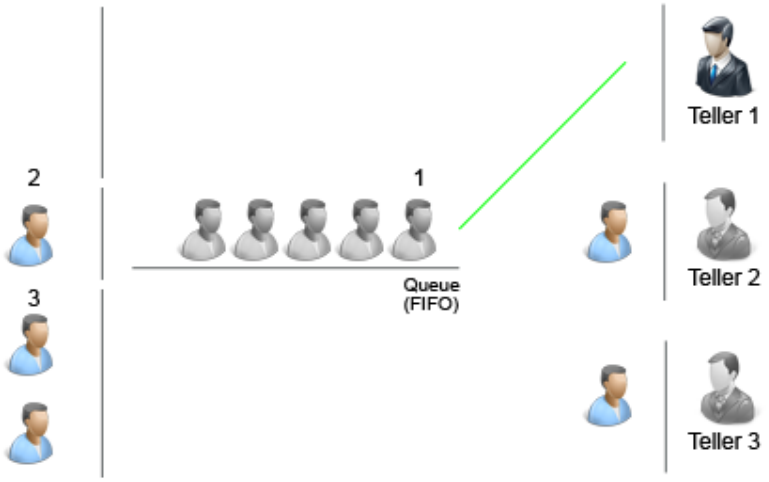


Figure 50 Representation of the bank model example

Costumers form a FIFO queue when all tellers are busy and balking (walking out) is present if there are k costumers in front on the queue when joining, with probability p_k being:

$$p_k = \begin{cases} 0 & if\ k \leq 5 \\ \dfrac{(n-5)}{5} & if\ 5 < k < 10 \\ 1 & if\ k \geq 10 \end{cases} \qquad 3)$$

Therefore, when customer 2 at Figure 50 arrives at the Bank he will join the queue ($p_k = 0$) as there are only 4 customers in the queue and the first one is already going to move to service. On the other hand, when customer 3 arrives he might join the queue or balk, the same for the upcoming customers up until the queue has size 10 time when all arriving customers will walk away. Also the number of working tellers at the bank is not fixed and it assumes a q_t probability where:

$$\begin{aligned} q_1 &= 0.05 \\ q_2 &= 0.15 \\ q_3 &= 0.05 \end{aligned} \qquad 4)$$

The service times are independent Erlang random variables, as for the arrival times they are random Poisson variables with constant rate $\lambda(t),\ t \geq 0$ as is depicted in Figure 51.

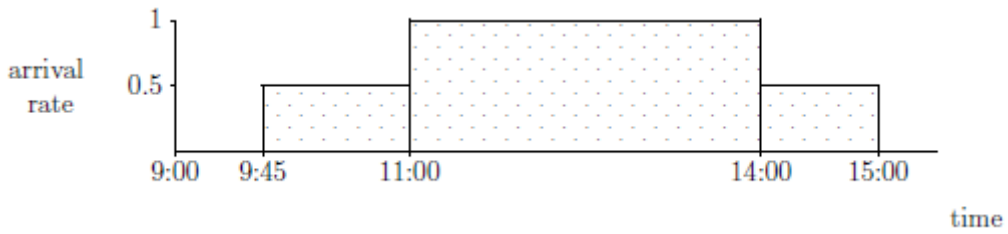


Figure 51 Customer arrival times distribution over the working hours [78]

This is an event-oriented model and those events are time based so there will be four time triggered events where the arrival rate is changed to meet the specifications set at Figure 51. The flowchart on Figure 52 explains the flow of the events happening throughout a working day at the bank. When the bank opens and the process starts, an event generates the number of tellers working. Upon arrival, customers assess the size of the queue to decide if they join it or balk. If they decide to join the queue or if there are no other customers waiting they will get their service at the first free teller.

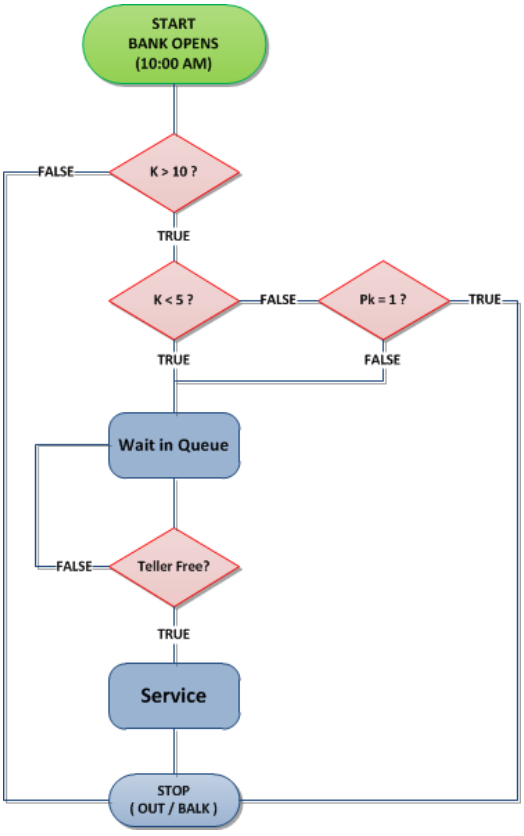


Figure 52 Flowchart of the bank process

Figure 53 shows the results for a program run simulating 100 work days at the bank, generated output statistics for the number of customers served per day as well as the average waiting times (hours).

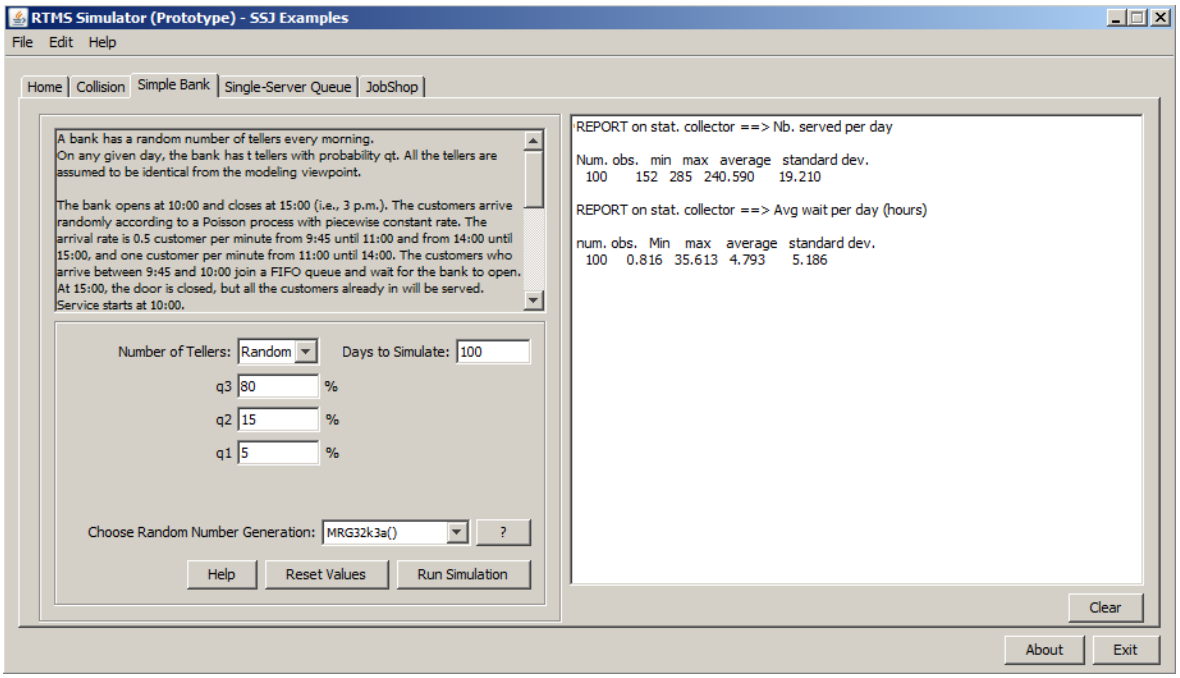


Figure 53 SSJ Simple Bank example interface with output

5.1.4. JOB-SHOP PROCESS-ORIENTED MODEL

The Job-shop model is an example of a process oriented model adapted from [18]. The original example creates a model and then shows examples of its implementation and simulation in both C and SIMLIB. It illustrates how simulation can be used to identify bottlenecks in the production process.

However, the example presented by the SSJ library simply computes the average time in the Job-Shop for each type of task as well as the average utilization rate, average waiting queue length and average waiting time for each type of machine.

A Job-Shop contains *M* groups of machines with *Sm* identical machines and is modeled as a network of queues each group of machines having a single FIFO queue. There are two main processes, the arrival function when a part arrives at a machine and the departure function when the machine finishes processing.

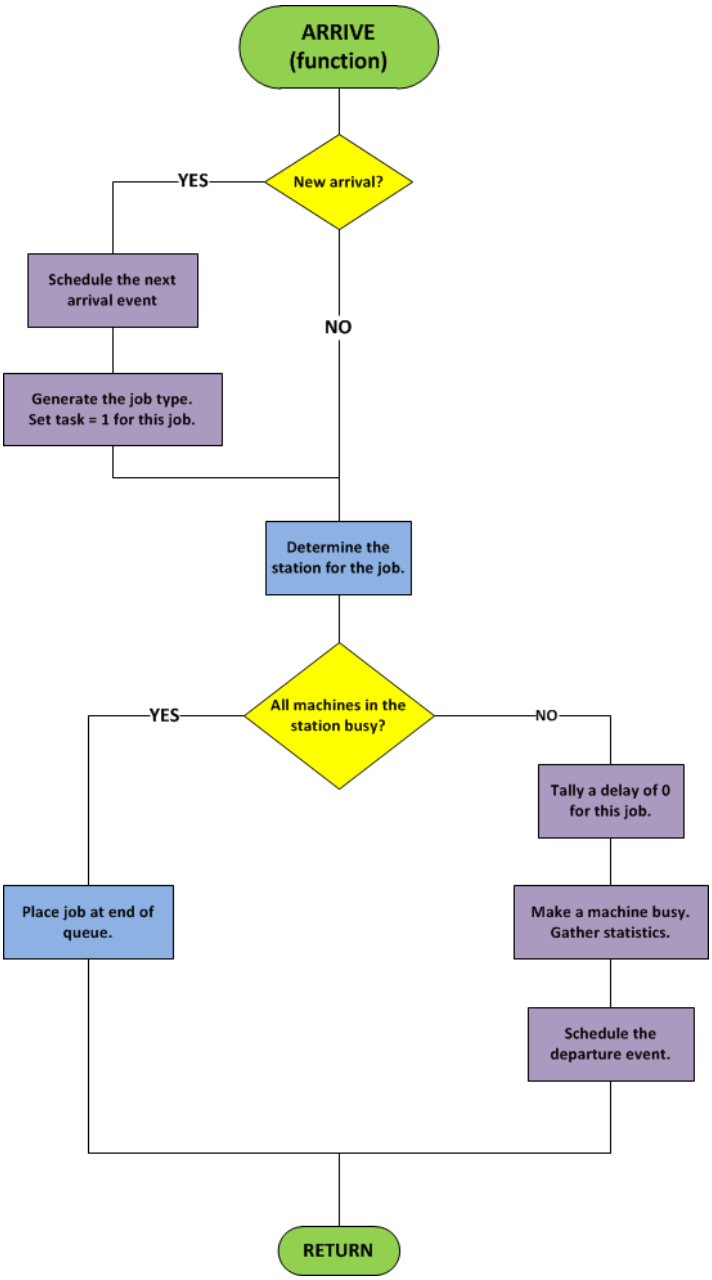


Figure 54 Job-shop arrival event

The tasks arrive randomly at the shop (according to a Poisson distribution) and each type of tasks requires a fixed sequence of operations that must be performed by a specific and pre-determined type of machine with a deterministic duration. The flowchart for the arrival event is depicted in Figure 54.

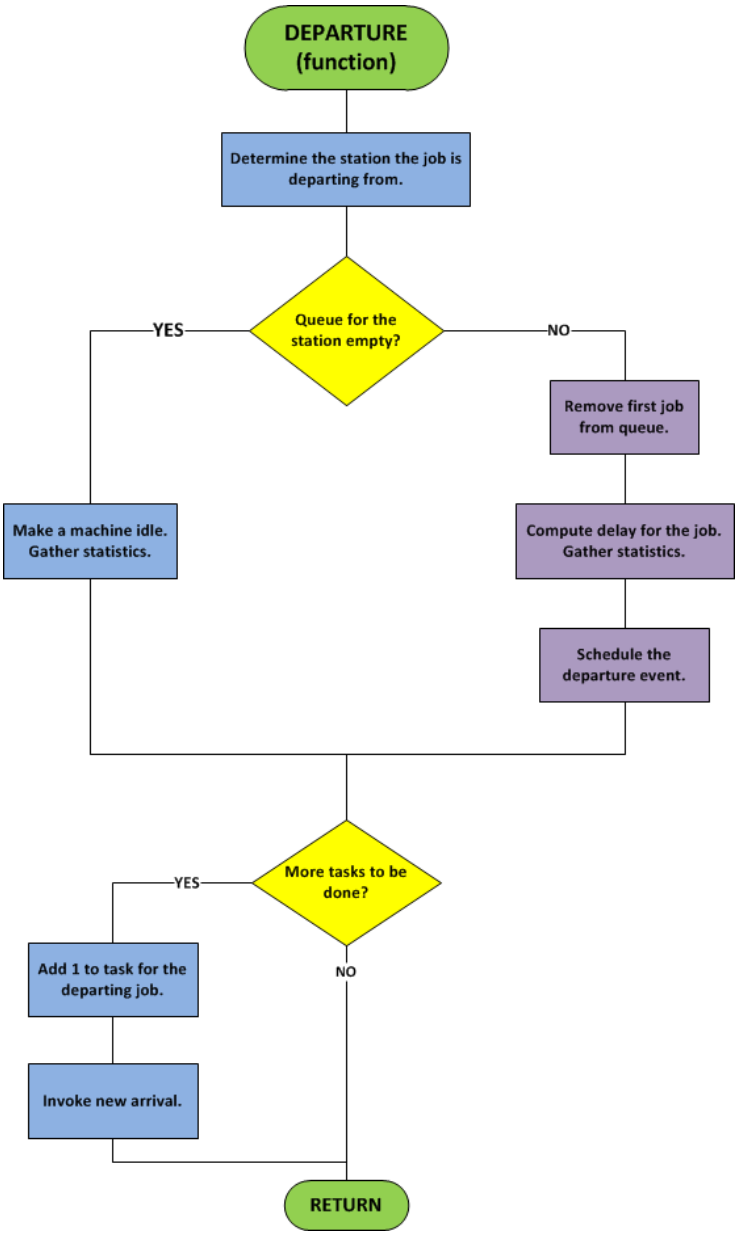


Figure 55 Job-shop departure event

Each group of machines is viewed as a resource whilst the different types of tasks being executed in the Job-Shop are viewed as objects of a class. The tasks circulate in the shop and an action method of its class describes their behavior from the arrival until its departure. For each operation the appropriate type of machine is requested as well as the duration of the operation. When the task is finished it collects its time in the system. The procedure is depicted in Figure 55.

The example is then able to generate individual reports for both the machines in the job-shop and the tasks performed. Following is an example of such reports outputted after a simulation run (see Figure 56).

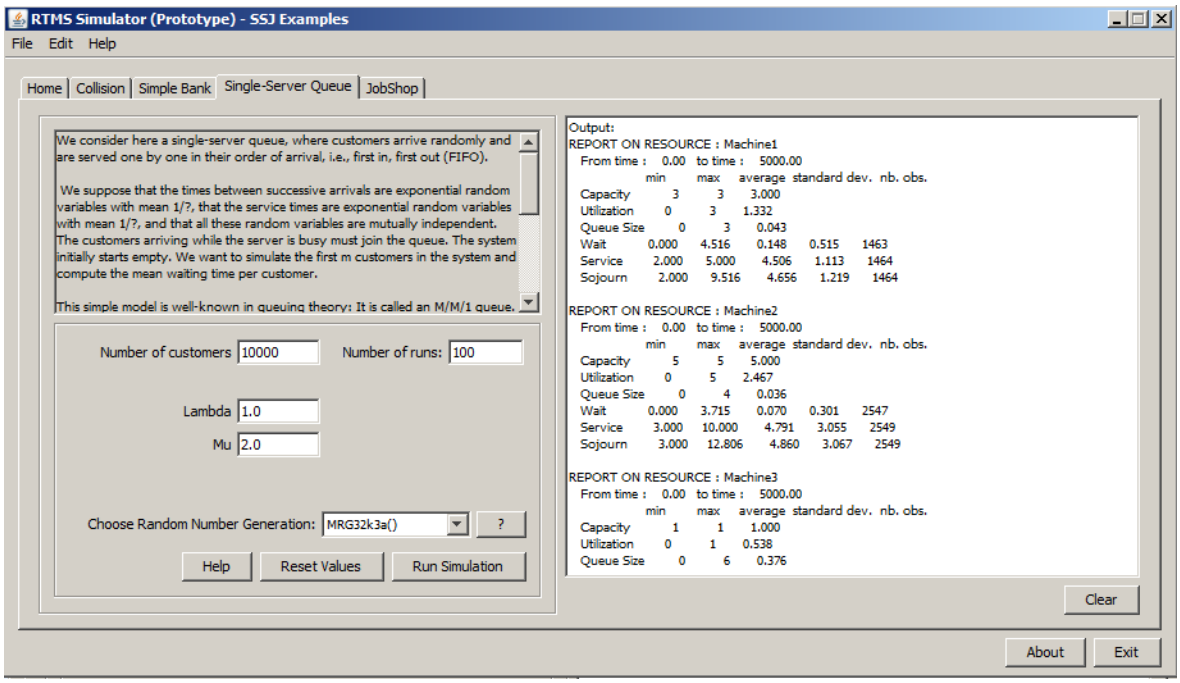


Figure 56 SSJ Examples' GUI showing the Job Shop example interface

5.1.5. FUNCTIONAL REQUIREMENTS SPECIFICATIONS

The functional requirements of this work are based on the motivations presented in section 1 and what is expected from the prototype. Table 12 presents the functional requirements set as goals for the design of the prototype and defines their priorities.

Table 12 - Functional requirements

Number	Requirement	Priority
1	Create simulation info plots	High
2	Edit Jobs' info	High
3	Edit Stations' info	High
4	Graphical engine for animation	High
5	Stop, pause and reset animation	High
6	Add, remove and re-order Jobs	Medium
7	Add, remove and re-order Stations	Medium
8	Control animation speed	Medium
9	Live-stream output	Low
10	Random-number generator for part arrival times	Low
11	Random-number generator for service times	Low

5.1.6. SYSTEM'S DESIGN

The system is, as expected, divided in several classes. Figure 57 details the hierarchy of the several modules used to build the program.

There are four main modules, *RTMSS* (described in Figure 57 as *RTMSS Prototype*), *SimGui* where the main simulation window, as well as all the visual components, is drawn and the information is handled, *DrawSim* where the information gathered and passed on by *SimGui* is handled, organized and converted into the animation procedures, *Report* is responsible for outputting the simulation results and charts and *ExamplesGui* the secondary interface already mention in section 5.1.1.

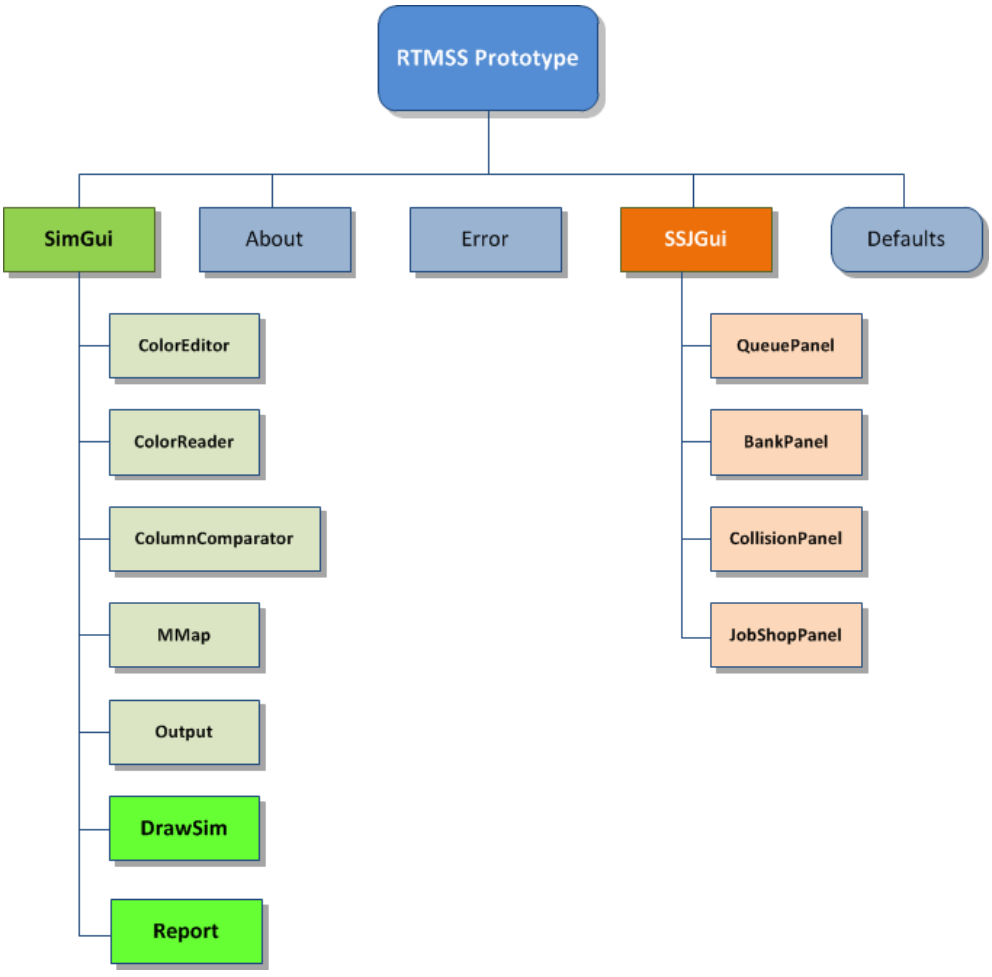


Figure 57 RTMSS' system architecture overview

To describe the structure of the system, its classes, their attributes, operations and the relationships among the classes will be presented as follows as class diagrams according to the Unified Modeling Language (UML). Some diagrams, due to its class's complexity,

have been shortened, however, full listing of all the application classes UML diagrams are listed further on Appendix B.

5.1.7. RTMSS MODULE

This is the main class for the application, where the initial interface is drawn. Figure 58 shows the RTMSS Class Diagram.

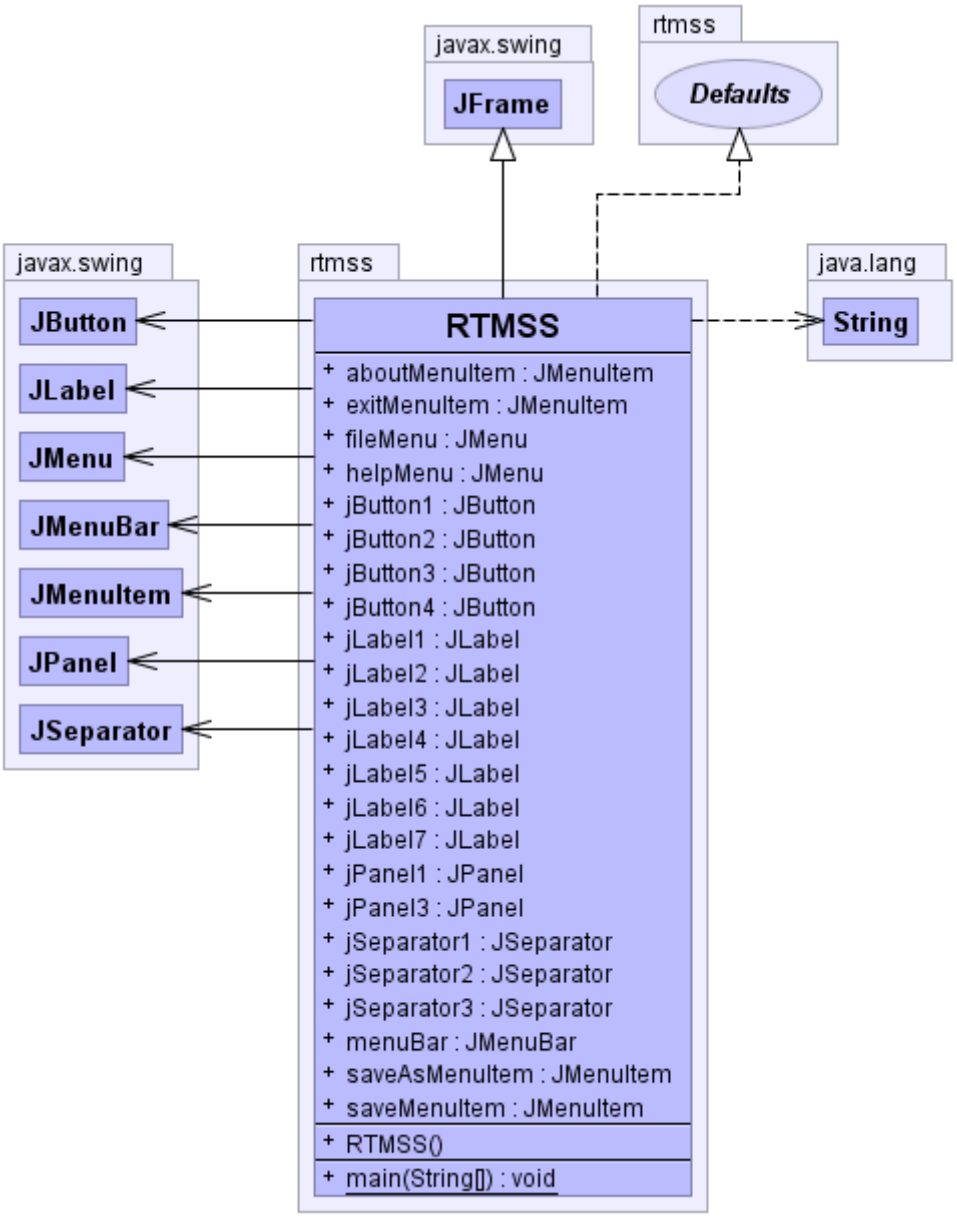


Figure 58 RTMSS Class Diagram

5.1.8. SIMGUI MODULE

As mentioned, this class is responsible for the handling of the simulation data and the visual output through the animation panel. Figure 59 shows the SimGui Class Diagram.

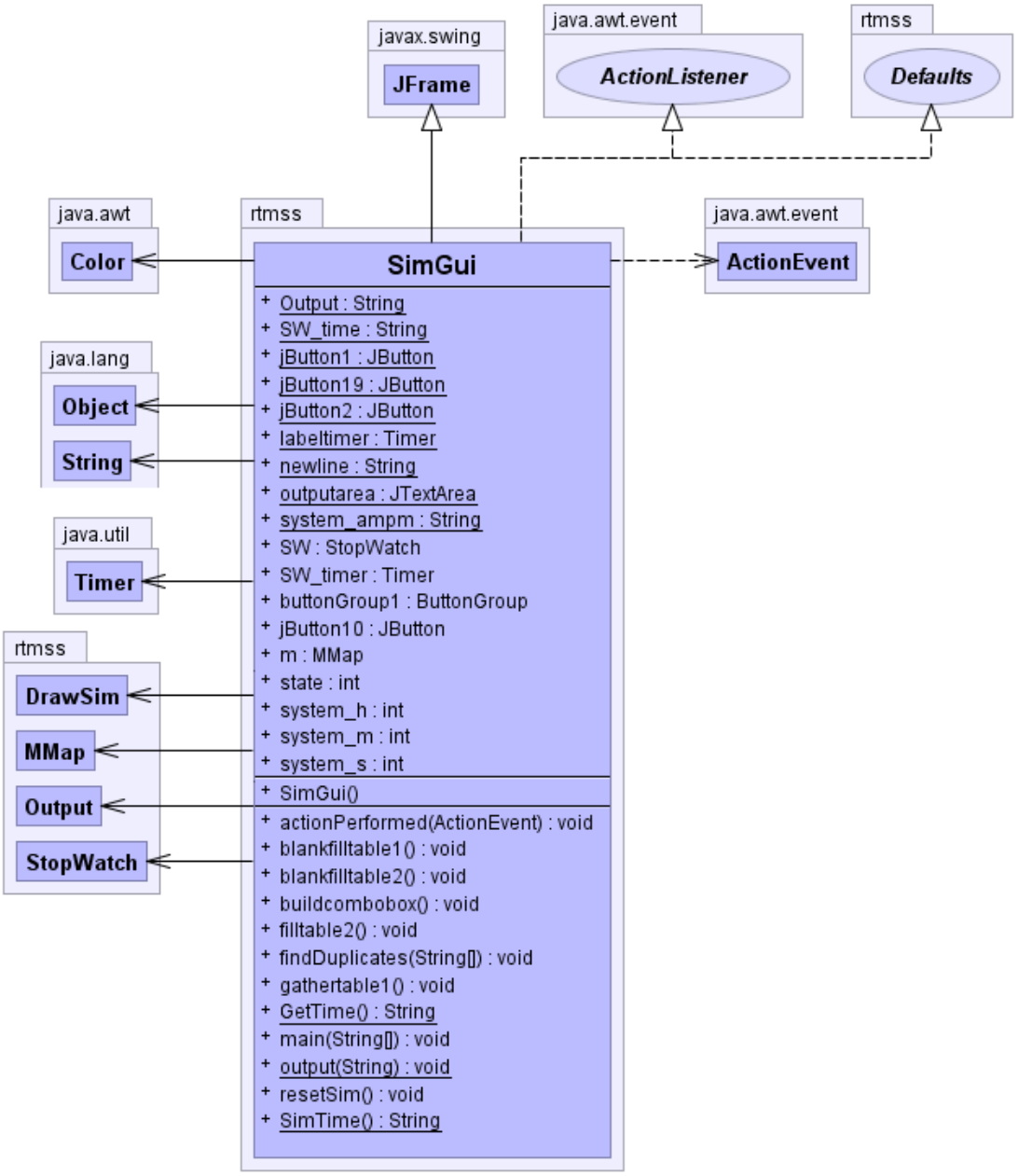


Figure 59 SimGui Class Diagram

5.1.9. DRAWSIM MODULE

This is the class responsible for the drawing of the components in the animation panel displayed in SimGui's interface. Figure 60 shows the DrawSim Class Diagram.

