
Universidade Federal
do Rio de Janeiro
Escola Politécnica

# PROCESSOS COM RECURSOS COMPARTILHADOS: MODELAGEM E CONTROLE UTILIZANDO FILAS E SIMULAÇÃO BASEADA EM EVENTOS

HENRIQUE WERNER CASTELO BRANCO

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador:
Prof. Amit Bhaya, Ph.D

Rio de Janeiro, RJ - Brazil

Março - 2015

# PROCESSOS COM RECURSOS COMPARTILHADOS: MODELAGEM E CONTROLE UTILIZANDO FILAS E SIMULAÇÃO BASEADA EM EVENTOS

HENRIQUE WERNER CASTELO BRANCO

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Aprovado por:

_____
Prof. Amit Bhaya, Ph.D

_____
Prof. Eugenius Kaszkurewicz, D.Sc.

_____
Prof. Oumar Diene, D. Sc.

Rio de Janeiro, RJ - Brasil

Março - 2015

Werner Castelo Branco, Henrique.
Processos com recursos compartilhados: modelagem e controle utilizando filas e simulação baseada em eventos/ Henrique Werner Castelo Branco - Rio de Janeiro: UFRJ/ Escola Politécnica, 2015.
X, 101 p.: il.; 29,7 cm.

Orientador:
Prof. Amit Bhaya, Ph.D

Projeto de Graduação - UFRJ/ Escola Politécnica/Curso de Engenharia de Controle e Automação, 2015.

Referências Bibliográficas: p. 89 - 92.

1. Controle de processos 2. Teoria das filas 3. Compartilhamento de recursos 4. SimEvents 5. Pattern Search I. Bhaya, Amit. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

# Acknowledgement

This work would never be completed if it was not for the support of some people, whom I would like to acknowledge below:

To my parents, Domingos and Ana Maria, who have always been by my side, giving support and advices in times of need.

To my brother Guilherme, who even though is the youngest of us two, has already taught me so much.

To my family, to those who are still physically with me and to those who are long gone, you will be always be in my thoughts to enlighten my path.

To all my friends, whether new or old, thanks for your caring support.

To my supervisor, Prof. Amit Bhaya, that has helped me along this work with his guidance, comments and suggestions.

To the Federal University of Rio de Janeiro professors and administrative staff, for helping me in the way to become an engineer.

To Elizabeth Moore, who has never let me stop believing in myself and in better days.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação.

# Processos com recursos compartilhados: modelagem e controle utilizando filas e simulação baseada em eventos

Henrique Werner Castelo Branco

2015

Orientador:
Prof. Amit Bhaya, Ph.D
Curso: Engenharia de Controle e Automação

Esse trabalho apresenta a modelagem de processs utilizando simulação baseada em eventos, a partir da biblioteca SimEvents do software Simulink, focando em um exemplo de restaurante "fast food" com recursos compartilhados. Aspectos financeiros do restaurante são incluídos no modelo a fim de maximizar o Economic Value Added, pelo uso do algoritmo Pattern Search para determinar os parâmetros ideais para o controlador PID que regula o preço cobrado para os clientes. Esse modelo de restaurante "fast food" pode então ser utilizado como uma ferramenta de apoio por um gerente de um restaurante. Ao inserir informaçẽos sobre o sistema a ser analisado no simulador, experimentos podem ser realizados e impactos reais previstos a partir dos dados de simulação.

*Palavras Chaves:* Controle de processos, Teoria de filas, Compartilhamento de recursos, SimEvents, Pattern Search.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfilment of the requirements for the degree of Control and Automation Engineer.

# Processes with Shared Resources: Modelling and Control using Queues and Discrete Events Simulation

Henrique Werner Castelo Branco

2015

Advisor:

Prof. Amit Bhaya, Ph.D

Course: Engenharia de Controle e Automação

This work presents the modelling of processes in discrete event simulation, using the SimEvents library of the Simulink software, targeting the implementation of a fast food restaurant example with shared resources. Financial aspects of the restaurant are included in the model in order to maximize the Economic Value Added, using the Pattern Search algorithm for optimization, thus determining the ideal parameters for a PID controller that regulates the price charged for the clients. This model of the fast food restaurant can then be used as a support tool for a real restaurant manager. When information about the system to be analysed is fed to the simulator, then experiments can be performed and the real impacts can be predicted from the simulated data.

*Keywords:* Process control, Queueing theory, Resource sharing, SimEvents, Pattern Search.

# Contents

# List of Figures

# List of Tables

# List of abbreviations

CC - Capital Charge

CT - Cycle Time

D - Deterministic Process

DES - Discrete Event Simulation

EVA - Economic Value Added

FIFO - First-In-First-Out

G - General Description Process

GA - Genetic Algorithm

GUI - Graphical User Interfaces

IIT - Interarrival Time

LIFO - Last-In-First-Out

MTBF - Mean Time Between Failures

M - Exponential/Markovian Process

MTTR - Mean Time To Repair

NOPAT - Net Operating Profit after Tax

ODE - Ordinary differential equation

PS - Pattern Search

QoS - Quality of Service

TR - Total Revenue

WIP - Work in Progress

# List of Symbols

$C(s)$: controller's transfer function

$e$: error signal to the controller

$Ref$: reference signal to the controller

$y$: measured signal from process to the controller

$u$: control signal to the plant

$G(s)$: plant's transfer function

$I(k)$: retail store's stock in day $k$

$W(k)$ : retail store's shipped items in day $k$

$O(k)$ : retail store's ordered items in day $k$

$O(k-\tau)$ : retail store's receive items in day $k$

$R(k)$ : retail store's receive items in day $k$

$S(k)$ : retail store's sold products in day $k$

$D(k)$ : retail store's product demand in day $k$

$k$ : discretized time of the retail store's model, in days

$\tau$ : lead time of the retail store's model

$\frac{1}{\lambda}$: Interarrival time (IIT)

$\frac{1}{\mu}$: Service time

$\lambda$: Interarrival rate

$\mu$: Service rate

$\rho$: Traffic intensity

$L_q$: Average queue length

$s$: number of parallel servers in the process

$P(j \geq s)$: Probability of the amount of customers to be bigger than the amount of available servers in a Parallel Queueing System

server $ij$: server from process $i$ that uses resource $j$

$E$: Coefficient of Price Elasticity of Demand

$Q$: Demand

$P$: Price

$\Delta Q$: Demand variation between two points in a price elasticity of demand curve

$\Delta P$: Price variation between two points in a price elasticity of demand curve

$a$: constant of the Linear Demand x Price curve

$b$: slope of the Linear Demand x Price curve

$E_0$: Coefficient of Price Elasticity of Demand at point $0$

$Q_0$: Demand at point $0$

$P_0$: Price at point $0$

$T$: Customer tolerance

$P$: PID's Proportional gain

$I$: PID's Integral gain

$D$: PID's Derivative gain

$E[X]$: Expected value of variable X

$\sigma^2$: Standard Deviation

# Chapter 1

# Introduction

Systems with Shared Resources are, in a simplified definition, systems with a physical constraint on the availability of the equipment that the system can use at a given time. This type of system appears quite frequently in our daily routine. Some everyday examples of Shared Resources are as follows:

- Traffic intersections, at which two or more different roads share the common or intersection space, and must do so in sequential order;
- At a fast food restaurant, in which there is only one soft drink machine to serve the whole store;
- At work places, where, typically a limited number of employees must perform all the tasks required to keep the business running;
- As you are reading this text on your computer, sharing of the CPUs computing resources occurs millions of times per second, as it manages all services such as Hard Drive accesses, programs and inputs from the user (keyboard, mouse, microphone and so on), all at the same time.

Usually, resource sharing in a system becomes necessary when the entity in charge of the shared resource(exemplifying, in the above cases, such as the city's traffic control department, the restaurant manager, the company boss and the CPU designer, respectively), wishes to optimize resource utilization available, in order to minimise costs, without compromising the

Quality of Service (QoS) too much.

In this work, some examples of such systems will be given, with a focus on the particular problem of a fast food restaurant that will be studied in details. Subsequently, taking into account the specific characteristics of a fast food restaurant with shared resources, a control law is suggested to maximize its financial results.

## 1.1 Motivation

Several fields of study have already targeted this type of problem: at least two different approaches have been taken, one from the viewpoint of network systems engineers [26] who work with the transmission of data over shared links and one from business managers [36] who work with processes that share resources and must generate revenue for the stakeholders. Network engineers typically use a mathematical optimization based approach to achieve congestion control without losing QoS, while the managers expend much effort on modelling and graphical understanding of the system usually using ad hoc control methods to increase the process revenue.

This project attempts a greater level of formalization, focusing on business systems with shared resources as well as their economic evaluation, using the Simulink environment as a modelling tool to represent the dynamics of the problem and investigate the impact of the Control Law proposed. The Simulink environment is a block diagram environment for simulation, which is a part of the Matlab® program. Most of the blocks used in this project are from the SimEvents library, since it provides a framework for the proposed modelling schemes.

## 1.2 Objectives

The objectives of this work are as follows:

**i** To give an overview of the different tools available for the modelling, simulation and control of systems with shared resources and select the most suitable of them;

**ii** To replicate an existing model, due to [16], of a fast food restaurant with two shared resources, using the chosen tool;

**iii** To modify the fast food restaurant model to include financial aspects;

**iv** To develop a control law capable of maximizing, in a given time frame, a financial criterion, the Economic Value Added (EVA) metric that measures the economic performance of the model;

**iv** To perform a feasibility study and sensitivity analysis for a set of parameters for the fast food restaurant example.

## 1.3 Organization of the Project

This First Chapter gave the reader a general introduction to the main topic of this work, systems with shared resources, together with a presentation of the objective and the motivation for this study.

Chapter 2 begins by presenting the theoretical definitions necessary for this project, as well as, the basic notions of Processes. A brief comparative discussion of the possible modelling commercially available platforms is also going to be presented, with the critical points that lead to the choice of the Simulink® as the Environment for this project, with its fundamental blocks from the SimEvents library.

In Chapter 3, some process problems are explained leading up to the fast food restaurant problem with shared resources and detailing the basic model for it. Economic, control and optimization aspects of the problem are not considered at this point.

Chapter 4 introduces an objective function based on a financial metric and then presents tools to maximize the proposed objective function for the fast food example from chapter 3.

Chapter 5 presents the results obtained using the methods from chapter 4 implemented for the fast food restaurant example presented in chapter 3, including a feasibility study and sensibility analysis of the system for two different scenarios: with and without shared resources.

The last chapter summarises the content and results of this project by presenting conclusions and suggesting topics for future work.

# Chapter 2

# Preliminaries and review of simulation concepts

There are a variety of possible approaches to model real-life phenomena [3]. A model is an abstraction of the reality that can be used to experiment and predict the system's evolution. Standard approaches to a control problem are to model the controller and the plant in terms of their state-space matrices, transfer functions and/or dynamic equations, and most controller design methods use one of these approaches, see, for example, [14], [28] and [10]. For example, in the transfer function approach, a controller $C(s)$ receives the error $e$ between a reference $Ref$ and the measured signal $y$, generating a signal $u$ to control the plant $G(s)$, as shown in the block diagram in figure 2.1.

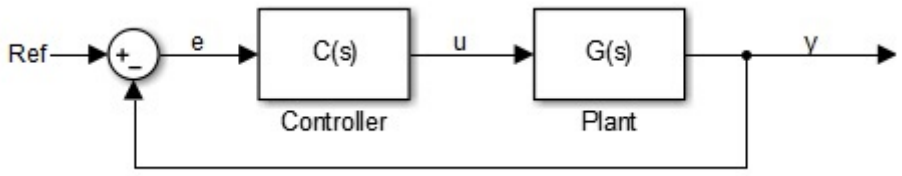


Figure 2.1: Typical block diagram of a control system

A part of a logistic system is modelled in figure 2.2 using a discrete-time state space approach, focusing on the retail process. This process has been much studied and described, for example, in the textbook [16], and has been investigated in, for example, [4], [25], and [12].

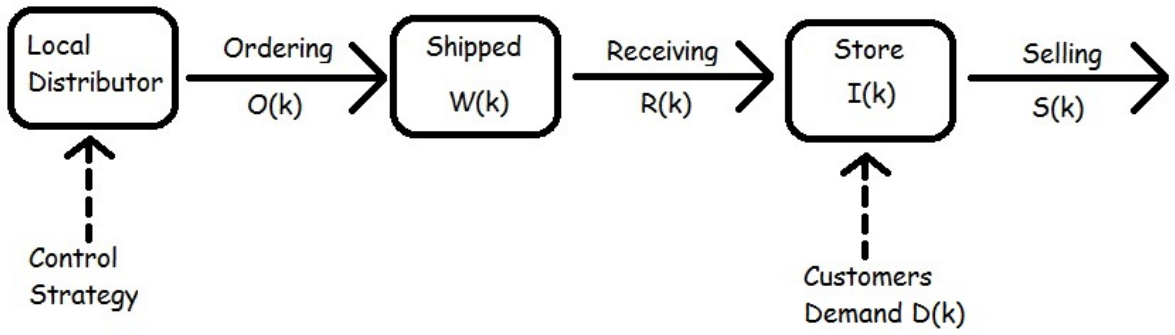


Figure 2.2: Model of the retail store to be investigated

The daily sale of the store $S(k)$ is based on the customers demand $D(k)$, which is the amount of customers that shows up at the store, and the amount of items in the inventory, which has a limited size. It is assumed that the store is obliged to keep at least one unit for display. Following an ordering control logic, $O(k)$, the store's manager requests daily to the local distributor more items, who takes a certain amount of time $\tau$ to deliver these products. The sum of the products that are on their way to the store is denominated Shipped or $W(k)$. The incoming items at each time step are the receiving items $R(k)$ and $R(k) = O(k - \tau)$.

From [4], for $k > \tau$, the discrete-time recurrence model of this system can be written as:

$$I(k) = I(k-1) + O(k-\tau) - S(k) \tag{2.1}$$

$$W(k) = W(k-1) + O(k) - O(k-\tau) \tag{2.2}$$

$$S(k) = f(D(k), I(k)) = \begin{cases} D(k), \text{ if} I(k-1) - D(k) > 1, \\ 1, \text{ otherwise} \end{cases} \tag{2.3}$$

$$R(k) = O(k-\tau) \tag{2.4}$$

Thus, the variables $I(k)$ and $W(k)$ are the outputs of two integrators, natural choices for state variables. The independent variable $k$ represents the discretized time of the model, in days. The amount of items that arrive at the store's inventory ($O(k$ - $\tau)$) is the same as the amount

ordered ($O(k)$) $\tau$ days before, thus the lead-time of the system is equal to $\tau$. The vector $x(k)$ is defined as:

$$x(k) = \begin{bmatrix} I(k) \\ W(k) \end{bmatrix} \tag{2.5}$$

Thus, from equations 2.1, 2.3 and 2.2, we get the overall discrete-time recurrence representation:

$$x(k) = x(k-1) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} O(k) + \begin{bmatrix} 1 \\ -1 \end{bmatrix} O(k-\tau) + \begin{bmatrix} -1 \\ 0 \end{bmatrix} S(k) \tag{2.6}$$

Modelling a system using an event-based approach, also known as a Discrete Event Systems (DES), differs from the usual difference equations modelling paradigm for discrete systems. The DES terminology describes a system in terms of processes, events, servers and queues, explained in more detail below. As a comparison, the example presented in this section will be remodelled in chapter 3 using the DES approach.

## 2.1 Basic Terminology for Discrete Event Simulation and Processes

The main purpose of a business organization is to create more wealth for its owners than they could get by investing elsewhere. In order to achieve this, the organization's managers have to find the best allocation of the available resources. An organization is a complex system, which can be defined as a regularly interacting or interdependent group of items, the component processes, composing a unified whole. Like any other complex system, tools for better comprehension of the organization's dynamics are necessary.

Each of the component processes is, by definition, a series of intermediate actions, activities or operations leading to the end product. Decomposition can be used recursively on the system's component processes until the desired level of detail is obtained for the model, going from a high-level abstraction of the process to its most fundamental operations. Each component process consumes inputs (raw materials, electricity, memory, ...) and yields physical (manufacturing processes) or information (service processes) outputs.

### 2.1.1 Process classification

Processes can be divided with respect to their intrinsic properties. In fact, real-life processes can be classified into regions depending on how detailed the model is and the specific part of the process that is being analysed. The main groups related to processes conditions are explained below, using the retail store example explained before to compare both categories:

- **Discrete versus Batch Processes**

  A discrete process produces outputs in separate units, while in a batch process, the items are grouped together into batches and then processed.

  In the retail store, for example, the ordering process can be classified as a discrete process if each individual unit is requested as soon as necessary or as a batch process, in case the store's manager waits for a certain amount, a backlog, to send a consolidated request to the local distributor for a batch of units.

- **Single versus Multistep Processes**

  In a single-step Process, only one component process is present in the model and, in the multistep, there are multiple component processes present in the model, which means that the whole process is divided into several phases, i.e. steps, that can be analysed individually. Single-step process can be transformed into a multistep through the addition of more details about the system, and, in the other direction, a multistep process can be compressed into a single-step process by merging intermediate steps.

  In the retail store example, a multistep process could be described as follows: (i) the customer enters the store and asks an employee for one product, as the first component process, (ii) then pays for the product, the second component process and (iii) finally receives the product in the third and last component process before leaving the store. As a single-step process, these three operations would occur in only one component process without halting, representing the shopping process as a whole.

- **Sequential versus Parallel Processes**

In a sequential or serial process, all items follow the same path through the process, subjected to the same operations, since each one is processed individually. For a parallel process, one or more of the above conditions is not satisfied, implying that items possibly take different routes through the process.

In the examples, for a sequential process, the customer enters the store and asks for one product, in the first component process of the system, then, tries the product as the second component process, finally paying and receiving the product as the last component process before leaving the store, with only one employee in the store. A parallel process could have more than one employee at the store, being able to serve two customers at once, or the customer can have an option of trying the product or not before buying it, or also each employee at the store can server multiple customers at the same time.

- **Balanced versus Unbalanced Processes**

  A balanced process has no excess capacity in the process, which means that, in case all items arrived at a certain constant rate, the process would always be executing without stopping. In an unbalanced process, the component processes do not execute without stopping, due to restrictions on the arrival of items in each unit.

  For the retail store model, the customer enters the store, asks for and receives one product, in one component process of the model, then, in another component process, the customer pays for the product before leaving the store, with one employee at each of these component process. If this process was balanced, all customer would go straight from the first component process to the second component process, without stopping. Otherwise, there would be a delay between concluding at the first component process and being able to enter the second one. Alternatively, the employee at the second component process could finish his task faster than the first one, generating excess capacity.

- **Open versus Closed Processes**

  In a closed process, all the items involved were already present in the process before the

period of time analysed, being served over and over (cycling through the process), never exiting the system. On the other hand, an open process, entities are generated during the simulation, entering the process, getting served and then leaving the process.

An employee's work at the retail store can be described an closed process, when no hiring or firing of employees is allowed. The employee serves the customer, in the system's only component process, and then returns to a buffer to wait to serve the next incoming customer, never actually leaving the system. An open process can be considered as the customer entering the store, being served and then leaving the store.

- **Tightly Coupled versus Uncoupled Processes versus Loosely Coupled**

  A tightly coupled process consists of several component processes in series, with no waiting areas or buffers separating them. Thus if one of the component process is busy processing an item, and a new item arrives, then it is hindered from proceeding further and the busy component process is said to be in a blocked state. Clearly, a component process in a blocked state affects the flow of units (entities) to all component processes downstream. The blocking phenomenon was humorously described by the former baseball player Yogi Berra as: "Nobody goes to that restaurant any more; it's too crowded." In an uncoupled process, there is an infinite storage space between two successive component processes, which can accommodate all the items that have finished from the upstream step to follow to the downstream step, leading to no rejection of a request to be processed (known as balking) at any given time. A loosely coupled process lies somewhere in between these two situations, having limited storage between two successive component processes.

  Suppose that, in the retail store example, there is only one cashier and one salesperson, with the customer entering the store and asking for one product, in the first component process to the salesperson, then, moving to another system's component process to pay for the product to the cashier before leaving the store with the product. As an example of a tightly couple process, suppose that the salesperson can only serve another customer after transferring his current one to the cashier. Then, if it happens that the cashier is busy

with a customer and the salesperson wants to transfer his served customer to the cashier, he would have to wait until the cashier is free. In an uncoupled Process, the salesperson can ask for his served customers to enter in a queue for paying for the item, from where the cashier would request the first in line whenever he is available, freeing the salesperson to serve another customer meanwhile.

The simplest representation of a system, a single-step process model, is now introduced. In this model, there is only one component process, to which the inputs are fed, spending a given amount of time inside it, in order to exit the process. A real-life process that can be completely modelled as a single-step process is a conveyor belt in figure 2.3.

Figure 2.3: A single-step process: A conveyor belt. Figure source: [23]

In the single-step process, there are three fundamental characteristics, which are exemplified for the conveyor belt model:

- **Thruput**: The rate, in inputs per time unit, at which units exit the process while flowing through the process to the output, in this case, flowing though the conveyor belt;
- **Cycle Time (CT)**: the time one unit spends within the external and internal boundaries of the process, in the conveyor belt, the CT is the time that takes for one input to go through the whole conveyor belt, or, in other words, the length of time that it is resident in the conveyor belt;
- **Work in Progress (WIP)**: the amount of units in a process at a given instant, that is, the amount of inputs on the whole conveyor belt;

In figure 2.4, there is a simple side representation of a conveyor belt. The conveyor belt is $1$ m long and works in a constant speed of $0.1$ m/s. Only one product, which has negligible size , can be put in the beginning of the belt at a time, which happens every $2$ seconds, being removed from the belt instantaneously when it arrives at the end of the belt. In figure 2.4, a new item was inserted in the conveyor belt $1$ second ago.

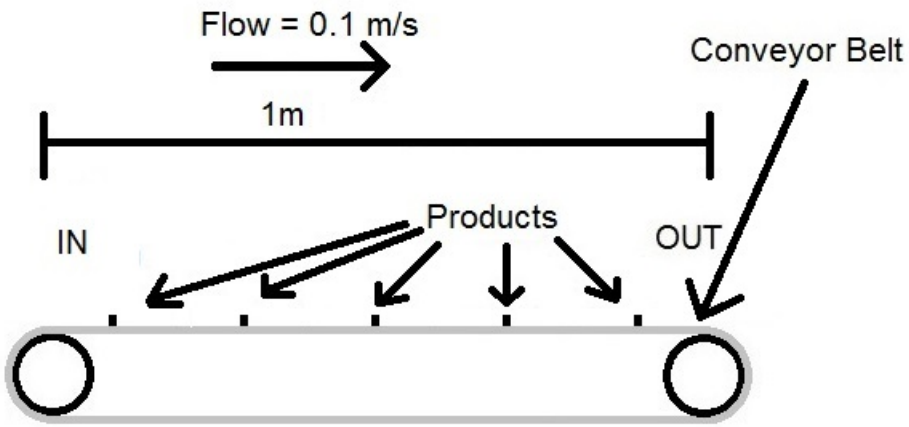


Figure 2.4: The conveyor belt problem

Since the belt's speed is $0.1$ m/s and the belt's length is $1$ m, the time that the product will spend in the process , the CT, is $10$ seconds, and, as a new product is inserted in the beginning of the belt every $2$ seconds, the distance between two successive products on the belt is $0.2$ m. For the instant represented, there are $5$ products in the conveyor belt, thus the WIP, for this particular instant is $5$ products. The average thruput is $0.5$ products per second, as, when a product is removed at the end of the belt, it takes $2$ seconds (the distance between two successive products divided by the belt speed) for another unit to arrive at the belt's end.

Of course, the conveyor belt has other characteristics, such as manufacturer, series number, mean time to repair (MTTR), Mean Time Between Failures (MTBF), etc., that are not relevant for the understanding of a single-step process, thus these characteristics are not necessary at this point.

### 2.1.2 Entity

An entity is a discrete item of interest in a model that flows through the process, such as the customer or the products of the conveyor belt. The entity can carry information, called attributes, that can alter its flow in the process. For the customer, an attribute could be the payment method, the gender or any other relevant characteristic that needs to be specified for the model. Each process can have multiple different types of entities, each of them with its own set of attributes, for example, a variety of products can be sold in the store, with each of them having a specific price. Different entities can be combined to create new entities, for example, a customer and a product can be merged into an new entity, a satisfied customer.

### 2.1.3 Event

An event is an instantaneous asynchronous change in the system state and/or outputs, e.g., the pressing of a button or the arrival of a new customer. An event can trigger other events in the process, creating a cascade effect in the system, for example, in a soft drink machine, the payment for a product by a customer triggers the release of the product to the customer, as well as the change, in case of a payment larger than the value of the product.

### 2.1.4 Time-driven discrete and event-driven discrete system

From [3], in time-driven systems, the system state changes continuously, synchronized with every clock "tick", thus, time (continuous, $t$, or discrete , $k$) is the independent variable which appears as the argument of all input, state and output functions. In event-based systems, the state evolution depends entirely on the occurrence of asynchronous instantaneous transitions, the events, over time, jumping from one state to another whenever an event occurs.

Hybrid systems combine both of these state evolution possibilities, working with many time-based changes, however, allowing event-based state transitions to occur as well. A computer is an example of a hybrid system, as it has many periodic functions synchronized with the clock, yet also responding to asynchronous calls that may occur at any time, e.g., user

requests.

### 2.1.5 Generator

A generator creates entities. It can be triggered to create new entities in a time based manner, by specifying an interarrival time (IIT), or in a event based manner, when a specified situation occurs. Generators are often modelled based on a probability distribution function, usually exponential distribution for its IIT. This models random disturbances in the arrival of entities to the process. The probability density function and the cumulative distribution function for the exponential distribution in addition to some relevant properties of this distribution can be found in appendix A.

### 2.1.6 Queue

After being generated and entering the process, an entity usually has to wait for the component process to be vacant. This waiting area is called buffer or queue, storing the entities with a predetermined capacity, which is the maximum number of entities that can be accommodated in the actual queue space. A queue is ruled by a sorting disciple to dispatch the entities when the following component process is available, such as FIFO (First-In-First-Out), LIFO (Last-In-First-Out), priority (the entities have an attribute that gives them more or less priority to be served), random (stock), etc.

### 2.1.7 Server

A server is the most fundamental component process of a system, serving the incoming entities for a certain period of time, known as service time. The service time can be deterministic or be ruled by a probability function, usually exponential distributed with average $\mu$. The server, also, has a finite capacity, limiting the amount of entities that can be served at a time, thus justifying the presence of the queues in the process.

### 2.1.8 Sink

A sink terminates the path of the entity in the process, thus restricting the scope of the model. The exit door of the retail store is a sink, as, after the customer pass through the door, he is no longer relevant to the retail store process.

### 2.1.9 Router

Finally, one single queue can be connected to multiple servers and multiple queues can be connected to one single server. The routers perform these switching between inputs and outputs paths, based on predetermined rules of the system, such as an entities attribute or a random variable. Routers can only appear in parallel queueing systems, as they create multiple possible paths for the entities in the process.

## 2.2 The Kendall-Lee notation for queues

The Kendall-Lee notation for the description of queueing processes is described now, specifying a queuing process as a sextuple, in the format: $1/2/3/4/5/6$ [16].

- 1 - Nature of the Arrival Process: it is described by a single letter (M, G or D) indicating the distribution that models the arrival process, with M standing for exponential distribution, G for general and D for deterministic. The mean arrival rate is denoted as $\lambda$, i.e., mean IIT is $\frac{1}{\lambda}$. In steady state, $\lambda$ is the mean value of the process thruput;
- 2 - Nature of service times: indicated by a single letter (M, G or D), in the same way, as the arrival process. The mean service rate is denoted as $\mu$, hence the mean cycle time is given by $\frac{1}{\mu}$;
- 3 - Number of parallel servers: this parameter is the number of parallel servers in parallel. In a sequential process, this parameter is set to $1$;
- 4 - Queue discipline: indicates how queues are serviced in the process: FIFO (First-In-First-Out), LIFO (Last-In-First-Out), priority (where a certain parameter specifies the

entities order in the queue) or GD (a general discipline queue, in which the order is not relevant to the process). Usually, this parameter is omitted and GD is chosen by default;

- 5 - The maximum population size in the Process: is the maximum entity capacity of all servers and queues in the process. Usually, this parameter is chosen as infinity;
- 6 - The size of the population from which the entities are drawn: the amount of entities in the pool at the entrance to the system. Usually, it is infinity for open processes and this parameter is omitted, but, this parameter is crucial for closed processes, where only a fixed amount of entities cycle through the process;

Thus, using the above notation, a simple process with one generator, one General Description queue of infinity capacity and one server, in which the generator and the server operate with a exponential distribution, can be modelled as M/M/1/GD/$\infty$/$\infty$, or for short, M/M/1, since the default values of the last three parameters are understood to be GD/$\infty$/$\infty$.

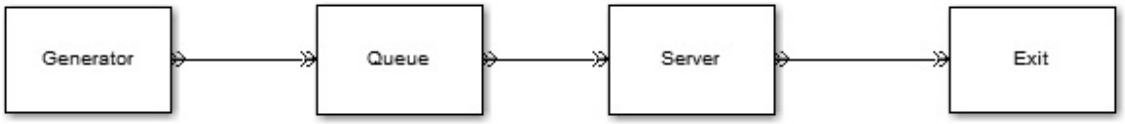


Figure 2.5: An example of a single step M/M/1 process

In the literature [3], a standardized abstract graphical format of the process configuration is used, in which a server is represented by a circle, a queue is represented by an open slotted box preceding the server and arrows show the flow of entities in the process, as in figure 2.6.

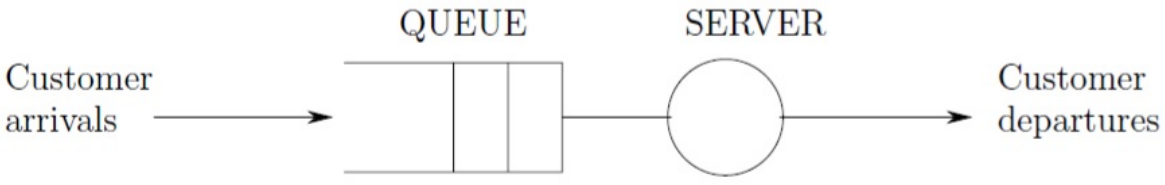


Figure 2.6: An example of a single step M/M/1 process. Figure source: [3]

In a more complex processes, in which a particular characteristic of the process cannot be fully described just by using the previous notations, or when the modeller wants to highlight

this particular characteristic, as in the case of the shared resources restrictions, this particular characteristic is appended to the Kendall-Lee description: for example, M/M/2 with Shared Resources. An example of this is a dual-core computer, with each core attempting to access a shared hard drive, possibly simultaneously.

## 2.3 Modelling Softwares

In order to develop the desired models, software options that facilitate the implementation of a DES system were investigated. The different options of available software are described below, along with their main characteristics and the reasons that led to the adoption of the Simulink's SimEvents Library as the software most appropriate for this project.

### 2.3.1 Software iThink

The iThink programming language was the starting point in the quest for the most suitable program for this project since it was the one originally used in the fast food restaurant example in [16]. The iThink platform is a proprietary program from ISEE Systems [37] targeting business dynamics which uses two interconnected graphical user interfaces (GUI): one to model the process with block connections (the Model tab) and one to visualise, exercise static control using manual switches and plot the results (the Interface tab). There is also an Equation tab, which displays the model state equations, generated from the block diagram, but does not permit editing these equations.

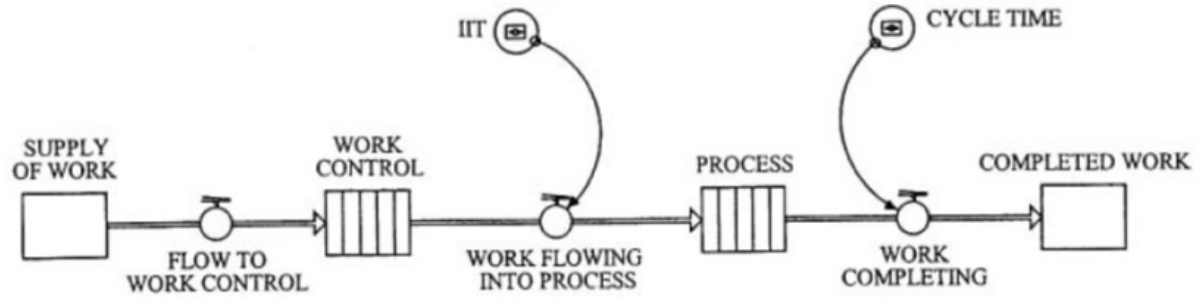


Figure 2.7: An example of a single step process modelled in iThink [16]

Figure 2.7 shows a single-step process, with a block named Work Control that regulates

the arrival time of the units in the Process block in iThink. This model will be further explained in chapter 3.

The graphical interface that displays the results obtained by simulating the above model is shown in figure 2.8.

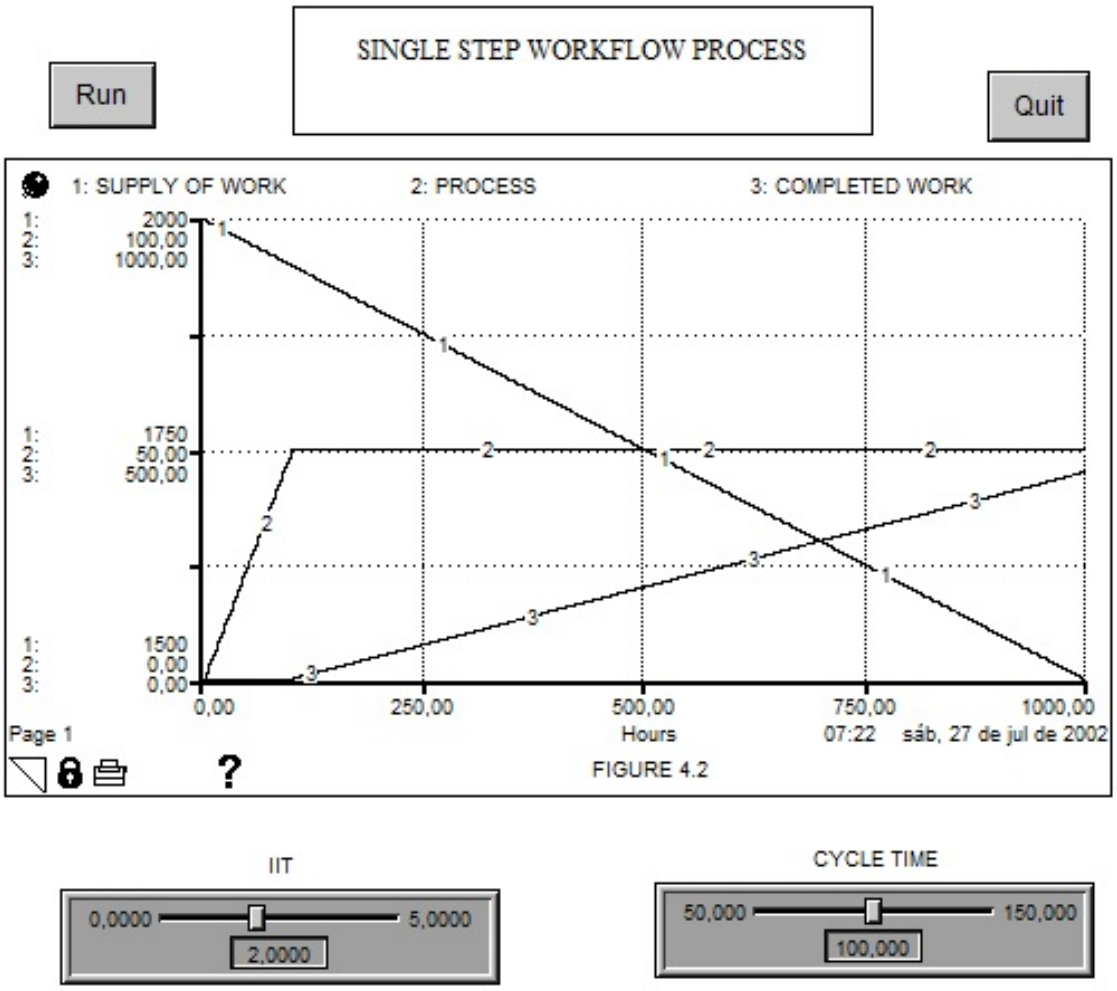


Figure 2.8: The graphical interface for the single step process modelled in iThink. Figure source: [16]

Pressing on the equation tab displays the difference equations that underlie the model and were automatically generated from the block diagram by the iThink software, as in appendix B.

The software iThink is very suitable for quick learning and modelling, since it uses only three basic blocks types (flows, converters and stocks, which can be further subdivided into

into reservoirs, queues and conveyor) and information arrows that create logical links between blocks. However, this simplification results in considerable difficulty when it is necessary to build anything more complex using only these few resources through the GUI, and thus also hinders the development of complex control mechanisms.

### 2.3.2 SEMoLa

The SEMoLa program was developed at the Department of Agricultural and Environmental Sciences at the University of Udine (Italy), intending to be a Simple, Easy to use Modelling Language, as implied by its name [7]. This program is free and theoretically able to represent any system, particularly suiting agricultural, biological and ecological systems, at different scales and complexity levels.

Few references [9] actually cite the use of this software and the only available manual is written in Italian. However, the main point that led to the discard, prior to any trial, of this software for this project is that its development and maintenance have been discontinued, the last version having been released in 2006, accordingly to [8].

### 2.3.3 SimPy

The SimPy technology is a framework for discrete simulation based on standard Python under the MIT License. It was originated from the merge of two previous Python packages, Simula and Simscript. It uses Python Object-Oriented terminology and structures to create process-based discrete-event simulation [34].

The program uses text commands to create processes, much like computer threads, as they run, yield or can suspended for a given period in time. These processes reside in a virtual location called environment that executes for a window of time, defined by the user. During environment execution, the user is not allowed to insert new dynamics into existing processes. It works particularly well with fixed step programs but has problems when steps smaller than one time unit are chosen in order to gather statistics.

Some example processes, including one with shared resources were modelled in this software in order to test the capacity of this software. The example in appendix C models the same process as figure 2.3.1.

Although the SimPy programming environment is well suited to the task of simulating a shared resource program [24], it does not have tools to facilitate getting process statistics from the model that was running, even elementary ones, such as the thruput and the cycle time of the process. Furthermore, for the next stage of the project, which involves the control of the model, no tools or references appear to be available.

### 2.3.4 SimEvents

SimEvents is one of the various Simulink's libraries in the Matlab platform. It comes with the complete version of Matlab or can be purchased individually and added to a Simulink and Matlab basic version. Simulink is a graphical environment inside the Matlab program, used for developing simulation through the combination of a variety of predefined off-the-shelf blocks, which perform some specific function in the model. Simulink is capable of interfacing with the general Matlab platform and its functions, with the usage of special blocks that embed customised function scripts, or, in the opposite direction, the Simulink model can be called straight from Matlab's command line and the script's editor.

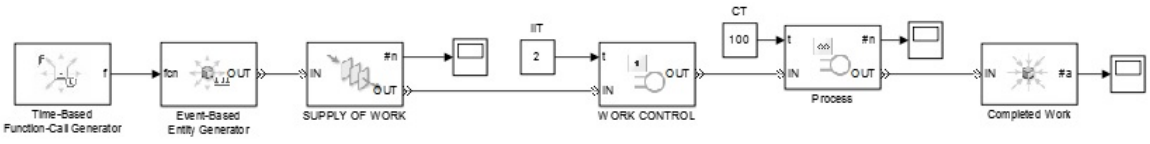


Figure 2.9: SimEvents model of the single step process of figure 2.7

SimEvents by itself has about the same basic functions as the other programs presented in the previous subsections of this work and is not a frequently used Simulink tool as the platform was originally designed to deal with ordinary differential equations (ODE) in standard ODE-based simulations [6]. Although the SimEvents library has a good statistics gathering tool, which is an improvement in relation to SimPy, the major advantage is the Simulink's framework and the possibility to connect with the vast array of available tools in this environment, through some special conversion blocks from event based signals to time based signals and vice versa, plus all the toolboxes available for Matlab from scripts programs, such as optimization toolboxes and curve fitting. It is important to point out that there is a

numeric restriction in SimEvents related to the number of events simultaneously occurring in a particular block and in the simulation, but this number has the relatively large value of $2^{31} - 1$.

The possibility of using other Simulink Libraries and Matlab tools, which a special mention of the Optimization Toolbox, led to the choice of SimEvents for this work.

## 2.4 Literature on modelling fast food restaurants

The US fast food industry is a 200 billion dollar market [30]. Developing strategies to increase the company's market share, and, therefore, net worth and profit, is a key role for the managers of the major brands and the incoming players to that market.

As for any other complex system, it is not an easy task to make decisions based exclusively in intuition. Complex systems usually involve a large number of components, influenced by uncertainty, feedback loops, time delays and non linearities [16]. One of the tools available to support in this arduous task is the development of simulations, which, although is not a recent technology [38] and [11], has become more powerful and easy-to-use in more recent times, with a vast availability of off-shelf tools to create customized models, e.g. the ones presented in section 2.3.

The first articles related to fast food restaurant simulation targeted only the process, focusing on strategies to process measurements, such as, reduce the waiting time for the customers, without evaluating the economical impacts of these decisions, as in [38] and [5]. This approach tries to create solutions without considering if it is economically or physically viable to be implemented in the restaurant. An economic-based approach, on the other hand, focus on the financial status of implementing a new technology in the restaurant, comparing the financial impact of the new technology to the present situation as, for example, being able to reduce the waiting time for the customers is not a guarantee of a better financial performance [20].

No reference was found involving a financial evaluation of a fast food restaurant in shared resources restrictions, which is a real-life fast food restaurant situation, as the equipments for producing the meals are limited to the availability in the store, i.e., there is no point in hiring another cook if the kitchen has only one stove and expect this to solve a low production level

problem, without taking into consideration the costs related to this new cook.

# Chapter 3

# Process Modelling using SimEvents

Chapter 2 introduced the basic terminology and illustrated the decision process that led to the adoption of SimEvents as the simulation platform for the models presented in this chapter. A few basic examples are implemented to illustrate general processes before finally presenting the fast food restaurant with shared resources using the Simulink environment.

## 3.1 A D/D/$\infty$/GD/$\infty$/2000 Model

The D/D/$\infty$ is one of the most basic models of a queuing process and it was used to exemplify the usage of iThink and SimPy programs in chapter 2. The Kendall-Lee notation implies that the model has a deterministic generation of entities that enter the process, a deterministic service time at the parallel server of infinite capacity that is served by a general description queue with infinite capacity and a sample population of 2000 entities that can enter the process.

The finite value of the last parameter does not influence the model execution at this point and was only included to meet with the specification for the model in [16].[1]

[1]The author can provide all the models developed for this work in a digital format under request by email (henriquewcb at poli dot ufrj dot br), in "slx" format, as it is not a pleasant task to reproduce the SimEvents models based only on their block diagram representation.

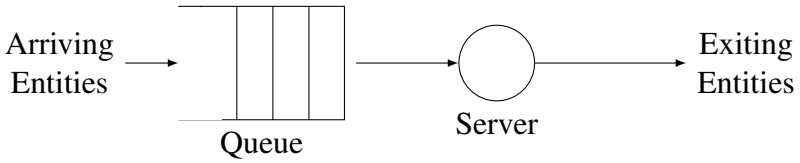


Figure 3.1: The DD1 abstract model

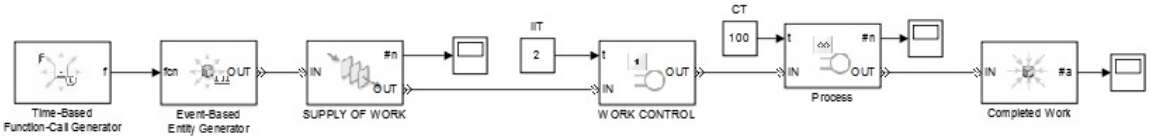


Figure 3.2: Example extracted from [16] implemented in SimEvents

This model starts with a function-call generator, which is specified to send a signal, only at the beginning of the simulation, to the event based entity generator attached to it, that produces and dispatches the 2000 entities specified for this system to the following queue *Supply of Work*. [2]

The queue *Supply of Work* is followed by a single server, *Work Control*, to regulate the IIT of the entities to the server with infinite capacity, *Process*. The server *Work Control* could be discarded in case the event-based generator were to be replaced by a deterministic time-based entity generator with an IIT of 2 time steps. Notice, however, that this change would impact the model in the sense that it would not be constrained to relative to the size of the population from which the entities are drawn, therefore altering the model proposed in [16]. The sink block *Completed Work*, that follows the server *Process*, provides a way to terminate the path followed by the entities.

In [16], the following parameters are specified for this problem:

[2] Whenever a general description (GD) queue with infinite capacity is specified for the model, which indicates that the order of dispatching is irrelevant, a FIFO queue block with infinite capacity is used instead, since it is a default block in SimEvent's library.

| | |
|---|---|
| Model time unit | hours |
| DT | 0.0625 |
| End Time | 1000 hours |
| IIT | 2 hours |
| Cycle time | 100 |
| Supply of Work | initial value 2000 |
| Process | initial value 0 |
| Completed Work | initial value 0 |

Table 3.1: Parameters for the model from figure 3.2

Figures 3.3, 3.4 and 3.5 present the results for model using the parameters in 3.1. It is possible to see the *Supply of Work* reducing at a constant slope of 1 unit every 2 hours, as specified by the IIT parameter, because the dispatching event to the server *Process* was modelled as a deterministic function, which means that there are no random effects acting on this variable.[3]

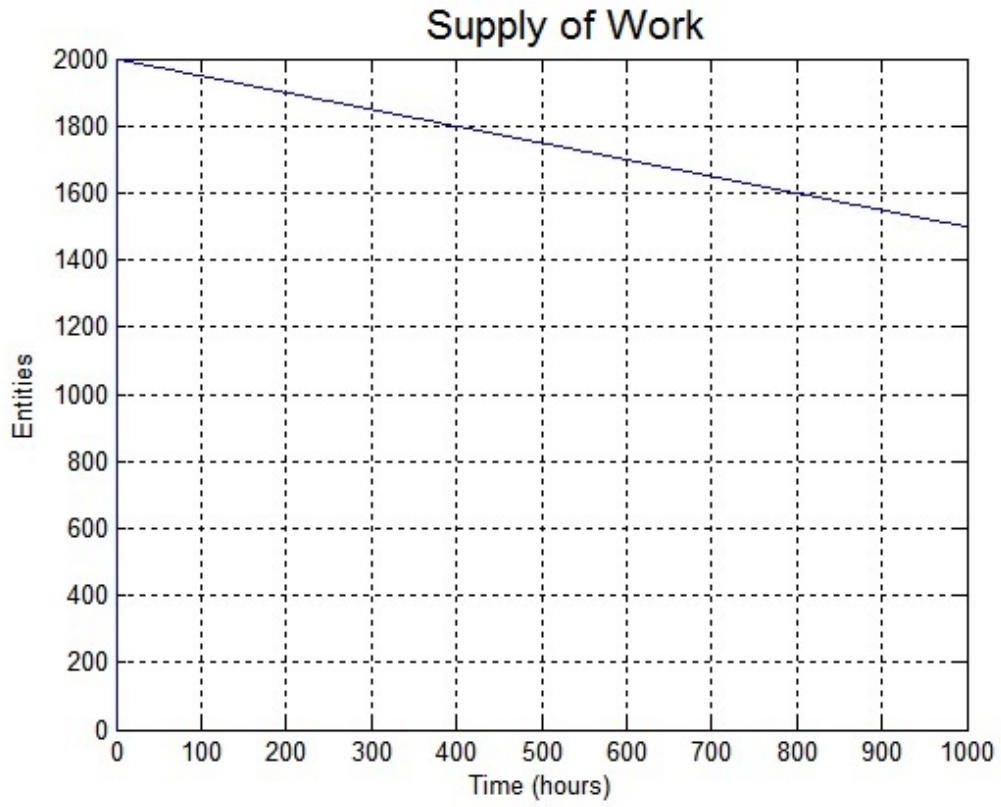


Figure 3.3: Time plot of *Supply of Work*

[3]The time step, DT, was not specified in the Simulink program since its internal structure uses an Event Calendar, instead of a solver [6].

The server *Process* is able to receive an infinite number of units. However, since the IIT of the process is 2 hours, as specified in 3.1 for *Work Control* and the service time of the server *Process* is deterministic, therefore always equal to 100 hours, this means that the number of entities inside the block *Process* will increase until units are able to exit, after residing in the process for one cycle time. Observed that, since both IIT and service time are deterministic, the server *Process* will reach a steady state, in which the number of entities inside the server will remain constant, because, for each entity exiting the server *Process*, another one is exiting the server *Work Control* and entering the server *Process*. This implies that the number of units in the server Process (WIP) is going to be equal to 50 until the *Supply of Work* becomes 0.

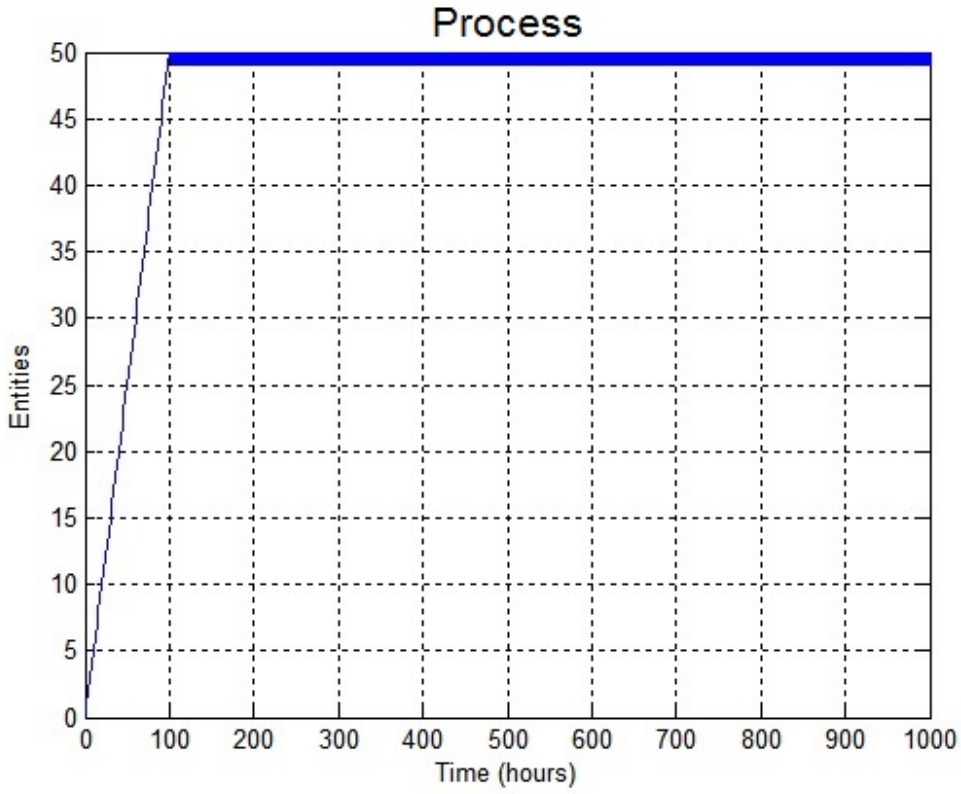


Figure 3.4: Time plot of entities within the server *Process*

The first entity to leave the server *Process* and enter the block *Completed Work* arrives at time $t = 100$, which is the CT of *Process*. Also, it is possible to see that the number of entities inside the entire system (*Supply of Work*, *Work Control*, *Process* and *Completed Work*) is constant and equals to 2000 at any given time, because all entities are generated at the beginning of the simulation.