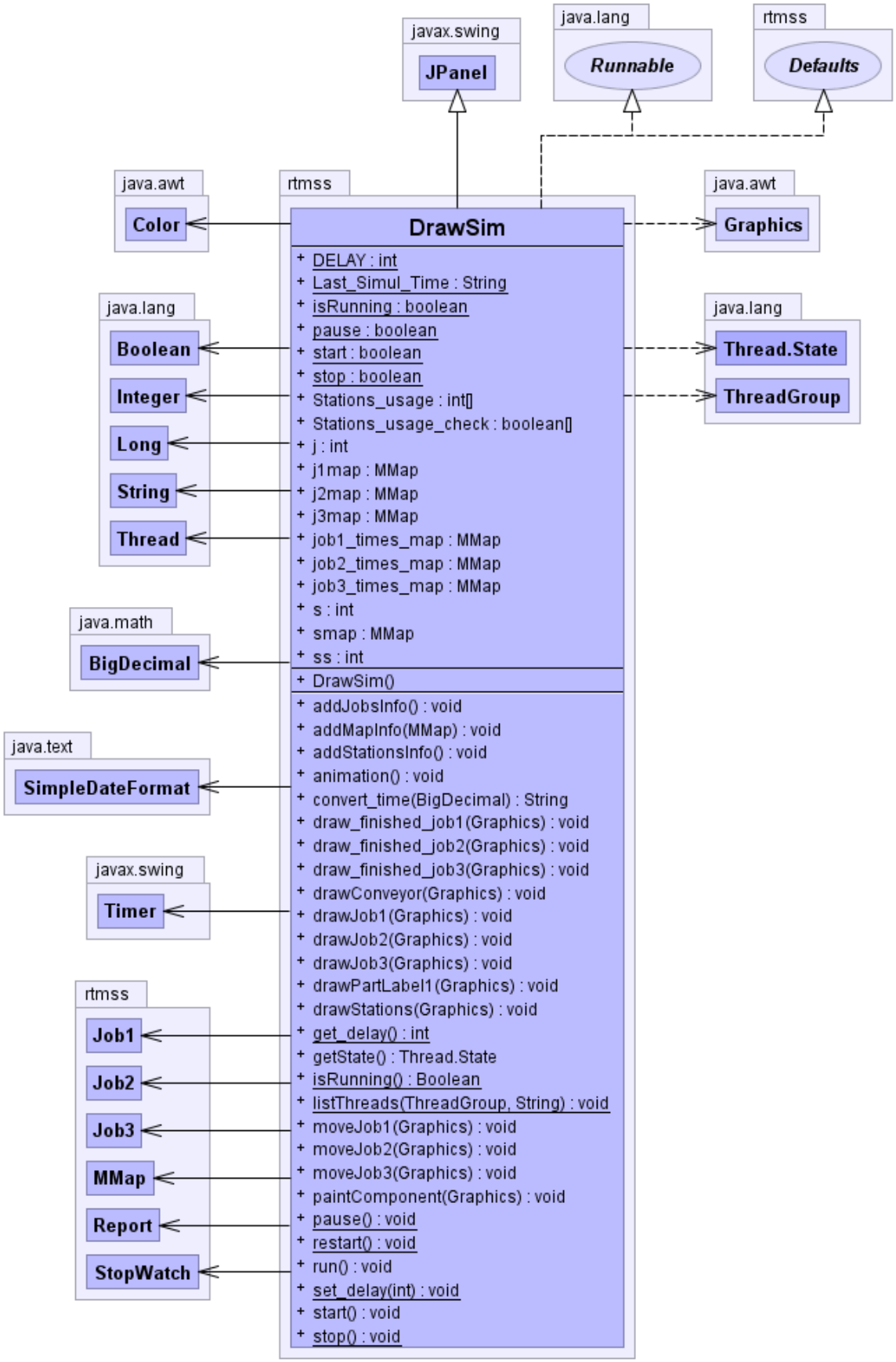


Figure 60 DrawSim Class Diagram

5.1.10. REPORT MODULE

The report class generates the interface where the simulation results are presented. The times for the job parts as well as its totals and averages are collected and some charts are drawn as will be presented further in the document. Figure 61 shows the Report Class Diagram.

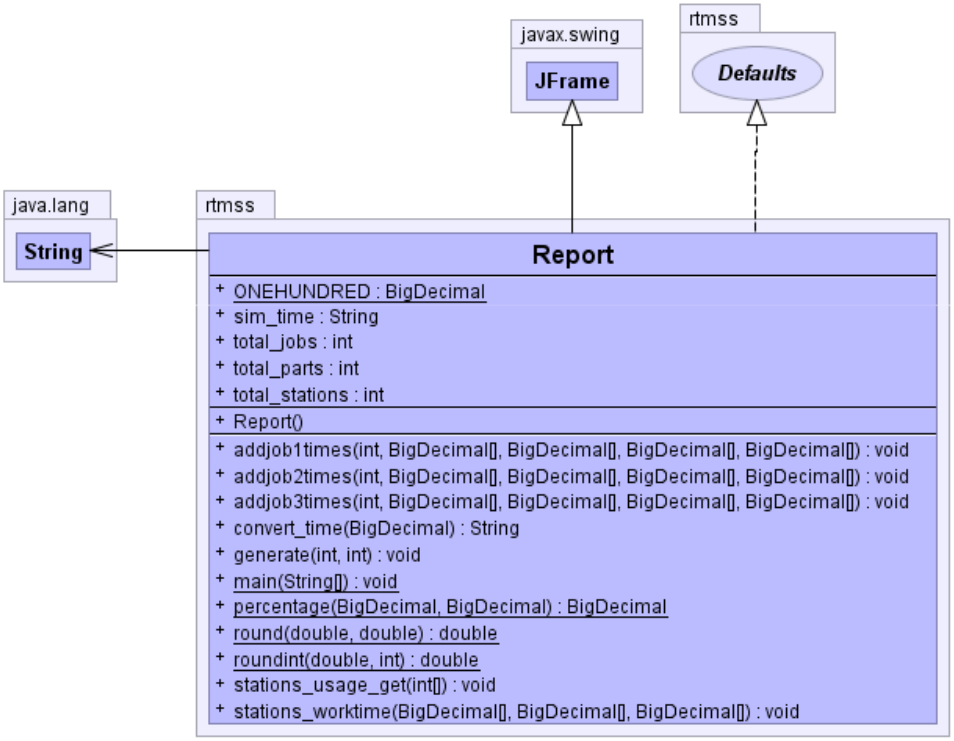


Figure 61 Report Class Diagram

5.1.11. ABOUT MODULE

As the application might be distributed or used for academic purposes an *About* dialog was created containing relevant information about both the project and the author. Figure 62 shows its Class Diagram.

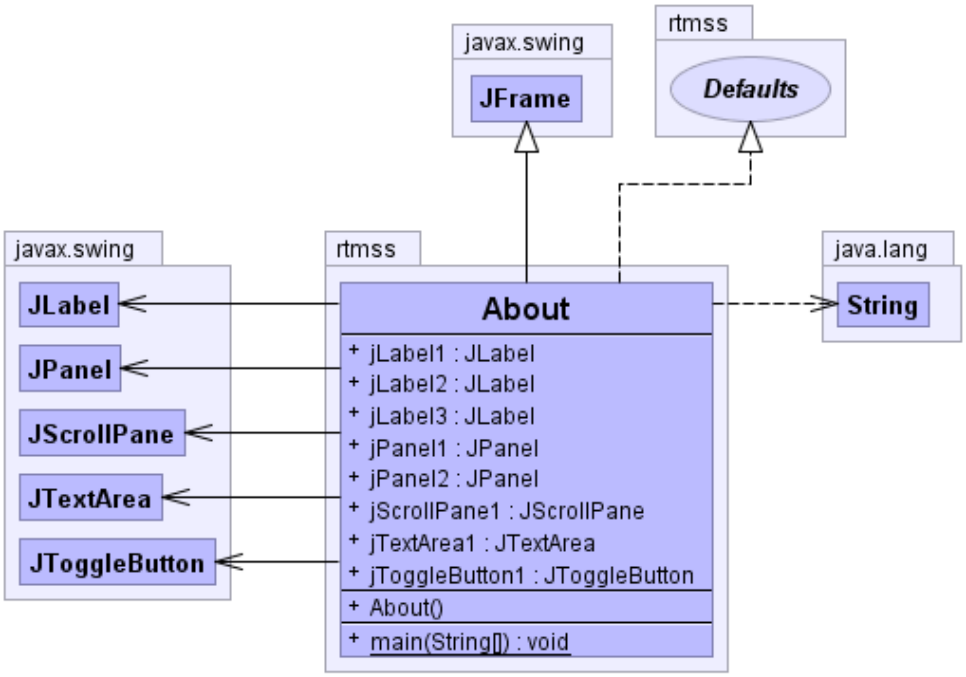


Figure 62 About Class Diagram

5.1.12. ERROR MODULE

An error dialog was also created to inform the application users of invalid actions and operations. The messages can be easily altered as they are defined in the *Defaults* interface and are easily changeable.

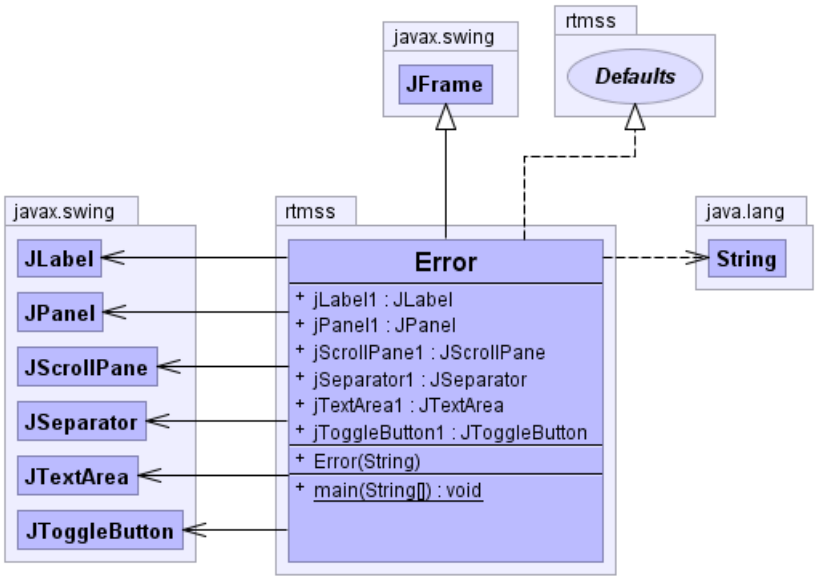


Figure 63 Error Class Diagram

5.1.13. SSJGUI MODULE

This class creates the previously detailed interface to run the SSJ based examples.

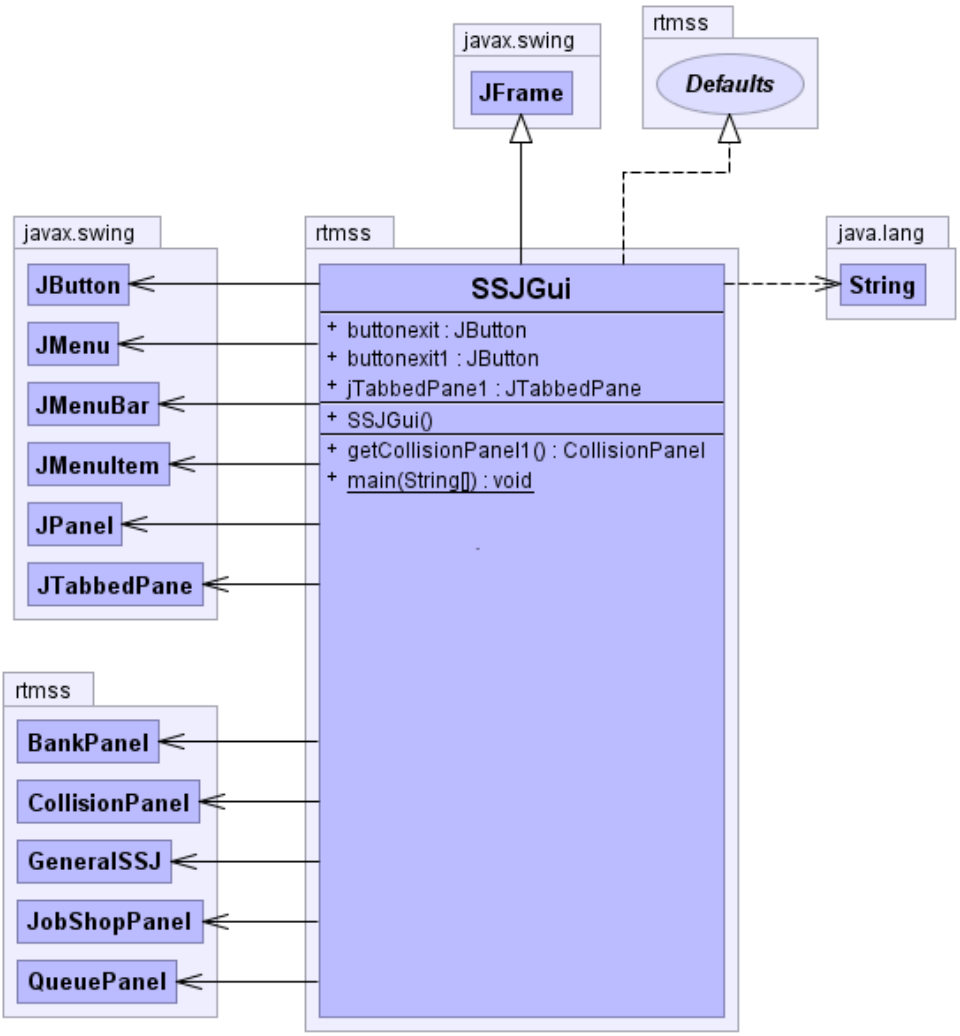


Figure 64 SSJGui Class Diagram

5.1.14. APPLICATION INTERFACE

The interface of the application to allow the interaction and configuration of the simulation models was designed by using the Java Swing library (*javax.Swing*). Java Swing is an application programming interface (API) for developing GUI's for programs. The GUI of Java applications are based on windows which contain several components, i.e. buttons, labels, textboxes, and tables. A window in Swing is represented by a JFrame but it does not have the characteristics needed to draw components on it, in order to do so, it is necessary to add panes or panels (*JPanel*) to it, a Java object representative of a surface in which components can be drawn. Oracle in its *Visual Guide to Swing Components* provides a good representation of the windows' architecture, as seen in Figure 65.

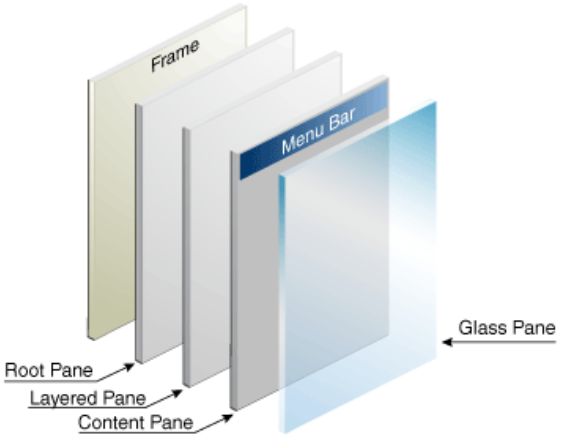


Figure 65 Java window layered architecture.

As mentioned previously the IDE used to develop the application was *NetBeans* v.7.1.2. It provides a significantly improved performance and coding experience and makes the creation of the user interfaces easier with the use of the *NetBeans Swing GUI Builder*. It is possible to design Swing GUIs by dragging and positioning GUI components from a palette onto a canvas as show in Figure 66.

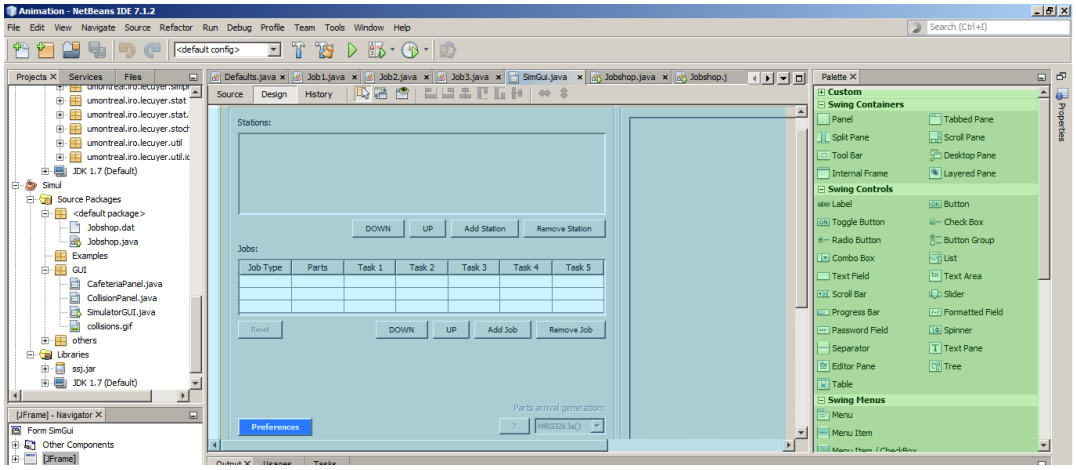


Figure 66 NetBeans IDE. Highlighted in green, the Swing components palette, in blue the design canvas.

The Simulator GUI was designed to offer a user friendly interface as is depicted in Figure 67 (image with increased size and resolution can be seen in Appendix C). The windows' look and feel chosen was the *SystemLookandFeel* which guarantees that the application will use the native look and feel of the system it is running on. Swing is designed to allow the change of the appearance of the GUI components, defined by the *LookandFeel* parameter.

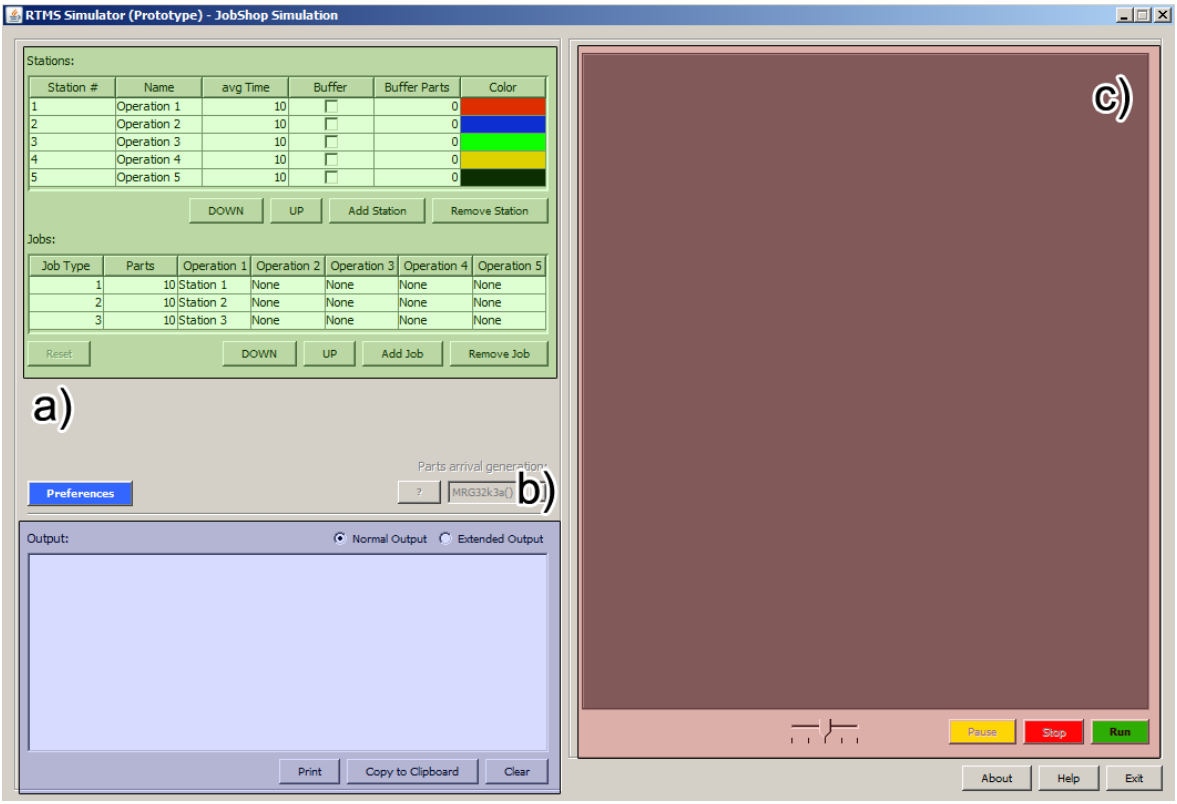


Figure 67 RTMS Simulator GUI a) Stations and jobs configuration parameters (green) b) output area (blue) c) animation drawing panel and simulation commands (red)

Four different sections can be identified on the GUI, two where the user can edit the configuration parameters regarding the stations and the jobs that will be processed in the system, another where the outputs are generated and finally the animation section where the animation of the simulation is processed. To edit the stations and jobs two tables are presented (see Figure 67.a). Regarding the stations' parameters users can edit:

- Station name;
- Average service time;
- Buffer and number of buffer parts;
- Station color;

As for the Jobs', users can define the number of parts as well as the tasks precedence and sequence for the jobs known, for instance, parts from *Job 1* will go to *Station 1*, then *Station 4*, then *Station 3* and are finished.

For now, the number of possible stations and jobs is limited to five and three respectively. On the other hand, all the buttons are functional and allow users to add and remove stations or jobs (if the limit is not reached) and to change their order. A button to reset the values back to the default ones is also present. Also, the simulator does not allow for a user to set

the same name for two different stations neither will it allow to allocate the same station to two consecutive tasks for a job.

An error dialog was created to generate error messages informing the user that an invalid action was performed, as is depicted in Figure 68.

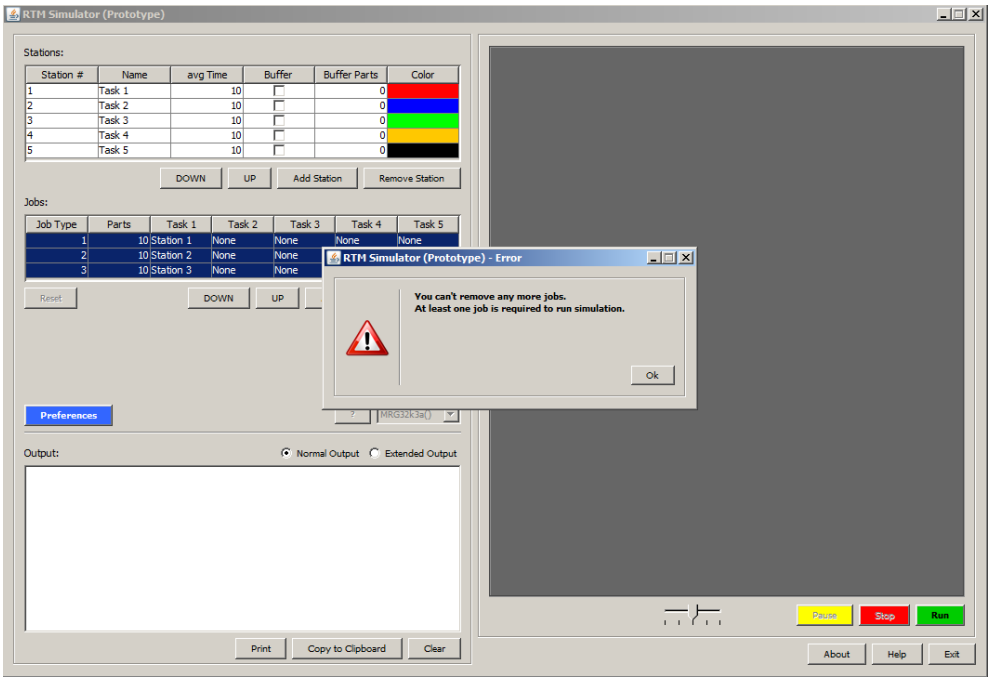


Figure 68 - Error dialog.

As shown, the user tried to remove all the jobs and the simulator did not allow the action as at least one job is required to run a simulation. This message, as well as all others, are easily configurable through a default values interface where all the messages are stored along with other program configurations. This was thought of to, in the future, facilitate the creation of a translation engine to display all the text present in different languages.

As for the output, a text area was created were the relevant information from the simulation run is displayed as is depicted in Figure 67.b.

The user can select to view the default normal output or the extended output (see Figure 69) where all the detailed info from the run is displayed. Besides, the user is presented with functional buttons which allow clearing the information, copying it to the clipboard or printing it.

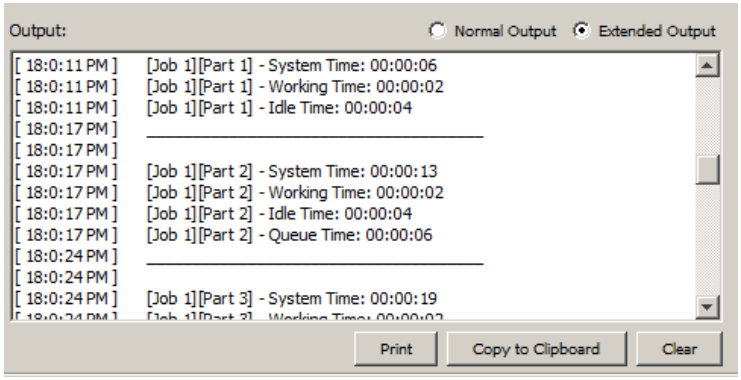


Figure 69 - Output text area displaying the extended output information from a simulation run.

Considering that the application might be distributed or used for academic purposes, an *About* dialog was also designed (see Figure 70) where the basic information regarding the project and the author is presented (bigger resolution image at Appendix.C).

Figure 70 - About dialog for the RTMSS with relevant information regarding the project and the author.

The most important component of the prototype is the animation engine in which the simulation runs are drawn and animated. From now further, the examples presented derive from a simulation run where all three jobs are executed (*Job 1* with operation at *Station 1*, *Job 2* with operation at *Station 2* and *Job 3* with operation at *Station 3*) containing ten parts, with *Work Time* of one second at each attributed station.

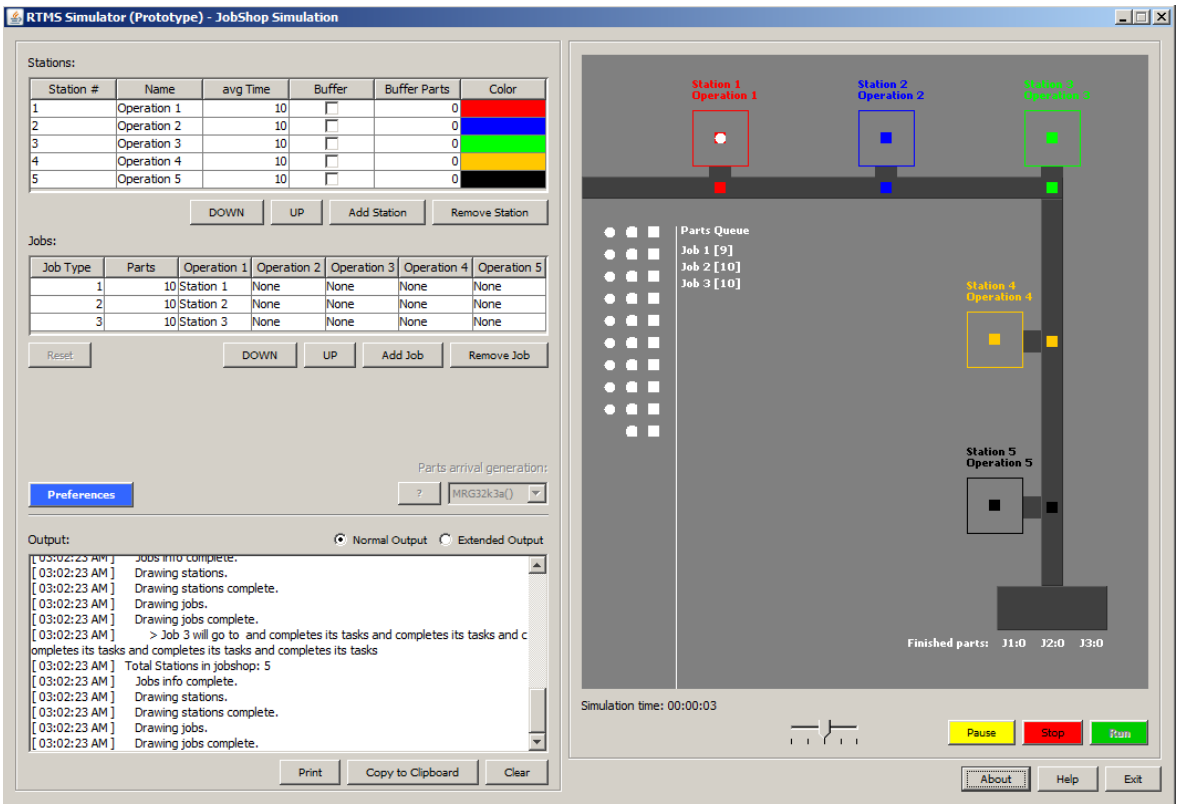


Figure 71 - Animation drawing of a simulation run.

The engine was entirely coded using the Java 2D API which provides several classes that define common geometric objects such as points, lines, curves, and rectangles. It was intended to be as simple and intuitive as possible from the visual point of view (as seen in Figure 71), so the use of images and icons was set aside. Therefore, all the stations are drawn as rectangles with the according color defined by the user and the parts for the jobs can be either squares, circles or rounded rectangles as is depicted in Figure 72.

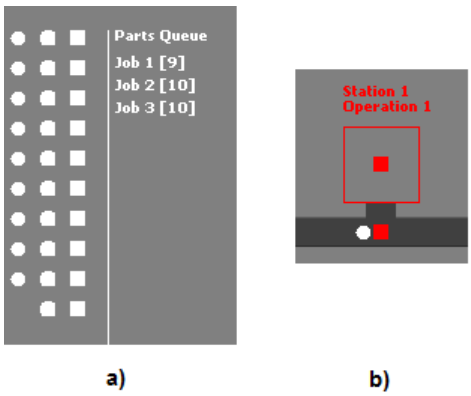


Figure 72 a) Job parts details showing the different shapes. b) Station drawing details.