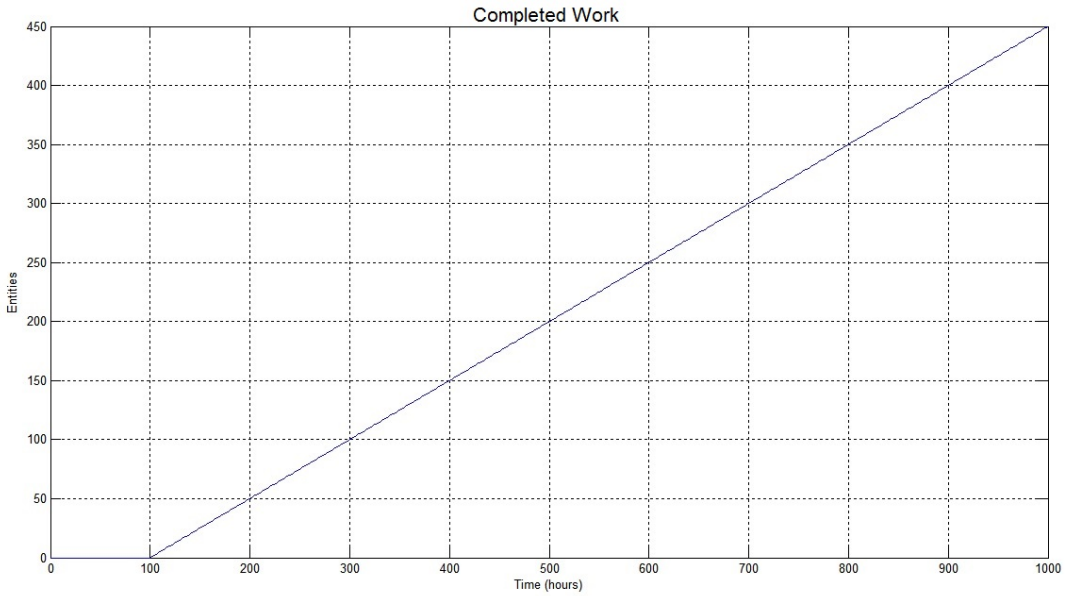


Figure 3.5: Graph of the *Completed Work* versus time

For a comparison, the results presented [16] for the same problem in iThink are shown below:

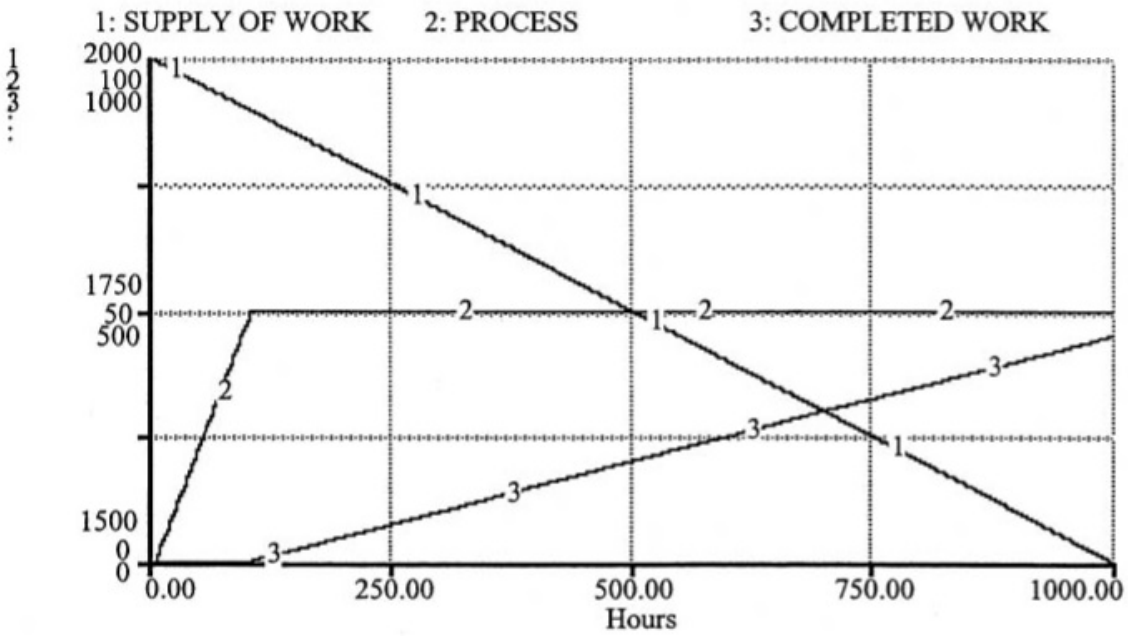


Figure 3.6: Results for example 3.2. Figure source: [16]

This process illustrates one of the most fundamental relationships for a process model,

known as Little's Law.

**Definition 1** *Little's Law states that, under steady state conditions, the average number of items in a system [the Work In Process (WIP)] is equal to the average rate at which items arrive [the thruput of the system ($\frac{1}{IIT}$)] multiplied by the average time that an item spends in the system [the Cycle Time (CT)][22].*

Thus,

$$WIP = Thruput * CT = \frac{CT}{IIT} \tag{3.1}$$

In case this process is executed for a time larger than IIT $*$ *Supply of Work*'s initial value, a starvation problem appears at the server Process since all the $2000$ entities available have been processed. This starts affecting the number of entities inside the server *Process* that had already reached its steady state, constant value of $50$. Since no new entities enter the server *Process*, the number of entities in the blocks starts to reduce until it reaches zero, when the last unit to enter has completed its cycle time. At that point, all the entities have reached the block *Completed Work*.

*Remarks on the SimEvents simulation aspects of this example:*

In this model, all the blocks accept event-based signals, except for the event-based entity generator, which receives a function-call signal from the function-call generator. Thus it was not necessary the usage of signal conversion blocks.

When a scope block is connected to a time-based signal, its graph is plotted at all instants of the simulation, even if no signal variation is occurring. In contrast, when a scope is connected to an event-based signal, plotting takes place only when an event occurs in the signal.

As an example, if starvation occurs, the graph of the *Completed Work*, for an event-based signal, would stop being plotted right after the arrival of the last entity, no matter for how much longer the simulation was executed. Thus, the scope would be locked at that point waiting for a new event. However, if a conversion block from event-based signal to a time-based signal were inserted between the *Completed Work* block and its scope, then the signal for the

Completed Work would be plotted until the end of the simulation as a constant signal.

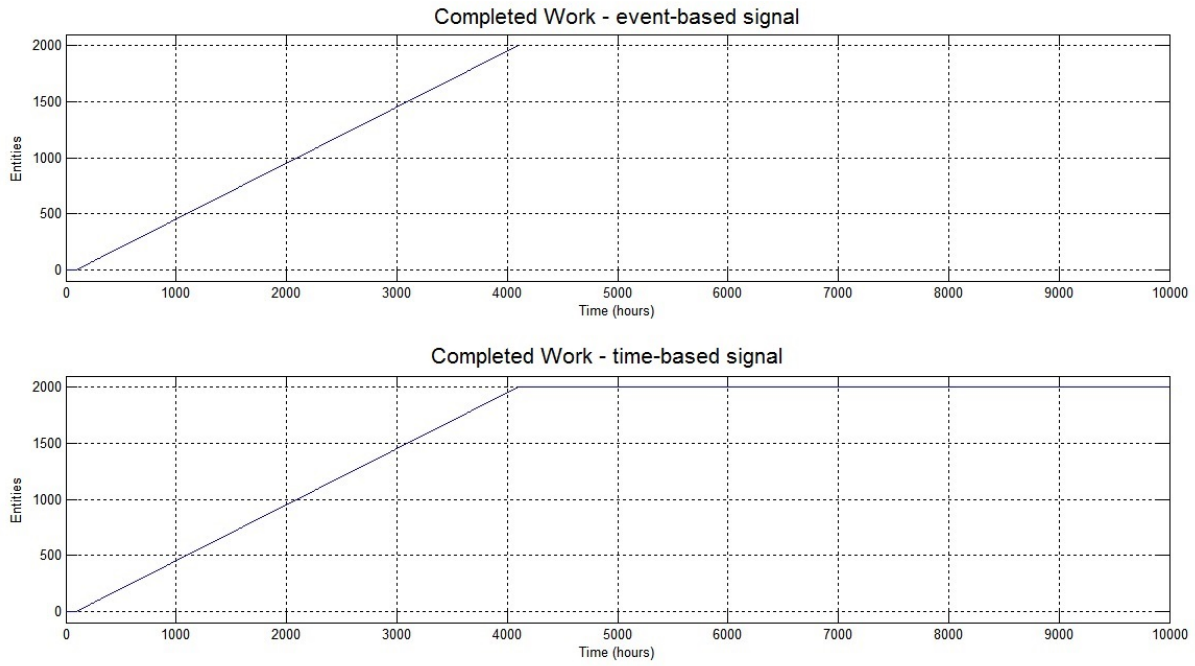


Figure 3.7: Comparison between event-based signal plot and time-based signal plot for *Completed Work*

The conversion between event-based signals and time-based signals can be done through the Event to timed signal block and the conversion between time-based signals to event-based signal can be done through the Timed to event signal block. The corresponding block icons are shown below:


Figure 3.8: Signal converters: the time to event signal, on the left, and the event to timed signal, on the right

Another important point in the difference between time-based signals and event-based signals is that, while the time-based signal can only have one value for each time step, the event-

based signal can have multiple values at the same instant. Analysing the number of entities inside the server *Process* event-based signal, it is possible to see this phenomenon when one entity exits and a new one enters the server. Since the service time and the IIT of this server are deterministic and the service time is a multiple of the IIT, in steady state condition, both situations occurs simultaneously, therefore, for that particular instant, the number of entities inside this block is $49$ and $50$, as depicted in figure 3.9.

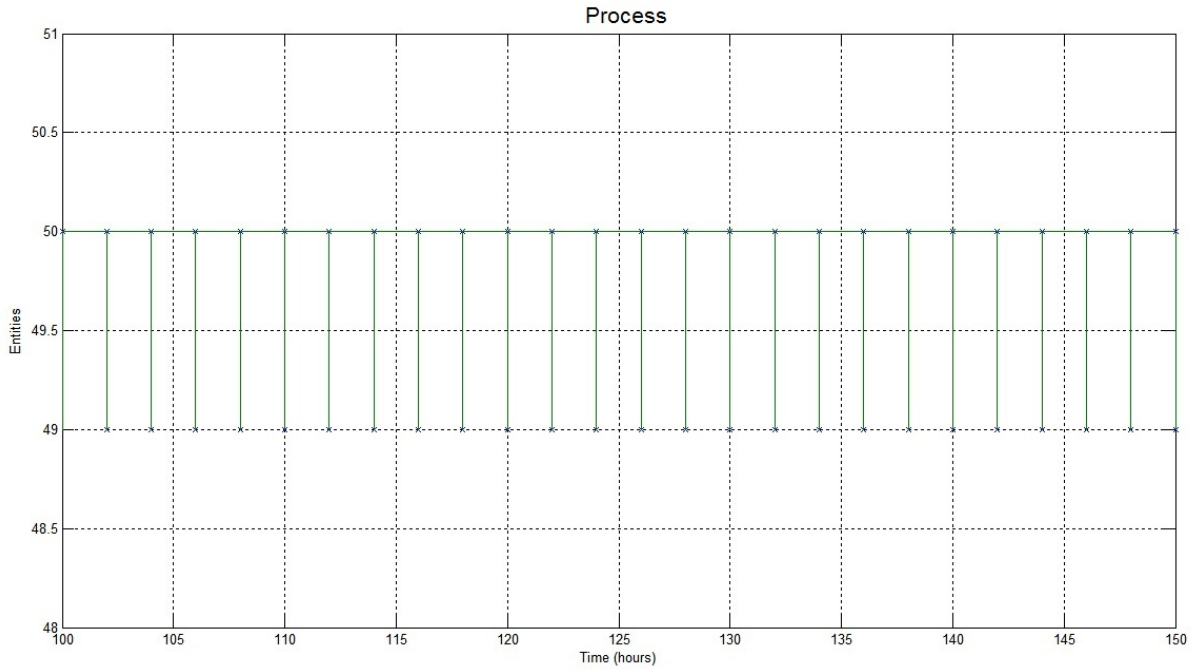


Figure 3.9: Event-based signal can have multiple values at the same instant

## 3.2 A M/M/1/GD/$\infty$/$\infty$ Model

The example from section 3.1 made unrealistic choices, such as a server with infinite capacity and that deterministic IIT and service time for server *Process*. This section introduces random effects, based on exponential density functions to model the IIT and the service time. For this M/M/1 queueing process, there is only one server available. This new assumption does not always hold, for example, in processes that do not have a stationary average IIT, such as peak and off-peak hours, or if the service time for a server is dependent on the number of entities being processed at each instant, e.g. a crowded restaurant.

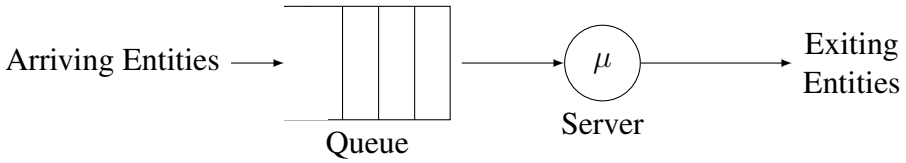


Figure 3.10: The MM1 abstract model

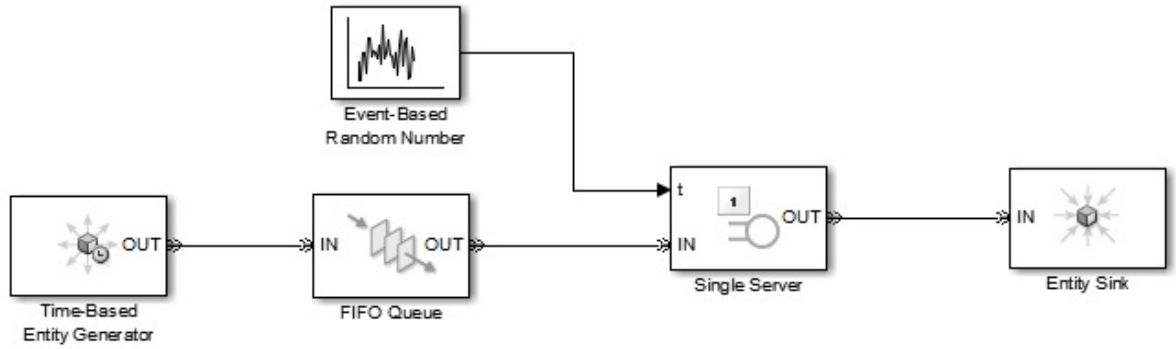


Figure 3.11: The M/M/1 Model implementation in SimEvents

From appendix A, since both arrival and service processes are assumed to be exponentially distributed, the average arrival rate, $\lambda$, is the inverse of the average IIT, $\frac{1}{\lambda}$, and the average service rate, $\mu$, is the inverse of the average service time $\frac{1}{\mu}$. Thus, the traffic intensity, $\rho$, also known as the server's utilization, parameter is given by the ratio of the average arrival rate and the average service rate, i.e.:

$$\rho = \frac{\lambda}{\mu} \tag{3.2}$$

According to [39], for a M/M/1 process with $\rho < 1$, the process has a steady state, based on the probabilities of entity arrival and departure events to occur, which are directly related to the mean values of the exponential distributions associated with these events. For processes with traffic intensity $\geq 1$ in presence of random effects, the queue, if not limited, will grow unbounded.

The formulas to calculate the steady state average CT and average WIP from the arrival rate and the average service rate are [39]:

$$\overline{CT} = \frac{1}{\mu - \lambda} \tag{3.3}$$

$$\overline{WIP} = \frac{\rho}{1 - \rho} \tag{3.4}$$

The average queue length ($L_q$) can be calculated by subtracting the traffic intensity, $\rho$, which is the expected number of customer in service, from the WIP: [39]

$$\overline{L_q} = \frac{\rho}{1 - \rho} - \rho = \frac{\rho^2}{1 - \rho} \tag{3.5}$$

In fact, for a model with $\mu = 1$ and $\lambda = 0.8$, from the formulas we get:

$$\rho = \frac{\lambda}{\mu} = 0.8 \tag{3.6}$$

$$\overline{WIP} = \frac{\rho}{1 - \rho} = 0.8/0.2 = 4 \tag{3.7}$$

$$\overline{CT} = \frac{1}{\mu - \lambda} = 1/0.2 = 5 \tag{3.8}$$

$$\overline{L_q} = \frac{\rho^2}{1 - \rho} = 0.64/0.2 = 3.2 \tag{3.9}$$

Simulating the model for these choices of parameters yields the following results:

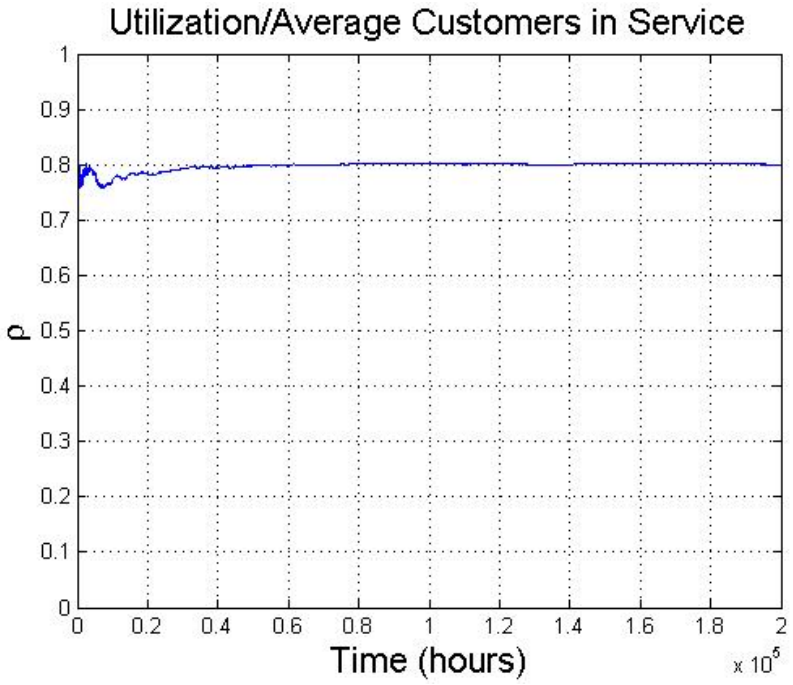


Figure 3.12: Average server utilization/traffic intensity ($\rho$) for the model with $\mu = 1.0$ and $\lambda = 0.8$

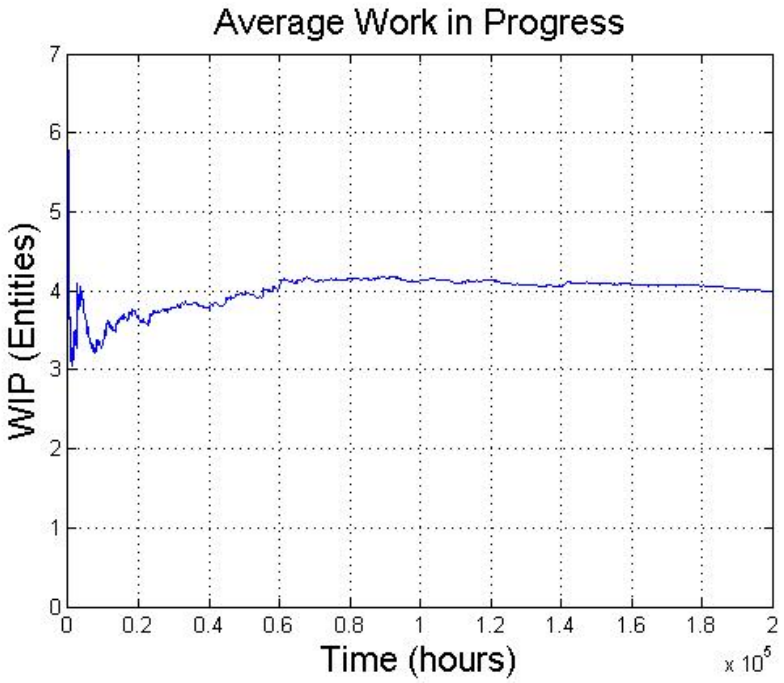


Figure 3.13: Average WIP for the model with $\mu = 1.0$ and $\lambda = 0.8$

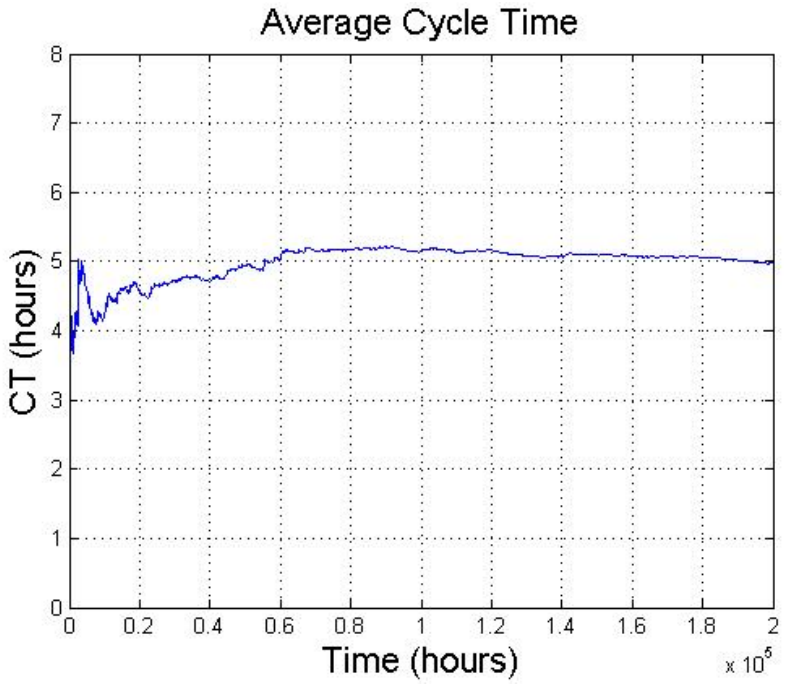


Figure 3.14: Average CT for the model with $\mu = 1.0$ and $\lambda = 0.8$

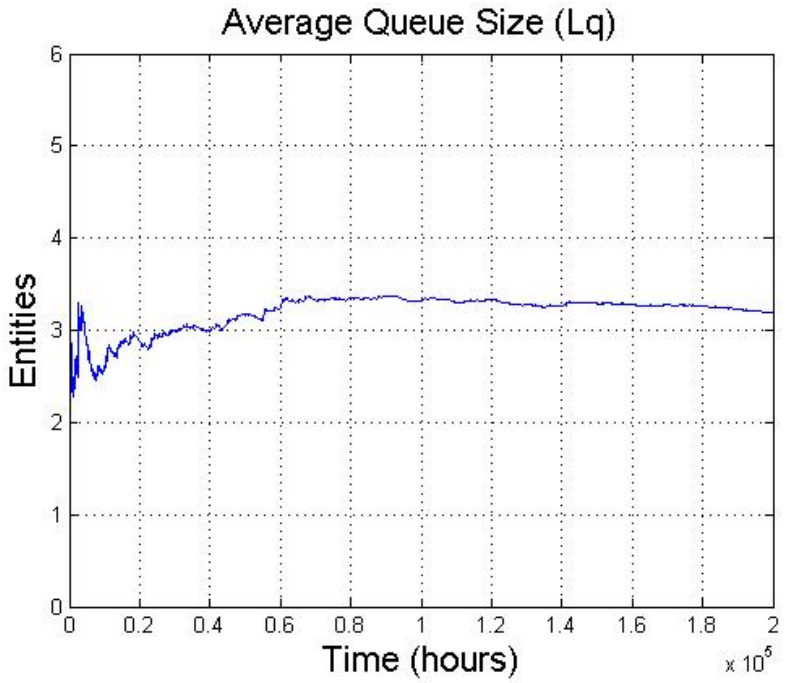


Figure 3.15: Average queue size ($L_q$) for the model with $\mu = 1.0$ and $\lambda = 0.8$

Since the statistics for SimEvents blocks are gathered from the start of the simulation without a warm-up time for the process to reach a steady state, the influence of the transient queueing behaviour is present in the measurements. This influence is smoothed over time as

new measurements are collected, thus the final value tends to the average predicted value of the variable if the simulation is executed for enough time.

As a consistency check, simulating the model at a traffic intensity ($\rho$) of 1, the queue length yields the following profile, eventually becoming infinite:

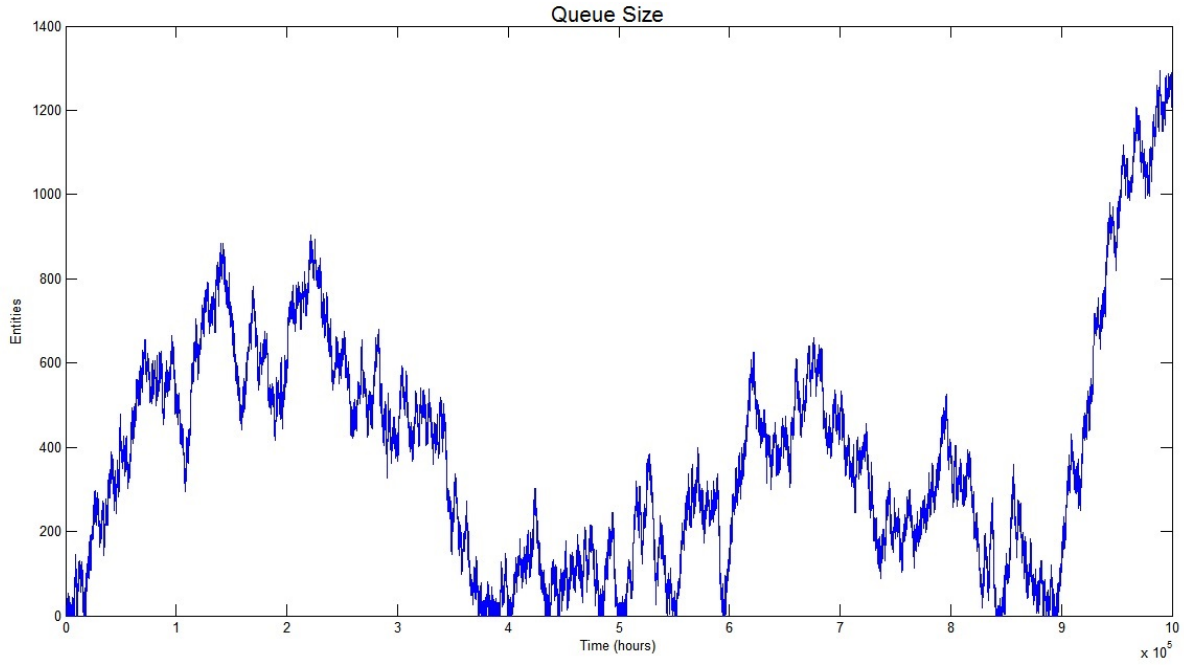


Figure 3.16: Queue length for the model with $\mu = 1.0$ and $\lambda = 1.0$

## 3.3 The Fast Food Restaurant example without Shared Resources

This section introduces the reader to the fast food restaurant example, which is described as an M/M/2 system, therefore having two parallel processing servers, *Server 1* and *Server 2*, with one general description queue, known as the *Arriving Customer* queue, feeding these servers through an output switch. After being served in the process, the entity, in this case, the customer, exits from both parallel production lines which meet in a sink, terminating the entities path in this model, as depicted in figure 3.18.

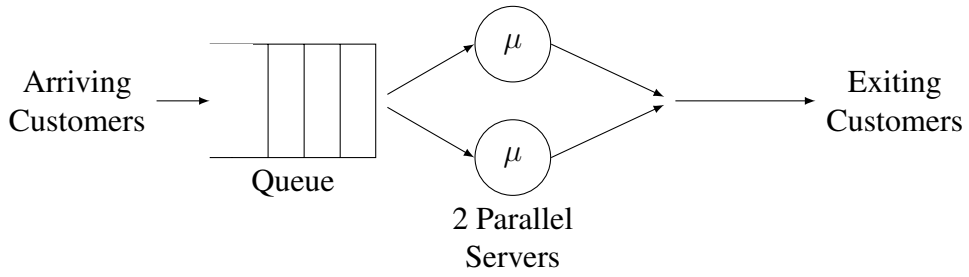


Figure 3.17: The fast food restaurant abstract model without shared resources

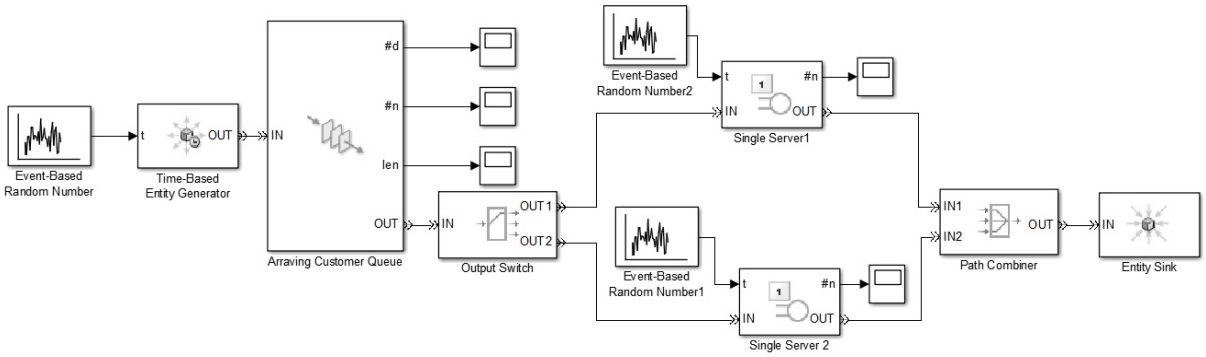


Figure 3.18: Model of the fast food restaurant with one step service and no Shared Resources

In this model, the queueing process takes place in the following order: the entities, i.e., the customers, enter the queue and are dispatched to one of the two parallel single servers when one of them is not busy with an output switch. The positive integer $s$ is introduced, corresponding to the number of servers in parallel configuration, that alters the definition of the traffic intensity ($\rho$), for this configuration:

$$\rho = \frac{\lambda}{\mu s} \tag{3.10}$$

The equations from 3.2 need to be altered to meet this new configuration, as they are now dependent on the number of servers $s$ and the number of customers at a given moment $j$. The probability that the number of customers exceeds the number of parallel servers, denoted $P(j \geq s)$, is computed as a function of the number of servers $s$ and the traffic intensity $\rho$. A table of this probability is shown below for some sample value of $\rho$ and $s$ from [39]:

| $\rho$ | $s=2$ | $s=3$ | $s=4$ | $s=5$ | $s=6$ | $s=7$ |
|---|---|---|---|---|---|---|
| 0.1 | 0.02 | 0 | 0 | 0 | 0 | 0 |
| 0.2 | 0.07 | 0.02 | 0 | 0 | 0 | 0 |
| 0.3 | 0.14 | 0.07 | 0.04 | 0.02 | 0.01 | 0 |
| 0.4 | 0.23 | 0.14 | 0.09 | 0.06 | 0.04 | 0.03 |
| 0.5 | 0.33 | 0.24 | 0.17 | 0.13 | 0.1 | 0.08 |
| 0.55 | 0.39 | 0.29 | 0.23 | 0.18 | 0.14 | 0.11 |
| 0.6 | 0.45 | 0.35 | 0.29 | 0.24 | 0.2 | 0.17 |
| 0.65 | 0.51 | 0.42 | 0.35 | 0.3 | 0.26 | 0.21 |
| 0.7 | 0.57 | 0.51 | 0.43 | 0.38 | 0.34 | 0.3 |
| 0.75 | 0.64 | 0.57 | 0.51 | 0.46 | 0.42 | 0.39 |
| 0.8 | 0.71 | 0.65 | 0.6 | 0.55 | 0.52 | 0.49 |
| 0.85 | 0.78 | 0.73 | 0.69 | 0.65 | 0.62 | 0.6 |
| 0.9 | 0.85 | 0.83 | 0.79 | 0.76 | 0.74 | 0.72 |
| 0.95 | 0.92 | 0.91 | 0.89 | 0.88 | 0.87 | 0.85 |

Table 3.2: Table with some calculated values for $P(j \geq s)$

Thus, using the data from the table 3.2 and the formulas from [39], it is possible to calculate the expected (average) queue size, WIP, CT and utilization ($\rho$) of each server shown below:

$$\overline{L_q} = \frac{P(j \geq s) * \rho}{1 - \rho} \tag{3.11}$$

$$\overline{WIP} = \frac{P(j \geq s) * \rho}{1 - \rho} + \frac{\lambda}{\mu} \tag{3.12}$$

$$\overline{CT} = \frac{P(j \geq s)}{s * \mu - \lambda} + \frac{1}{\mu} \tag{3.13}$$

$$\overline{Util} = \frac{\lambda}{s * \mu} \tag{3.14}$$

Thus, for $\mu = 0.5$ and $\lambda = 0.8$ and 2 parallel servers:

$$\overline{L_q} = \frac{0.71 * 0.8}{1 - 0.8} = 2.84 \tag{3.15}$$

$$\overline{CT} = \frac{0.71}{2 * 0.5 - 0.8} + \frac{1}{0.5} = 5.55 \tag{3.16}$$

$$\overline{WIP} = \frac{0.71 * 0.8}{1 - 0.8} + \frac{\lambda}{\mu} = 4.44 \tag{3.17}$$

$$\overline{Util} = \frac{0.8}{0.5 * 2} = 0.8 \tag{3.18}$$

Running the model simulator for the given parameters $\mu = 0.5$, $\lambda = 0.8$ and $2$ parallel servers, the following results are obtained:

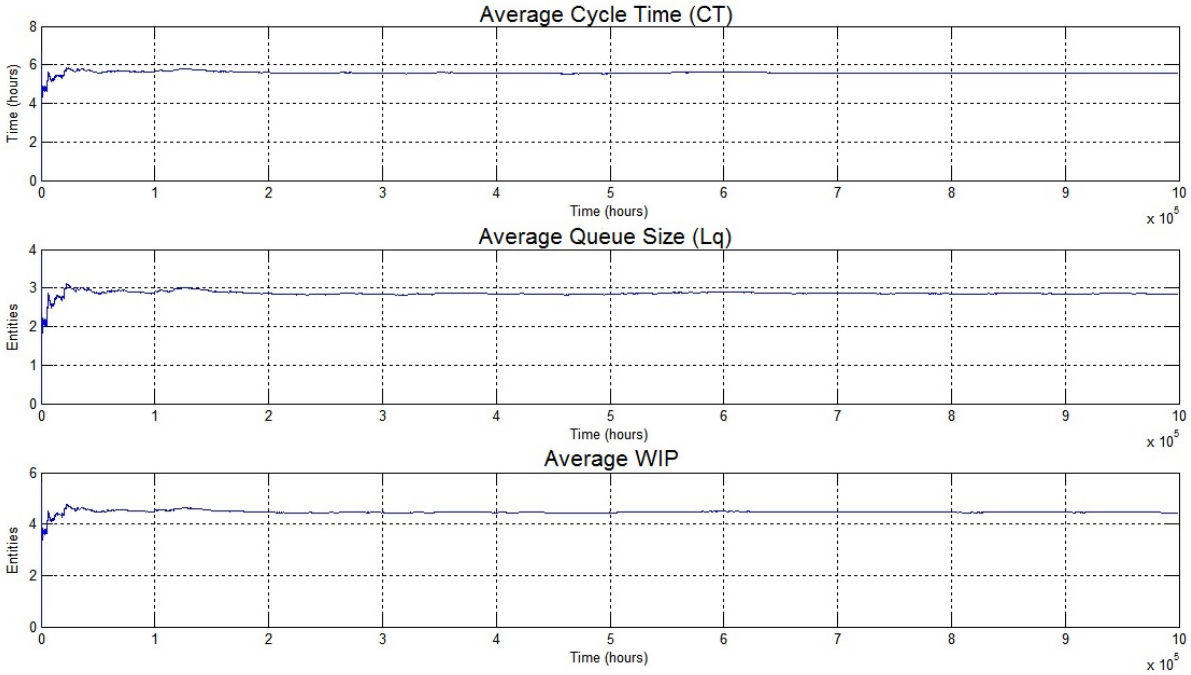


Figure 3.19: Average queue length, WIP and CT for the model with $\mu = 0.5$ and $\lambda = 0.8$

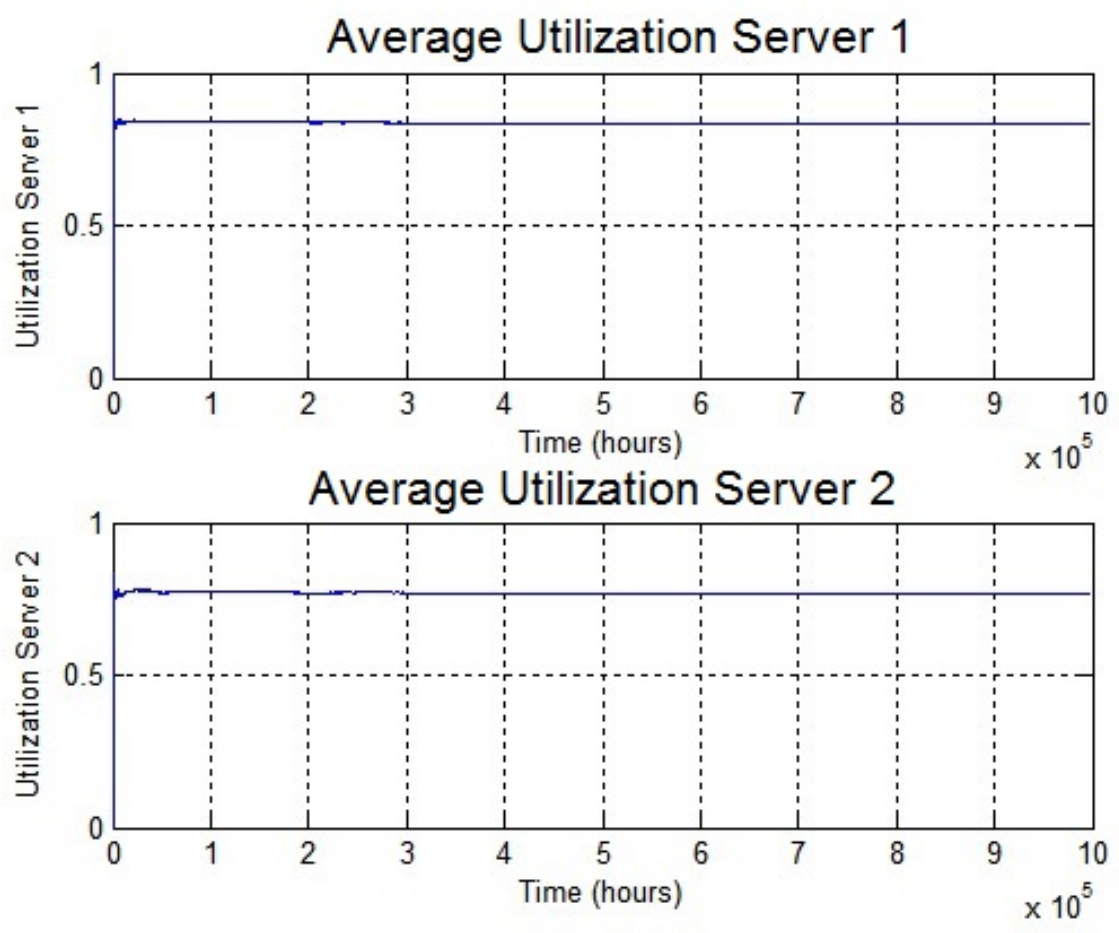


Figure 3.20: The utilization ($\rho$) of each server for the model with $\mu = 0.5$ and $\lambda = 0.8$

Figure 3.20 shows that the average utilization of *Server 1* $(0.85)$ is slightly higher than the utilization of *Server 2* $(0.75)$. This effect is caused by the output switch that has a preference for the first port that is not blocked, thus, every time that both servers are available, it picks the first available server to dispatch the entity.

At this point, a new feature is introduced. Each of the parallel servers present in the previous model is broken up into two servers, in sequential configuration. Thus, *Server 1* is replaced by two servers, *Server 1A* and *Server 1B*, and *Server 2* is replaced by, *Server 2A* and *Server 2B*. The single letters that compose the servers names indicate the resource that must be utilised for the server to be able to serve the customer, i.e., *Server 1A* requires the entity resource A to serve a customer and *Server 1B* requires the entity resource B. In this new configuration, the service time of all servers are exponential distributed with the same mean value.

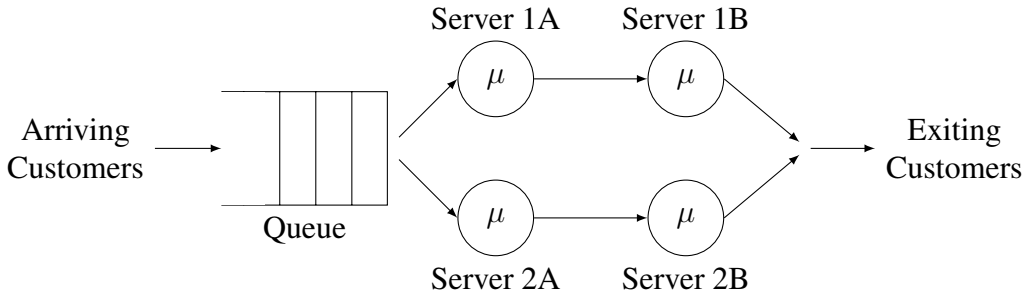


Figure 3.21: The fast food restaurant abstract model without shared resources in 2 parallel sets of servers configuration

In addition to this new feature, only one customer can be in *Server 1A* or *Server 1B*, therefore, either *Server 1A* is serving the customer or *Server 1B* at any instant. This restriction is imposed as only one employee can be serving each customer, serving the customer through the whole process. The whole process is now broken up into several subsystems, which are a special type of block that contains inner blocks, i.e., the model is divided into two levels of complexity, a top level that contains the subsystems and a lower level that is depicted in each subsystem. Figure 3.22 shows 4 different subsystems: *Process 1*, *Process 2*, *Resource A* and *Resource B*, which are explained below.[4]

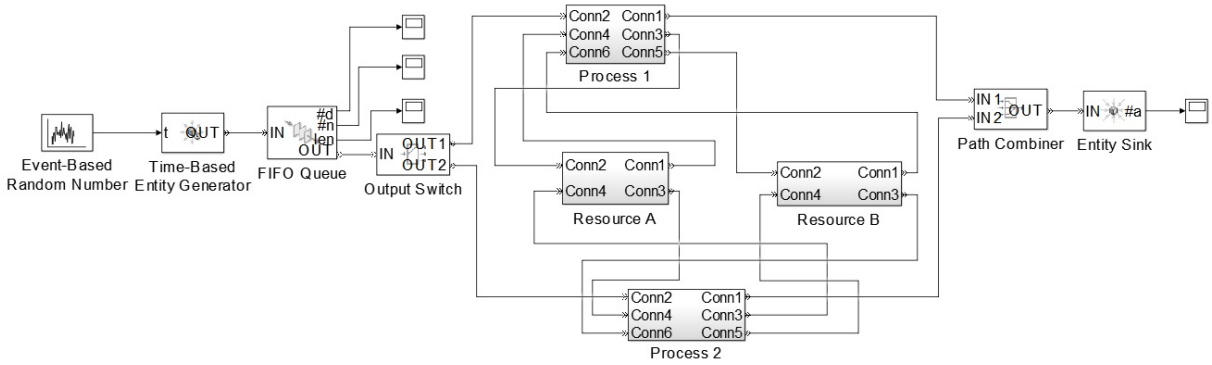


Figure 3.22: The fast food restaurant model with two parallel blocks with two sequential servers each

The generator block in subsystem *Process 1* creates one entity that represents the

[4]The process could also be divided into systems, which means that blocks would be just grouped together following an affinity logic, yet still in the model's top level, just to facilitate the explanation of a large model as in chapter 5.

employee only at the beginning of the simulation. This entity does not leave this subsystem, merging with the customer entity as soon as it arrives at the subsystem and goes with the customer through the whole subsystem, splitting off and going back to its FIFO queue when the customer is fully served (and departs from the subsystem), to wait for the next incoming customer. A new incoming customer can only enter the *Process 1* subsystem when this unique entity employee is available.

In order to serve the incoming customer in *Server 1A*, besides the employee entity, the *Process 1* subsystem requires the availability of one entity of type resource A that resides in the subsystem *Resource A*. While an entity of type resource A is not available in its subsystem to be used in *Server 1A*, the subsystem *Process 1* halts to wait for this resource. In case a resource A entity is requested by the subsystem *Process 1* for *Server 1A* and this entity is available, it travels from the subsystem *Resource A* to the *Process 1* subsystem, merging with the entities employee and customer to enter *Server 1A*, in which they reside for the service time specified.

After the completion of the service in *Server 1A*, the three entities (resource A, customer and employee) split off. The entity resource A goes back to the subsystem *Resource A*, where it will wait for another request for the subsystems *Process 1* or *Process 2*. At this point, the two other entities (customer and employee) halt until an entity of type resource B is available in order to enter *Server 1B*. The entity resource B resides in the subsystem *Resource B* and, as resource A, is requested and merges with the other two entities when available. The merged entity is then served in *Server 1B*, from which departs after the fulfilment of the server's service time. Thus, the merged entity splits off, with the entity resource B going back to the *Resource B* subsystem, the customer entity exiting for the sink connected to the subsystem *Process 1* and the employee entity waiting for the next customer at its own FIFO queue.

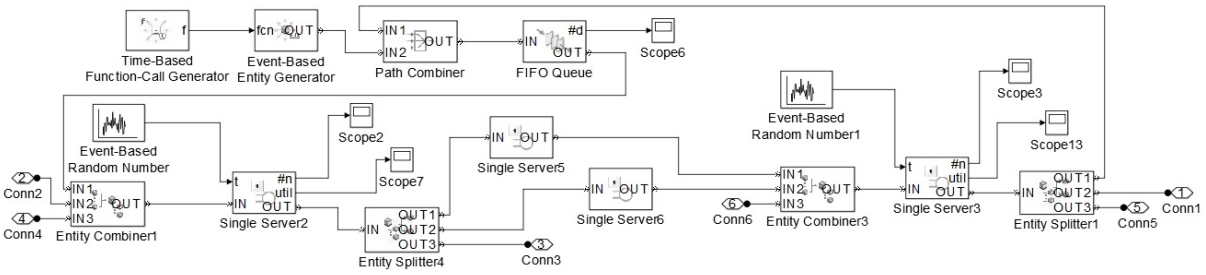


Figure 3.23: The *Process 1* subsystem of the fast food restaurant

The *Process 2* subsystem is designed in the same way as the *Process 1* subsystem, expect for the variable names that have been changed to match the description.

The *Resource A* subsystem has an entity generator that only executes at the start of the simulation in order to create the available resource A entities for subsystems *Process 1* and *Process 2*. These entities flow around the model, going to both *Processes 1* and *Process 2* when requested, to be used in the *Server 1A* and *Server 2A* of these subsystems and then returning to the FIFO queue in the *Resource A* subsystem.

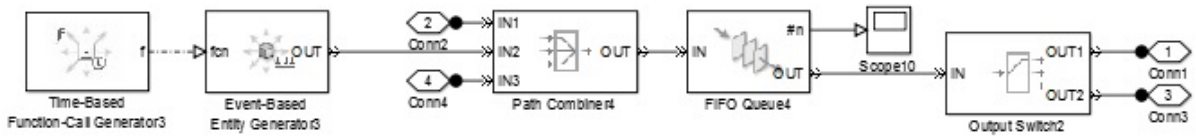


Figure 3.24: The *Resource A* subsystem of the fast food restaurant

The *Resource B* subsystem is designed in the same way as the *Resource A* subsystem, expect for the variable names that have been replaced to match the description.

In this model, as there are no shared resources restriction, one resource entity of each type (A and B) is going to be created for each server, i.e., as two entities resource A, one for *Server 1A* and one for *Server 2A* is going to be generated at the beginning of the simulation, as two entities resource B are generated to satisfy *Server 1B* and *Server 2B*.

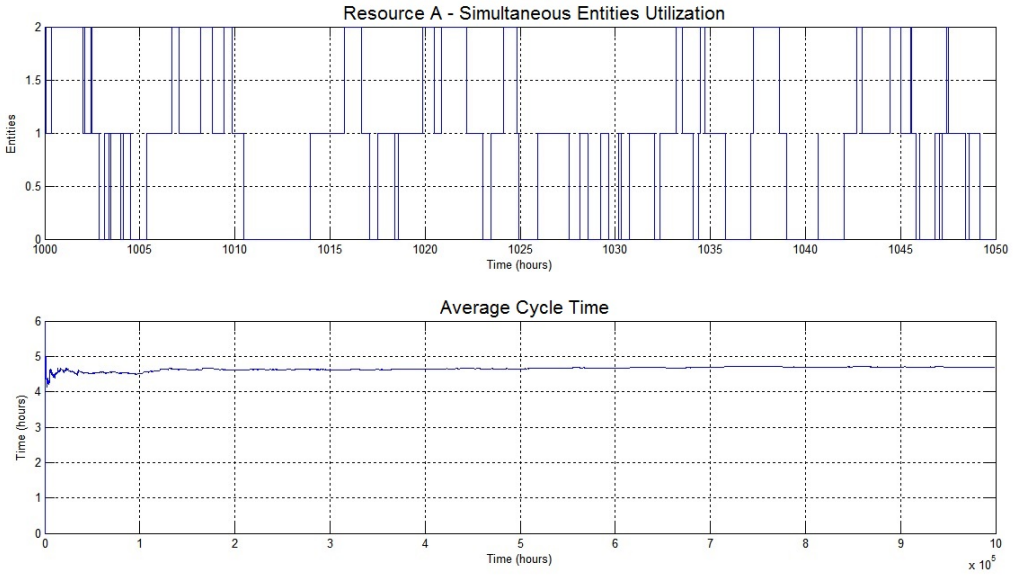


Figure 3.25: The resource A utilization over time and the average cycle time of the process

Figure 3.25 shows that, in some moments, the two entities resource A are being utilised, in *Server 1A* and *Server 2A* simultaneously, thus proving that the *Process 1* and *Process 2* are not restricted to resource availability, i.e., these processes do not share resources.

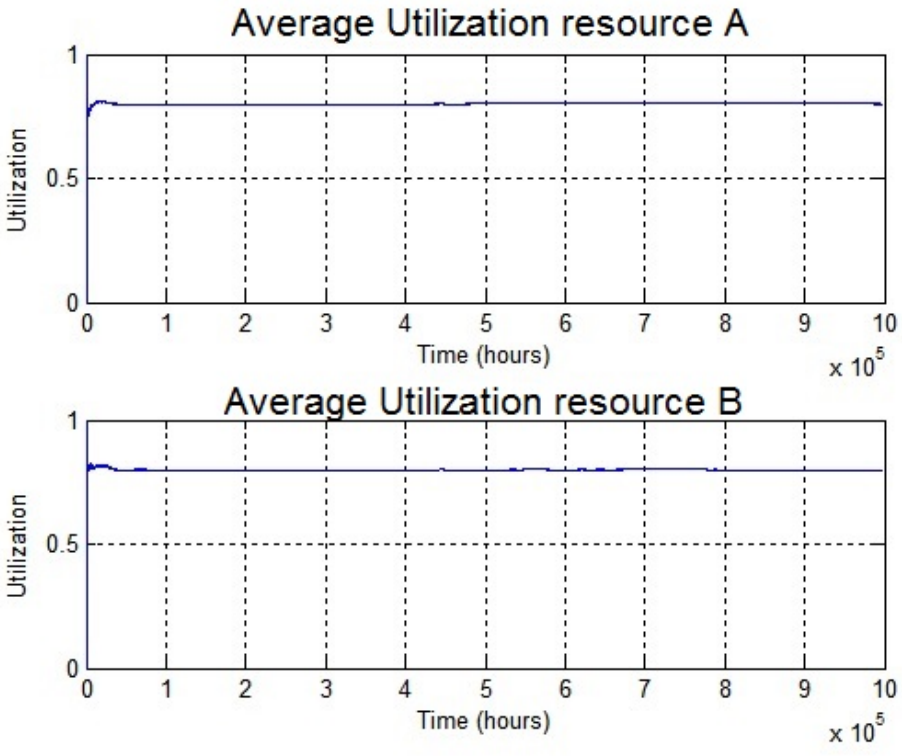


Figure 3.26: Average utilization of resources A and B

The entities types introduced in this model (resource A, resource B and employee) allow the model to have shared resources restrictions that are going to be explored in section 3.4.