In Figure 72.a) it is seen that only 10 parts are drawn on the waiting queue. To avoid cluster of the animation area in cases where the user is simulating jobs with more than ten

parts, only ten will be drawn and as soon as they start to enter the system they leave that queue. In the image we can see that one of the ten parts from *Job 1* has already entered the system and is being transported for operation on the conveyor (see Figure 72.b).

The procedure for the animation is fairly simple, when a part is finished and has been stored in the finished parts batch, another part joins the system. As it enters the conveyor it is moved through the production line and as it passes the station where it should stop to execute a task it is moved to that particular station. In Figure 73 we can see the finished parts batch in the lower right corner, where there are all the ten parts for *Job 1* finished, and *Job 2* is currently undergoing with a part being handled in *Station 2*. The system was programmed to only allow one *Job* type at a time, with only one part on the conveyor but it is possible to have all jobs running at once with little modifications as there was attention to make sure concurrency was possible as will be explained further.

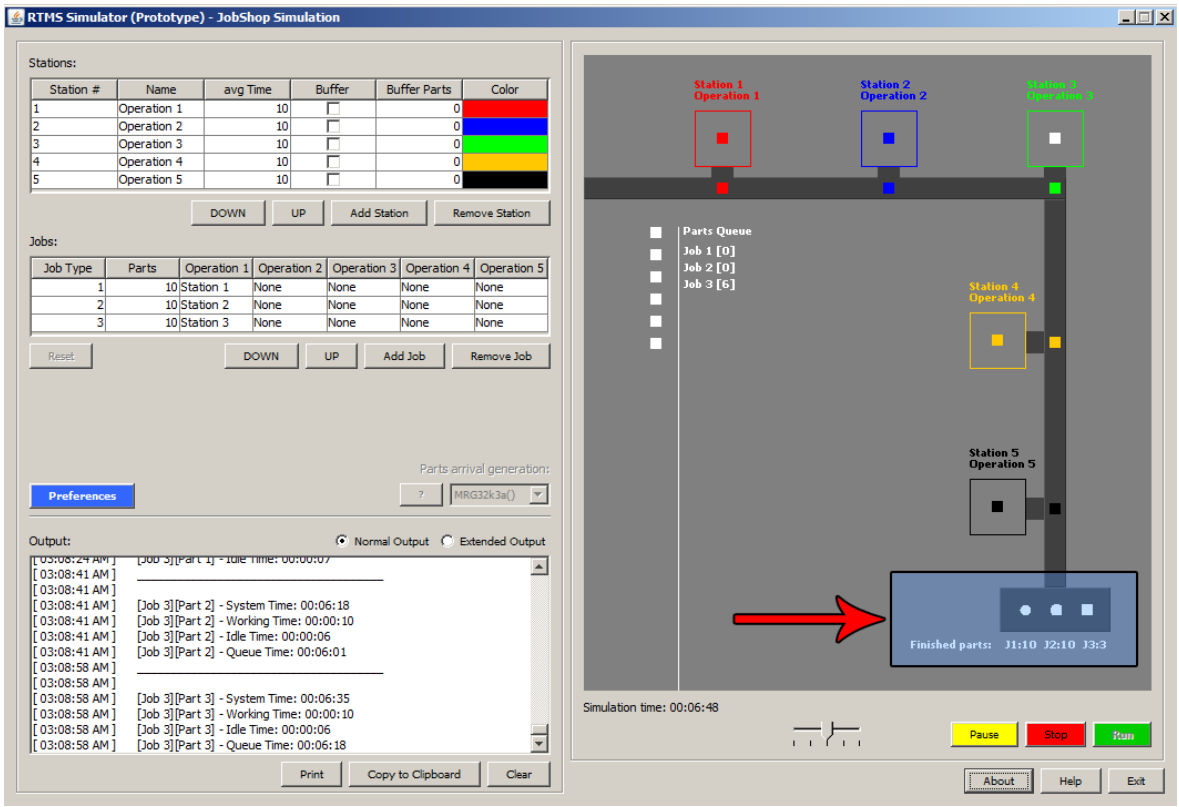


Figure 73 Animation of a simulation run.

As referred, the application was developed to guarantee that concurrency was possible. Concurrency is the ability to process several parts of a program or several programs in parallel, which can highly improve the throughput of a program if certain tasks can be performed asynchronously or in parallel, in this case, executing all the jobs simultaneously

if desired. In concurrent programming, there are two basic units of execution: processes and threads. In the Java programming language, concurrent programming is mostly concerned with threads. Threads are sometimes called lightweight processes. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process. In this application, multithreading is present as a new thread is assigned for each job. From a logical point of view, multithreading means multiple lines of a single program can be executed at the same time, therefore, multiple jobs can be executed at the same time if intended.

As soon as all parts from all Jobs have been processed, the queue is erased and all the parts have its place in the finished parts batch, as is depicted in Figure 74. As is expected from simulators, as mentioned previously (see section 4.6.1), the capability of gathering statistics and generating reports should be present. Taking that into consideration, when the simulation run is concluded a report is presented with some gathered statistics and some charts regarding the system information.

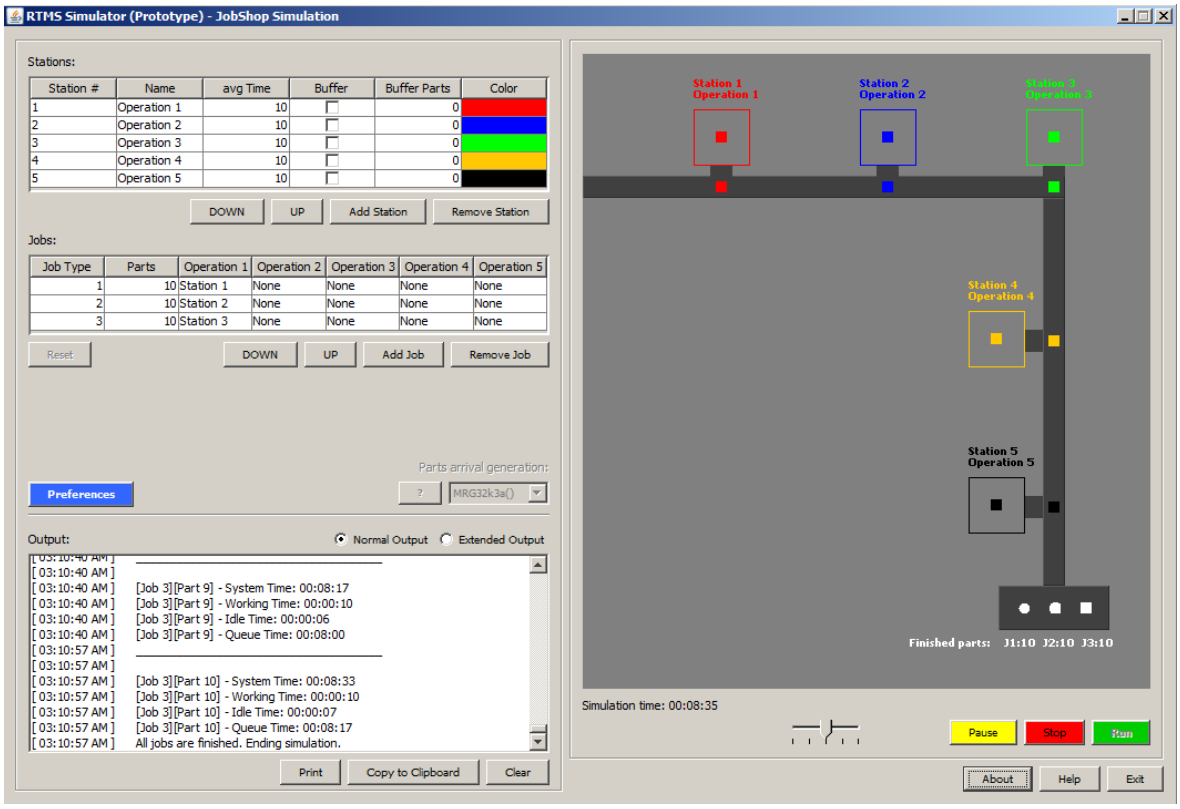


Figure 74 Finished animation of a simulation run.

The *Report* interface is simple and clear, at first the times collected for the parts from the jobs processed in the system are presented (see Figure 75). For each job a table presents the *System Time*, *Idle Time*, *Work Time* and *Queue Time* for each part. Also, the average

times for each job are also presented as well as the overall statistics such as *Total Parts Processed*, *Total Jobs Processed*, *Total Stations*, *Total System Time*, *Total Working Time*, *Total Idle Time*, *Total Queue Time* and *Simulation Time*.

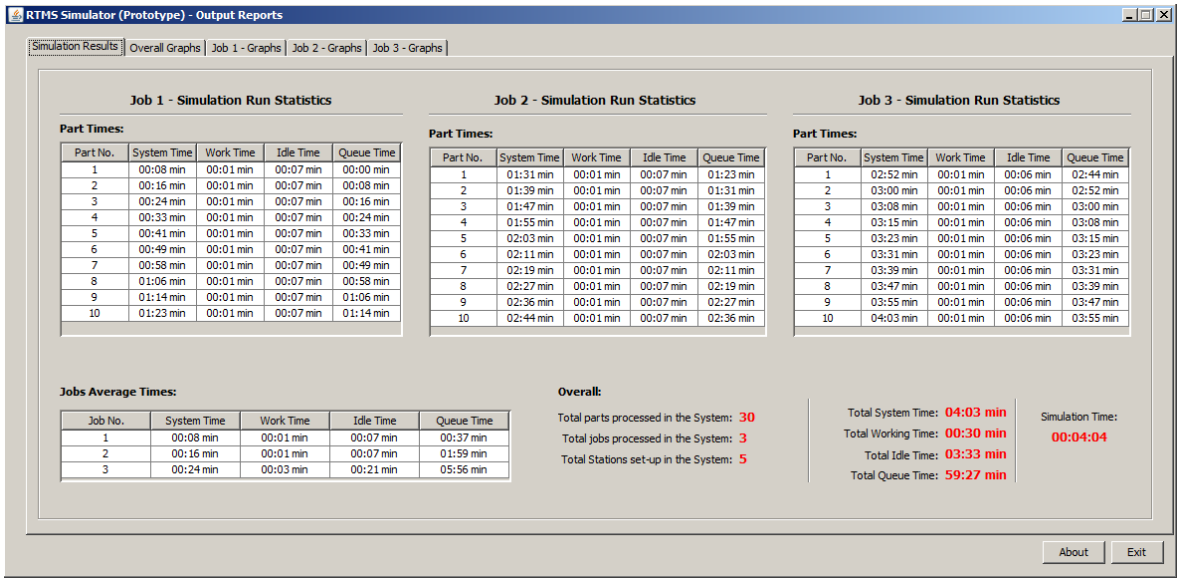


Figure 75 RTMSS Output Report interface.

As soon as the simulation starts, the parts are marked as being in the system, each having its own counter for their *System Times*. As soon as they enter on a station to perform a designed operation their *Work Time* counter is activated and incremented. The *Idle Time* counter discriminates the time parts spend traveling between operations and the *Queue Time* is, as it implies, the time parts spend in queue waiting for service. Figure 76 shows the interface in detail, where the times for the *Job 1* are presented as well as the average times for all three jobs.

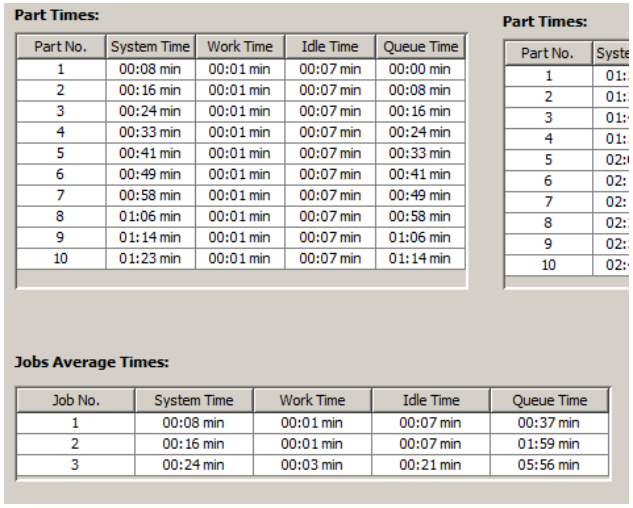


Figure 76 Detail of the output report interface collected statistics.

As an example of possible output information that can be present in simulation reports some charts are generated and presented in the different tabs. First, in the *Overall Charts* tab, two charts are drawn the first (see Figure 77.a) shows the *Stations Usage* as for the second (see Figure 77.b), the *System Time* distribution.

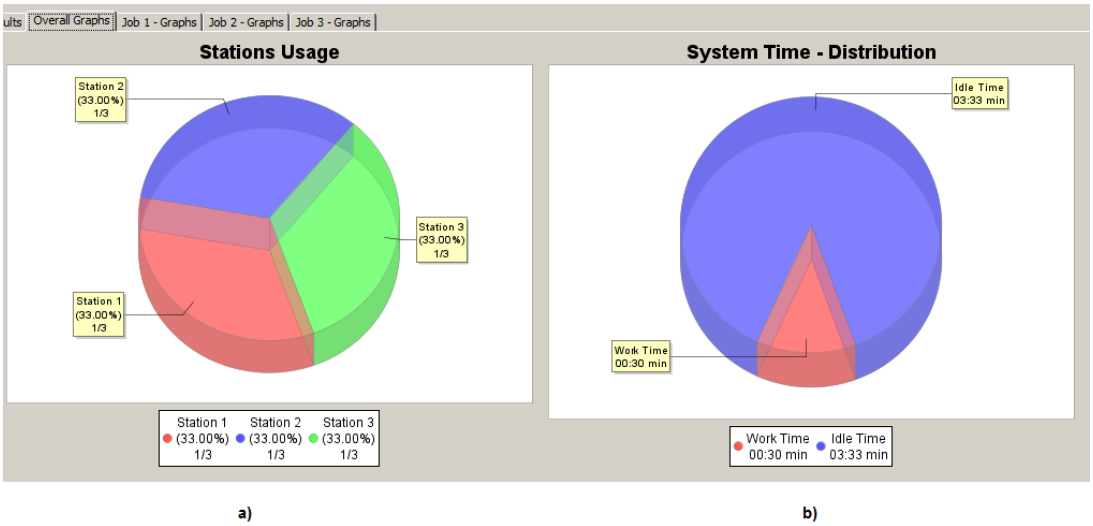


Figure 77 Output Report interface showing two of the generated charts.

As visible in Figure 77.a, only *Station 1*, *Station 2* and *Station 3* where used in the portrayed example, therefore, each station being used only once will have an overall usage of approximately 33.00% (1/3). Also, in Figure 77.b, the time overall time distribution is presented. In a total simulation run time of 04:04 *min*, only 00:30 *min* were spent actually operation on parts, as for the remaining 03:33 *min* correspond to the *Idle Time*. Times differ by a second due to rounding.

Another type of chart is also generated to display the stations occupation for each job processed (see Figure 78). Again, this is a simple example of possible output information generation but several other charts are be plausible to be presented according to the detail reached in the simulation application. In the chart shown, the occupation of Station 1, the only station where operations for Job 1 occurred, is of 12% for the Work Time and 88% for the System Time, thus meaning that it only spent 12% of the time available to finish all parts processing for Job 1 operating, whilst the rest of the time it was idle.

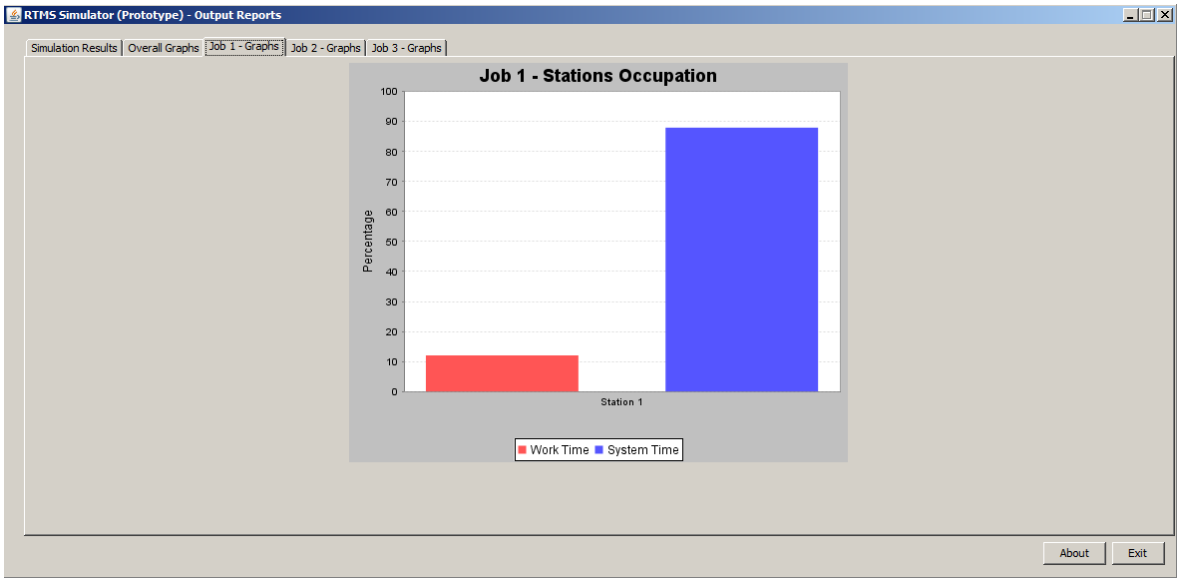


Figure 78 Output Report interface showing the stations occupation chart regarding *Job 1*.

Several examples with different amount of detail and complexity regarding the tasks to be performed will be presented further on section 6 so it will be possible to observe the differences in the outputs generated in the *Report* interface.

5.2. CONCLUSIONS

This chapter describes the setup used, and the process of prototype developing, relating:

- The basis behind all the work developed;
- The prototype design and features;
- Details of the SSJ examples available for experimentation in the prototype;
- The system architecture with a description of the modules implemented;
- A step-to-step walkthrough of all the features present in the application were detailed, as well as all the designs implementations.

6. EXPERIMENTAL RESULTS AND DISCUSSION

This chapter describes the experimental study based on case studies (Single Server Queue, Bank and Job-Shop) and on Real Time Manufacturing Systems.

Four case studies were analyzed with two different scenarios (default and alternative parameter configurations).

For the Real Time Manufacturing Systems several scenarios with different model configurations were evaluated as well as the generated outputs and reports.

6.1. SSJ EXAMPLES EXPERIMENTAL RESULTS

This subsection describes the experimental study based on the different case studies with different configurations. All case study model examples were detailed previously in sub-sections 5.1.2, 5.1.3 and 5.1.4.

6.1.1. SINGLE SERVER QUEUE EXAMPLE WITH DEFAULT CONFIGURATION

The model presented in this example, as well as all others were previously detailed so we will focus on the proposed prototype, possible configurations for the examples and

obtained results. It is important to state the different configuration parameters that can be modified when running the model simulation

In the particular case, for the single server queue with Lindsey's recurrence, users can define the number or runs to simulate, the number of customers that will arrive at the queue and the λ and μ values responsible for the mean times of the exponential random variables generated for the randomization of the customers arrivals as seen in Figure 79.

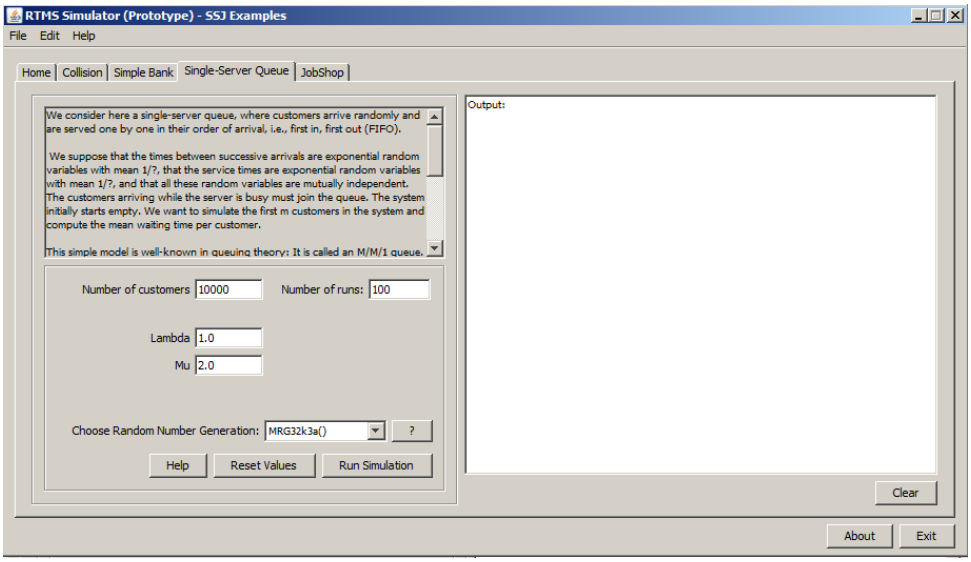


Figure 79 Single Server Queue example interface with default configuration

Also, the random number generators can be modified. In all examples a thorough explanation of the model as well as the parameters effects are provided.

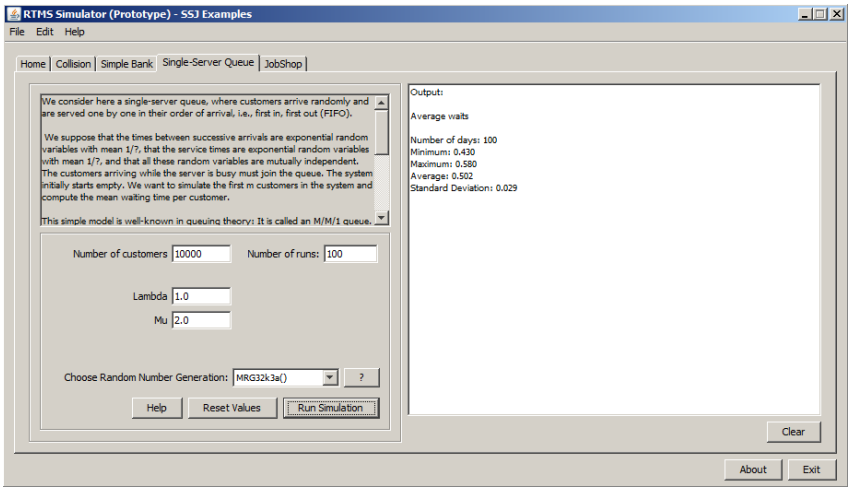


Figure 80 Single Server Queue example interface with default configuration and the generated output

In this example the simulation run will output the average wait, in hours, for the number of days previously set as well as the minimum and maximum wait values and its standard

deviation. For the standard values provided, the output values generated are presented in Figure 80.

6.1.2. SINGLE SERVER QUEUE EXAMPLE WITH ALTERNATIVE CONFIGURATION

As example of alternative configurations, three runs took place to demonstrate the impact of the different possible changes in the model. First, a simulated run with reduced number of customers joining the queue (dropped from 10000 to 5000) whilst maintaining the parameters for the distributions responsible for the arrivals at the queue, produced the results show in Figure 81.

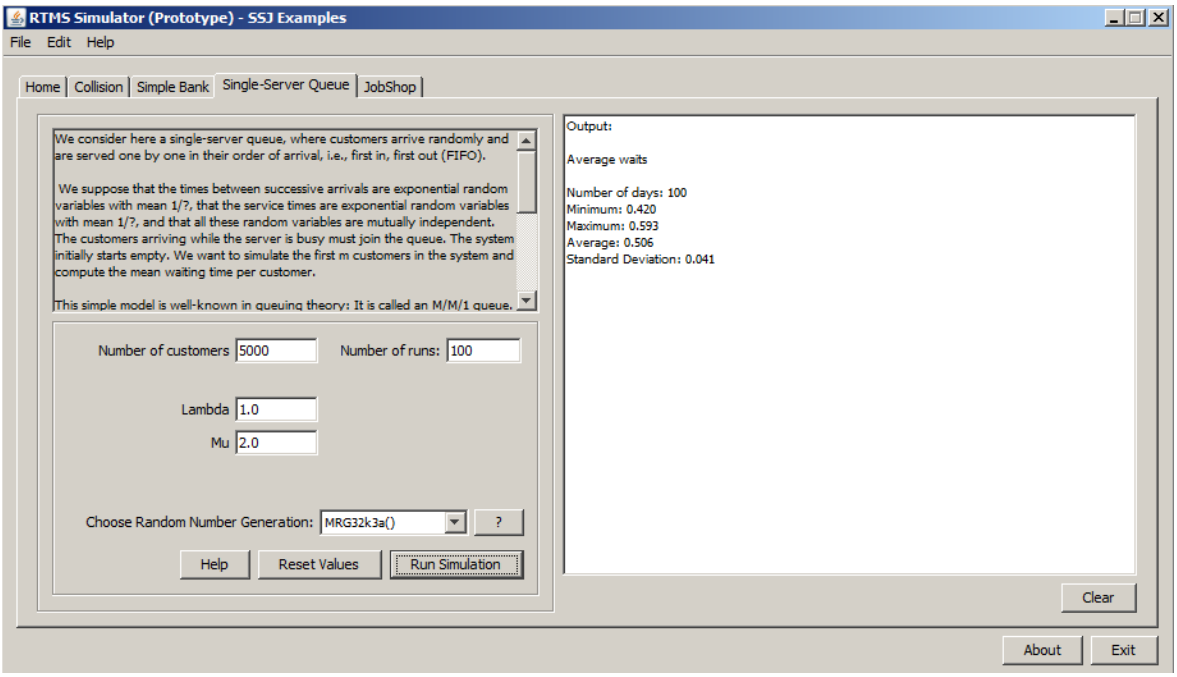


Figure 81 Detail of the output values from the single server model simulation run with 5000 customers

A second run, with the increased number of customers (raised from 10000 to 15000) reveals that the average waits in the queue barely produce any changes at all (see Figure 82). When comparing all three outputs from the simulation runs in this model, it is clear that the changes in the customer's number barely had any impact in the average waiting on the queue, the only noticeable variation being notable in the standard deviation of the waiting times.

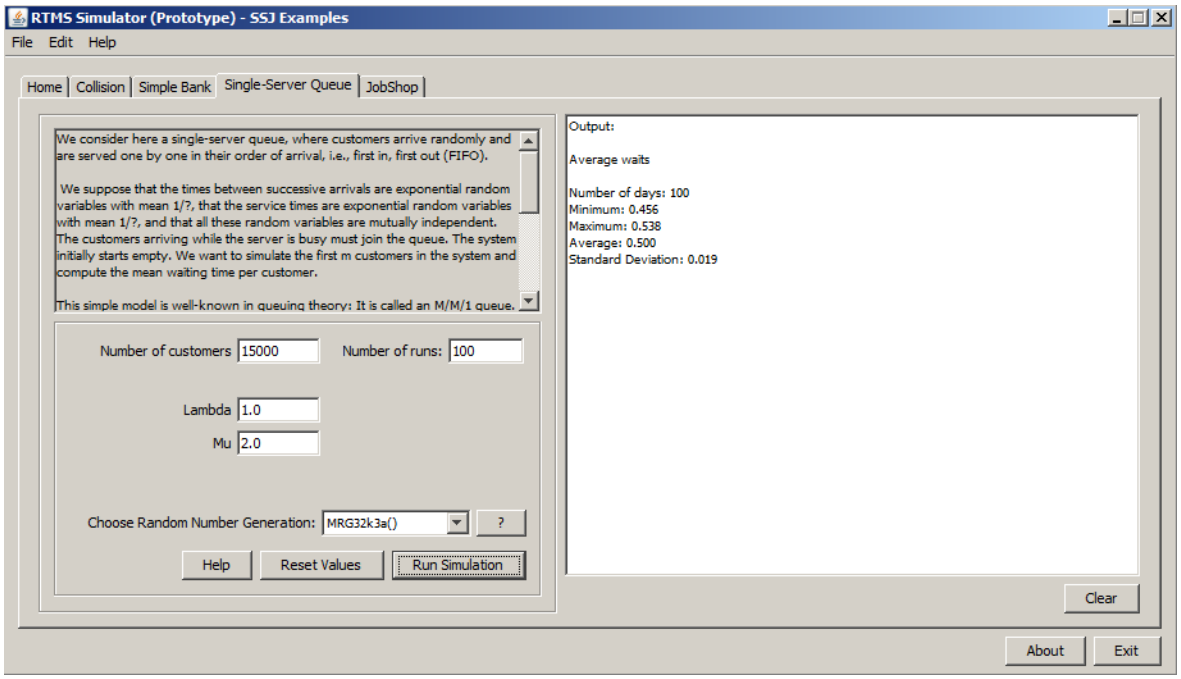


Figure 82 Detail of the output values from the single server model simulation run with 15000 customers

When it comes to change the parameters for the exponential distributions that generate the customers arrival times at the queue, a simulation run with both λ and μ set at 1,5 reveals that different mean times for those distributions will have a major impact in the waiting times as is depicted in Figure 83.