*Remarks on the SimEvents simulation aspects of this example:*

This model introduced the concept of the combination and splitting of entities. The *Entity Combiner* block creates new types of entities from the different incoming entities, that includes the attributes of theses entities. The *Entity Combiner* only combine the incoming entities when all its ports has an available entity. On the other hand, the *Entity Splitter* can only be used after an *Entity Combiner* block, as it separates a merged entity into its original structures, with the same attributes as they had before merging.

An interesting characteristic about modelling these process with entities is that it is possible to set individual attributes for each entity, which permit a more customized model, for example, it is be possible to set an attribute that would specify that certain customers wouldn't want to have the output of the Resource A in their meal.

## 3.4 The Fast Food Restaurant example with Shared Resources

The model in this section is the same as the final version of the model used in the previous section, with the difference that, at this point, only one resource A entity and one resource B entity are going to be available for the Processes to use.

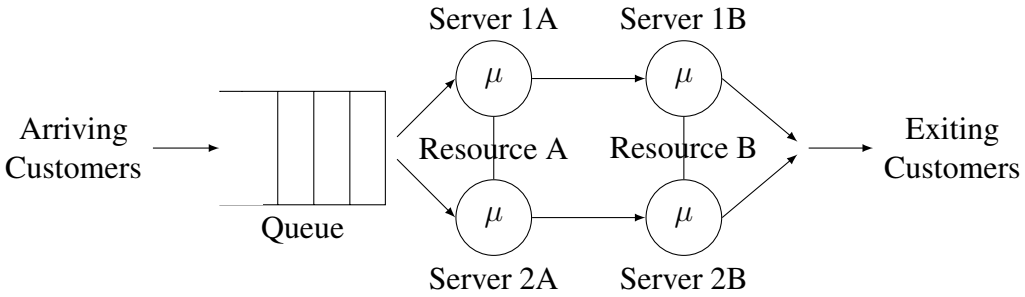


Figure 3.27: The fast food restaurant abstract model with shared resources in 2 parallel sets of servers configuration

It is possible to see now that, as there is only one entity resource A, either *Server 1A* or

*Server 2A* is going to be working at any instant. This same property is true for the resource B, considering *Servers 1B* and *Server 2B*.

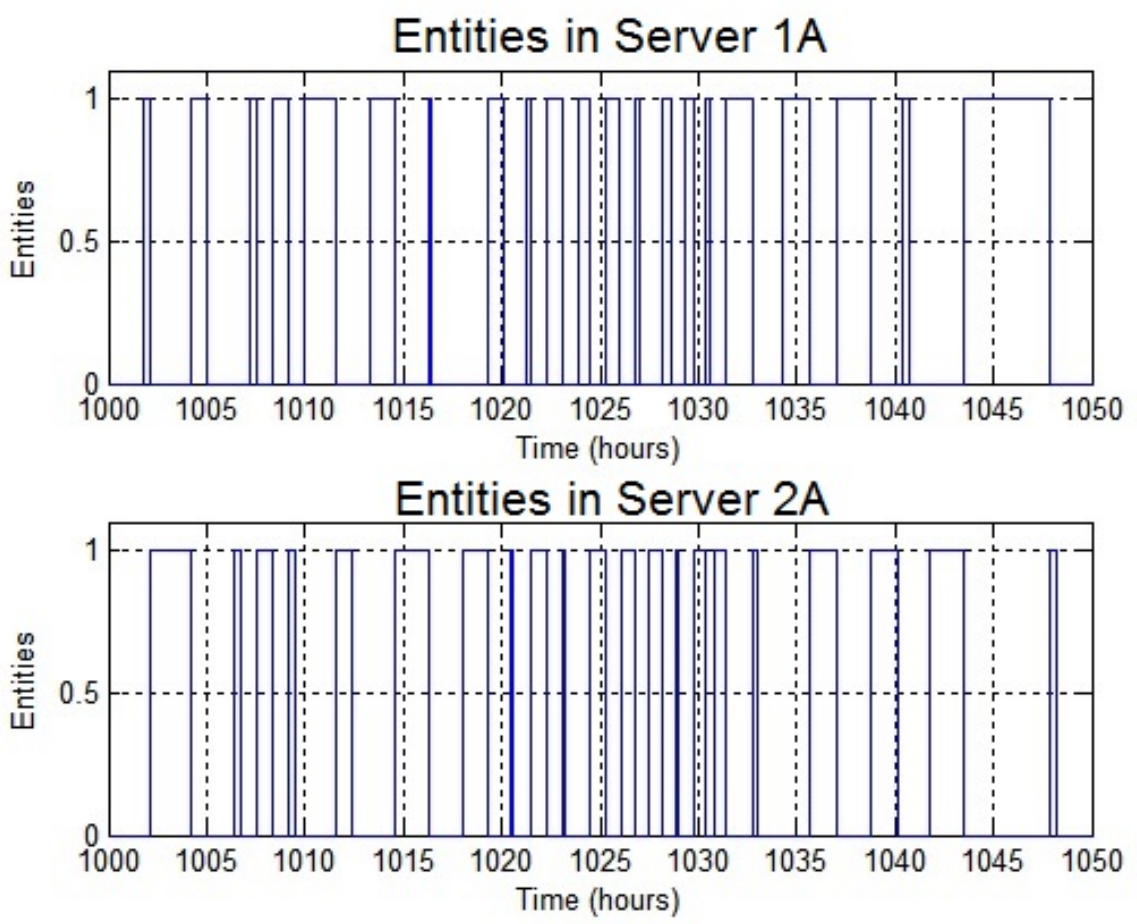


Figure 3.28: Entities in *Server 1A* and *Server 2A*

Running the model for the same parameters as of the last section, it is possible to see that process capacity now has drastically dropped, as the queue length starts to grow unbounded for the same IIT and service time.

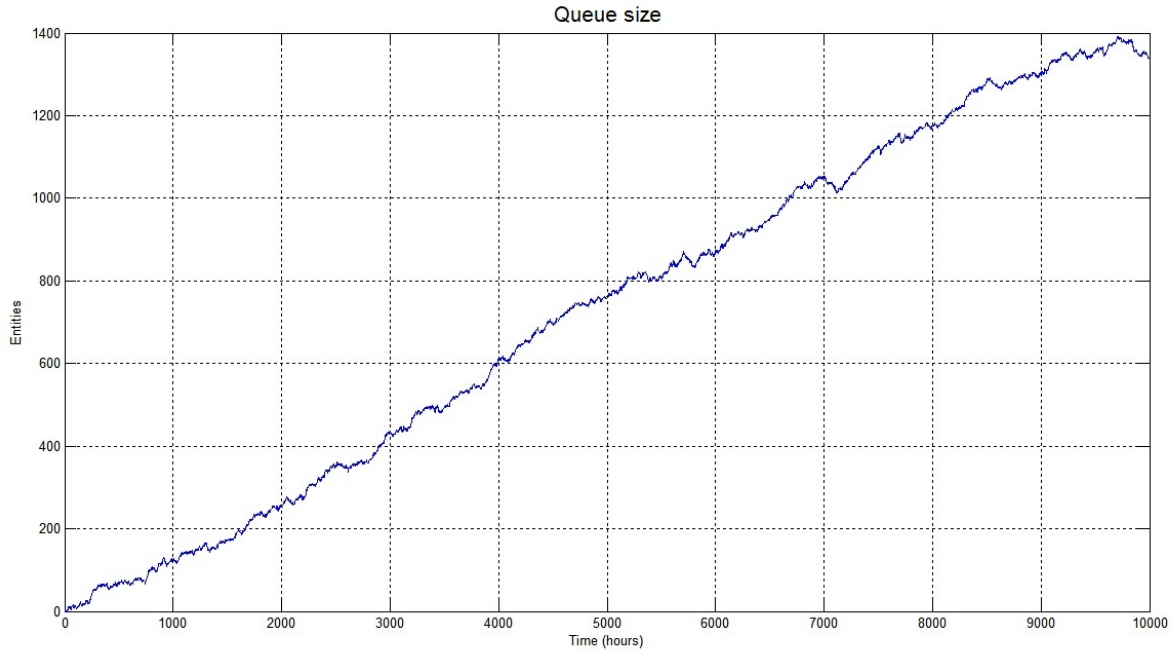


Figure 3.29: Queue length grows unbounded for $\mu = 0.5$ and $\lambda = 0.8$

It is possible to find a new value for $\lambda$ that gives a traffic intensity ($\rho$) smaller than 1, by gradually decreasing the value for $\lambda$ to find that for a $\lambda$ smaller than $0.67$, the queue reaches a steady state.

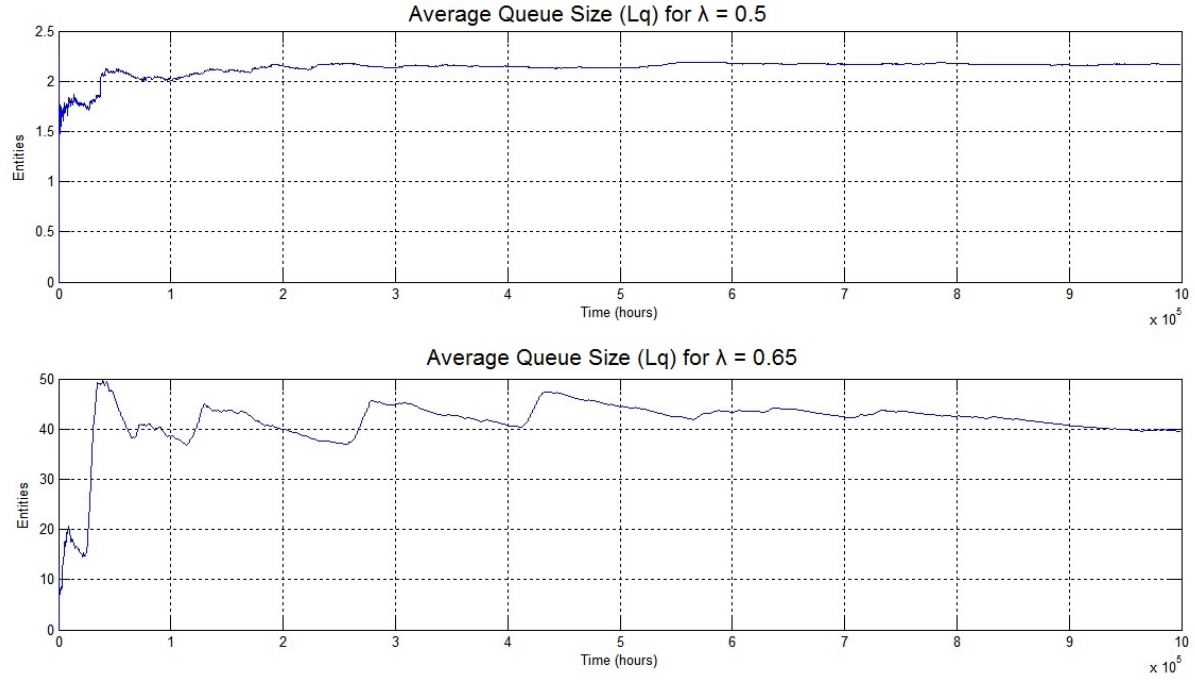


Figure 3.30: Average queue length for $\lambda = 0.5$ and $\lambda = 0.65$

## 3.5 Other Sample Modelled Processes

As a reference for more enthusiasts of this approach, two additional examples are modelled in SimEvents and explained. The first one is the same model as the described in difference equations in section 2.1 from chapter 2, for a comparison between the two approaches. The second model is a traffic intersection, which has the intersection itself as a shared resource.

In figure 3.31, the complete model of the retail store is displayed for a daily based simulation. From left to right, following the entities path, the local distributor subsystem is connected to the daily orders counter $O(k)$, which instantaneously counts and dispatches the entities from the local distributor to the server shipped $W(k)$, which has infinite capacity and an exponential service time $\tau$. Similarly the receiving $R(k)$ block, is an instantaneous counter block, connected between the server shipped and the subsystem store. When there is both a customer entity in the customer demand $D(k)$ and a product entity in store $I(k)$, respecting the restriction of the display product specified, a merge between entities occurs in selling $S(k)$, dispatching the combined entity to a sink, terminating the entities path. The function-call generator and the function-call split block generate the reset signals for both $O(k)$ and $R(k)$ signals on a daily basis.

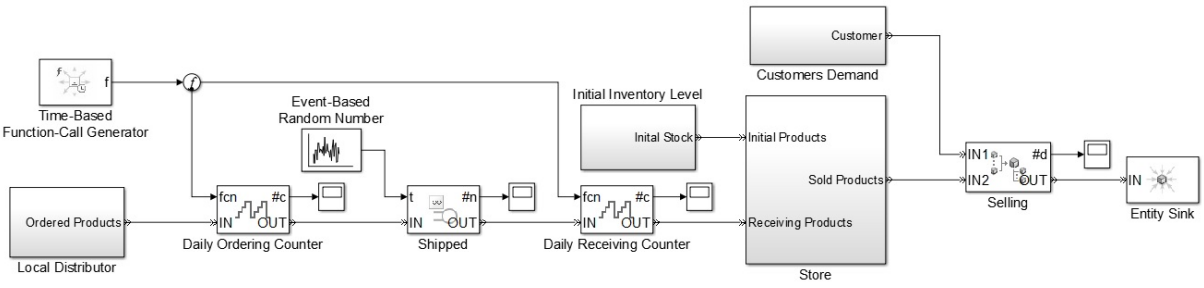


Figure 3.31: The retail store model using SimEvents

The store subsystem, from figure 3.32, has two paths for entity inputs, one coming from the *Initial Stock* subsystem and one from the *Daily-based Receiving* block, thus a path combiner merges both paths for the single input of the FIFO queue *Stock* that dispatches entities to the *For Selling* block. The *For Selling* block is a single server that stores the next unit to be sold, being refilled with another entity from the *Stock* if and when that event occurs. The *Enabled Gate*

interrupts the passage of entities from the *For Selling* when the display restriction is achieved, since the $I(k)$ is the combined number of entities in the blocks *Stock* and *For Selling*.

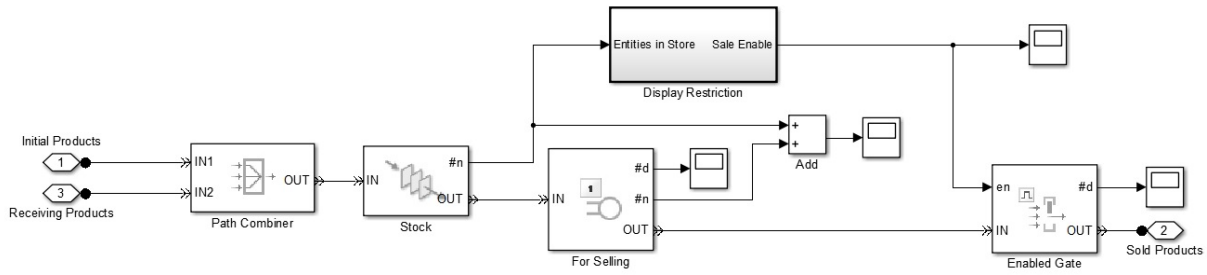


Figure 3.32: The *Store* subsystem (which includes inventory and display) for the retail store example

The *Display Restriction* subsystem, shown in figure 3.33, in the *Store* subsystem creates the conditional signal for the *Enabled Gate* to allow the entities to be sold to customers. This block gives a constant $1$ output signal if the combined number of entities in *Stock* and *For Selling* blocks is bigger than one entity or a constant $0$ output signal, otherwise.

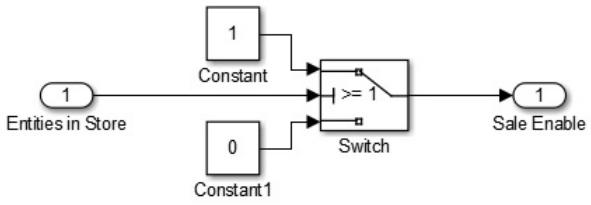


Figure 3.33: The *Display Restriction* subsystem in the *Store* subsystem for the retail store example

The demand $D(k)$, in this case, is expressed by a time-based entity generator, shown in figure 3.34, with an exponential IIT in the subsystem *Customers Demand*. The entities generated are sent to the backlog block where they wait to be merged with the product entity from the *Store* subsystem. This block can be altered to express an actual demand curve of an product using the approach of chapter 4.

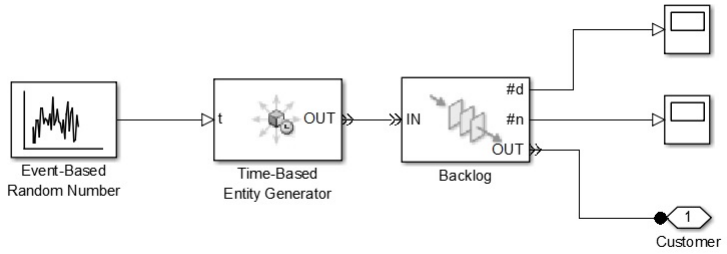


Figure 3.34: The *Customers Demand* subsystem for the retail store example

The subsystem *Initial Stock* is just a event-based entity generator, shown in figure 3.35, triggered only at simulation start to create the entities that are in the inventory $I(k)$ when the simulation starts to execute. The time-based function-call needs to specify at least one entity, the display unit, for the store inventory.

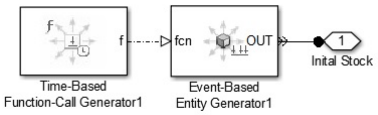


Figure 3.35: The initial stock subsystem for the retail store example

The *Local Distributor* subsystem, shown in figure 3.36, follows the same principle of the *Customers Demand* subsystem, however, with a deterministic generation, as it dispatches one new unit every day. The *Enabled Gate* can be configured to interrupt the entities to go from the subsystem *Local Distributor* to the subsystem *Store* before the start of the simulation.

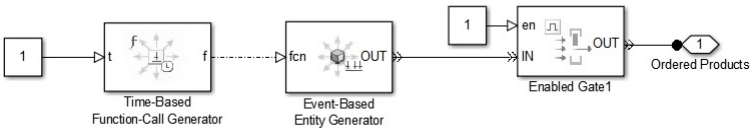


Figure 3.36: The *Local Distributor* subsystem for the retail store example

Although, the same methodology applied to the fast food restaurant in chapter 4 could be used, an actual control law is not going to be implemented for this example, since the purpose of this model just was to compare the difference equation and a DES approach. Therefore, plots and results were not included as they were not relevant to this work.

*Remarks on the SimEvents simulation aspects of this example:*

Another interesting point in the *Display Restriction* subsystem, shown in figure 3.33, is that, although the switch block can only accept time-based signals, as this subsystem is set to be treated as an atomic unit, the conversion conversion from event to time-based and back again after the block is not necessary. From [18], this configuration causes Simulink software to treat the subsystem as a unit when determining the execution order of block methods. For example, when it needs to compute the output of the subsystem, Simulink software invokes the output methods of all the blocks in the subsystem before invoking the output methods of other blocks at the same level as the subsystem block.

The second model presents another shared resource situation. It involves a traffic intersection between two roads controlled by a traffic light with only one lane, as in figure 3.37. To simplify matters, it is assumed that cars can either come to the intersection from North to South or from West to East.

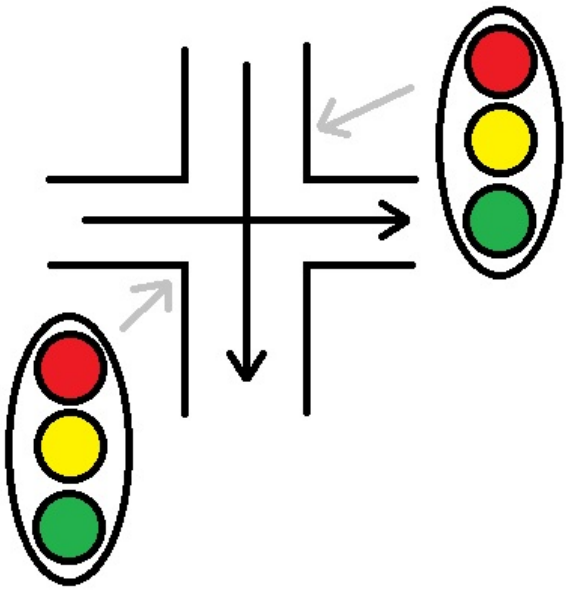

Figure 3.37: The traffic intersection representation

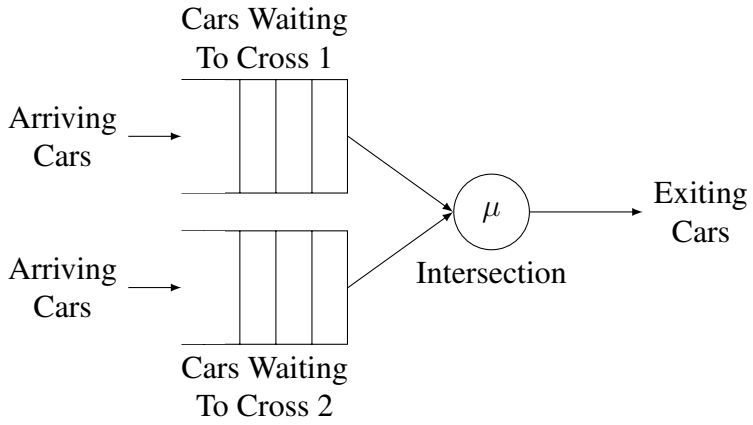


Figure 3.38: The traffic intersection abstract model

These problem is slightly simpler than the complete fast food restaurant example from section 3.4, as there is only one shared resource. However it introduces an interesting feature in the management of its shared resource, as it switches from one incoming car lane to the other in a time-based manner.

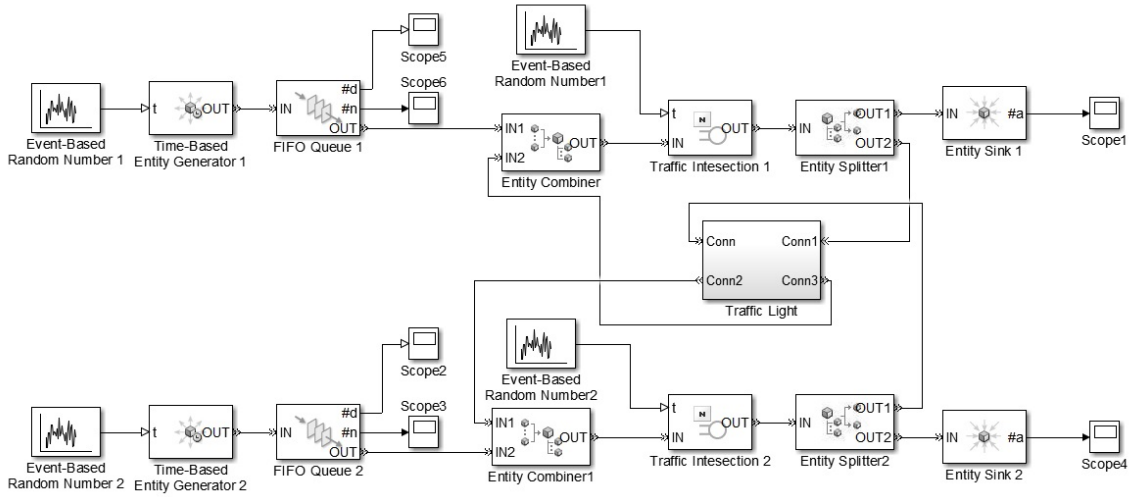


Figure 3.39: The traffic intersection model

Figure 3.39 shows the model for the traffic intersection problem. Two distinct entity generators, which can be configured to generate entities with different IITs, create the cars that cross the intersection in the two different directions. Both FIFO queues have infinite capacity to store the cars that are waiting to cross the intersection whenever the traffic light permits.

Although two servers called *Traffic Intersection* are displayed in the model, only one or none can be activated at any instant, representing the shared resource restriction, i.e., either the

entities from the top entity generator are crossing the server *Traffic Intersection 1*, then terminating their path in *Entity sink 1* or the entities from the bottom entity generator are crossing the server*Traffic Intersection 2*, then terminating their path in *Entity Sink 2*, or even no entity is crossing these servers, as the safety measure determined by the *Traffic Light* subsystem explained below.

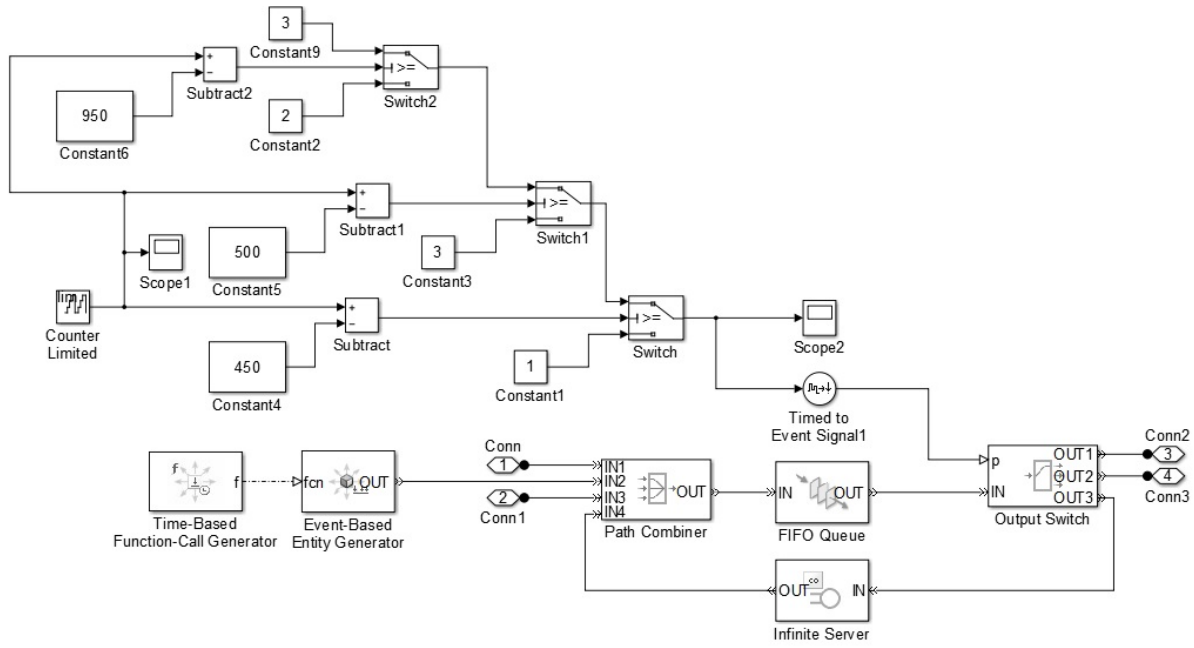


Figure 3.40: The *Traffic Light* subsystem from the traffic intersection example

The *Traffic Light* subsystem, shown in figure 3.40, represents the logic behind the traffic light. In this model, only one car can pass through the intersection at a time, as the entity generator in this subsystem only creates one entity at the start of the simulation. Multiple cars could go through each activated server *Traffic Intersection* at a time as both have infinity capacity, in the case that multiple entities were generated in this subsystem.

This traffic light works in the following fashion: for $45$ time units the entities from the top entity generator are allowed to enter *Traffic Intersection 1* by merging with the entity available from the *Traffic Light* subsystem, then for $5$ time units there is a safety time, when the entity from the*Traffic Light* subsystem does not leave this subsystem, after switching to allow the entities from the bottom entity generator to enter the *Traffic Intersection 2* by merging with the entity available, finally for $5$ time steps there is another safety time, when the entity from the*Traffic Light* subsystem does not leave this subsystem. The signal that controls the selection

of the active server is generated by a block *Limited Counter*, which counts the elapsed time and is set to reset its counting every 100 time units, in addition to a series of if clauses that select the correct state of the traffic light for each instant.

Figure 3.41 shows the result for running this model for 400 time units for exponentially distributed IIT for both entity generators of 2.5 time steps and mean service time of 1 for each exponentially distributed server traffic intersection.

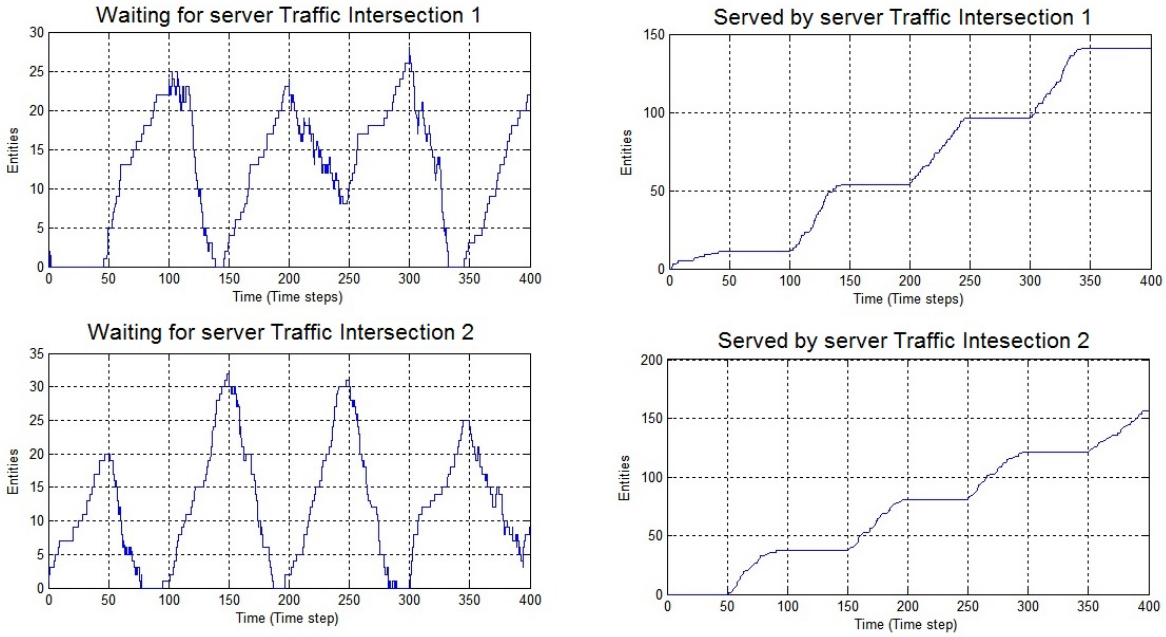


Figure 3.41: Results for the traffic intersection model

This traffic light logic is simple since it does not take into consideration any measured signals from the models, i.e., the IIT of each entity generator or the number of cars waiting to cross each FIFO queue. Surprisingly, most of the existing traffic lights still work in this manner.

# Chapter 4

# Economic modelling and control

So far, this work has targeted only the modelling part of business processes. From this point onwards, the control part of these process is investigated, measuring variables from the models to manipulate a variable in order to target a goal. A control law is proposed to control the financial status of the model, evaluating how this affects the model's dynamics. Therefore, it is necessary to characterise the demand of the service in relation to its price and how it affects the organization's financial measurements, expanding the fast food restaurant model presented in chapter 3 to include financial aspects.

Pursuing this goal, it is reasonable to assume that, for the fast food restaurant example as modelled before, customers can arrive and, based on the current price and the WIP, decide whether it is worth entering the queue or not [29]. Thus, although the model's IIT is going to remain the same, the actual number of customers entering the queue is varying over time. This tool is necessary for the Economic Value Added (EVA), as it permits the calculation of lost customers, consisting of those who would have eaten, but desisted (reneged) on entering the restaurant, basing their decision on a combination of the current price and the current WIP.

## 4.1 Price Elasticity of Demand

The price elasticity of demand is the responsiveness of quantity demanded ($Q$) to changes in price ($P$) [35]. It is represented by the variation in quantity divided by the variation in price, yielding the coefficient of price elasticity of demand ($E$). The arc elasticity is the measurement of the elasticity between two points in the price-demand curve and is an approximation of a normalized derivative of the demand with respect to price, given by the equation 4.1:

$$E = \frac{\Delta Q}{\Delta P}\frac{P}{Q} \tag{4.1}$$

The values for $P$ and $Q$ used in equation 4.1, by convention, are the mean values between the two points for $P$ and $Q$.

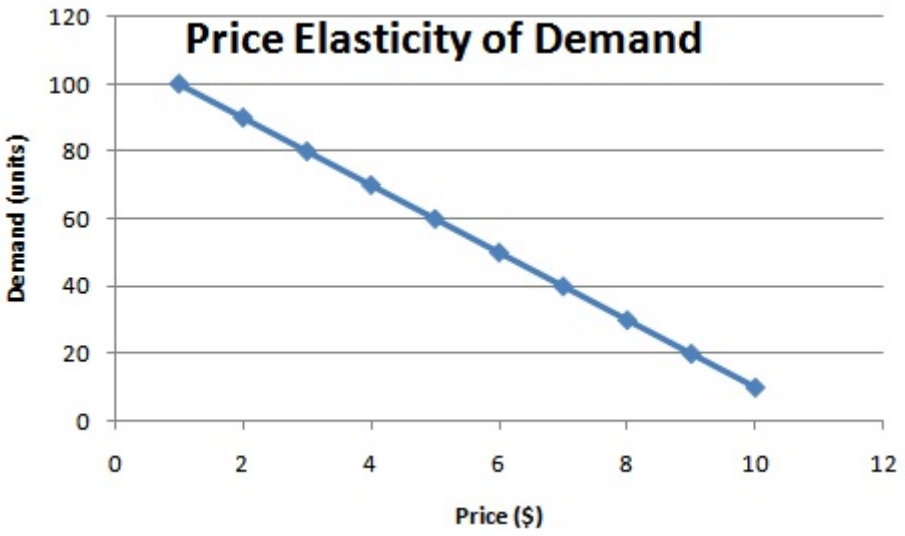


Figure 4.1: An example of a price elasticity of demand curve

For figure 4.1, the price elasticity of demand between the points A (for price = 6) and B (for price = 4) is:

$$E = \frac{\Delta Q}{\Delta P}\frac{P}{Q} \tag{4.2}$$

$$= \frac{70-50}{4-6}\frac{\frac{6+4}{2}}{\frac{70+50}{2}} \tag{4.3}$$

$$= -\frac{20}{2}\frac{5}{60} \tag{4.4}$$

$$= -0.8\bar{3} \tag{4.5}$$

Each elasticity value can be analysed in terms of the total revenue (TR) for the firm that the sales the product, before taxes or other deductions. The TR can be calculated from equation 4.6.

$$TR = UnitCost * QuantitySold \tag{4.6}$$

The complete range of values that E can assume, as well as the corresponding economic jargon and the TR tendency for a price modification in that region, is shown in table 4.1, with respect to the pertinent region:

| Elasticity | Description | TR situation for price rising | TR situation for price falling |
|---|---|---|---|
| E = 0 | Perfectly inelastic demand | Rises | Falls |
| -1<E <0 | Inelastic demand | Rises | Falls |
| E = -1 | Unit elastic | Remains the same | Remains the same |
| - $\infty$ <E <-1 | Elastic demand | Falls | Rises |
| E $\rightarrow$ - $\infty$ | Perfectly elastic demand | Falls | Rises |

Table 4.1: Price elasticity of demand regions

Usually, the coefficient of price elasticity of demand for a certain good changes as price changes, i.e., the $E$ is not constant over the entire price elasticity of demand curve for the product, being able to go from elastic to inelastic depending on the price of the product.

A related equation for calculating the price elasticity of demand is shown in 4.7 from [35], which represents the Point Elasticity, assuming a infinitesimal variation in the price to calculate the correspondent $E$ for a certain point of interest:

$$E = \frac{P}{Q}\frac{dQ}{dP} \tag{4.7}$$

A linear regression can be used to create a price elasticity of demand curve. A straight-line demand curve, as in equation 4.8, has different values for E at each point, except for the perfectly inelastic and perfectly elastic scenarios [35]. As the curve represents the demand as a function of the price, its slope ($-b$) is $dQ/dP$.

$$Q = a - bP \tag{4.8}$$

Therefore, replacing the slope of the straight-line curve from equation 4.8 in equation 4.7:

$$E = -b\frac{P}{Q} \tag{4.9}$$

From [32], the price elasticity ($E_o$) of demand of a fast food restaurant is $-0.743$ in one given point. Supposing that this elasticity was calculated for a price ($P_o$) and a demand ($Q_o$), the coefficients $a$ ($> 0$) and $b$ ($> 0$) of equation 4.8 are:

$$b = -E_o \frac{Q_o}{P_o} \tag{4.10}$$

$$Q_o = a - \frac{E_o Q_o}{P_o} P_o \tag{4.11}$$

$$a = Q_o - E_o Q_o = (1 - E_o) Q_o \tag{4.12}$$

The equation 4.8 gives a quantity demanded, that could be used directly as the inverse of the process's IIT (the arrival rate), for a given price for the fast food restaurant example. In this work, however, this demand is used in a slightly different manner in order to account the reneging customers due to dissatisfaction with the service (the current price charged and the current WIP). The IIT of the process is set as a constant and the customers have a service tolerance ($T$), which is exponentially distributed with the demand of equation 4.8 as its mean shown in equation 4.13.

$$\bar{T} = Q = (a - bP) \tag{4.13}$$

A price elasticity of the mean tolerance can be developed based on equation 4.13, similar to the one presented in figure 4.1. Thus, whenever a new customer arrives at the system, she checks if the current status of the process is consistent with her tolerance expectation. If yes, the customer enters the queue for waiting to be served, else she just leaves the process and is accounted for as a lost customer.

## 4.2 Economic Value Added

In [16], the Economic Value Added (EVA) is defined as a financial measure of the process performance: the generation of value to an organization's shareholders over a certain time period. This measurement can be used for an organization to evaluate and compare the company's alternatives for future investments. It is mathematically described as:

$$EVA = NetOperatingProfitAfterTax - CapitalCharge \tag{4.14}$$

The first term, the Net Operating Profit After Tax (NOPAT), as described in the name, is the profit after tax deduction, and it is given by equation 4.15:

$$NOPAT = (SalesRevenues - OperatingExpenses) * (1 - TaxRate) \tag{4.15}$$

As the fast food model only works with one type of meal, the sales revenues is calculated as the number of meals sold times the price charged for the meal. The Operating Expenses can be divided into Fixed Costs, relative to expenses that do not have a big variation of business activities over a period of time, such as rent, water bills, payments to employees (given that the organization did not hire, fire or promote anyone), and the Variable Costs, that is expressed as the number of meals sold times the unitary cost for the meal, which includes the costs associated with the materials and storage of that meal. The sales revenues minus the operating expenses yields the gross profit of the organization.

$$NOPAT = [meals * (PriceCharged - UnitCost) - FixedCost) * (1 - TaxRate)] \tag{4.16}$$

The second term of the equation 4.14, the Capital Charge (CC) is expressed by the product of the cost of capital, i.e., the expected return for the shareholders at a given time, and the amount of money tied up in the organization, which was invested by the shareholders.

$$CC = (CostOfCapital * ShareholdersInvestment) \tag{4.17}$$

Therefore, replacing equations 4.17 and 4.16 in 4.14:

$$\begin{aligned} EVA &= NOPAT - CC \\ &= (SalesRevenues - OperatingExpenses) * (1 - TaxRate) \\ &\quad - (ExpectedReturn * CapitalInvested) \end{aligned} \tag{4.18}$$

From equation 4.18, it is possible to notice for a positive EVA, the NOPAT must be bigger than the CC, thus the gross profit needs to be bigger than zero, and that, in order to increase the EVA, it is necessary either to increase the organization's profitability or to reduce the Capital Invested. Expanding equation 4.18, the EVA can be calculated as:

$$\begin{aligned} EVA &= [meals * (Price - UnitCost) - FixedCost) * (1 - TaxRate)] \\ &\quad - (ExpectedReturn * CapitalInvested) \end{aligned} \tag{4.19}$$

In [16], an extra term is added to equation 4.19, that refers to the cost of turning away a customer, which is an indirect cost for not trying to retain a customer, who could become loyal to the fast food restaurant. This new feature is a penalization for not meeting the customer's service tolerance, which is justified in [31]. Thus the EVA final equation becomes:

$$\begin{aligned} EVA &= \{[meals * (Price - UnitCost) - FixedCost] * (1 - TaxRate)\} \\ &\quad - (ExpectedReturn * CapitalInvested) \\ &\quad - (Price - UnitCost) * (TurnedAwayCustomers) \\ &\quad * (TurnAwayPenaltyFactor) \end{aligned} \tag{4.20}$$

Depending on how severely the manager wants to penalise the control for not being able to correspond to the customer's expectation in relation to the price charged and WIP, the manager can vary the parameter turn away penalty factor. A bigger penalty factor obviously has a bigger negative impact in the EVA than a small penalty factor. Equation 4.20 is used as the objective function for the optimization of the fast food restaurant model.

## 4.3 PID Controller

In industry, the continuous Proportional-Integral-Derivative (PID) controller is widely prevalent. It receives an error signal between a given reference, called the setpoint, and a measured plant signal, $y$, and generates an output control signal to the plant by attempting to minimise this error signal. A typical PID has 3 parameters that are used to create the control signal, $u$, is shown in figure 4.2:

- Proportional ($P$): produces a part of the actuation signal that is proportional to the error
- Integral ($I$) : produces a part of the actuation signal that is proportional to the integral of the error

- Derivative ($D$): produces a part of the actuation signal that is proportional to the derivative of the error

These three signals, parametrized by the so-call P,I,D gains, are then summed in order to obtain the plant input $u$. A block diagram representation of the control system is shown in figure 4.2.

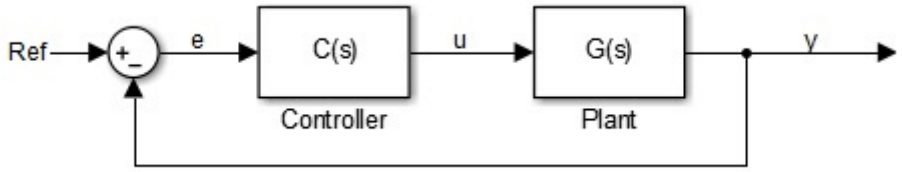


Figure 4.2: An example of a control system with a controller and a plant in a feedback loop

The PID used in Simulink has 4 parameters, the 3 standard ones just discussed as well as $N$, which is a filter coefficient used to carry out approximate differentiation. A representation of the internal structure of the PID block and its dynamic equation in terms of the complex variable, $s$, are shown, respectively, in figure 4.3 and equation 4.21, in a parallel configuration:

$$PID(s) = P + I * \frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}} \tag{4.21}$$

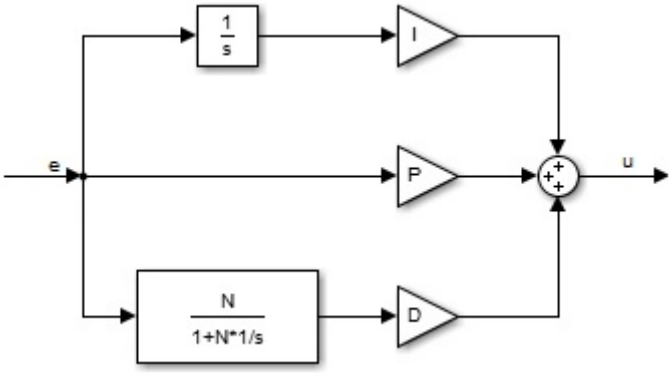


Figure 4.3: A representation of a PID controller's internal structure

Another feature present in the PID implemented in Simulink is an anti-reset windup method, in this case, back-calculation, with unitary gain. The anti-reset windup is a necessary response to the effects of saturation of the actuator, as it "turns off" the integral action when the

control signal reaches saturation. The anti-reset windup reduces the control effort and the overshoot whenever the latter occurs [13].

The fast food restaurant problem targets the optimization of the EVA, through an objective function, and not the waiting time in the system or the WIP directly. The controller is then responsible to control the WIP of the system by varying the price charged for each meal to optimize the EVA. For this reason, the gains for the PID controller ($P$, $I$ and $D$) and the Optimal WIP for each set of conditions passed for the simulation are chosen by an optimization method, using a fixed filter coefficient, $N$.

The fast food restaurant problem then becomes an Optimal control problem, in which, the optimization method is going to determine the parameters for the controller that yields a maximum value for the objective function. Furthermore, in this case, as a black box as the input becomes the price and the output is the WIP of the model. The stability of this problem is then related to the state of the waiting queue, thus, for a stable instance of this problem, for a bounded arrival rate of the customers, a bounded queue is expected.

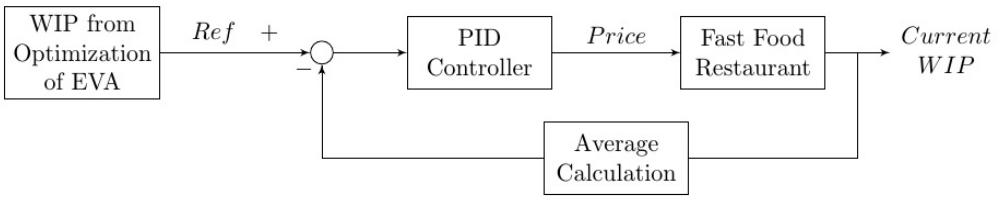


Figure 4.4: The block diagram of the problem with gains for the PID controller and Optimal WIP chosen by the optimization method using the EVA (from equation 4.20) as the objective function

## 4.4 Optimization

The optimization to find the best controller parameters that maximize the EVA over a certain time period is necessary in order to effectively control the price charged in the fast food restaurant example. Since this model is subjected to random variations, the objective function is not smooth and an optimization method based on the gradient of the objective function is not suited for this task [1]. Therefore, a derivative-free method was chosen to carry out this

optimization. Pattern Search (PS) [1] and the Genetic Algorithm (GA) [15] methods are the most commonly used methods of this class [17].

Genetic algorithms were directly inspired by Darwin's theory of the survival of the fittest, creating a random original population (a set of points in the search space) that, utilizing a set of operations based on the fitness of each individuals in the current generation, creates new fitter generations until a stopping criteria (average fitness, number of generation, ...) is met.

Pattern search uses a pattern, which is a vector of directions, and a scalar, the original mesh size, specified by the user to find the optimal point. The current point is set as the starting point. Then, at each iteration, based on the current point, the current mesh size and the pattern, the algorithm creates a mesh, which is a group of points surrounding the current point in the directions specified in the pattern. At this stage, a poll is done, in which the algorithm compares the objective function's evaluation for the current point and the objective function's evaluation for the points in the mesh. In case there is a point in the mesh with a smaller objective function value than the current point, this point becomes the new current point and the poll is said to have been successful, doubling the current mesh size (the mesh is expanded), else, the poll is unsuccessful and the current mesh size is halved (this operation is called mesh refinement). This process is repeated until the stopping criterion (mesh size, number of iterations, ...) is reached.

A comparison between these two methods can be found in [2], which concludes that, although the both yields results similar for the problem presented in [2], GA is more computationally expensive than PS, hence, in this work, PS is selected for the maximization of the EVA.[1]

[1]Matlab's Optimization toolbox only works with minimizing functions [19], therefore, it is necessary to transform the optimization problem from $\max_x f(x)$, where $f(x) =$ EVA, to $\min_x g(x)$, thus, the objective function used in this problem becomes $g(x) = -f(x) = -$ EVA.

# Chapter 5

# Implementation of the control and results

In this chapter, the model for the fast food restaurant from chapter 3 is modified to include the price elasticity of demand for customer's demand, representing by the reneging of dissatisfied customers, and the control features presented in chapter 4. Figure 5.1 presents the complete modified model that included the features mentioned before, then, it is broken up into its component systems to facilitate explanation.

After the introduction of this new modified model of the fast food restaurant, a comparison is made between running a fast food restaurant without shared resources and with shared resources, in terms of their financial results for different turn away penalty factors, service time and interarrival time. For each situation presented, the PS algorithm determines the parameters of the PID that maximize the EVA, determining the financial feasibility of this project to the given conditions.

## 5.1 Implemented model with inclusion of EVA and control

The model from chapter 3 in SimEvents is expanded to include the reneging of dissatisfied customers and the financial status of the system and the PID controller that sets the current price based on the current WIP. Figure 5.1 shows the complete model divided into the seven main systems to facilitate the explanation of the complete model, as mentioned in chapter 3: (i) *Customers Arriving and Queueing System*, (ii)*Entering System*, (iii)*Lost Customers System*, (iv)*Customer Serving System*, (v)*Exiting Customers System*, (vi)*WIP Calculation System* and (vii)*Management System*. Each system is now explained in detail.

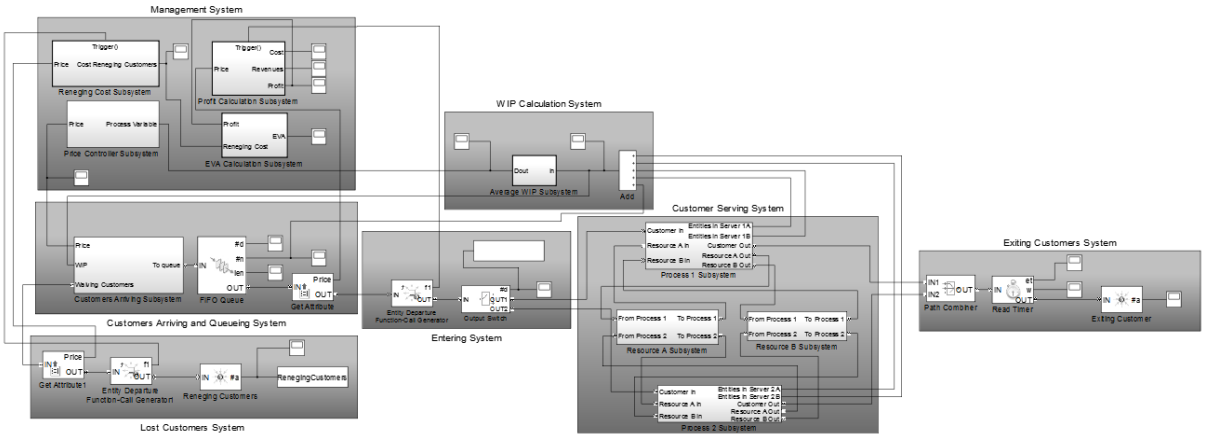


Figure 5.1: Fast food restaurant model in SimEvents format, modified to include EVA

Figure 5.2 presents three different parts of the system. The *Customers Arriving and Queueing System* system generates the incoming customers and stores them in a FIFO queue to dispatch to the *Entering System* that routes the customer to one of the two parallel sets of servers. In the *Customers Arriving Subsystem*, the reneging customers, who do not enter the queue based on their tolerance value, are selected and are therefore sent to the *Lost Customers System* in order to exit the process.