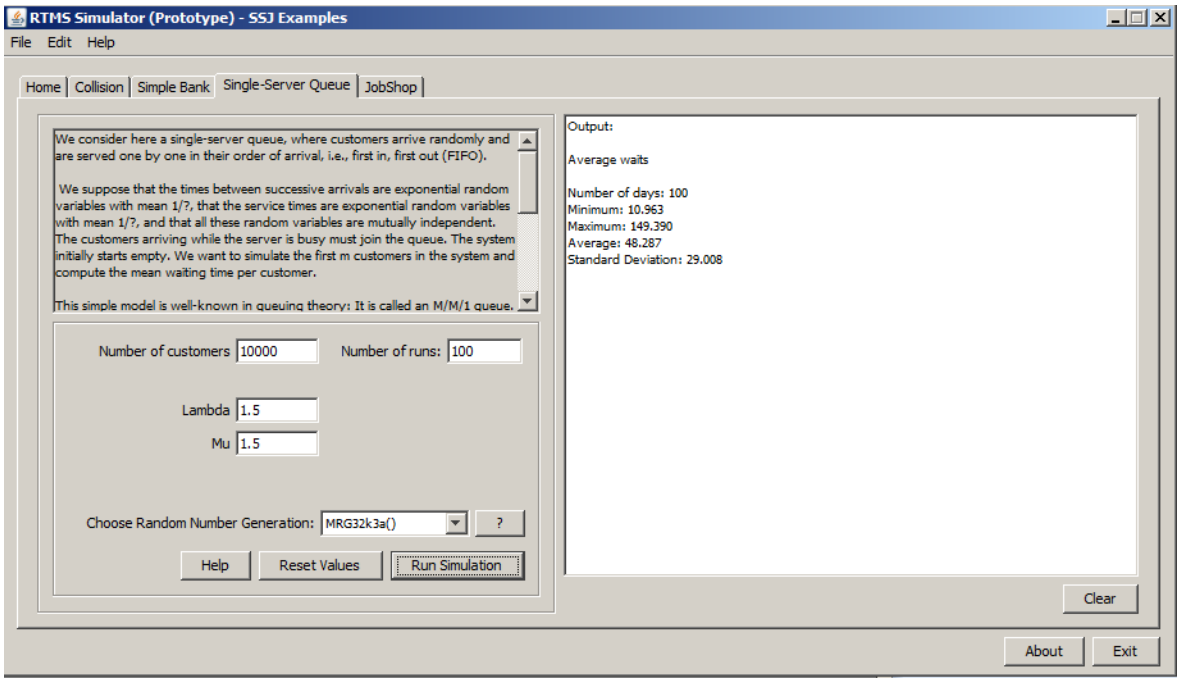


Figure 83 Detail of the output values from the single server model simulation run with 10000 customers but different parameters for the arrivals distributions

This is the type of practice that can be useful when addressing these subjects with similar examples in lectures or other pedagogical event, as it could allow for a better and faster understanding of the relationship between model characteristics and parameters, and the results of changes in the model configurations, thus providing a practical analysis of these subjects.

Table 13 Output values from all 4 single server simulation runs

Output	Run 1	Run 2	Run 3	Run 4
Number of days	100	100	100	100
Minimum	0.430	0.420	0.456	10.963
Maximum	0.580	0.593	0.538	149.390
Average	0.502	0.506	0.500	48.287
Standard Deviation	0.029	0.041	0.019	29.008

Table 13 depicts the waiting times, in the 100 simulated days of the single server queue. The minimum, maximum and average values are accounted and presented, as well as the standard deviation which shows how much variation or dispersion exists from the average value

6.1.3. SIMPLIFIED BANK EXAMPLE WITH DEFAULT CONFIGURATION

The Simplified Bank example is intended to simulate the number of customers the bank serves, in its pre-defined schedule, as well as the average waiting time per day.

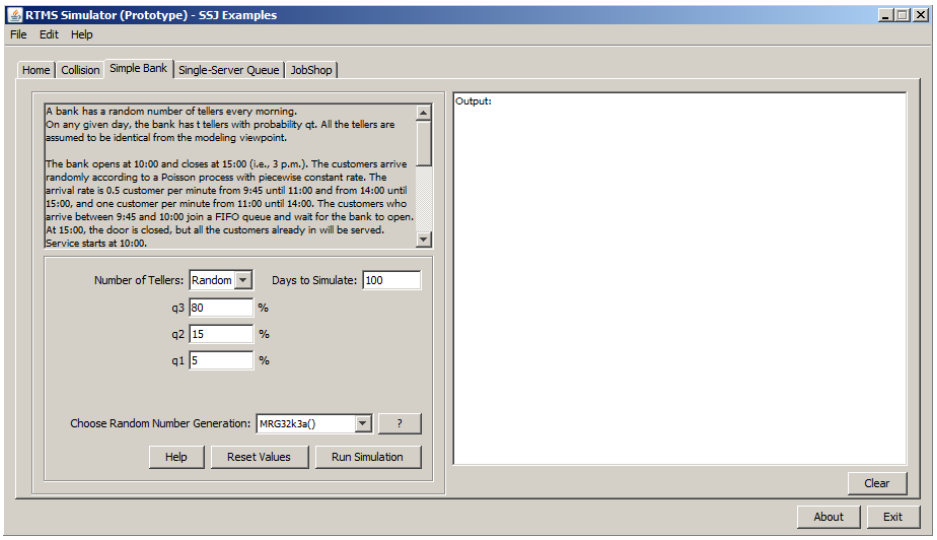


Figure 84 Simplified Bank example interface with default configuration

Once again, detailed information is presented so the user understands that the values generated derive from the pre-defined schedule. However, users can choose between a

fixed number of tellers or randomly generating them by defining the probabilities according to which they will be set as is depicted in Figure 84.

The number of days to run the simulation and the random number generator can be set. In this example, with the default values set for a random number of tellers with a generation probability for one teller of 5%, two tellers of 15% and three tellers of 80% the number of customers served and the average waiting time, in minutes, for the period of 100 days is as depicted in Figure 85.

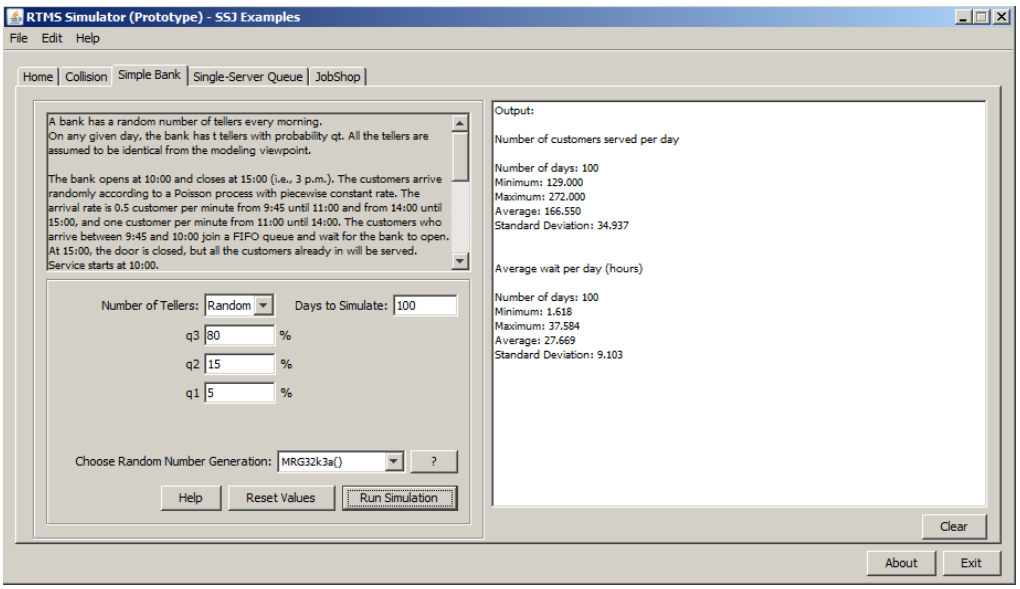


Figure 85 Detail of the output values from the simple bank model simulation run

6.1.4. SIMPLIFIED BANK EXAMPLE WITH ALTERNATIVE CONFIGURATION

With the default configuration, tellers working at the bank will vary from one to three. Alternatively we can run the model simulation with a fixed number of tellers for all days. The output results for those simulation runs are presented next, with the first run having a fixed number of one teller working, again, for 100 days. Alternative scenarios were evaluated for two and three tellers.

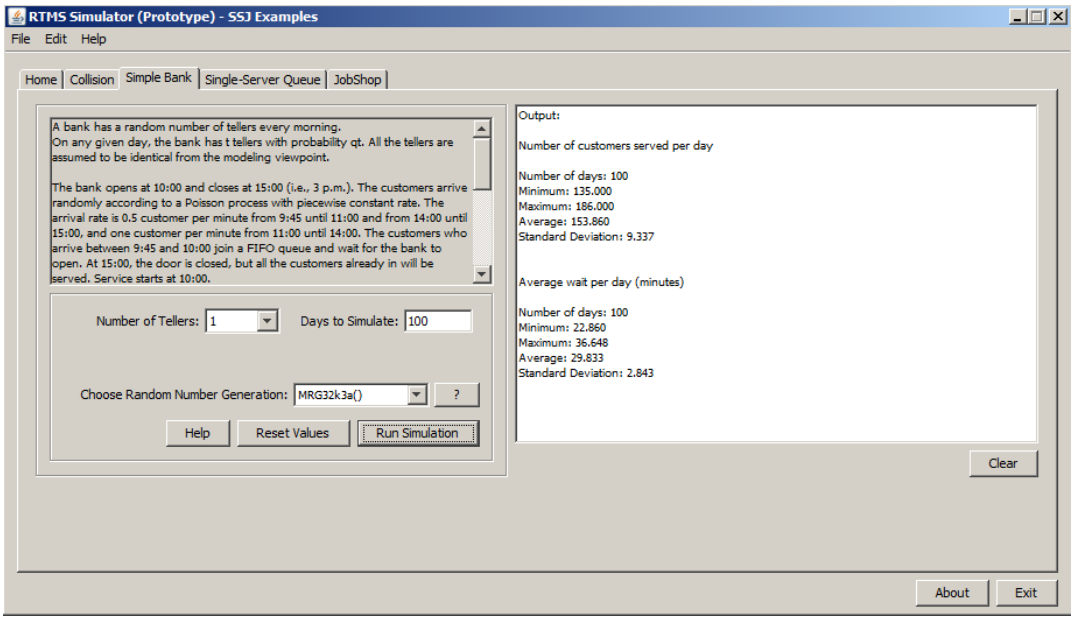


Figure 86 Detail of the output values from the simple bank model simulation run with one teller

Figure 87 shows the results for the simulation run with two tellers in the system.

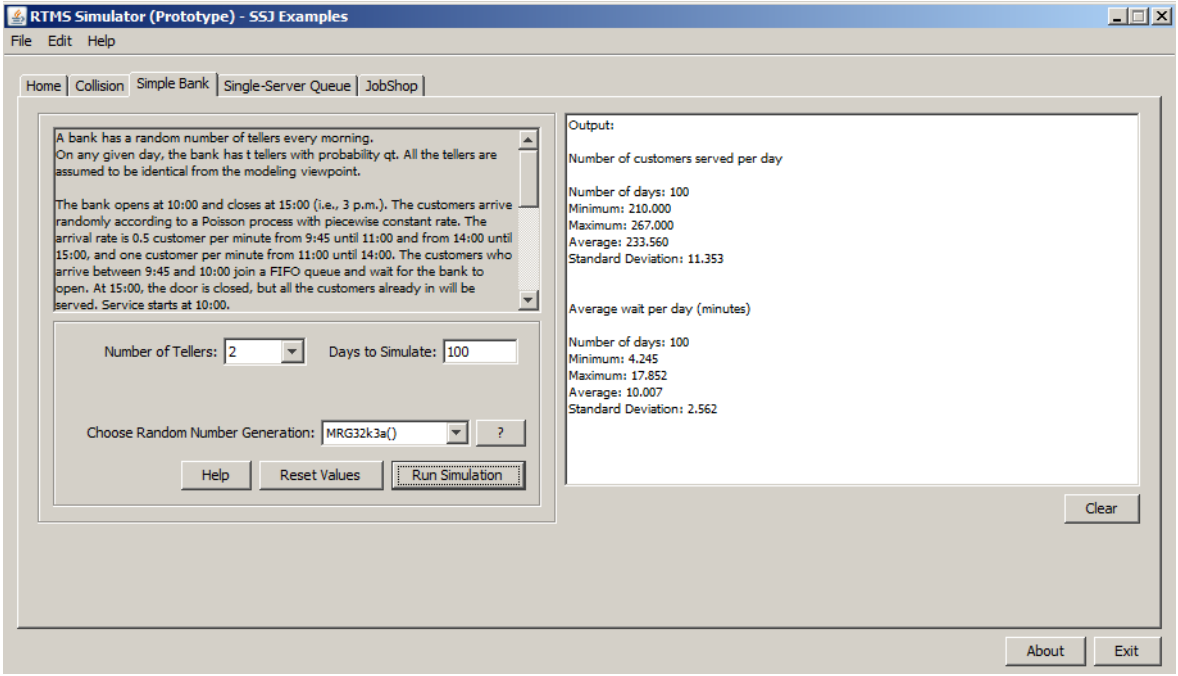


Figure 87 Detail of the output values from the simple bank model simulation run with two tellers

Figure 88 shows the results for the simulation run with three tellers in the system. By observing and comparing all three runs, the impact of the number of the tellers actively working in the system is noticeable.

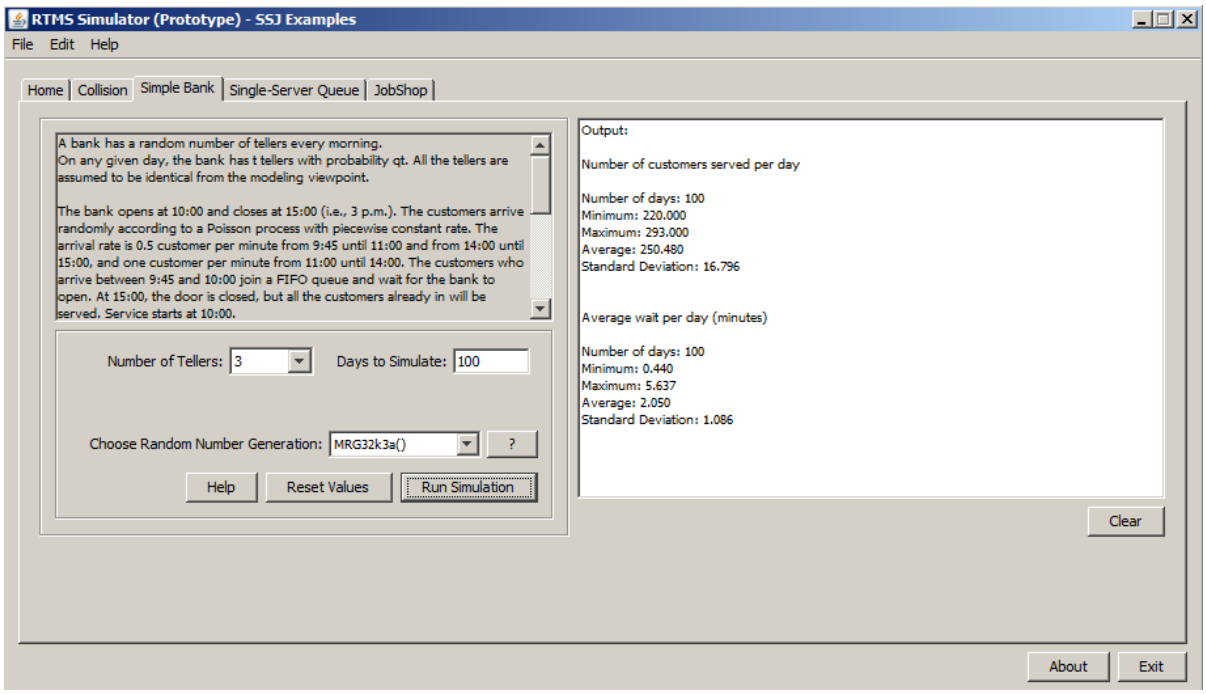


Figure 88 Detail of the output values from the simple bank model simulation run with three tellers

The following tables gather all the results generated from the various simulation runs of this example. Table 14 depicts the number of customers served per day, in the 100 simulated days of the bank model. The minimum, maximum and average values are always accounted and presented, as well as the standard deviation which shows how much variation or dispersion exists from the average value.

Table 14 Output values for the bank example runs – customers served

Output	Run 1	Run 2	Run 3	Run 4
Number of days	100	100	100	100
Minimum	129.000	135.000	210.000	220.000
Maximum	272.000	186.000	267.000	293.000
Average	166.550	153.860	233.560	250.480
Standard Deviation	34.997	9.337	11.353	16.796

Along with the number of customers served, the example model also outputs the average customer waiting times (in minutes) for the simulation period, as seen in Table 15.

Table 15 Output values for the bank example runs – average wait per day (minutes)

Output	Run 1	Run 2	Run 3	Run 4
Number of days	100	100	100	100
Minimum	1.618	22.860	4.245	0.440
Maximum	37.584	36.648	17.852	5.637
Average	27.669	29.833	10.007	2.050
Standard Deviation	9.103	2.843	2.562	1.086

6.1.5. JOB-SHOP MODEL EXAMPLE WITH DEFAULT CONFIGURATION

This example simulates a job-shop model with a pre-defined number of machines and tasks. Due to the complexity of the model and its associated code, it is only possible to modify the warm-up and the horizon time for the simulation as is depicted in Figure 89. In this model, identical machines are grouped together as a group of resources. There are four groups of machines and five different tasks. Table 16 describes the number of identical machines, grouped together.

Table 16 Machine group resources

Machine type	Number of Machines
Machine 1	3
Machine 2	5
Machine 3	1
Machine 4	2

The tasks arrive at the shop with exponential random variables, with mean 0.1, 0.25, 0.15, 0.04, 0.05 respectively.

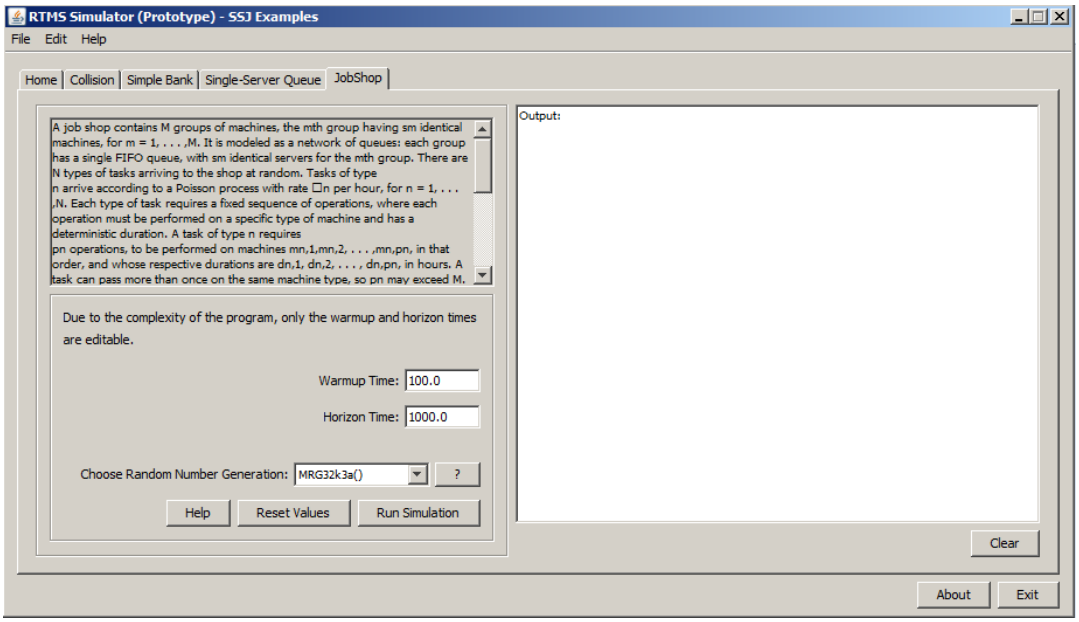


Figure 89 Job-Shop model example interface with default configuration

Also, the routing for the work stations of each job is defined as presented Table 17.

Table 17 Work stations routings for the different tasks

Task Number	Work stations routing
1	3, 1, 2
2	4, 2
3	1, 2, 3, 4
4	2
5	1, 4

Each job type has a service time defined for each machine type represented by an exponential distribution with mean defined as presented in Table 18.

Table 18 Mean service times for each task type.

Task number	Mean service time
1	3.0, 5.0 , 10.0
2	2.0, 3.0
3	5.0, 3.0, 1.0, 4.0
4	10.0
5	2.0, 2.0, 2.0

The simulation is able to compute a report regarding each machine time (see Figure 90) as well as the statistics for each task.

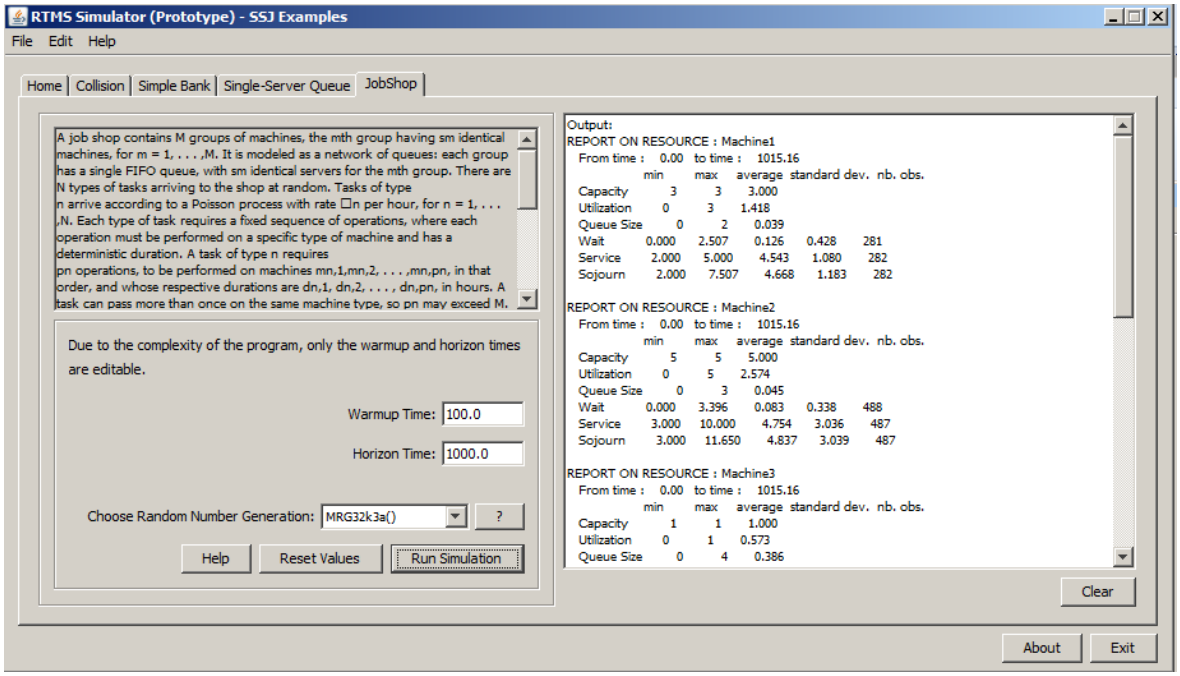


Figure 90 Detail of the output values from the job shop model simulation run

As is seen in Figure 90, the outputted information is capable of providing enough detail concerning the resources (machines) and the tasks performed.

For each machine utilized in the system, the capacity, utilization, queue size, wait time, service time, sojourn (temporary stay or *passing by*) time are presented. Again, all minimum, maximum, average as well as the standard deviations are presented for the number of observations registered in the horizon time period set for the run.

As for the tasks, the model registers the number of operations related to each particular task as well as the time values for those operations such as minimum, maximum, average and the respective standard deviation.

6.1.6. JOB-SHOP MODEL EXAMPLE WITH ALTERNATIVE CONFIGURATION

As mentioned, due to its complexity, there are few model configuration parameters available. However, they do affect the output statistics as expected. Figure 91 shows the results concerning the reports utilization, reflecting the changes due to the decrease in warm-up and increase horizon time length (from 100 to 50 hours and 1000 to 5000 hours, respectively).

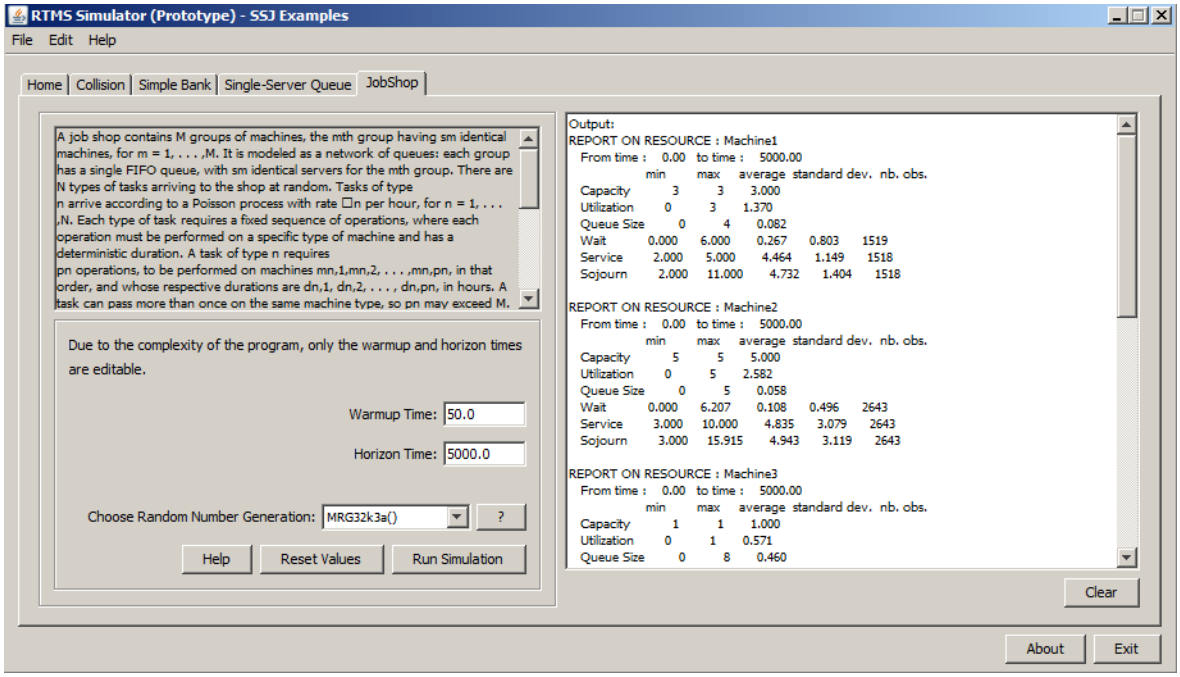


Figure 91 Detail of the output values from the job shop model simulation run with 50 hours as warm-up and 5000 hours as the run horizon time

6.2. REAL-TIME MANUFACTURING SYSTEM MODEL SIMULATION RESULTS

Three different simulation tests were executed with different timings, and consequent degree of complexity, to demonstrate the prototype response to different demands.

Table 19 Simulation runs general configuration parameters

Test No.	No. Stations	No. Jobs	No. Parts
1	3	3	24
2	5	2	23
3	5	2	8

Using second as the time base for all the simulation runs, the tests were executed with the configurations defined at Table 19.

The time for each operation was increased dramatically from run to run, thus allowing to show the prototype reliability when facing different scenarios and also, revealing how the changes from one run to another reflect in the obtained results.

6.2.1. FIRST SIMULATION RUN

As a first task, a simple scenario was created with relatively simplicity.

Considering the Job Shop model implemented for the algorithm, the application was configured using three different stations where the operations are performed, with a short work time. Figure 92.a shows the configuration details for the first simulation run. As is visible, the operation with the longest work time is *Operation 2* with 5 seconds. As for *Operation 1* and *Operation 3* their work times are of 2 and 3 seconds respectively.

All jobs have their own machine allocation for the operations to be performed, as detailed in Table 20.

Table 20 Job operation allocations for the first simulation run

Job No.	Operation 1	Operation 2	Operation 3	Operation 4	Operation
1	Station 1	Station 2	Station 3	None	None
2	Station 2	None	None	None	None
3	Station 1	Station 3	None	None	None

Therefore, all parts from *Job 1* will start their operations at *Station 1*. They will then proceed to *Station 2* after the first operation is complete and, finally, they will move to *Station 3*. After all three operations have been complete they are moved to the finished parts batch. All other jobs will act accordingly.

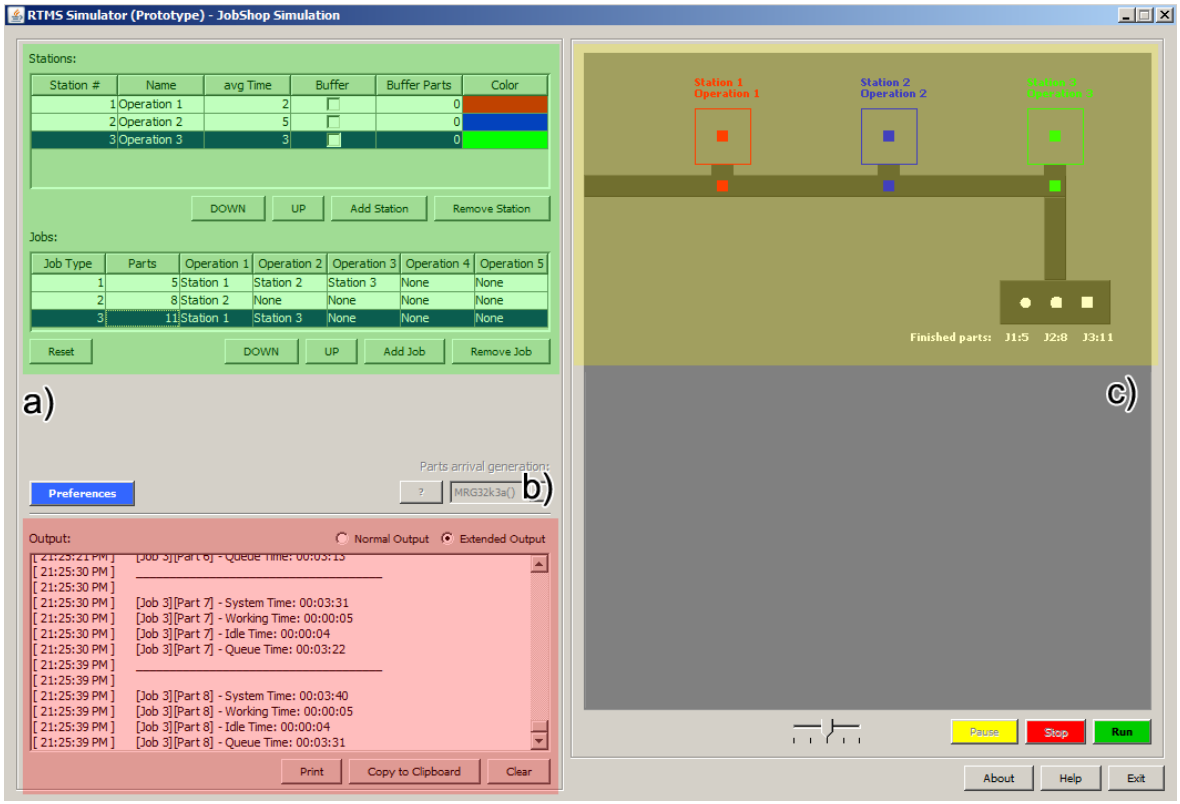


Figure 92 Simulation interface a) Configuration parameters b) output área c) animation panel

As the simulation run is being executed, users can choose to increase or decrease the simulation speed (up to two times faster or slower). Besides, when choosing the *Extended Output* option they can monitor information from the undergoing simulation such as when a certain job is finished or, as shown in Figure 92.b, the individual times for each part in the system.