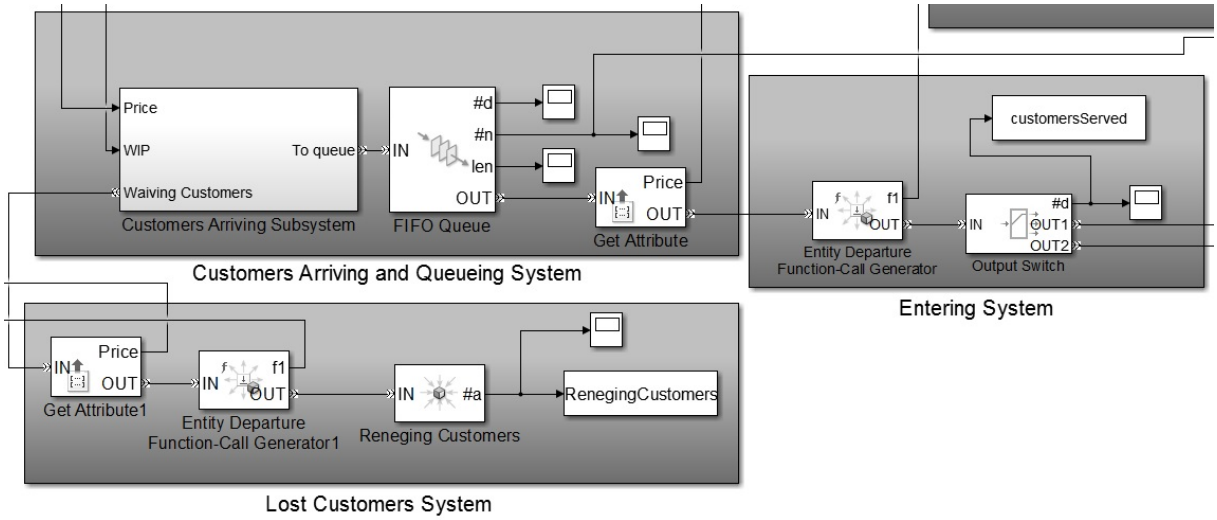


Figure 5.2: *Customers Arriving and Queueing System*, *Entering System* and *Lost Customers System*

Figure 5.3 presents the internal logic of the *Customers Arriving Subsystem*. This subsystem performs the actual calculation of the IIT based on the chosen mean IIT and the

calculation of each customer's tolerance to the current charged price and the WIP.

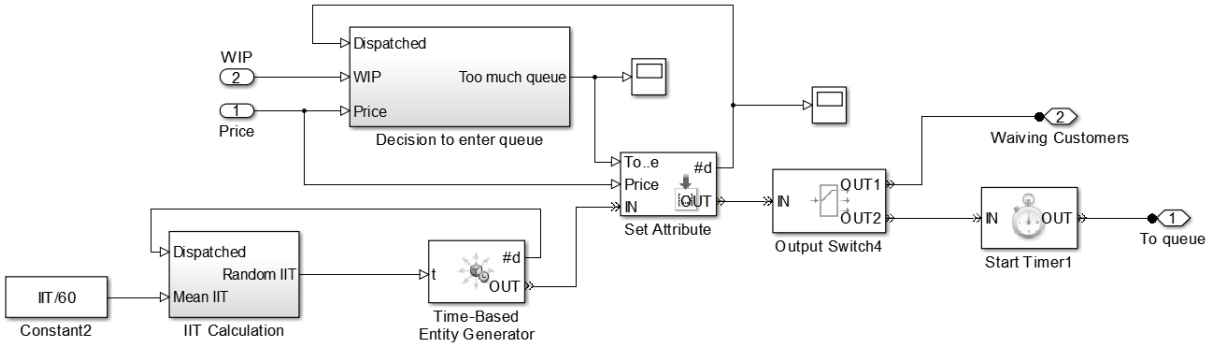


Figure 5.3: The *Customers Arriving Subsystems*

The *Random IIT* subsystem from the *Customers Arriving Subsystem* is shown in figure 5.4, which performs the mathematical transformation, explained in appendix A, to generate an exponentially distributed variable with mean IIT.

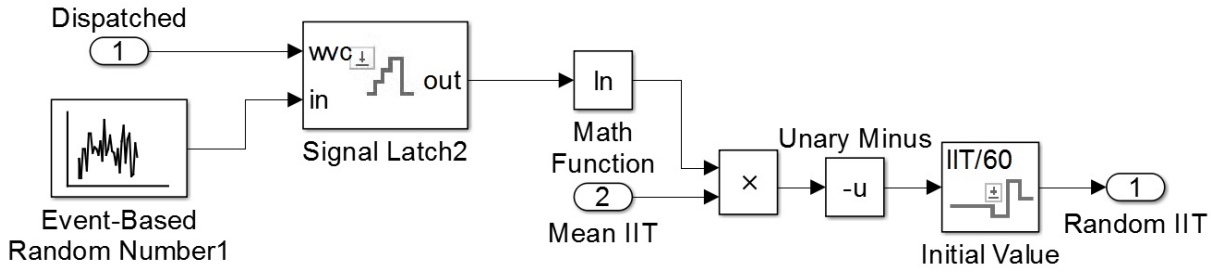


Figure 5.4: The *Random IIT Subsystem* from *Customers Arriving Subsystem*

The *Decision to enter queue* subsystem from *Customers Arriving Subsystem* is shown in figure 5.5, which calculates the customer's tolerance based on the price charged and subtracts it from the current WIP - 1, yielding the path chosen for the generated customer, i.e., whether the customer is going to the fast food's queue or neglect his desire to be served at the fast food restaurant. The subtraction of one unit from the WIP reinforces the parallelism of the process, as whenever the WIP is equal to 1, there is one parallel set of servers free.

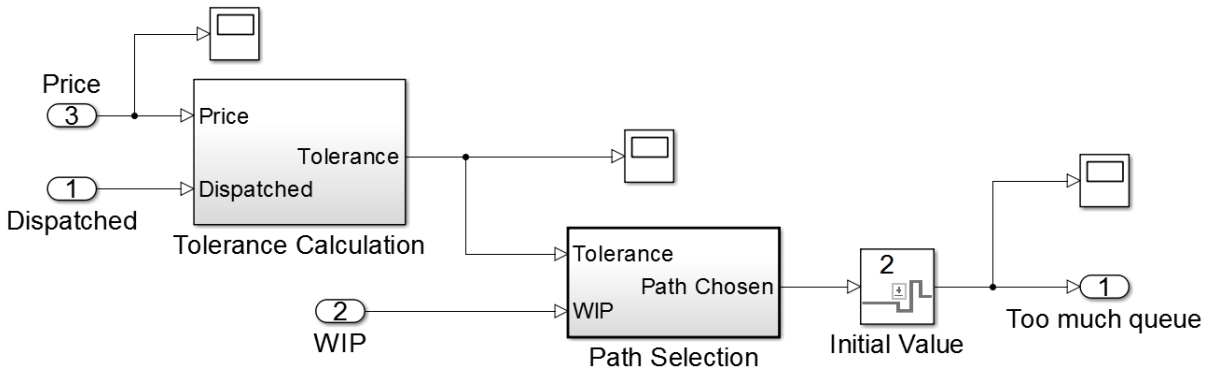


Figure 5.5: The *Decision to enter queue Subsystem* from *Customers Arriving Subsystems*

The *Tolerance Calculation* subsystem in *Decision to enter queue* subsystem from *Customers Arriving Subsystem* is depicted in figure 5.5, which calculates the customer's tolerance T, explained in chapter 4, using the transformation explained in appendix A to calculate the exponential distribution variable.

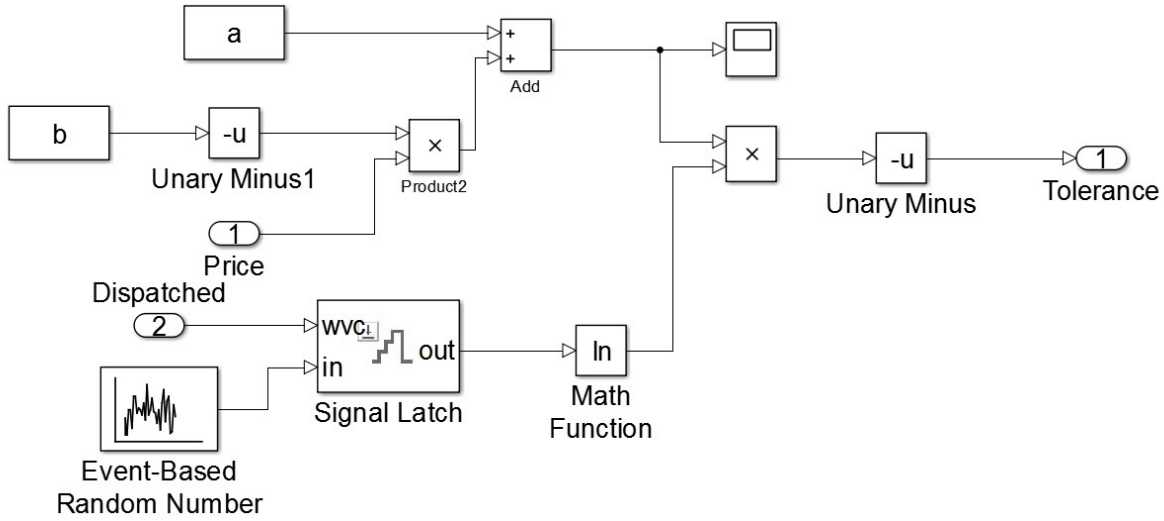


Figure 5.6: The *Tolerance Calculation* subsystem in *Decision to enter queue* subsystem from *Customers Arriving Subsystem*

The *Path Selection* subsystem in *Decision to enter queue* subsystem from *Customers Arriving Subsystem*, in figure 5.7, is basically a switch that, in case the incoming signal is smaller than zero, it sends a constant 1 signal and, in case it is bigger or equal, it sends a constant 2 signal, that feeds the output switch in order to decide whether the customer wants to

join the queue or not.

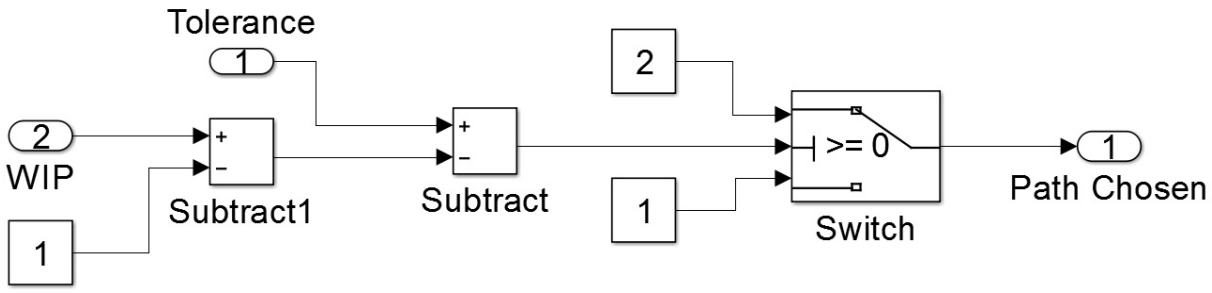


Figure 5.7: The *Path Selection* subsystemin *Decision to enter queue* subsystem from *Customers Arriving Subsystem*

The actual serving and exiting logic of the fast food restaurant model from chapter 3 is maintained in this modified version, shown in figure 5.8, i.e., the *Process 1 Subsystem*, *Process 2 Subsystem*, *Resource A Subsystem* and *Resource B Subsystem* remain unaltered in this new model, in the *Customer Serving System* and in the *Exiting Customers System* as before. Thus, the whole serving process and its subsystems are not explained in more details to avoid repetition. The only exceptions are for the exiting signals from the servers that indicate the presence or absence of a customer in such server.

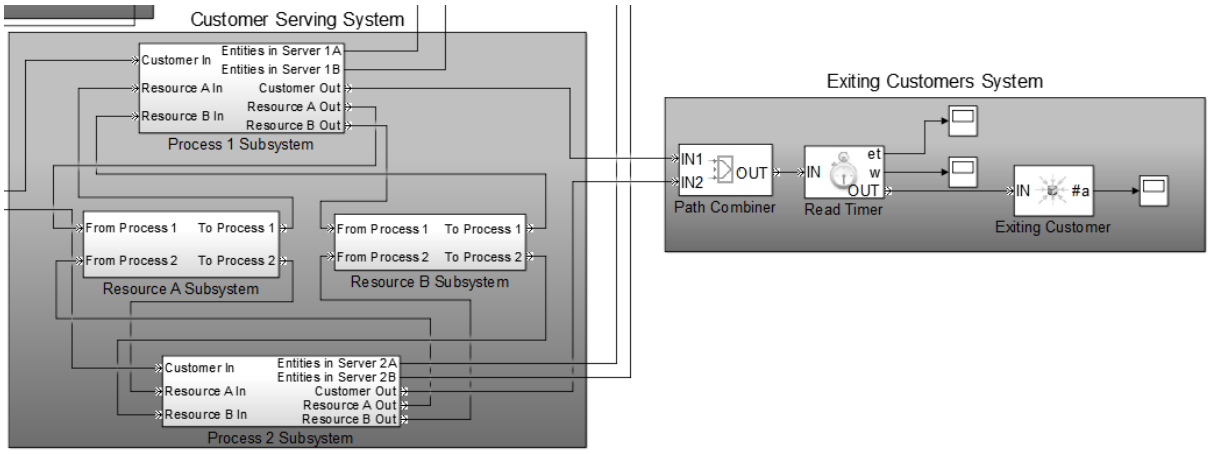


Figure 5.8: *Customer Serving System* and *Exiting Customers System*

The WIP calculation and its average value are generated in the *WIP Calculation System*. The *Average WIP Subsystem* calls a Matlab function to calculate and persists the value of the WIP over time, which is explained in detail in appendix D.

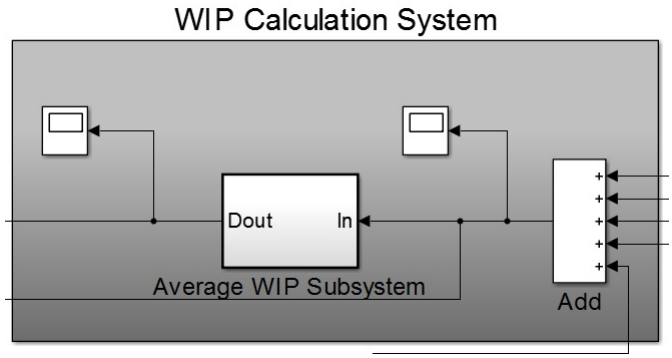


Figure 5.9: The *WIP Calculation System*

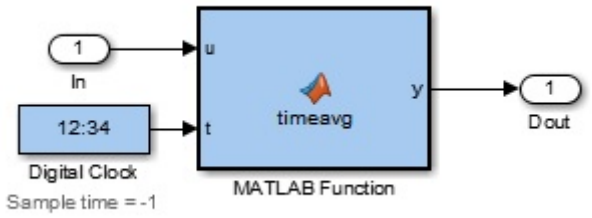


Figure 5.10: The *Average WIP Subsystem*

Figure 5.11 presents the *Management System* that computes all the costs and revenues involved in the process and calculates the major financial status of the system. [1] In this part of the system, the current price charged is calculated based on the average WIP in the *Price Controller* subsystem.

[1]Some of the subsystems present in this system have a trigger input, which causes the subsystem to only recalculates its output when there is a change in the trigger signal, i.e., the revenue of the process will only be incremented by the current price charged whenever a new customer is served.

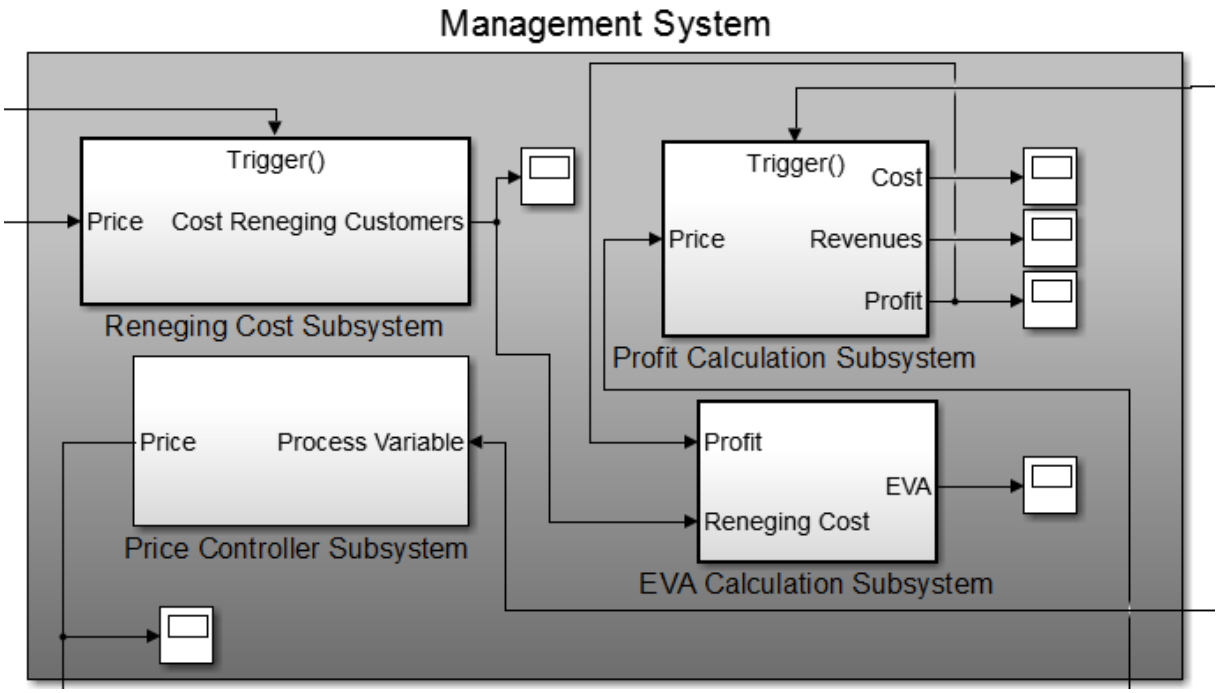


Figure 5.11: The *Management System*

In figure 5.12, the profit is calculated, triggered by the dispatching of an entity to the entering customer logic. When triggered, this subsystem calculates the process current revenue as a accumulation of previous revenues and the one from the new customer, i.e., the price paid for the new customer. The revenue is then subtracted from the total cost of the process, both variable and fixed, resulting in the current profit of the process.

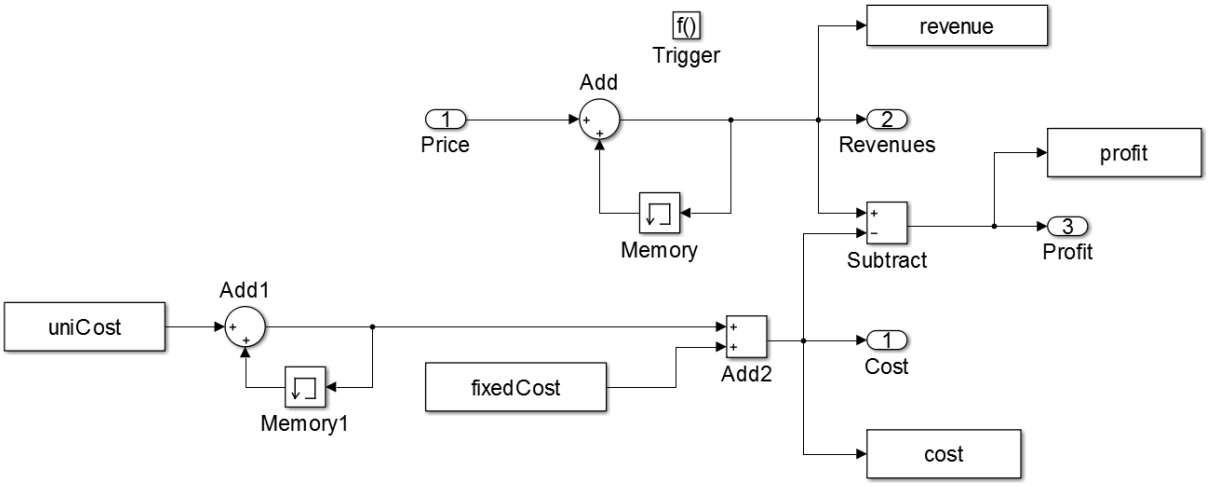


Figure 5.12: The *Profit Calculation Subsystem*

The cost of the lost costumers is depicted in the reneging cost subsystem in figure 5.13. This logic was explained in chapter 4, as the last added term to the EVA calculation, which will be shown in figure 5.14.

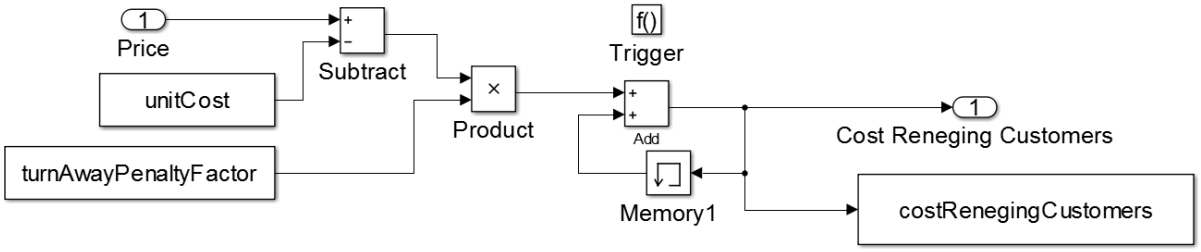


Figure 5.13: The *Reneging Cost Subsystem*

The figure 5.14 exposes the calculation for the EVA as it was explained in chapter 4, thus, the explanation is omitted here.

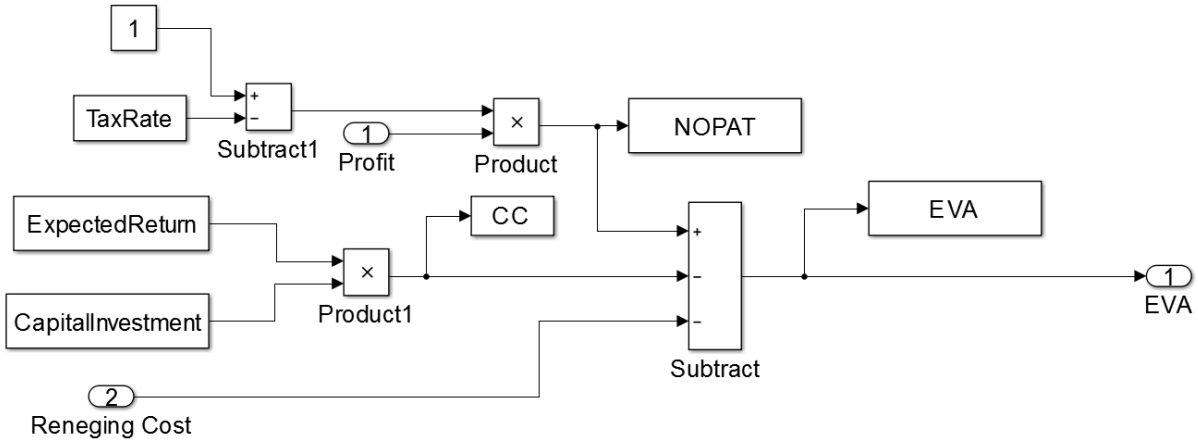


Figure 5.14: The *EVA calculation Subsystem*

Finally, figure 5.15 shows the *Price Controller Subsystem*, which calculates the price that is charged to achieve the EVA's maximization in the specified time interval. As explained in chapter 4, the PID's parameters are calculated targeting the optimal WIP for this process, yield by the optimization tool.

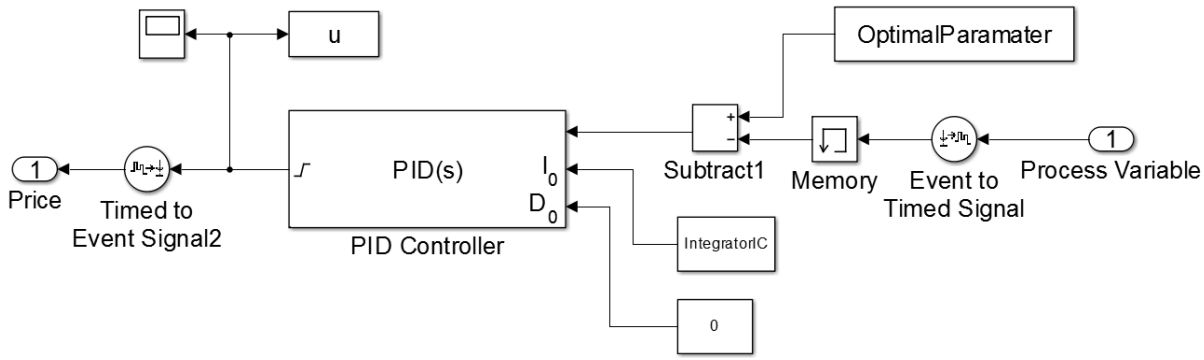


Figure 5.15: The *Price Controller Subsystem*

## 5.2 Feasibility Analysis

The Feasibility study for the fast food example proposed investigates the economic viability for implementing this project under a set of assumptions for the model and how it would react to a certain set of different turn away penalty factors. Two scenarios are studied as described below:

*Scenario 1:*

- Only one entity A and one entity B are available at the process, i.e., the resources are shared between the two sets of sequential servers;
- Capital invested: 400000;
- Monthly fixed cost: 10000.

*Scenario 2:*

- Two entities A and two entities B are available at the process, i.e., there is no shared resource, hence the corresponding capital invested and monthly fixed cost are higher;
- Capital invested: 600000;

- Monthly fixed cost: 15000;

The two scenarios have the common parameters from table 5.1:

| Parameter | Value |
|---|---|
| IIT (minutes) | 1/0.65 = 1.5385 |
| End time (hours) | 720 (= 1 month) |
| Restaurant Closing Hour | Never |
| Elasticity $E_0$ | -0.743 |
| Demand $Q_0$ | 10.0 |
| Price $P_0$ | 5.0 |
| Tax rate | 0.3 |
| Return expected | 0.15/12 = 0.0125 |
| Unit Cost ($) | 2.0 |
| Server 1A mean service time | 1.0 |
| Server 1B mean service time | 1.0 |
| Server 2A mean service time | 1.0 |
| Server 2B mean service time | 1.0 |
| Number of employees | 2.0 |
| Customers Tolerance | 1.0 |

Table 5.1: Default parameters for both scenarios

The PS Algorithm was set to use the GSS Positive Basis 2N polling method, which is one of the most efficient methods for polling, with a mesh tolerance of $10^{-3}$ and a initial mesh size of $5$, besides its default configuration in Matlab's Optimization toolbox, which are omitted here.