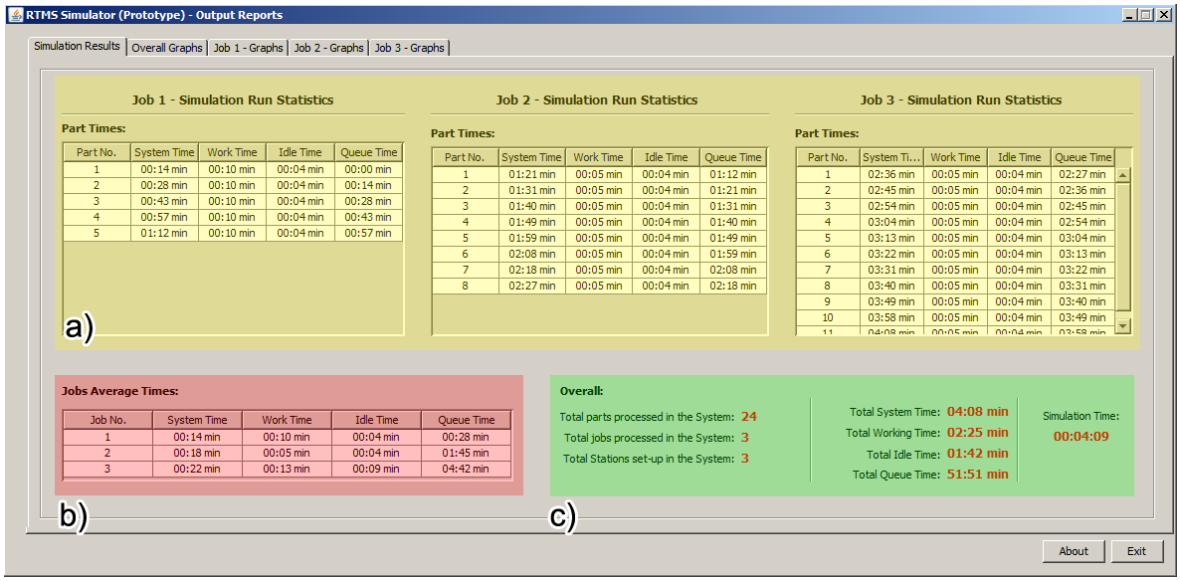


Figure 93 - Report interface a) Jobs individual part times b) average job times c) overall statistics

When all parts have been processed and, therefore, all jobs have been finished they are placed together in the finished parts batch as is depicted in Figure 92.c.

After a run is finished, the Report interface will be shown as the active window on the screen, so users can immediately analyze the generated statistics from the run. Figure 93 shows the Report interface with the values resulting from the run.

As result of the first simulation run, a total of 24 parts were processed from all 3 jobs and 3 work stations. The simulation took 4 minutes and 9 seconds to be completed with a total working time of 2 minutes and 25 seconds and a total idle time of 1 minute and 42 seconds. The individual waiting times in queue for all 24 parts groups together a total of 51 minutes and 51 seconds. These statistics are presented in the initial result interface tab as is depicted in Figure 93.c.

The average waiting time for the parts in each job is also accounted as Figure 93.b shows. Considering *Job 2* average times we can see that, in average, parts being processed spent 18 seconds in the system, from which 5 seconds were active working time and other 4 seconds corresponds to idle time in the system. As the queue times are summed throughout the run, they are larger than the other accounted times.

Considering *Job 2*, once again, it shows an average queue time of 1 minute and 45 seconds. It is calculated by summing all the individual queue times from that job parts as show in equation 6.

$$Avg.\,Queue\ Time = \frac{\sum \text{Queue Time}}{No.parts} \qquad 6)$$

For instance, for *Job 2*, calculations are as follows:

$$Avg.\,Queue\ Time_{Job\ 2} = \frac{(72 + 81 + 90 + 99 + 108 + 117 + 126 + 135)}{8}$$
$$= 103{,}5\ seconds$$

The timings summed in the above calculations are the queue times shown in Figure 93.a, in seconds. All the timings accounted in the application algorithm use second as the time base and are converted to another timestamp, when needed. The average queue time calculated for *Job 2* is of 103,5 seconds, equivalent to 1 minute and 44,5 which rounds up to 1 minute and 45 seconds.

Some charts are generated as an example of possible statistical outputs from the simulation runs. First, in the *Overall Charts* two pie-charts are generated, one for the stations occupation and, another, for the system time distribution of the run.

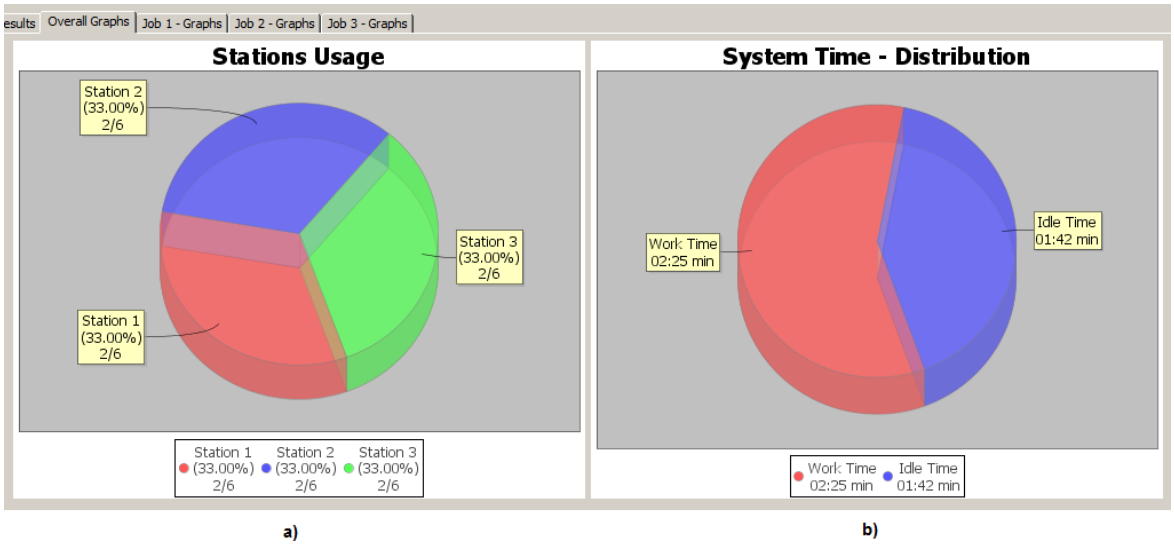


Figure 94 Charts a) Overall stations usage b) System time distribution

These charts allow for a quick and easy understanding of such distributions and others could be presented. Figure 94.a details the individual occupation for the stations where work has occurred throughout the simulation run. By analyzing Table 20, we can see that *Station 1* is used by both *Job 1* and *Job 3*, *Station 2* is used by both *Job 1* and *Job 2* and so on. Therefore in a total of 6 operation allocations, each station is used twice which translates in a rounded 33% of the number of total processed operations.

As for Figure 94.b, it displays the time distribution for the jobs in the simulation run. In the 4 minutes and 9 seconds the simulation was executed, 2 minutes and 25 seconds where active work time while 1 minute and 42 corresponds to the part idling times. Summing these times performs a total of 4 minutes and 7 seconds, instead of the total simulation time present in Figure 93.c, as a result of the rounding and conversions of the timings throughout the run.

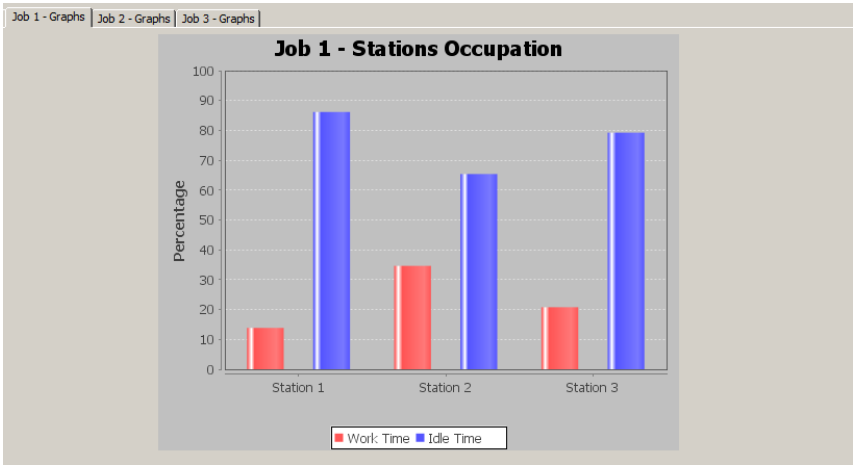


Figure 95 - Job 1 Stations time occupation distribution

Some different charts are presented for the several jobs. In this run, three bar charts are created in individual tabs for each one of the three jobs. These charts detail the percentage difference between work time and idle time in all the stations where operations from that job occurred.

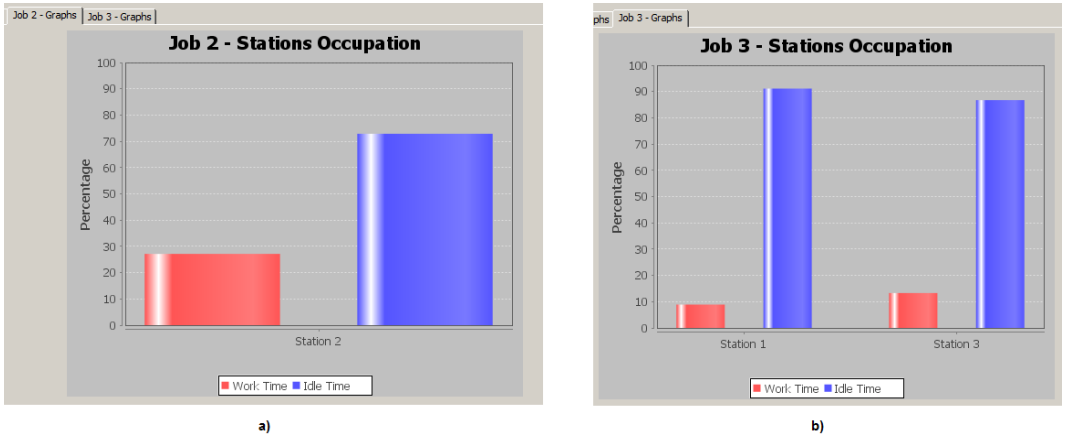


Figure 96 - a) Job 2 Stations time occupation distribution b) Job 2 Stations time occupation distribution

Figure 95 shows those differences for the work stations in *Job 1* where in *Station 1* in around 13% of the overall system time there was operations being executed whilst the remaining percentage of the time the work station was idle. Figure 96 displays the occupations for the remaining jobs.

6.2.2. SECOND SIMULATION RUN

The second test run features an increase in the operation times and the number of jobs was reduced to two with an increase in the number of stations allocated. The configuration parameters are presented in Figure 97.

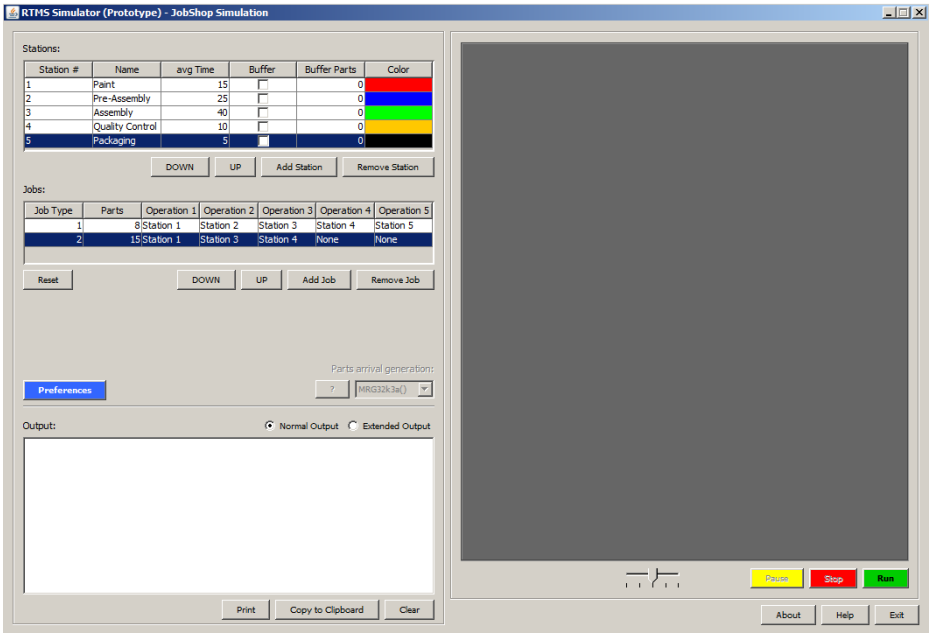


Figure 97 - Second simulation run configuration details

The job operation allocation is as follows in Table 21.

Table 21 Job operation allocations for the second simulation run

Job No.	Operation 1	Operation 2	Operation 3	Operation 4	Operation
1	Station 1	Station 2	Station 3	Station 4	Station 5
2	Station 1	Station 2	Station 4	None	None

The simulation run is complete in 31 minutes and 36 seconds, from which 28 minutes and 55 seconds where active working time and 2 minutes and 38 seconds as the remainder idle time. 23 parts where processed in the 2 existing jobs, with operation being distributed between 5 stations. Such statistics are depicted in Figure 98.

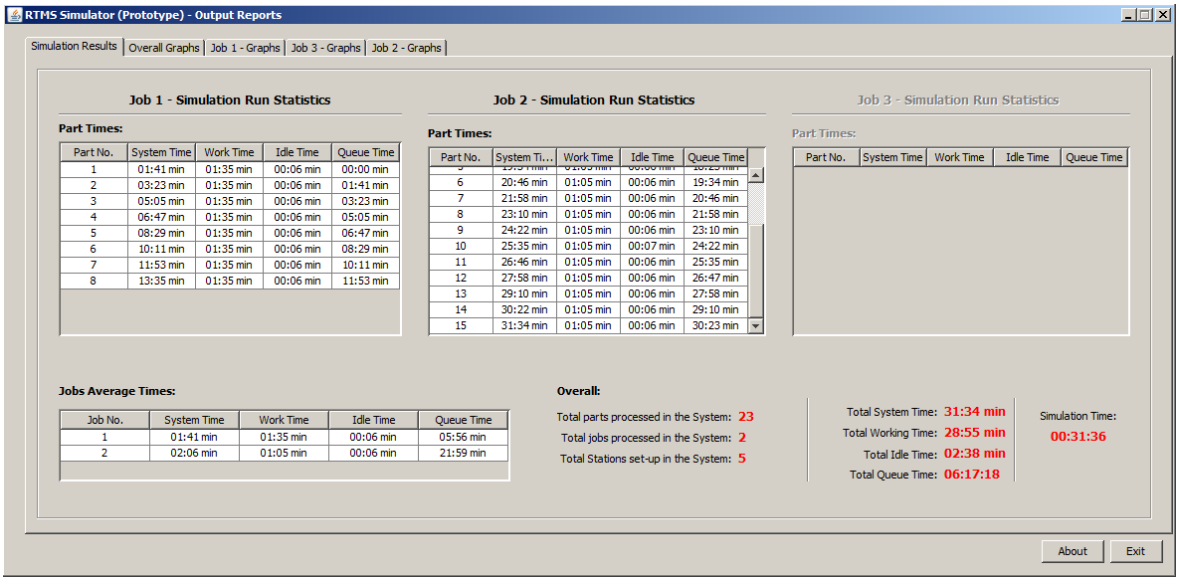


Figure 98 – Report interface

There are visible changes in the stations usage distribution (see Figure 99.a) but mainly in the distribution of the system times, as the majority of the time corresponds to working time, as is shown by the pie chart in Figure 99.b. This is due to the idle times in the system being constants meaning that are not other sources of randomness i.e., machines downtimes, transport times between stations or setup changes.

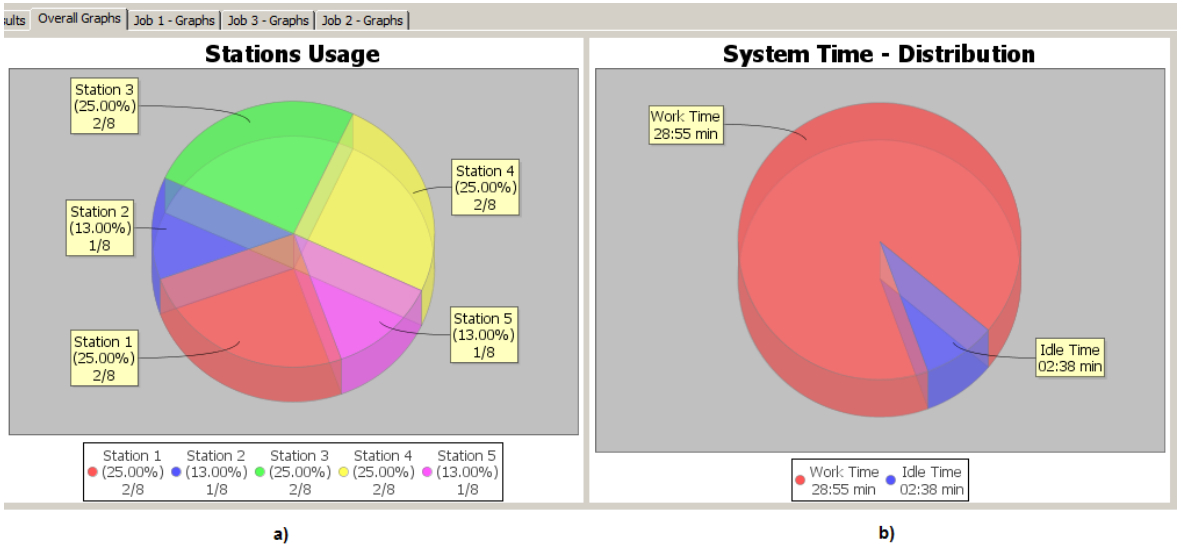


Figure 99 - Overall output charts generated a) Overall stations usage b) System time distribution

The remaining information regarding the time occupation distribution for the stations in both jobs are show in Figure 100.

Figure 100 - a) Job 1 Stations time occupation distribution b) Job 2 Stations time occupation distribution

6.2.3. THIRD SIMULATION RUN

The third and final run features another increase in the operation times with a reduction in the number of parts for both jobs.

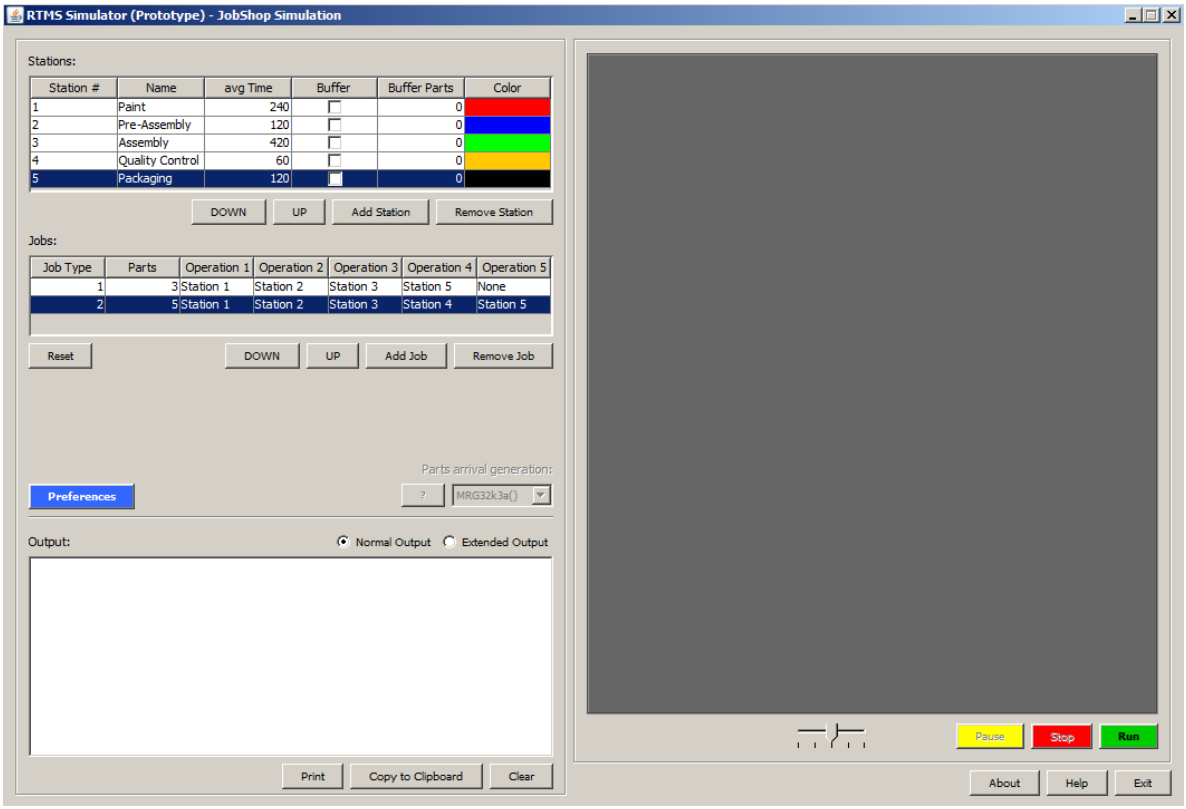


Figure 101 - Third simulation run configuration details

The configuration parameters are presented in Figure 101. The names of the stations where modified from *Operation 1*, *Operation 2* and so on, to real operation designations, thus allowing for an easier interpretation and visualization of the animation which translates into a more realistic simulation. These names are configurable by the users.

The job operation allocation is as follows in Table 22.

Table 22 Job operation allocations

Job No.	Operation 1	Operation 2	Operation 3	Operation 4	Operation
1	Station 1	Station 2	Station 3	Station 5	None
2	Station 1	Station 2	Station 3	Station 4	Station 5

The simulation run is completed in 2 hours 5 minutes and 55 seconds, from which 2 hours and 5 minutes where active working time 54 seconds as the remainder idle time. Although there was a significant increase in the total and system times, the idle time is reduced as there were fewer parts processed in the system, thus meaning few time moving through stations. A total of 8 parts where processed in the 2 existing jobs, with operation being distributed between 5 stations. Such statistics are depicted in Figure 102.

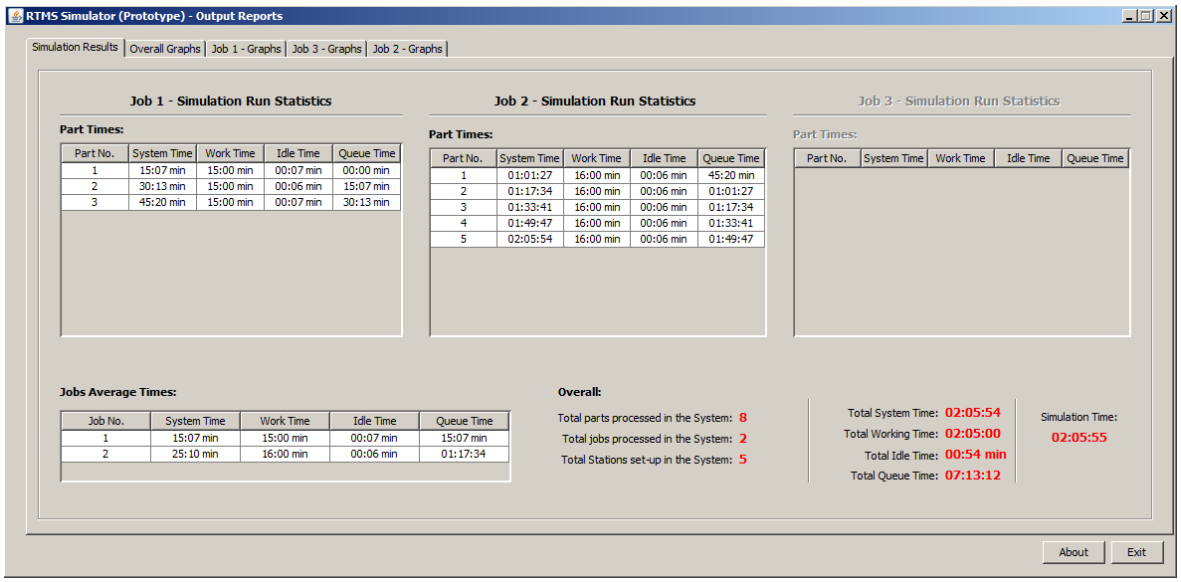


Figure 102 - Part of the statistics outputted in the report interface

There are visible changes in the stations usage distribution (see Figure 103.a) but mainly in the distribution of the system times, as the majority of the time corresponds to working time, as is shown by the pie chart in Figure 103.b. This is due to the idle times in the system being constants meaning that are not other sources of randomness i.e., machines downtimes, transport times between stations or setup changes.

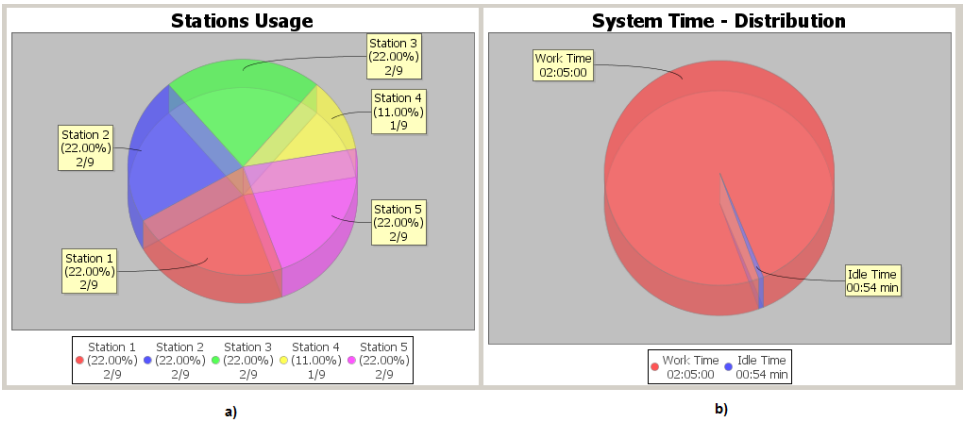


Figure 103 - Overall output charts generated a) Overall stations usage b) System time distribution

The remaining information regarding the time occupation distribution for the stations in both jobs are show in Figure 104.

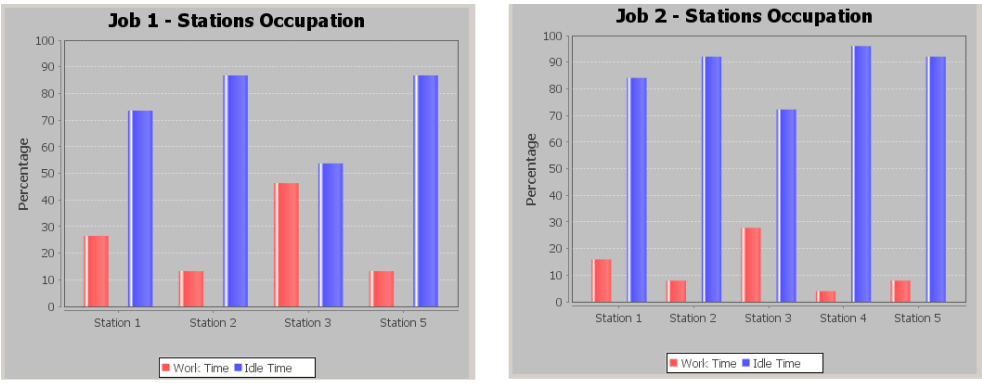


Figure 104 - a) Job 1 Stations time occupation distribution b) Job 2 Stations time occupation distribution

6.3. CONCLUSIONS

The tests regarding all the features and possibilities of use in the prototype where demonstrated.

Regarding the SSJ examples, it demonstrates that not only it is plausible to be used for academic purposes as mean of demonstration regarding such examples but it also shows how existing packages or libraries of simulation can be improved.

As for the job-shop prototype, it simulates a simple model with imposed limitations and is able to simulate output reports with additional graphs. The animation engine works as expected and provides a clear view and understanding of the model. The different outputs provide an abroad view on possible information that can be extracted from the simulation.

It would be an improvement to the prototype if the number of stations and jobs was editable, thus allowing for more than the pre-defined number, as it would increase the flexibility of the job-shop model. To be possible, besides removing the limits imposed in the simulation engine algorithm, changes would have to be done regarding the modification of the animation engine to be able to adapt to such changes and i.e. allow the placement of the stations and increase/decrease the zoom view on the animation model in case of a higher station number.

7. CONCLUSIONS AND FUTURE WORK

The main objective set for this work was to develop a prototype application capable of simulating the behavior of simple real time manufacturing systems, with provided animation, and being plausible for academic purposes as well. After performing the tests, the application presented in this thesis fulfills the determined requirements for the work, and it meets the needs and demands of such a tool being plausibly usable for academic purposes.

The general idea behind the prototype was to maintain its simplicity while performing the intended tasks, namely providing its users with an animation engine. The model used for the manufacturing system is only configurable to a certain extent to prevent an increase in the application complexity and also due to obvious time and knowledge limitations.

The recurrence to JAVA as the programming language allowed to create a modular application which translates in the ability to easily integrate other modules or features, or the integration of the prototype in other existing platforms.

There are numerous simulation applications, most of which were mentioned in the document, intended for commercial use in real world systems and the application

developed in this thesis is not to be compared with such solutions due to its different nature. However, when compared with similar open-source tools and other applications developed concerning academic purposes it is somewhat more complete due to the implementation of a graphic user interface to support its operations and mainly due to the inclusion of an animation engine capable of a visual output of the operations in the manufacturing system.

The prototype is, as expected, a viable tool to be used for academic purposes, i.e. lectures regarding the concepts of manufacturing systems, system layouts, measures of performance and simulation.

7.1. FUTURE WORK

The purpose of this project was the development of a working prototype for a simulator of real time manufacturing systems for academic purposes. However, the work presented in this thesis is far from being completely developed, as there are some features which can be implemented, as well as existing features which can be refined.

One of the most important tasks to develop in the future, if further development of the application is intended, is the implementation of a complete random number generation engine capable of implementing different sources of randomness. This is of crucial importance to correctly simulate more complex manufacturing systems where system randomness plays an important role in its operation. The application can significantly increase its potential if continuous distributions could be implemented and configured as follows:

- Inter-arrival times for parts;
- Processing, assembling or inspection times;
- Times to failure of machines;
- Times to repair machines;
- Setup times;

Although the application was thoroughly tested, there are some minor improvements that can be made in the graphic user interface and the code can be optimized even further. The inclusion of drag and drop features in the animation drawing panel and the removal of the station and job number limitations would increase the customization of the job shop model.

Also, although the algorithm has been developed recurring to the multi-threading concepts its complete implementation can be stated as a further development as it would allow for multiple jobs being held at the same time in the system, thus allowing the implementation of a working queue system.

References

[1] D. Goldsman, R. E. Nance, and J. R. Wilson, "A brief history of simulation," presented at the Winter Simulation Conference, 2009.

[2] J. P. Womack, D. T. Jones, and D. A. Roos, *A máquina que mudou o mundo*, 3rd ed. Rio de Janeiro: Campus, 1992.

[3] "Watt steam engine," ed, p. Watt steam engine image: located in the lobby of into the Superior Technical School of Industrial Engineers of the UPM (Madrid).

[4] W. Perales. Classificações dos sistemas de produção.

[5] "Business 101 — The Basics - Production and Operations Management," 14 ed.

[6] E. S. Buffa, *Modern Production Management*, 7th ed.: New York: John Wiley & Sons., 1977.

[7] N. Slack, "Administração da Produção," in *Administração da Produção*, E. A. S.A., Ed., ed, 2006.

[8] D. A. Moreira, *Administração da Produção e Operações*, 3 ed. São Paulo, 1998.

[9] M. A. Carravilla, *Layouts Balanceamento de Linhas*, 1998.

[10] W. Siler. (2008, July). *Ten ways the Model T changed the World*. Available: http://jalopnik.com/5057386/ten-ways-the-model-t-changed-the-world

[11] J. Banks, *Handbook of Simulation - Principles, Methodology, Advances, Applications and Practice*, 1998.

[12] J. B. Dilworth, *Production and Operations Managment* vol. Fourth Edition, 1989.

[13] "Ship building in Gdansk Poland," ed, 2012.

[14] A. Kumar. Lesson 8 - Production Planning and Control. 5-6.

[15] H. A. Simon, "Prediction and prescription in systems modeling," 1990.

[16] J. Banks, "Handbook of Simulation," Wiley, Ed., ed, 1998.

[17] A. A. B. Pristker, *Principles of Simulation Modeling. In Handbook of Simulation: principles, methodology, advances, applications, and practice*: New York: John Wiley, 1998.

[18] A. M. Law and W. D. Kelton, *Simulation modeling and analysis*: McGraw-Hill, 2000.

[19] J. Banks, I. Johh S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*: Prentice Hall, 2009.

[20] P. E. Babulak and D. M. Wang, "Discrete Event Simulation," University of the South Pacific, Suva, Fiji.

[21] C. D. Pedgen, "ADVANCED TUTORIAL: OVERVIEW OF SIMULATION WORLD VIEWS," presented at the Proceedings of the 2010 Winter Simulation Conference, 2010.

[22] J. F. Robinson, W. C. Copaccino, and R. E. Howe, *The Logistics Handboom*: Free Press, 1994.

[23] R. E. Shannon, *Systems Simulation – The Art and Science*: Prentice-Hall, 1975.

[24] D. K. P. Mohandas. Introduction to simulation.

[25] A. Maria, "Introduction to modeling and simulation," presented at the Proceedings of the 1997 Winter Simulation Conference, 1997.

[26] J. Banks and R. R. Gibson, "10 rules for determining when simulation is not appropriate," *IIE Solutions*, pp. 1-7, September 1997 1997.

[27] R. E. Nance, "A history of discrete event simulation programming languages," Department of Computer Science - Virginia Polytechninc Institute and State University1993.

[28] R. E. Nance, "A history of discrete event simulation programming languages," presented at the HOPL-II: The second ACM SIGPLAN conference on History of programming languages, 1993.

[29] N. Metropolis and S. Ulam, *The Monte Carlo Method* vol. 44: Journal of the American Statistical Association, 1949.

[30] L. Leemis and S. Park. (2004). *Discrete-event simulation: A first course*.

[31] M. Wang, G. Sun, and M. Nooh, "Application of Simulation to Reduce Manufacturing Cycle Time," presented at the 4th International Microelectronics Systems'95 Conference, Malaysia, 1995.

[32] D. K. Pace. (2004) Modeling and simulation verification and validation challenges. *APL Technical digest*. 163-172.

[33] S. Schlesinger, *Terminology for Model Credibility*, 1979.

[34] A. M. Law and M. G. McComas, "Pitfalls in the simulation of manufacturing systems," presented at the 18th Winter simulation Conference, 1986.

[35] A. M. Law and M. G. McComas, "Simulation of manufacturing systems," presented at the Winter Simulation Conference, 1997.

[36] R. G. Ingalls, "Introduction to Simulation," presented at the Winter Simulation Conference, 2011.

[37] J. Banks, "Handbook of Simulation - Principles, Methodology, Advances, Applications and Practice," ed, 1998, pp. 38-48.

[38] B. Copstein, C. E. Pereira, and F. R. Wagner, "The OBject-Oriented approach and the event discrete simulation paradigms," presented at the European Simulation Multiconference, 1996.

[39] D. J. Eck, *Introduction to Programming Using Java* vol. 5, 2006.

[40] Oracle. (August, 2012). *The Java Tutorials - What is an Object?* Available: http://docs.oracle.com/javase/tutorial/java/concepts/object.html

[41] Oracle. (August, 2012). *The Java Tutorials - What is Inheritance?* Available: http://docs.oracle.com/javase/tutorial/java/concepts/object.html

[42] Anon. (August). *Wikipedia - Sun acquisition by Oracle*. Available: http://en.wikipedia.org/wiki/Sun_acquisition_by_Oracle

[43] E. R. Harold. (1997, August, 2012). The Prehistory of Java. Available: http://www.cafeaulait.org/slides/hope/02.html

[44] (September). *History of Java programming language*. Available: http://www.freejavaguide.com/history.html

[45] R. A. Kilgore, "Introduction to Visual Modeling with Silk and JavaBeans," presented at the Winter Simulation Conference Proceedings, 1998.

[46] R. A. Kilgore and K. J. Healy, "THE FUTURE OF JAVA-BASED SIMULATION," presented at the Proceedings of the 1998 Winter Simulation Conference, 1998.

[47] E. H. Page, R. L. Moose, and S. P. Griffin, "Web based simulation in SimJava using Remote Method Invocation," presented at the Proceedings of the 1997Winter Simulation Conference, 1997.

[48] A. Sawhney, A. Mund, and H. Deshpande, "Construction engineering II: JavaBeans-based framework for construction simulation," presented at the Winter Simulation Conference, 2000.

[49] R. McNab and F. W. Howell, "simjava: A discrete event simulation library for Java," presented at the Proceedings of the 1998 International Conference on Web-Based Modeling & Simulation, 1998.

[50] P. S. Coe, F. W. Howell, R. N. Ibbett, and L. M. Williams, *A Hierarchical Computer Architecture Design and Simulation Environment*, 1998.

[51] M. C. Little. (1997). *JavaSim World Wide Web Home Page*. Available: http://javasim.ncl.ac.uk/

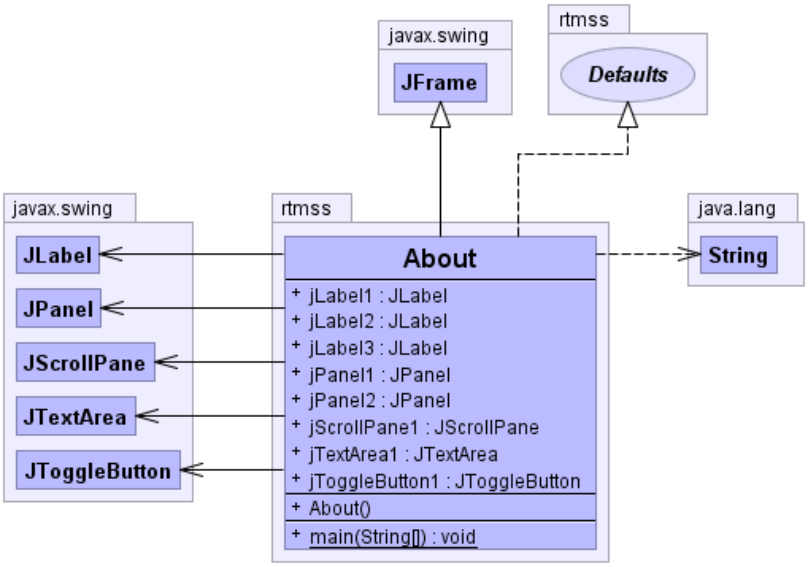
[52] A. H. Buss and K. A. Stork, "Discrete-Event Simulation on the World Wide Web Using Java," presented at the Proceedings of the 1996 Winter Simulation Conference, 1996.
[53] J. Kapuno, R.R., and N. N. Nagarur. (1999, SimProd: A Web-Based Flexible Simulation Package for Production Systems. Available: http://www.sat.ait.ac.th/ej-sat/articles/1.2/ng.html
[54] R. Nair, "JSIM: A Java-based query driven simulation and animation environment," Masters Thesis, The University of Georgia, 1997.
[55] M. D. Rossetti, *Simulation Modeling and Arena*: Wiley, 2009.
[56] R. E. Shannon, "Introduction to simulation languages," presented at the Winter Simulation Conference, 1977.
[57] S. M. Corporation. (October, 2012). Available: www.sm.com
[58] Autosimulations. (October, 2012). Available: www.autosim.com
[59] I. T. Inc. (October, 2012). Available: www.imaginetahtinc.com
[60] W. Software. (October, 2012). Available: www.wolverinesoftware.com
[61] C. P. Organization. (October, 2012). Available: www.simscript.com
[62] A. Corporation. (October, 2012). Available: www.aesop.de
[63] J. O'Reilly, "Introduction to AweSim," presented at the Winter Simulation Conference, 2002.
[64] J. Banks, *Handbook of Simulation - Principles, Methodology, Advances, Applications and Practice*, 1998.
[65] D. Krahl, "Extend: an interactive simulation tool," presented at the Winter Simulation Conference, 2003.
[66] *ARACOM - Simple++*. Available: http://www.aracomsys.com/html/simple.shtml
[67] AutoMod. (October, 2012). Available: www.simplan.de/en
[68] P. Corporation. (October, 2012). Available: www.extendsim.com
[69] PROMODEL. (October, 2012). Available: www.promodel.com
[70] F. H. Simulations. (October, 2012). Available: www.flexsim.com
[71] L. Group. (October, 2012). Available: www.lanner.com
[72] P. Dvorak. (2002). *Software fine-tunes a factory*. Available: http://machinedesign.com/article/software-fine-tunes-a-factory-1010
[73] Anon. (1998, Simulatuion software buyer's guide. 48-54.
[74] J. Banks and R. G. Gibson. (1997, Selection of simulation software. 30-32.
[75] L. Kleinrock, *Queueing Systems* vol. 1. New York, 1975.
[76] P. L'Ecuyer. Official SSJ2.5 Documentation. Available: http://www.iro.umontreal.ca/~simardr/ssj/doc/pdf/
[77] P. Bratley, B. L. Fox, and L. E. Schrage, in *A Guide to Simulation*, S.-. Verlag, Ed., ed New York, 1987, p. 14.
[78] P. L'Ecuyer. Official SSJ2.5 Overview and Examples. Available: http://www.iro.umontreal.ca/~simardr/ssj/examples/examples.pdf

Appendix A. Full code for the Lindsey Queue example

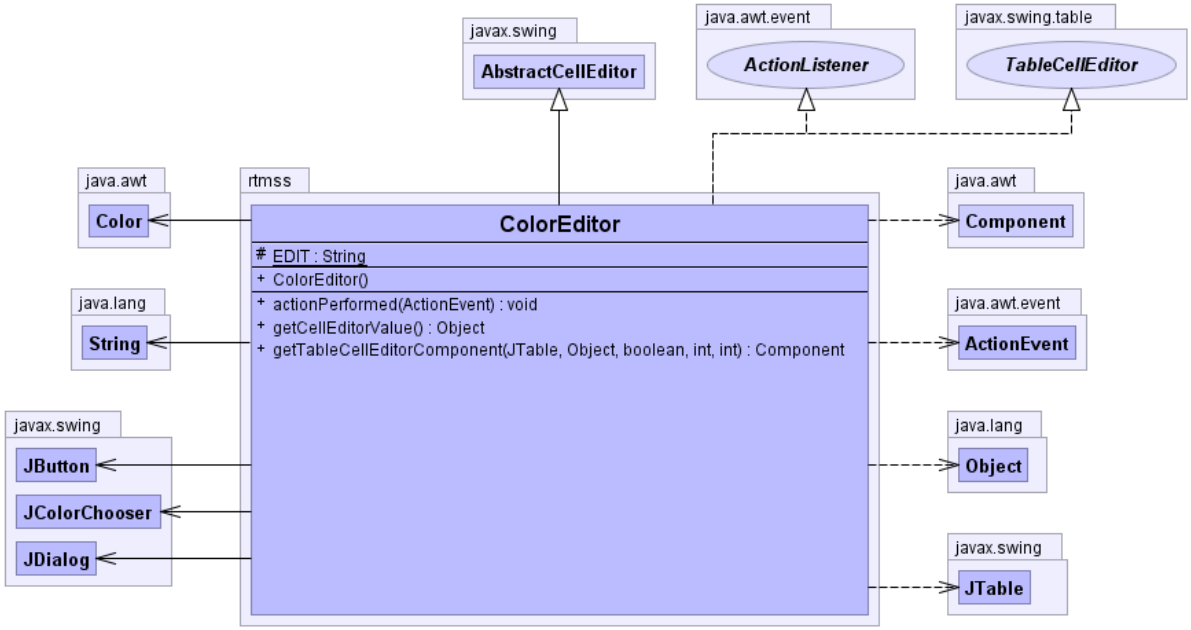
```
import umontreal.iro.lecuyer.stat.*;
import umontreal.iro.lecuyer.rng.*;
import umontreal.iro.lecuyer.probdist.ExponentialDist;
import umontreal.iro.lecuyer.util.Chrono;

public class QueueLindley {
RandomStream streamArr = new MRG32k3a();
RandomStream streamServ = new MRG32k3a();
Tally averageWaits = new Tally ("Average waits");
public double simulateOneRun (int numCust, double lambda,
double mu) {
double Wi = 0.0;
double sumWi = 0.0;
for (int i = 2; i <= numCust; i++) {
Wi += ExponentialDist.inverseF (mu, streamServ.nextDouble())
-
ExponentialDist.inverseF (lambda, streamArr.nextDouble());
if (Wi < 0.0) Wi = 0.0;
sumWi += Wi;
}
return sumWi / numCust;
}

public void simulateRuns (int n, int numCust, double lambda,
double mu) {
averageWaits.init();
for (int i=0; i<n; i++)
averageWaits.add (simulateOneRun (numCust, lambda, mu));
}

public static void main (String[] args) {
Chrono timer = new Chrono();
QueueLindley queue = new QueueLindley();
queue.simulateRuns (100, 10000, 1.0, 2.0);
System.out.println (queue.averageWaits.report());
System.out.println ("Total CPU time: " + timer.format());
 }
}
```

Appendix B. Complete UML Class Diagrams

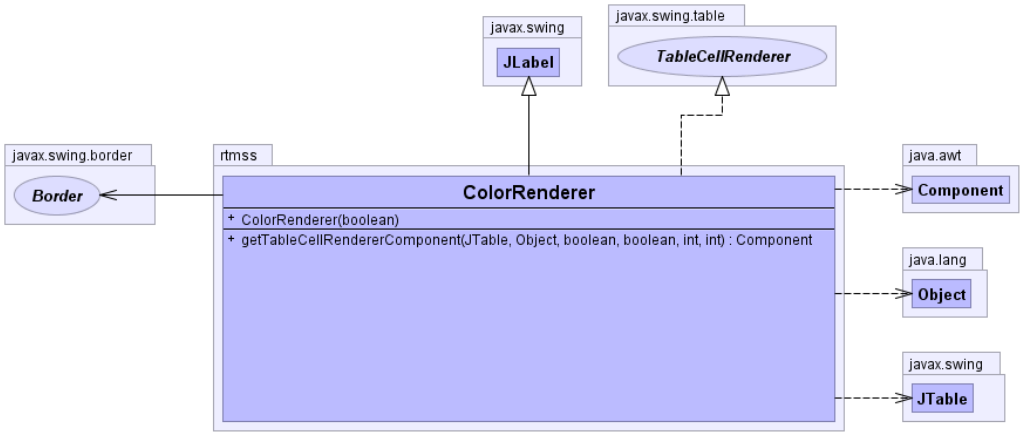
About Class Diagram:



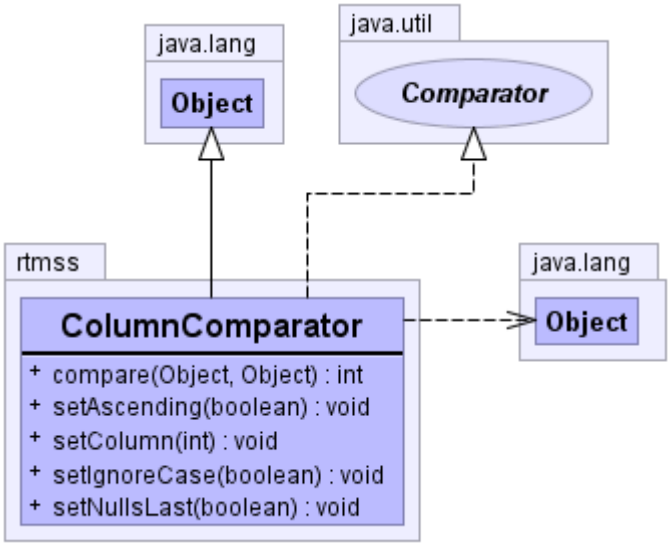
ColorEditor Class Diagram:



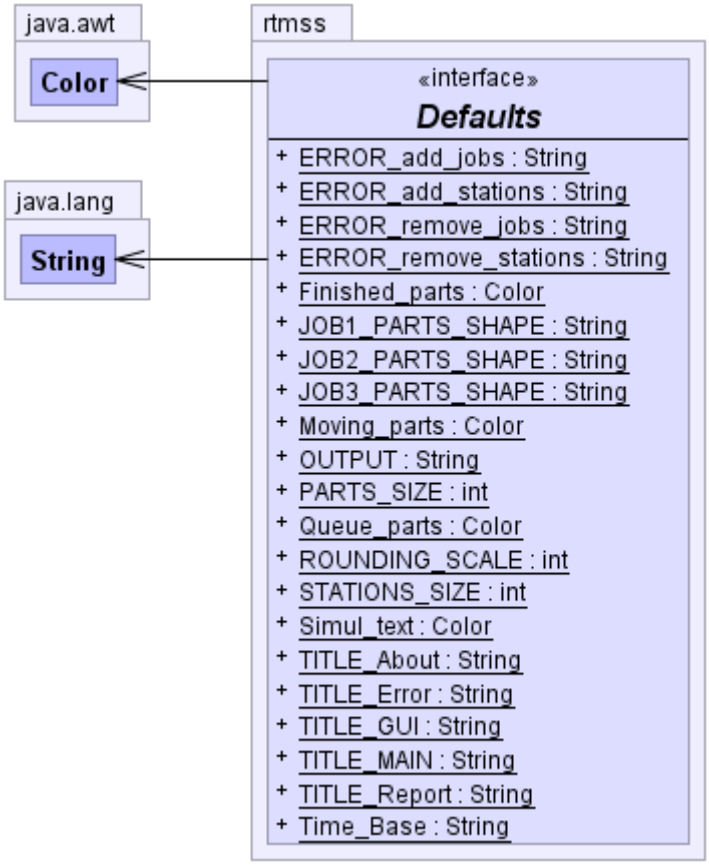
ColorRenderer Class Diagram:



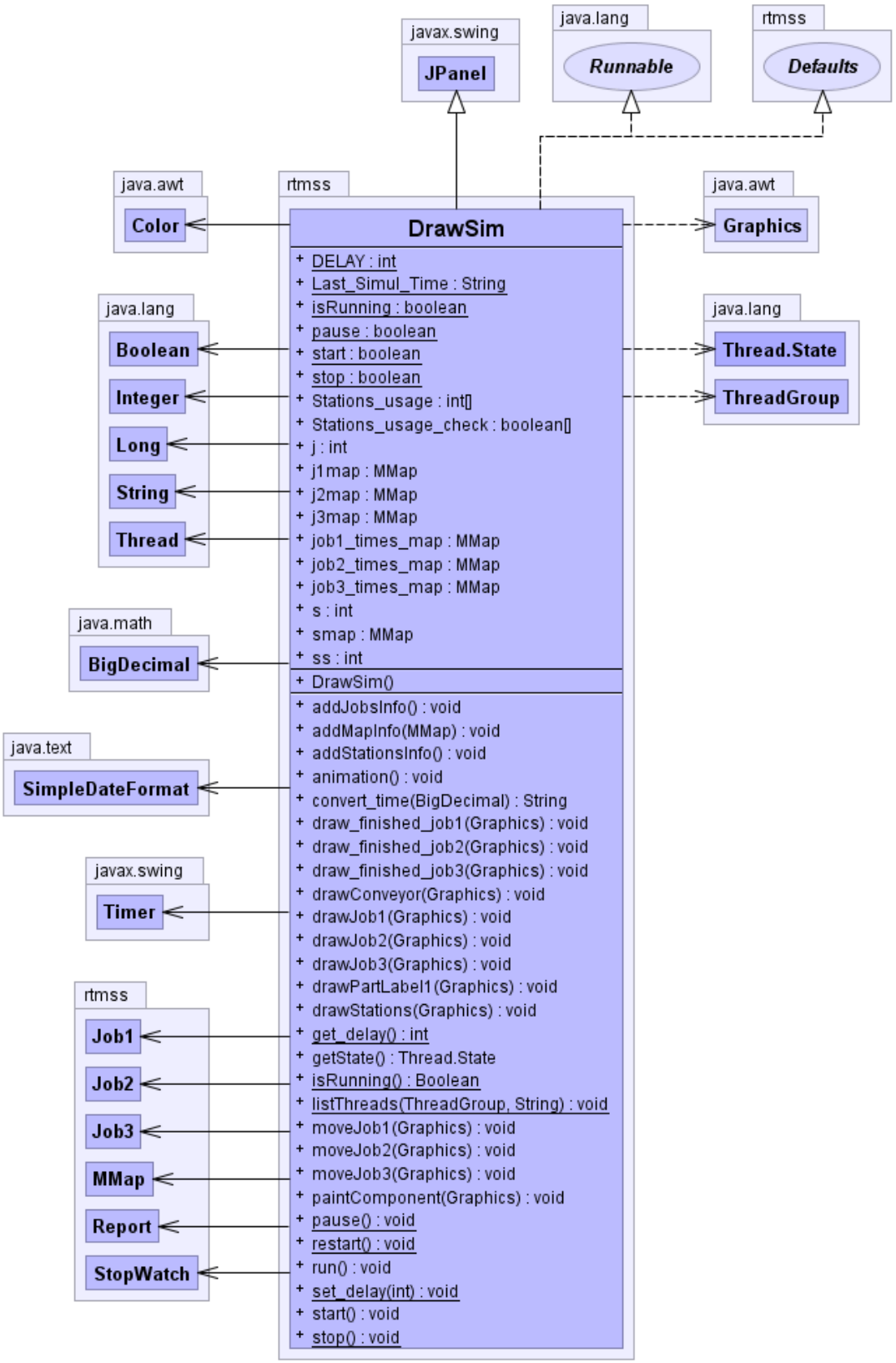
ColumnComparator Class Diagram:



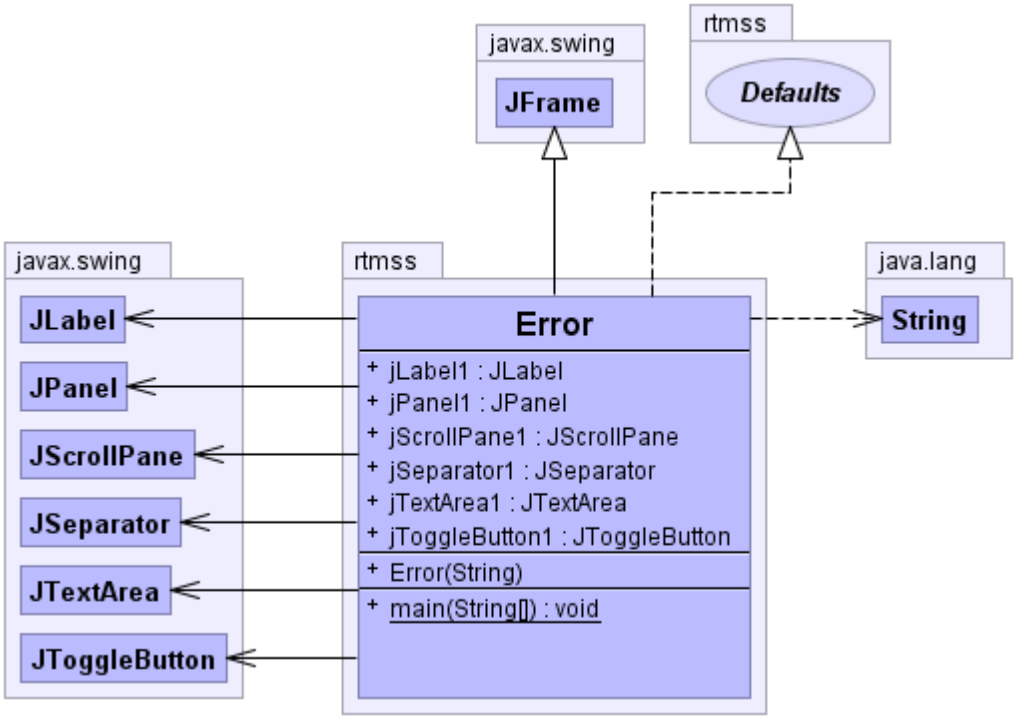
Defaults Interface Class Diagram:



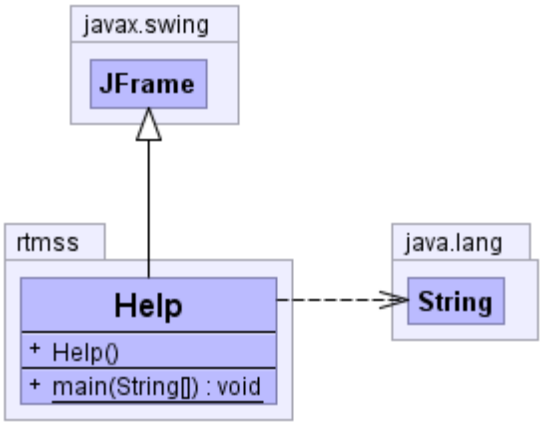
DrawSim Class Diagram:



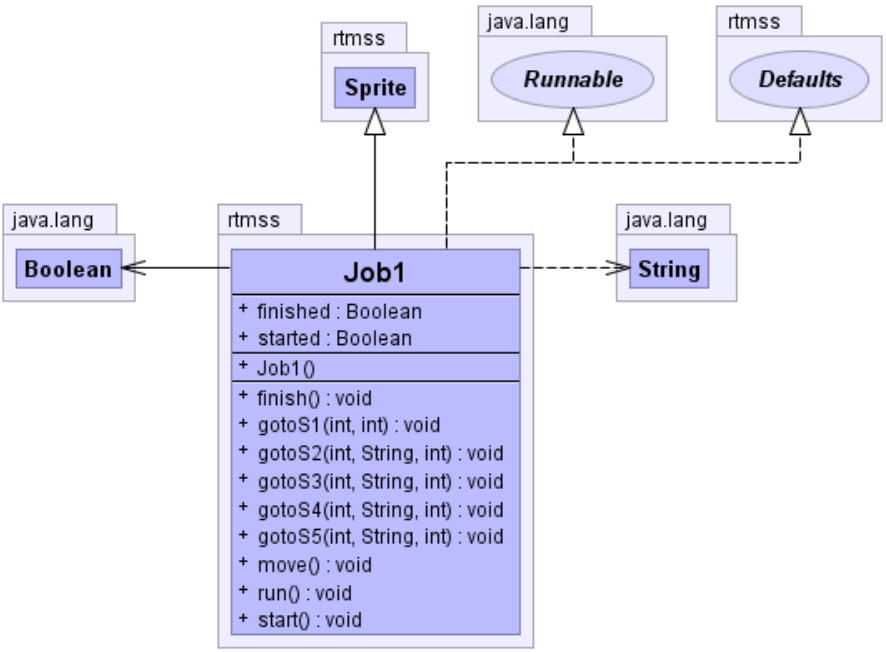
Error Class Diagram:



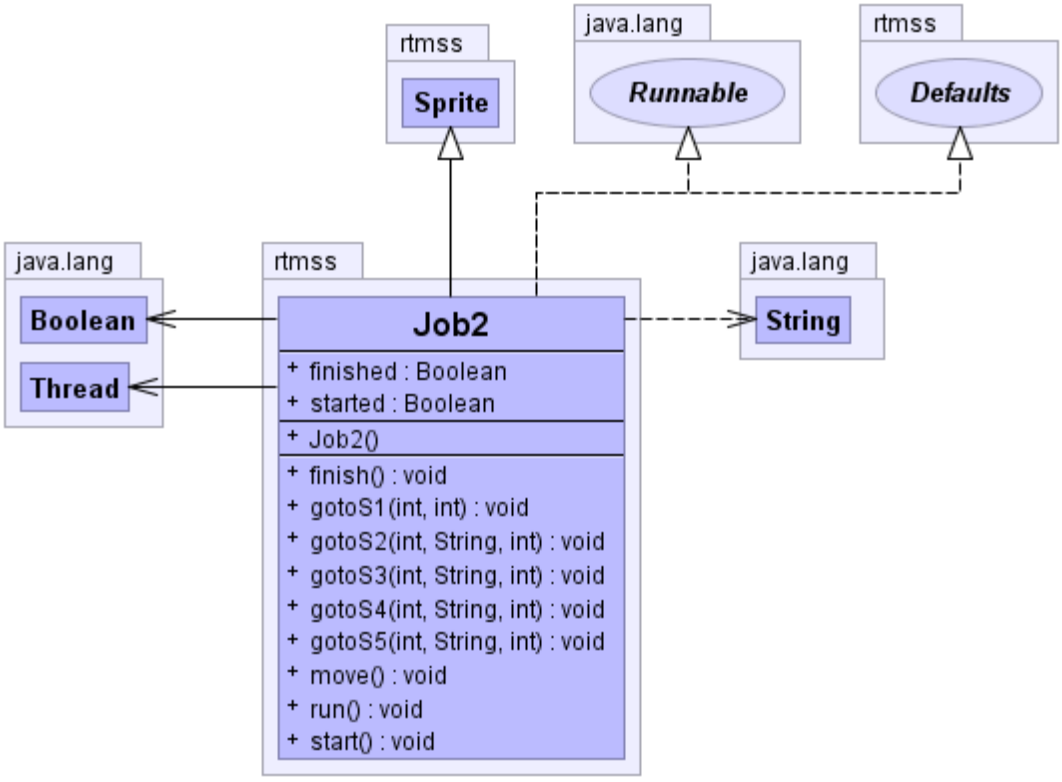
Help Class Diagram:



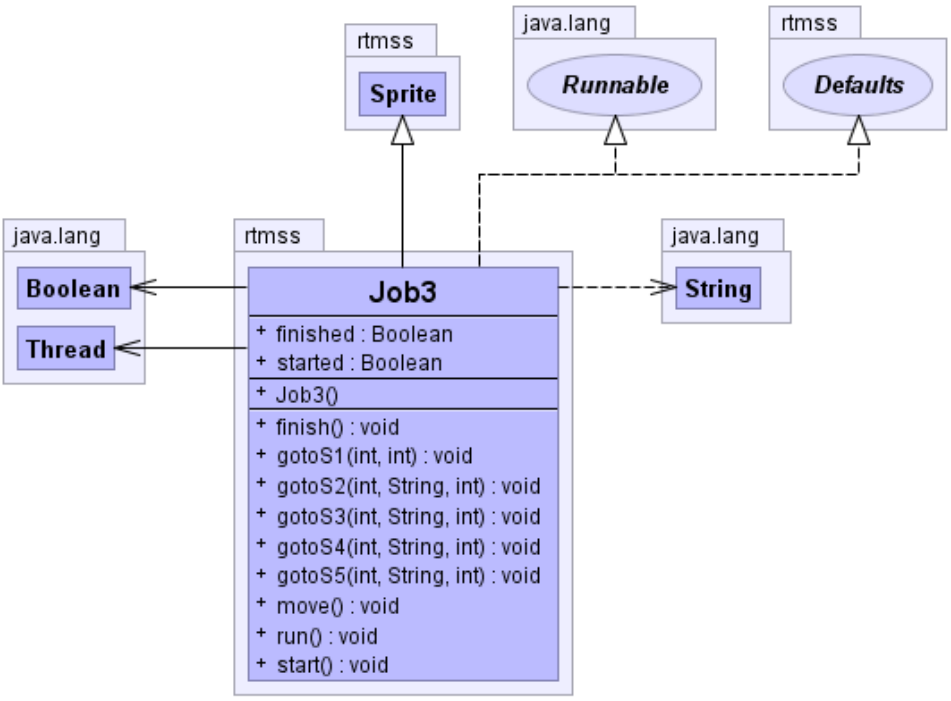
Job1 Class Diagram:



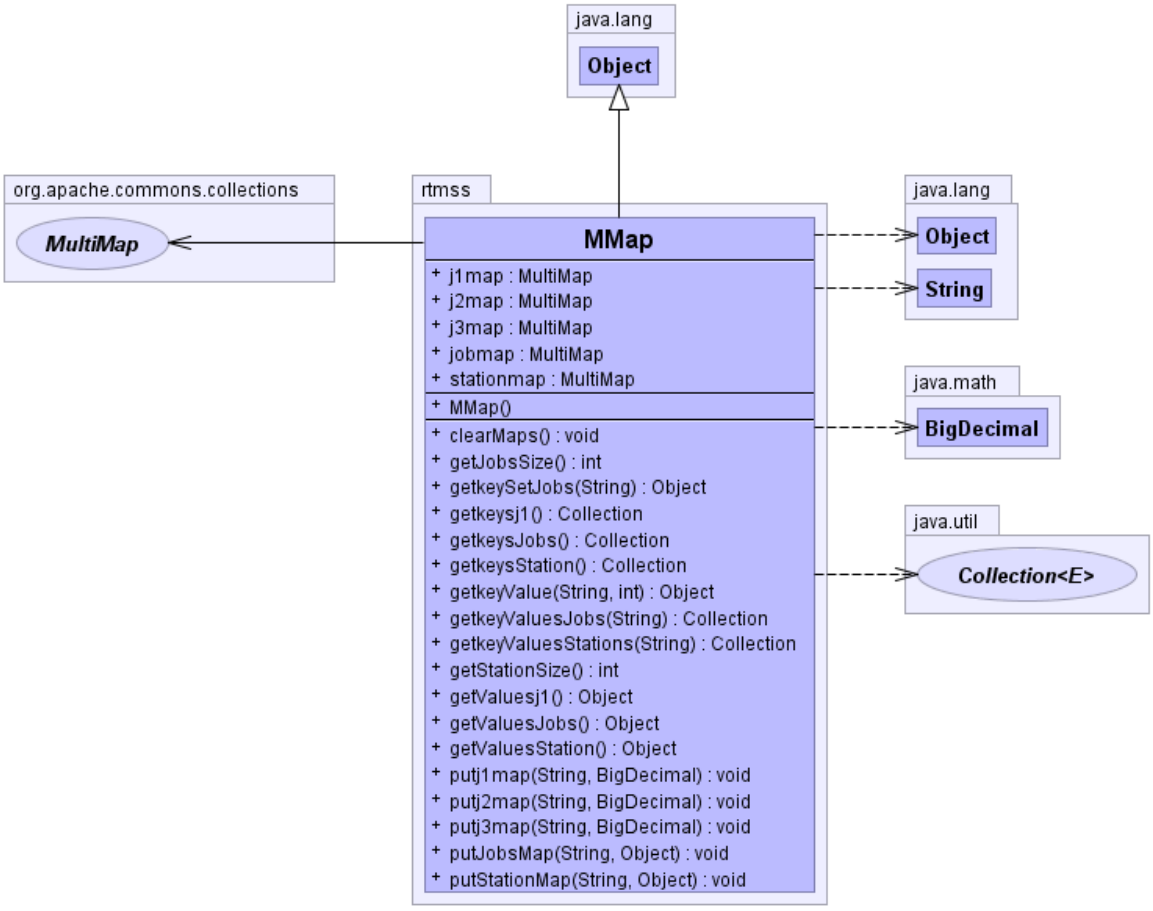
Job2 Class Diagram:



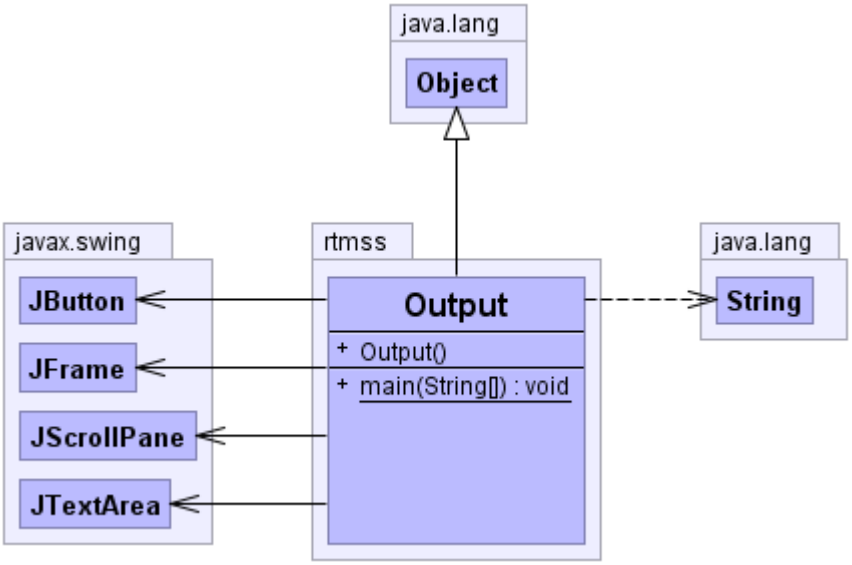
Job3 Class Diagram:



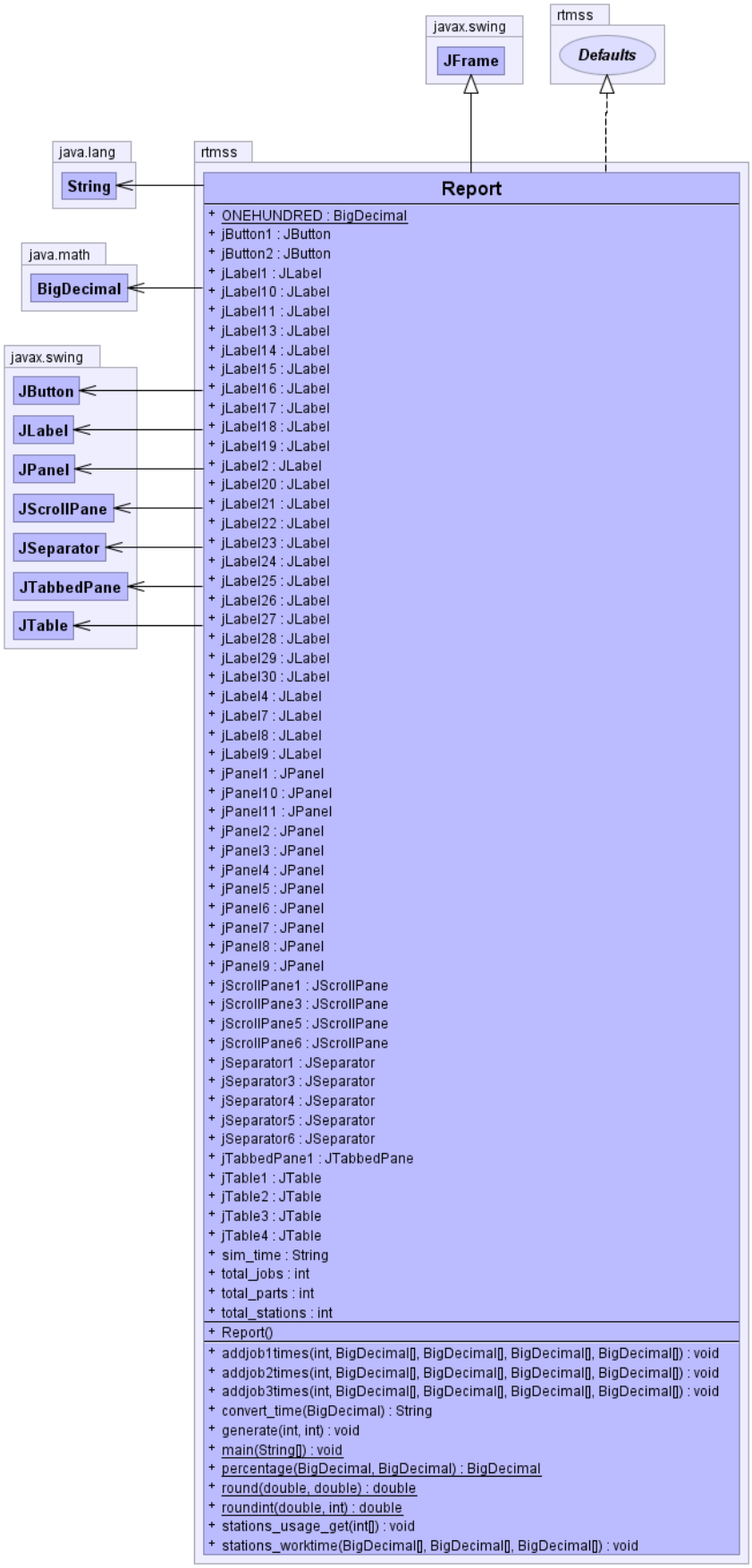
MMap Class Diagram:



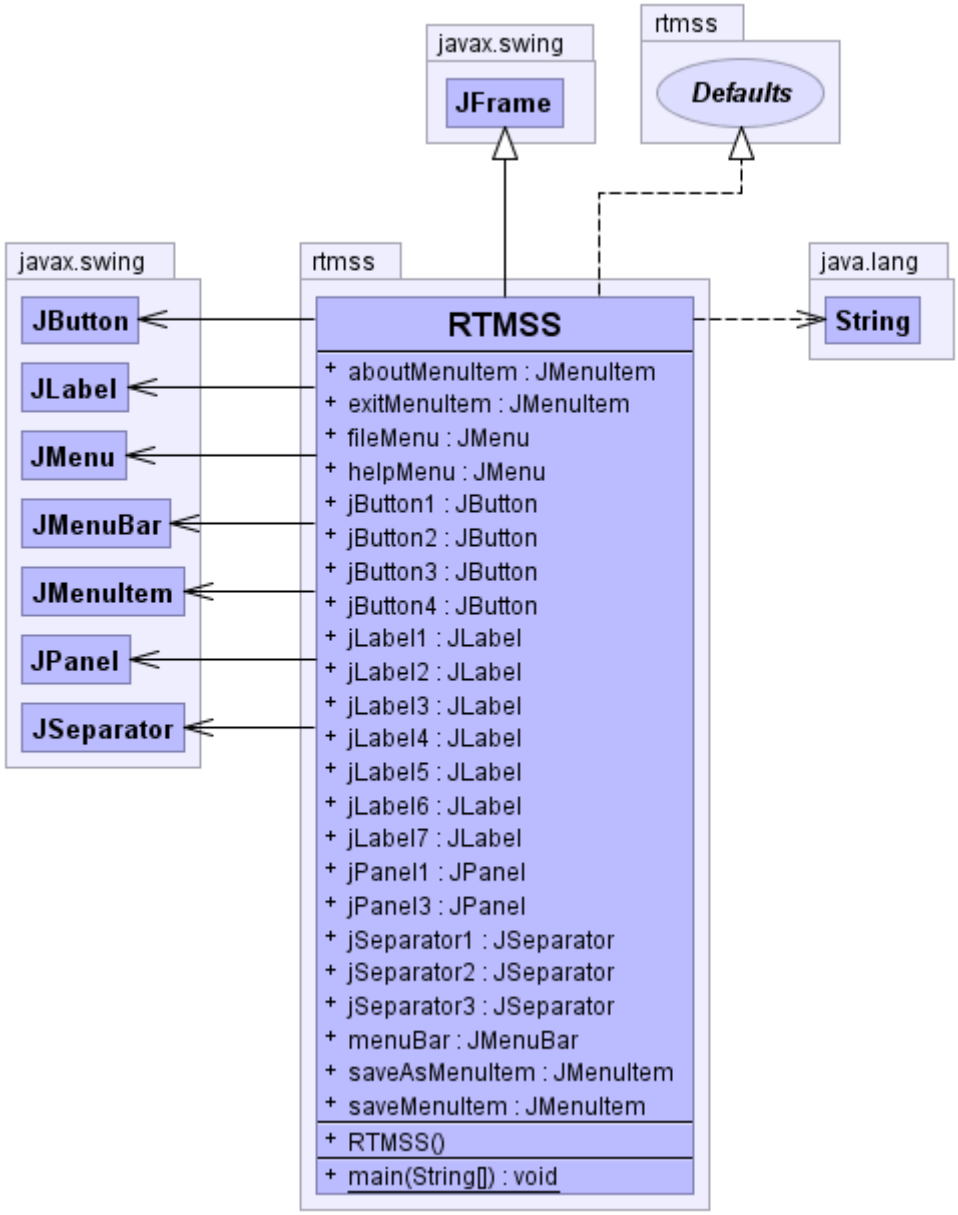
Output Class Diagram:



Report Class Diagram:

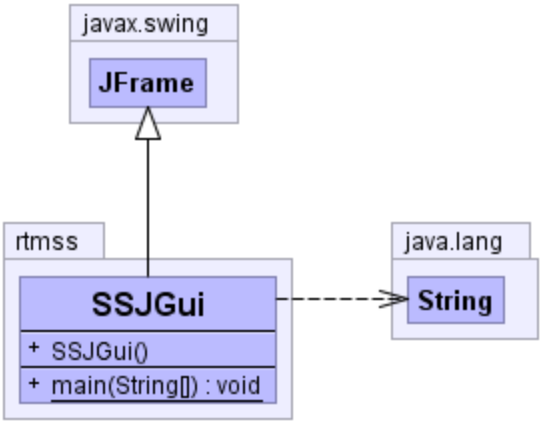


RTMSS Class Diagram:

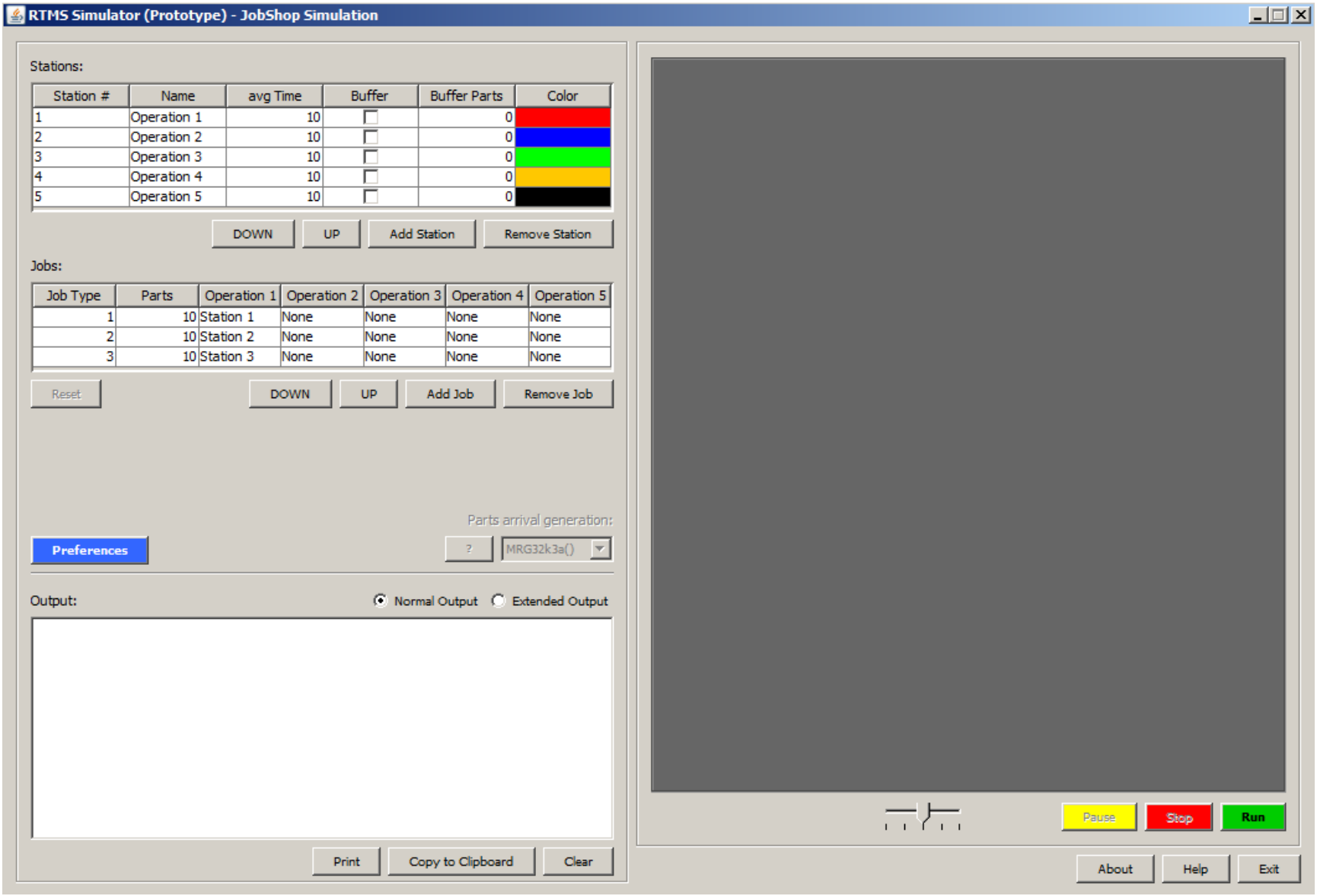


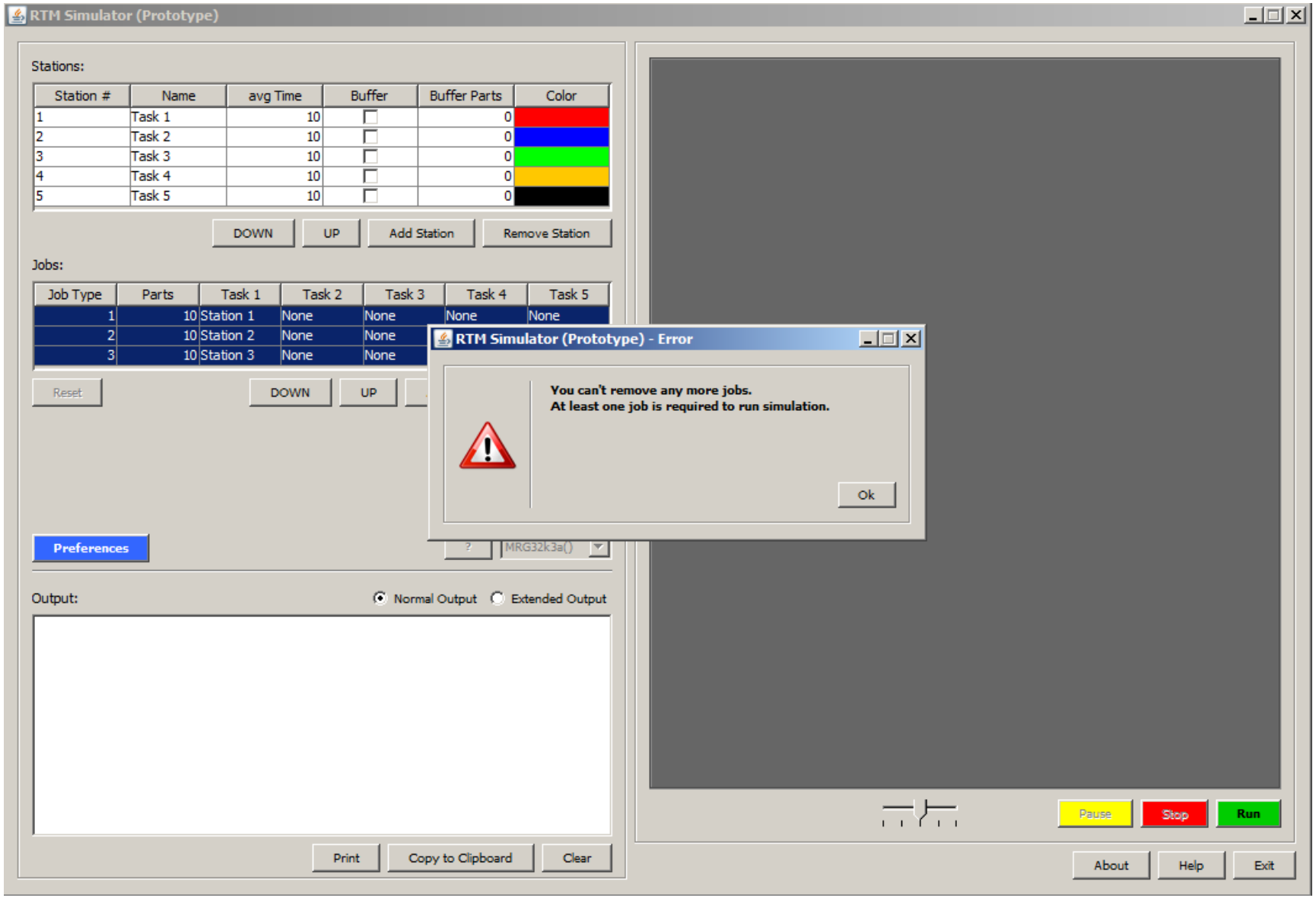
SimGui Class Diagram:

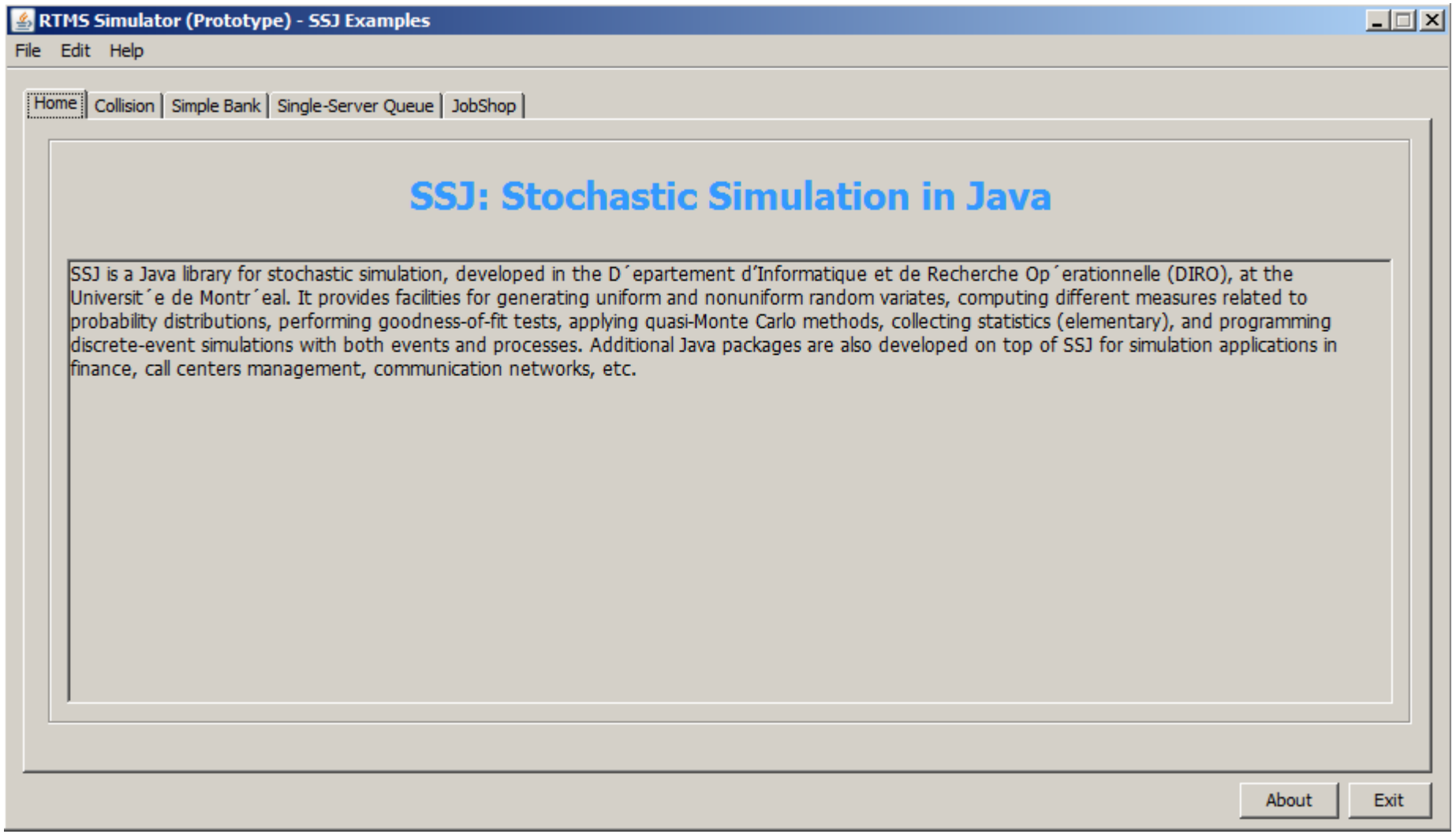
SSJGui Class Diagram:

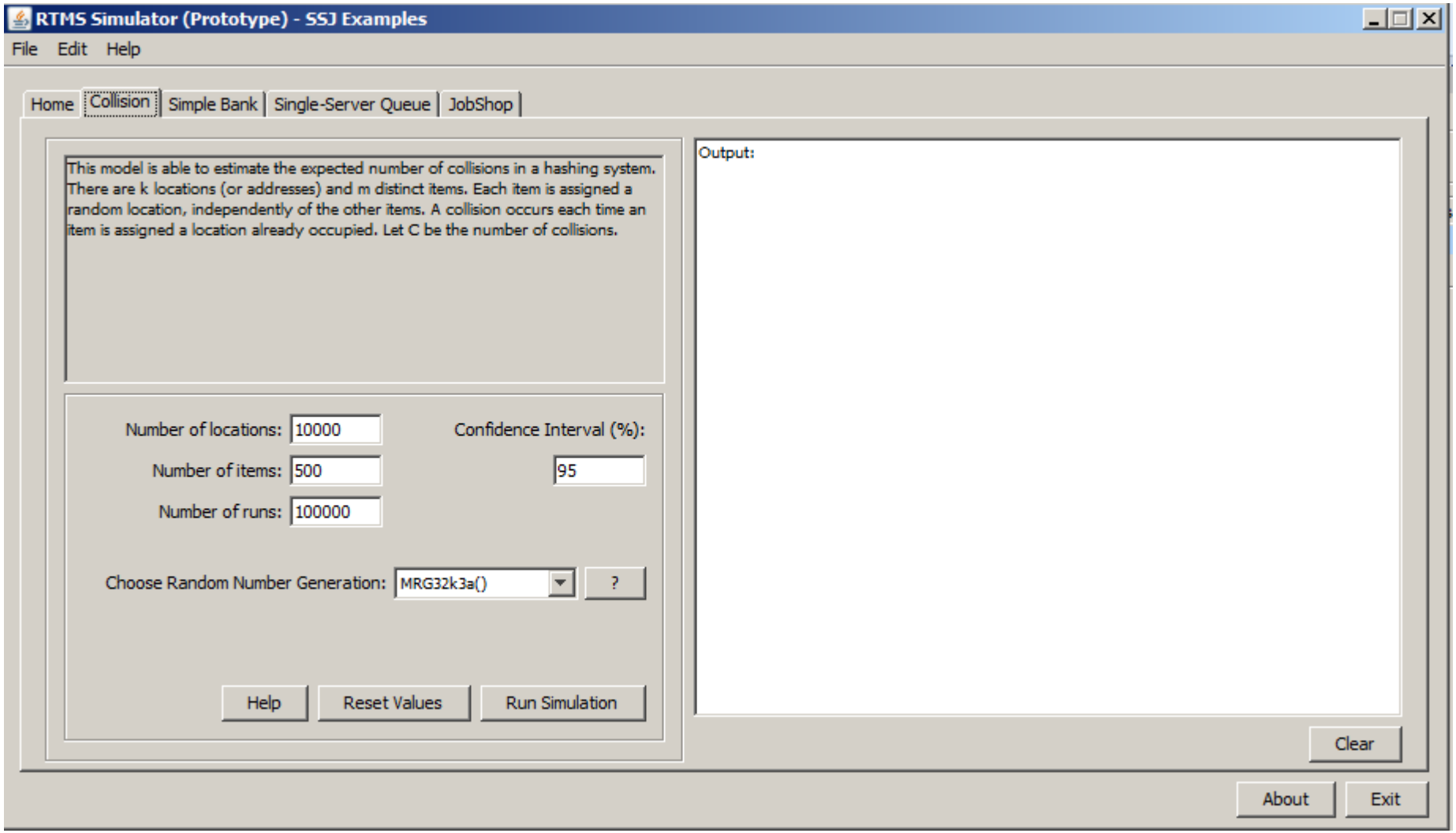


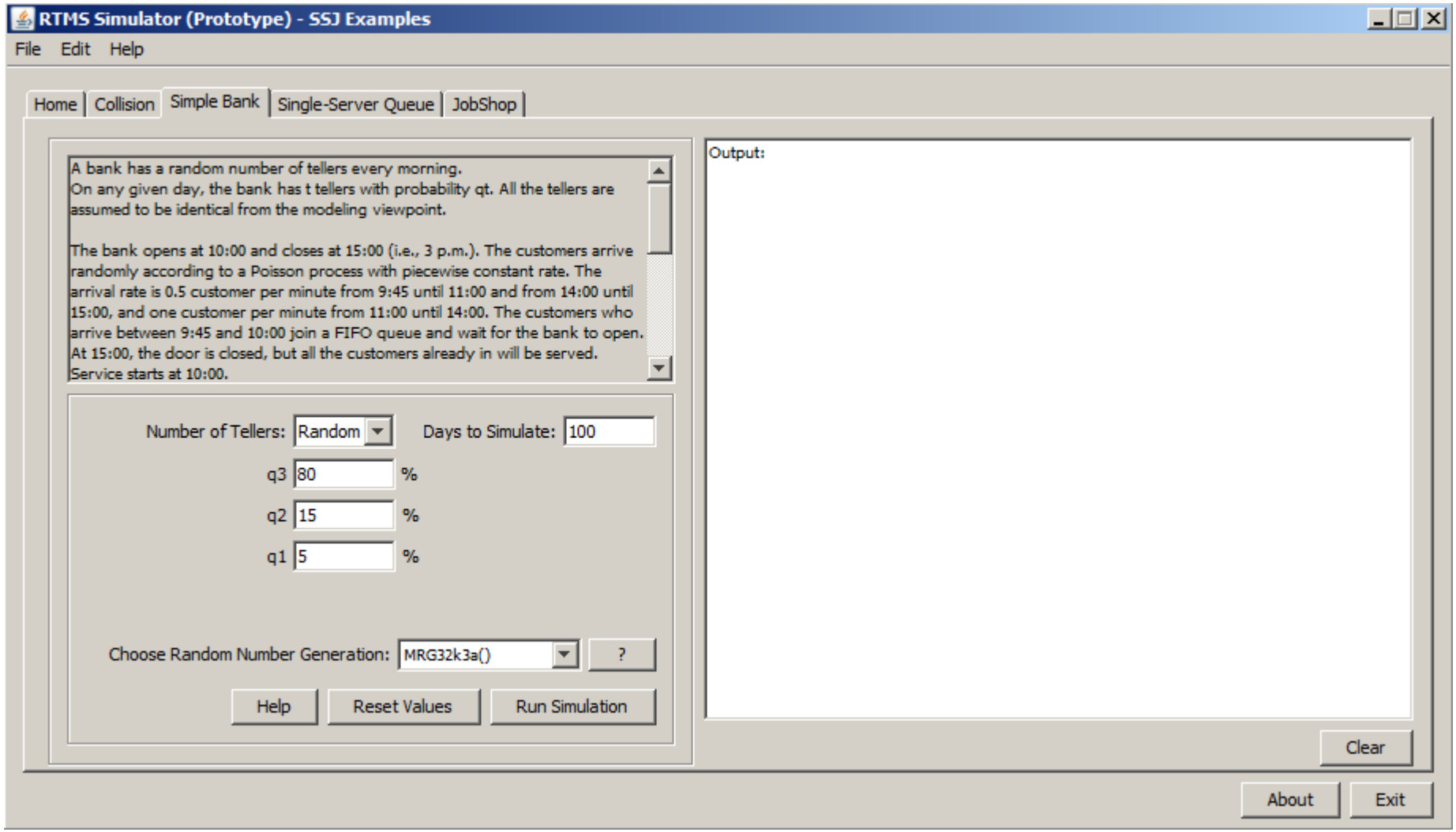
Appendix B. Prototype interface image

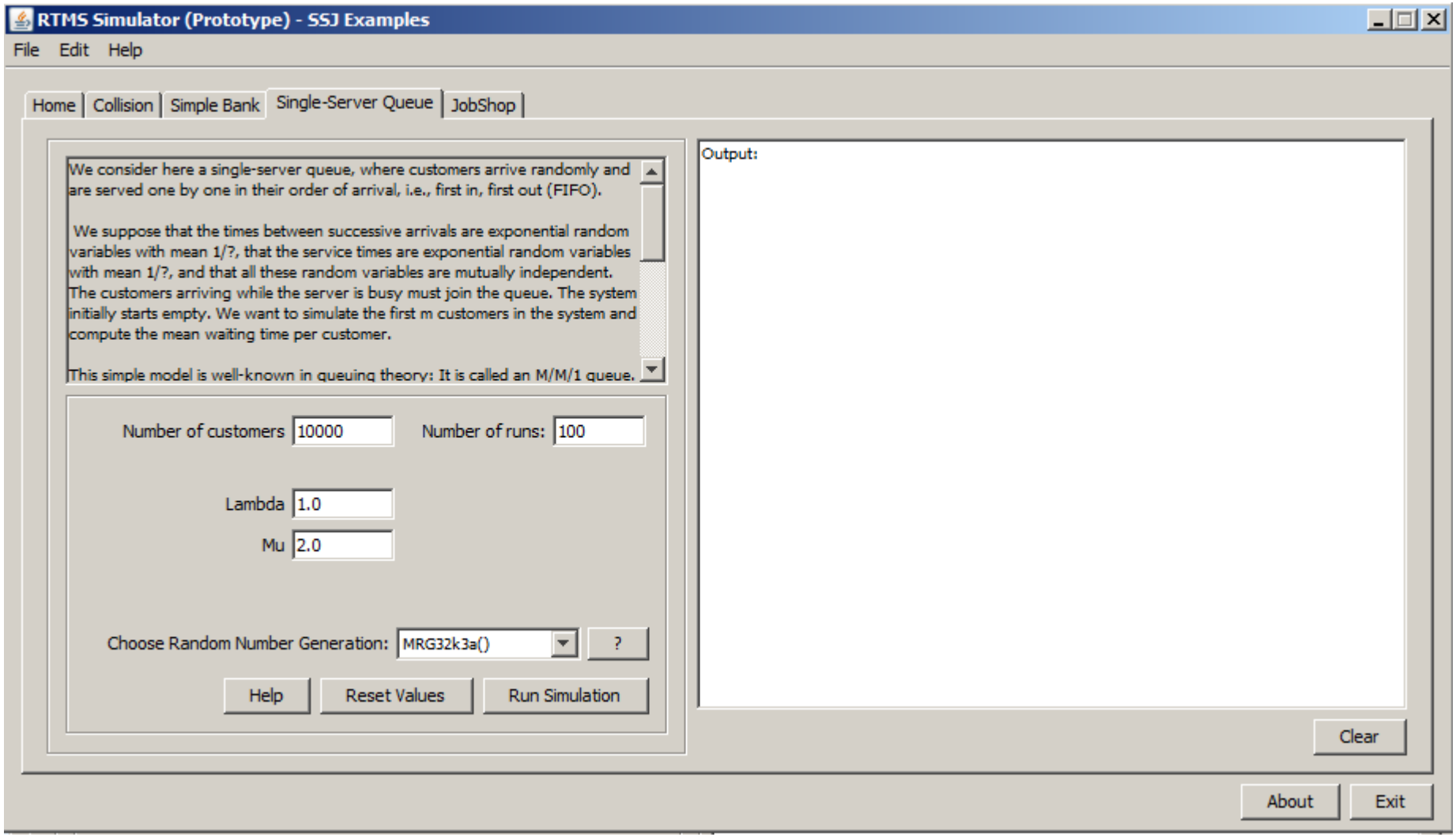

RTMS Simulator (Prototype) - JobShop Simulation
Stations:
Station # | Name | avg Time | Buffer | Buffer Parts | Color
1 | Operation 1 | 10 | 0
2 | Operation 2 | 10 | 0
3 | Operation 3 | 10 | 0
4 | Operation 4 | 10 | 0
5 | Operation 5 | 10 | 0
DOWN
UP
Add Station
Remove Station
Jobs:
Job Type | Parts | Operation 1 | Operation 2 | Operation 3 | Operation 4 | Operation 5
1 | 10 | Station 1 | None | None | None | None
2 | 10 | Station 2 | None | None | None | None
3 | 10 | Station 3 | None | None | None | None
Reset
DOWN
UP
Add Job
Remove Job
Parts arrival generation:
Preferences
?
MRG32k3a()
Output:
Normal Output
Extended Output
Print
Copy to Clipboard
Clear
Pause
Stop
Run
About
Help
Exit

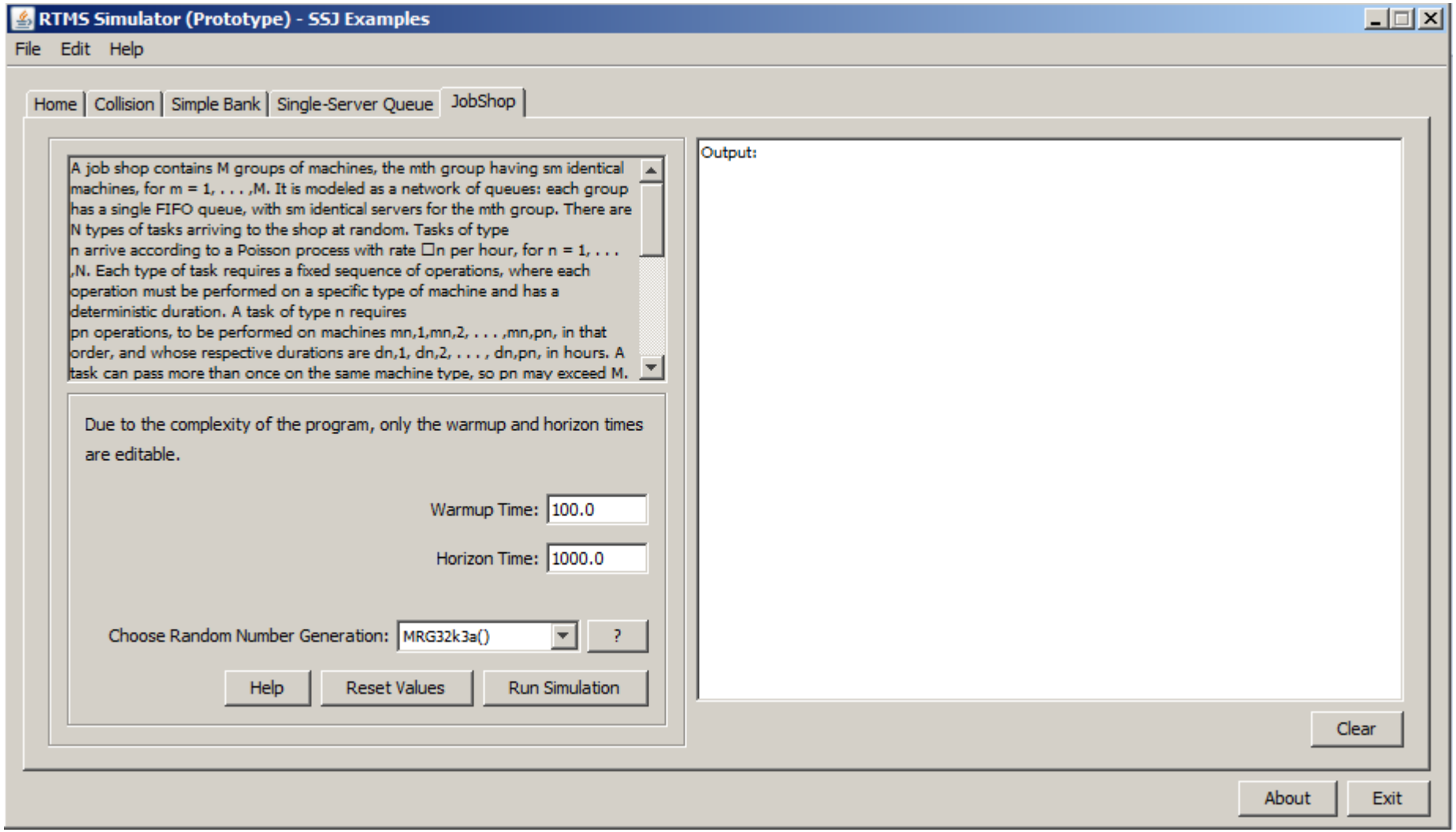

RTM Simulator (Prototype)
Stations:
Station # | Name | avg Time | Buffer | Buffer Parts | Color
1 | Task 1 | 10 | 0
2 | Task 2 | 10 | 0
3 | Task 3 | 10 | 0
4 | Task 4 | 10 | 0
5 | Task 5 | 10 | 0
DOWN
UP
Add Station
Remove Station
Jobs:
Job Type | Parts | Task 1 | Task 2 | Task 3 | Task 4 | Task 5
1 | 10 | Station 1 | None | None | None | None
2 | 10 | Station 2 | None | None
3 | 10 | Station 3 | None | None
Reset
DOWN
UP
RTM Simulator (Prototype) - Error
You can't remove any more jobs.
At least one job is required to run simulation.
Ok
Preferences
MRG32k3a()
Output:
Normal Output
Extended Output
Print
Copy to Clipboard
Clear
Pause
Stop
Run
About
Help
Exit

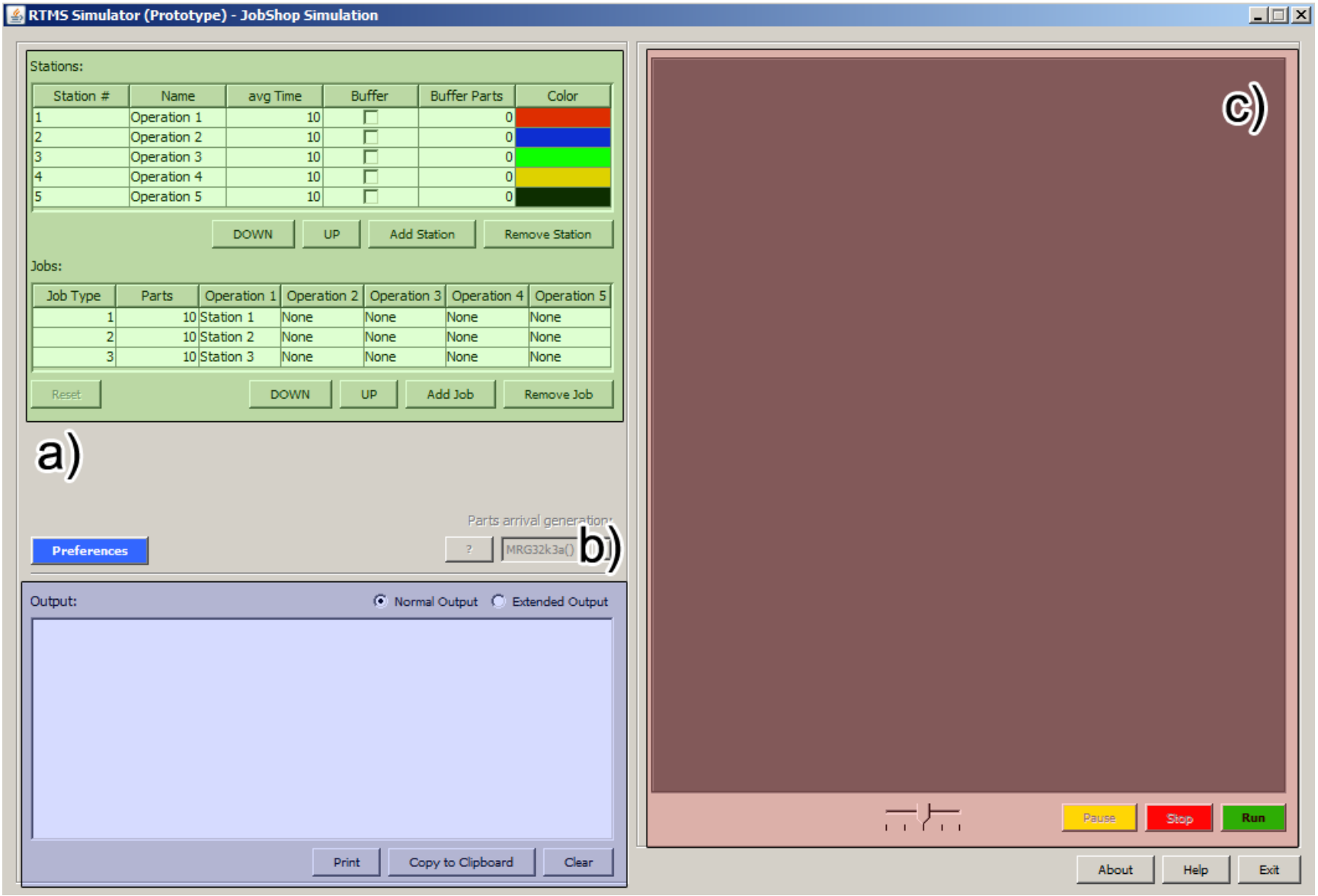

RTMS Simulator (Prototype) - SSJ Examples
File
Edit
Help
Home
Collision
Simple Bank
Single-Server Queue
JobShop
SSJ: Stochastic Simulation in Java
SSJ is a Java library for stochastic simulation, developed in the D´epartement d'Informatique et de Recherche Op´erationnelle (DIRO), at the Universit´e de Montr´eal. It provides facilities for generating uniform and nonuniform random variates, computing different measures related to probability distributions, performing goodness-of-fit tests, applying quasi-Monte Carlo methods, collecting statistics (elementary), and programming discrete-event simulations with both events and processes. Additional Java packages are also developed on top of SSJ for simulation applications in finance, call centers management, communication networks, etc.
About
Exit

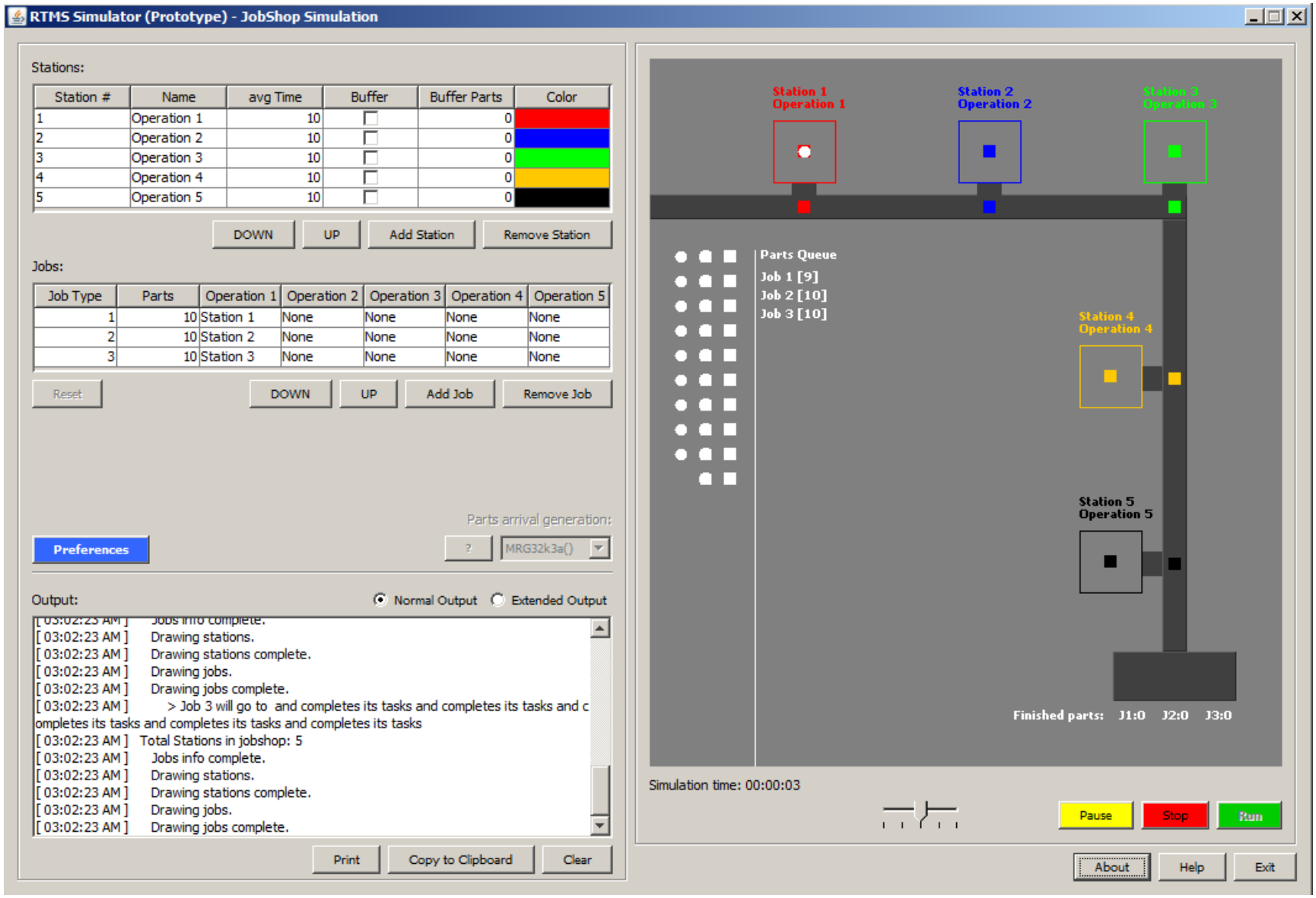

RTMS Simulator (Prototype) - SSJ Examples
File
Edit
Help
Home
Collision
Simple Bank
Single-Server Queue
JobShop
This model is able to estimate the expected number of collisions in a hashing system. There are k locations (or addresses) and m distinct items. Each item is assigned a random location, independently of the other items. A collision occurs each time an item is assigned a location already occupied. Let C be the number of collisions.
Number of locations:
10000
Confidence Interval (%):
Number of items:
500
95
Number of runs:
100000
Choose Random Number Generation:
MRG32k3a()
?
Help
Reset Values
Run Simulation
Output:
Clear
About
Exit


RTMS Simulator (Prototype) - SSJ Examples
File
Edit
Help
Home
Collision
Simple Bank
Single-Server Queue
JobShop
A bank has a random number of tellers every morning.
On any given day, the bank has t tellers with probability qt. All the tellers are assumed to be identical from the modeling viewpoint.
The bank opens at 10:00 and closes at 15:00 (i.e., 3 p.m.). The customers arrive randomly according to a Poisson process with piecewise constant rate. The arrival rate is 0.5 customer per minute from 9:45 until 11:00 and from 14:00 until 15:00, and one customer per minute from 11:00 until 14:00. The customers who arrive between 9:45 and 10:00 join a FIFO queue and wait for the bank to open. At 15:00, the door is closed, but all the customers already in will be served. Service starts at 10:00.
Number of Tellers:
Random
Days to Simulate:
100
q3
80
%
q2
15
%
q1
5
%
Choose Random Number Generation:
MRG32k3a()
?
Help
Reset Values
Run Simulation
Output:
Clear
About
Exit


RTMS Simulator (Prototype) - SSJ Examples
File
Edit
Help
Home
Collision
Simple Bank
Single-Server Queue
JobShop
We consider here a single-server queue, where customers arrive randomly and are served one by one in their order of arrival, i.e., first in, first out (FIFO).
We suppose that the times between successive arrivals are exponential random variables with mean 1/?, that the service times are exponential random variables with mean 1/?, and that all these random variables are mutually independent. The customers arriving while the server is busy must join the queue. The system initially starts empty. We want to simulate the first m customers in the system and compute the mean waiting time per customer.
This simple model is well-known in queuing theory: It is called an M/M/1 queue.
Number of customers
10000
Number of runs:
100
Lambda
1.0
Mu
2.0
Choose Random Number Generation:
MRG32k3a()
?
Help
Reset Values
Run Simulation
Output:
Clear
About
Exit


RTMS Simulator (Prototype) - SSJ Examples
File
Edit
Help
Home
Collision
Simple Bank
Single-Server Queue
JobShop
A job shop contains M groups of machines, the mth group having sm identical machines, for m = 1, . . . ,M. It is modeled as a network of queues: each group has a single FIFO queue, with sm identical servers for the mth group. There are N types of tasks arriving to the shop at random. Tasks of type
n arrive according to a Poisson process with rate □n per hour, for n = 1, . . . ,N. Each type of task requires a fixed sequence of operations, where each operation must be performed on a specific type of machine and has a deterministic duration. A task of type n requires
pn operations, to be performed on machines mn,1,mn,2, . . . ,mn,pn, in that order, and whose respective durations are dn,1, dn,2, . . . , dn,pn, in hours. A task can pass more than once on the same machine type, so pn may exceed M.
Due to the complexity of the program, only the warmup and horizon times are editable.
Warmup Time:
100.0
Horizon Time:
1000.0
Choose Random Number Generation:
MRG32k3a()
?
Help
Reset Values
Run Simulation
Output:
Clear
About
Exit


RTMS Simulator (Prototype) - JobShop Simulation
Stations:
Station # | Name | avg Time | Buffer | Buffer Parts | Color
1 | Operation 1 | 10 | 0
2 | Operation 2 | 10 | 0
3 | Operation 3 | 10 | 0
4 | Operation 4 | 10 | 0
5 | Operation 5 | 10 | 0
DOWN
UP
Add Station
Remove Station
Jobs:
Job Type | Parts | Operation 1 | Operation 2 | Operation 3 | Operation 4 | Operation 5
1 | 10 | Station 1 | None | None | None | None
2 | 10 | Station 2 | None | None | None | None
3 | 10 | Station 3 | None | None | None | None
Reset
DOWN
UP
Add Job
Remove Job
a)
Parts arrival generation
Preferences
?
MRG32k3a()
b)
Output:
Normal Output
Extended Output
Print
Copy to Clipboard
Clear
c)
Pause
Stop
Run
About
Help
Exit


RTMS Simulator (Prototype) - JobShop Simulation
Stations:
Station # | Name | avg Time | Buffer | Buffer Parts | Color
1 | Operation 1 | 10 | 0
2 | Operation 2 | 10 | 0
3 | Operation 3 | 10 | 0
4 | Operation 4 | 10 | 0
5 | Operation 5 | 10 | 0
DOWN
UP
Add Station
Remove Station
Jobs:
Job Type | Parts | Operation 1 | Operation 2 | Operation 3 | Operation 4 | Operation 5
1 | 10 | Station 1 | None | None | None | None
2 | 10 | Station 2 | None | None | None | None
3 | 10 | Station 3 | None | None | None | None
Reset
DOWN
UP
Add Job
Remove Job
Parts arrival generation:
Preferences
?
MRG32k3a()
Output:
Normal Output
Extended Output
[03:02:23 AM] Jobs info complete.
[03:02:23 AM] Drawing stations.
[03:02:23 AM] Drawing stations complete.
[03:02:23 AM] Drawing jobs.
[03:02:23 AM] Drawing jobs complete.
[03:02:23 AM] > Job 3 will go to and completes its tasks and completes its tasks and c
ompletes its tasks and completes its tasks and completes its tasks
[03:02:23 AM] Total Stations in jobshop: 5
[03:02:23 AM] Jobs info complete.
[03:02:23 AM] Drawing stations.
[03:02:23 AM] Drawing stations complete.
[03:02:23 AM] Drawing jobs.
[03:02:23 AM] Drawing jobs complete.
Print
Copy to Clipboard
Clear
Station 1
Operation 1
Station 2
Operation 2
Station 3
Operation 3
Parts Queue
Job 1 [9]
Job 2 [10]
Job 3 [10]
Station 4
Operation 4
Station 5
Operation 5
Finished parts: J1:0 J2:0 J3:0
Simulation time: 00:00:03
Pause
Stop
Run
About
Help
Exit