From table 5.1, the price elasticity of the mean tolerance curve becomes:

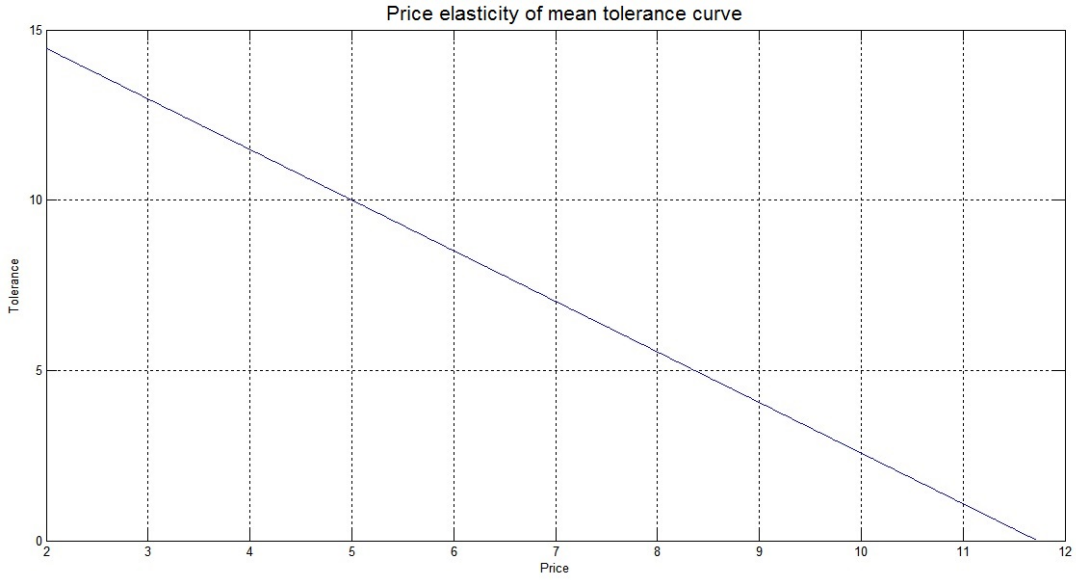


Figure 5.16: The price elasticity of mean tolerance curve for the fast food example

From figure 5.16, the mean tolerance, explained in chapter 4, has two extreme points: at the point in which the price is equal to $2$, the tolerance has its peak, however the process is not profitable as the meal is sold at its unit cost. The point in which the curve intersects the horizontal axis, for the given conditions, when price is equal to $11.7295$, the tolerance becomes zero and no customer enters the system as the elasticity becomes $-\infty$. As the demand is always non-negative, the tolerance is constant and equals to zero for any price bigger than $11.7295$.

The results for the maximum EVA for each turn away penalty factor in the scenario 1 is shown in table 5.2 and graphically represented as a mesh in figure 5.17.

| Turn away penalty factor | Optimal WIP | EVA |
|---|---|---|
| 0 | 0.03125 | 120507.3391 |
| 0.5 | 1.5313 | 86398.312 |
| 1 | 1.5313 | 59932.5404 |
| 1.5 | 2.0156 | 41890.3969 |
| 2 | 2.0156 | 26807.5928 |
| 2.5 | 2.0781 | 10078.6536 |
| 2.6 | 2.25 | 10074.7962 |
| 2.7 | 2.25 | 8372.6265 |
| 2.8 | 2.25 | 8516.2282 |
| 2.9 | 2.2813 | 5405.0953 |
| 3 | 2.2813 | 2547.242 |
| 3.1 | 2.2813 | 2429.7882 |
| 3.2 | 2.4063 | 1261.7554 |
| 3.3 | 2.4063 | -182.0804 |
| 3.4 | 2.4063 | -2291.5673 |
| 3.5 | 2.5313 | -3226.0185 |
| 4 | 2.5 | -9729.2227 |
| 4.5 | 6.75 | -11932.97 |
| 5 | 5.75 | -11854.798 |

Table 5.2: Results for the fast food restaurant example for scenario 1

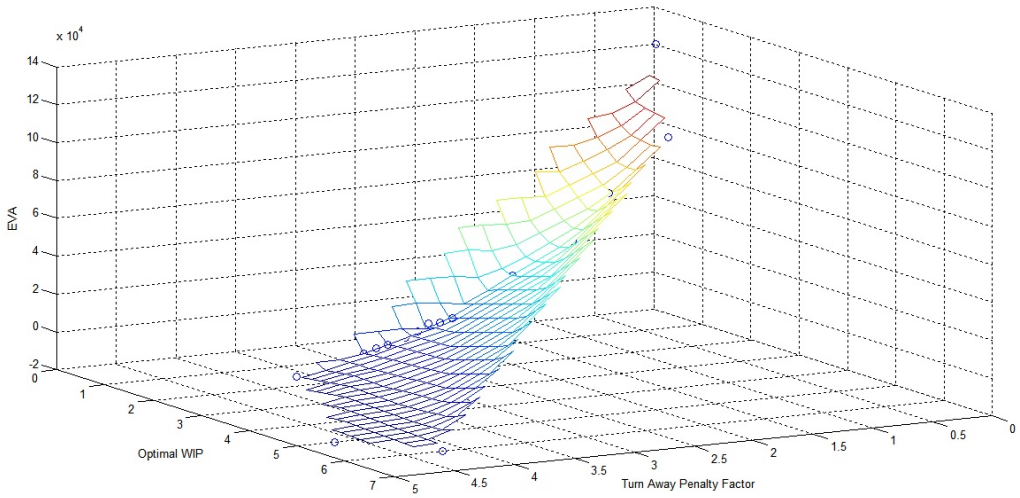


Figure 5.17: Fast food restaurant example graph for scenario 1

The results for the maximum EVA for each turn away penalty factor in the scenario 2 is shown in table 5.3 and graphically represented as a mesh in figure 5.18.

| Turn away penalty factor | Optimal WIP | EVA |
|---|---|---|
| 0 | 0 | 123572.5632 |
| 0.5 | 1.25 | 98021.2863 |
| 1 | 1.25 | 81449.2622 |
| 1.5 | 1.25 | 64757.6321 |
| 2 | 1.375 | 52436.1334 |
| 2.5 | 1.375 | 42688.1652 |
| 3 | 1.375 | 33557.8149 |
| 3.5 | 1.4375 | 26768.8272 |
| 4 | 1.5 | 21095.1067 |
| 4.5 | 1.5 | 14709.4633 |
| 5 | 1.5 | 11087.2519 |
| 5.5 | 1.5 | 7226.1699 |
| 6 | 1.5 | 1912.9077 |
| 6.1 | 1.5625 | 616.6918 |
| 6.2 | 1.5625 | -178.0798 |
| 6.3 | 1.5625 | 461.6558 |
| 6.4 | 1.5625 | 12.1271 |
| 6.5 | 1.5313 | -746.9661 |
| 6.6 | 1.5625 | -1351.2149 |
| 6.7 | 1.5938 | -2760.9409 |
| 6.8 | 1.5313 | -3046.3559 |
| 6.9 | 1.5313 | -4073.6266 |
| 7 | 1.5313 | -5769.6707 |
| 7.5 | 1.5313 | -8017.093 |

Table 5.3: Results for the fast food restaurant example for scenario 2

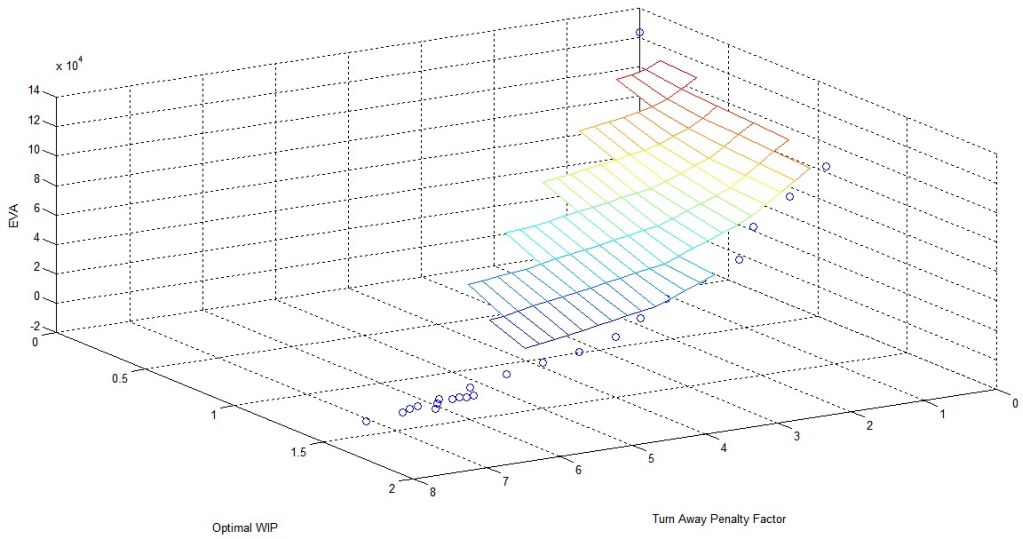


Figure 5.18: Fast food restaurant example graph for scenario 2

Some notable aspects of the results are pointed out for the fast food restaurant example:

- The PID's parameters yield by the PS algorithm, in tables 5.4 and 5.5, have negative values, except for the integrator initial condition, as they have a reverse influence in the process, i.e., a higher price diminishes the queue, as less customers enter the queue and a smaller price motivates customers to enter it;

| Turn away penalty factor | Optimal WIP | Optimal P | Optimal I | Optimal D | Optimal integrator initial condition |
|---|---|---|---|---|---|
| 0 | 0.03125 | -6.5625 | -1.625 | -1 | 8.3647 |
| 0.5 | 1.5313 | -6.5625 | -0.375 | -3.1875 | 5.2397 |
| 1 | 1.5313 | -16.5625 | -0.0625 | -3.1875 | 7.8179 |
| 1.5 | 2.0156 | -10.3125 | -0.023438 | -3.1875 | 6.5679 |
| 2 | 2.0156 | -10.3125 | -0.023438 | -3.1875 | 6.5679 |
| 2.5 | 2.0781 | -0.29297 | -0.10156 | -3.1875 | 11.5679 |
| 2.6 | 2.25 | -4.675 | -0.1 | -3 | 8.3647 |
| 2.7 | 2.25 | -4.675 | -0.060938 | -3 | 8.3647 |
| 2.8 | 2.25 | -1.55 | -0.060938 | -3 | 5.8647 |
| 2.9 | 2.2813 | -1.55 | -0.060938 | -3 | 4.6147 |
| 3 | 2.2813 | -1.55 | -0.060938 | -3 | 4.6147 |
| 3.1 | 2.2813 | -1.55 | -0.060938 | -0.5 | 7.7495 |
| 3.2 | 2.4063 | -3.425 | -0.060938 | -0.1875 | 9.312 |
| 3.3 | 2.4063 | -3.425 | -0.060938 | -0.1875 | 4.937 |
| 3.4 | 2.4063 | -3.1125 | -0.060938 | -0.1875 | 2.437 |
| 3.5 | 2.5313 | -2.9563 | -0.060938 | -0.10938 | 9.2241 |
| 4 | 2.5 | -4.25 | -1.225 | -0.1 | 5.8647 |
| 4.5 | 6.75 | -0.1875 | -0.2875 | -0.1 | 6.1772 |
| 5 | 5.75 | -0.5 | -0.2875 | -0.1 | 9.9272 |

Table 5.4: Table of PID's parameters that maximize the EVA for each turn away penalty factor for the fast food restaurant for scenario 1

- For a null turn away penalty factor, the controller ignores the term correspondent to the potential retention of the customers in the objective function, therefore focusing on the maximization of the NOPAT, as the CC is constant, behaving as an almost constant signal. This fact can be seen in figure 5.19 for the scenario 1, along with its correspondent reneging customers in figure 5.20.

| Turn away penalty factor | Optimal WIP | Optimal P | Optimal I | Optimal D | Optimal integrator initial condition |
|---|---|---|---|---|---|
| 0 | 0 | -2.6875 | -10.6 | -0.1 | 5.8647 |
| 0.5 | 1.25 | -17.375 | -5.2875 | -0.4125 | 8.3647 |
| 1 | 1.25 | -17.4141 | -5.2875 | -0.4125 | 5.7085 |
| 1.5 | 1.25 | -17.4141 | -6.5375 | -0.4125 | 5.7085 |
| 2 | 1.375 | -17.1016 | -1.5375 | -10.4125 | 5.7866 |
| 2.5 | 1.375 | -17.1016 | -1.5375 | -10.4125 | 10.7866 |
| 3 | 1.375 | -17.1016 | -0.2875 | -9.7875 | 2.0366 |
| 3.5 | 1.4375 | -18.8203 | -0.2875 | -9.7875 | 2.271 |
| 4 | 1.5 | -8 | -0.3 | -4 | 7.1147 |
| 4.5 | 1.5 | -8 | -0.3 | -4 | 6.4897 |
| 5 | 1.5 | -3 | -0.3 | -1.5 | 6.4897 |
| 5.5 | 1.5 | -2.8438 | -0.14375 | -1.5 | 7.1147 |
| 6 | 1.5 | -3 | -0.15 | -2 | 5.8647 |
| 6.1 | 1.5625 | -2.375 | -0.15 | -0.75 | 8.3647 |
| 6.2 | 1.5625 | -2.375 | -0.15 | -0.75 | 7.1147 |
| 6.3 | 1.5625 | -1.9063 | -0.15 | -0.125 | 3.3647 |
| 6.4 | 1.5625 | -1.9063 | -0.15 | -0.125 | 3.3647 |
| 6.5 | 1.5313 | -12.8438 | -0.14375 | -1.1875 | 4.5366 |
| 6.6 | 1.5625 | -1.3203 | -0.15 | -0.125 | 3.521 |
| 6.7 | 1.5938 | -1.3203 | -0.15 | -0.125 | 6.021 |
| 6.8 | 1.5313 | -8.5078 | -0.15 | -0.125 | 3.521 |
| 6.9 | 1.5313 | -18.4297 | -0.22812 | -0.125 | 3.521 |
| 7 | 1.5313 | -14.0938 | -0.14375 | -1.1875 | 9.5366 |
| 7.5 | 1.5313 | -14.0938 | -0.14375 | -1.1875 | 3.1304 |

Table 5.5: Table of PID's parameters that maximize the EVA for each turn away penalty factor for the fast food restaurant for scenario 2

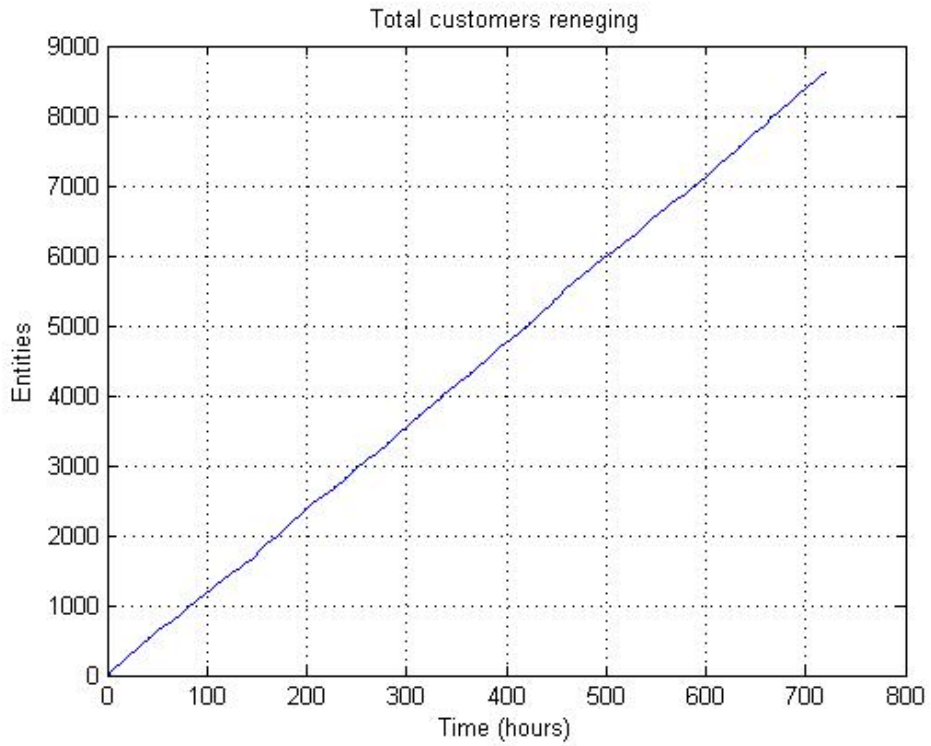


Figure 5.20: The reneging customers for the scenario 1 with a turn away penalty factor of 0

- During the beginning of the simulation (usually the first 24 hours), the model is still warming up. Thus, a strong variation of the price is observed until the model reaches a more steady condition, as depicted in figure 5.21 which represents the fast food example in scenario 1 with a turn away penalty factor of 0.5.

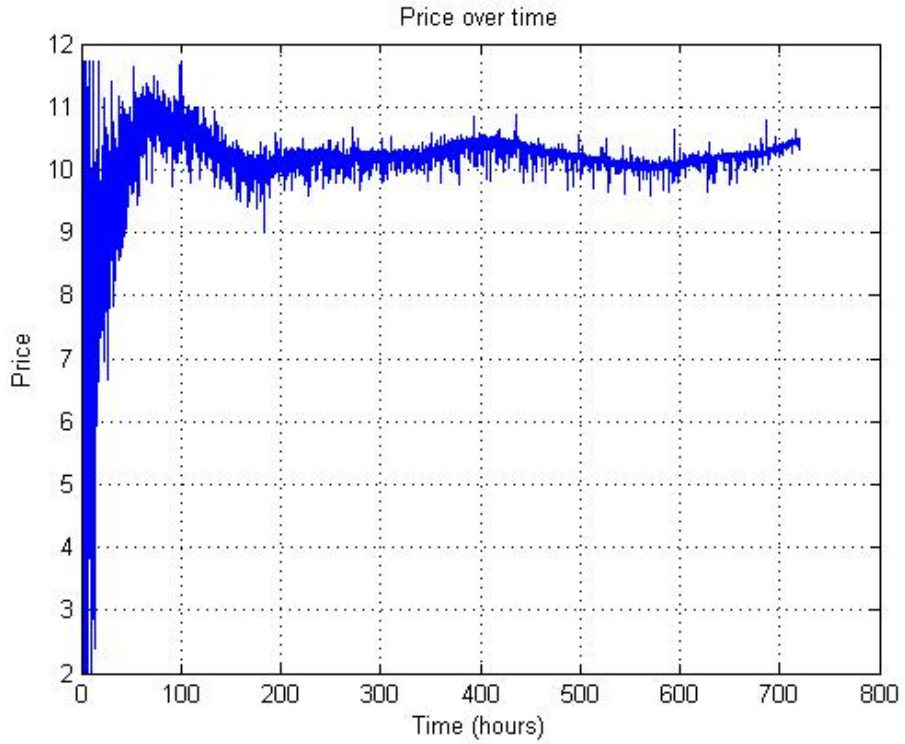


Figure 5.21: The price over time for the scenario 1 with a turn away penalty factor of 0.5

- For a small turn away penalty factor, the number of customers that waive the system does not have a strong influence in the EVA. This fact can be seen in figures 5.21, 5.22 and 5.23 for the scenario 1 and a turn away penalty factor of 0.5.

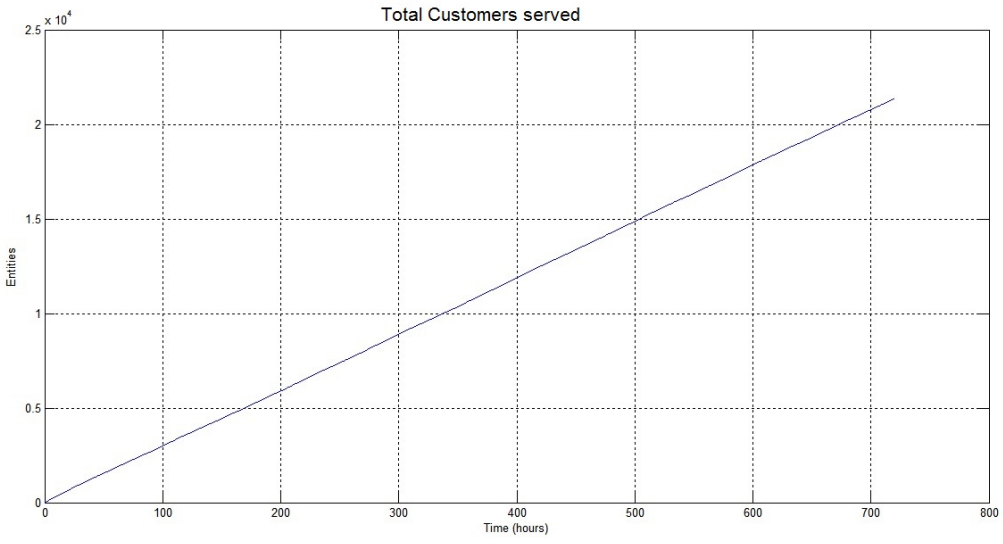


Figure 5.22: The served customers for the scenario 1 with a turn away penalty factor of 0.5

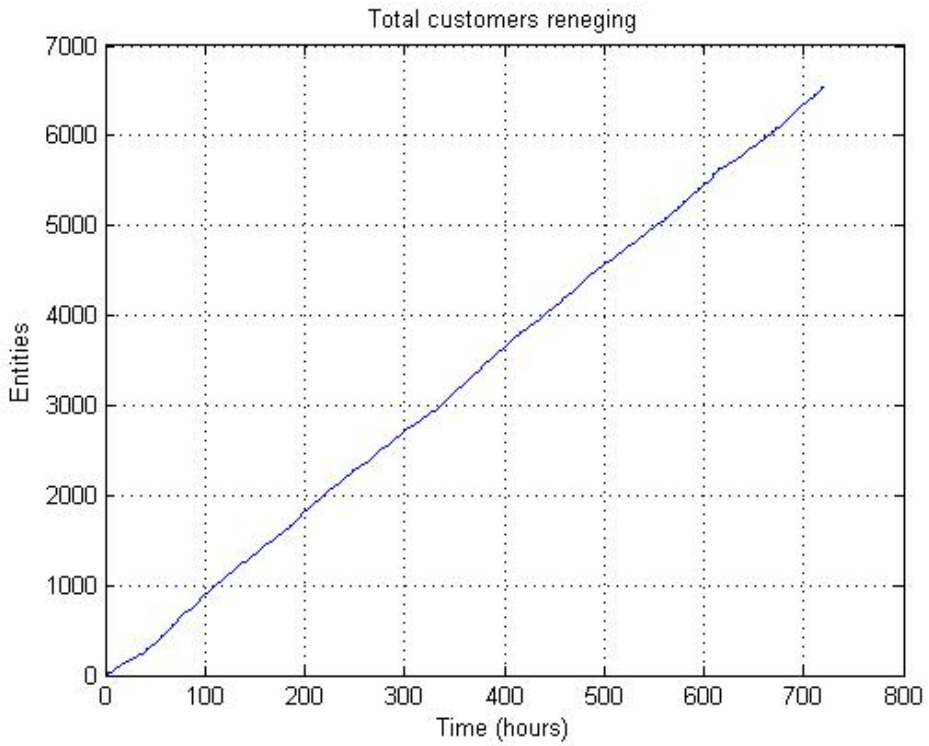


Figure 5.23: The reneging customers for the scenario 1 with a turn away penalty factor of 0.5

- The mean price chosen by the controller is inversely dependent on the turn away penalty factor. For a bigger turn away penalty factor, the negative influence of the lost of potential loyal customers increases in the evaluation of the objective function, thus the controller starts to reduces the price charge, and more customers enter the queue, instead of reneging. Figures 5.21 and 5.24 can be used for a comparison between two different price profiles for two different turn away penalty factor.

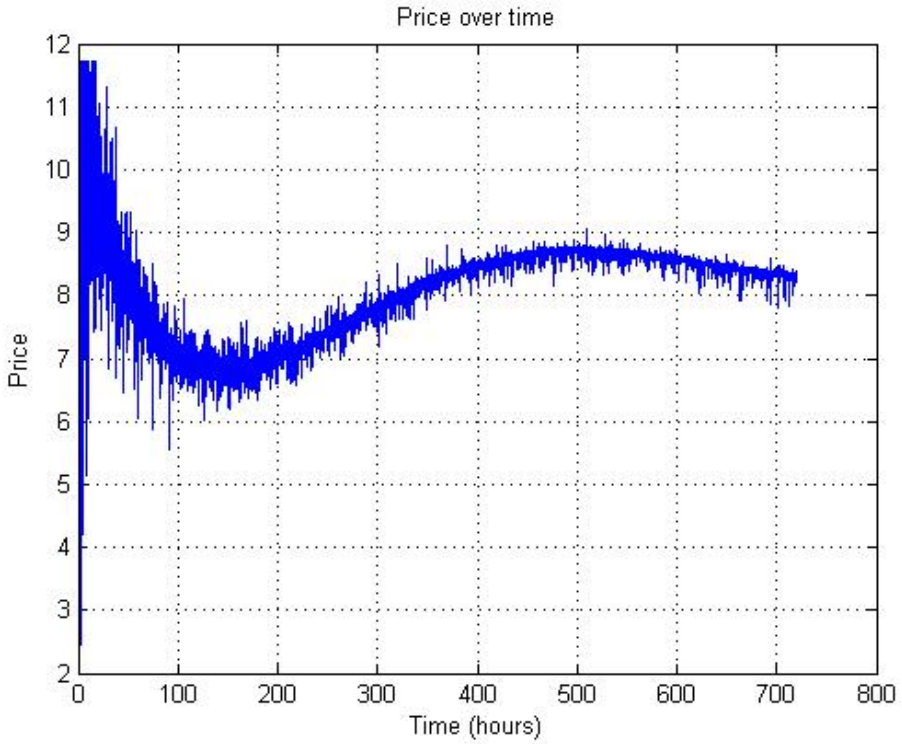


Figure 5.24: The price over time for the scenario 1 with a turn away penalty factor of 2.5

- As the process is subject to random influences, the values of the PID's parameters that yield the maximum EVA and the EVA itself for the same given assumptions vary slightly at each optimization. In table 5.6, the result of 10 sequential runs for the scenario 1 with Turn Away Penalty Factor 0.5 and using the PID's parameter that maximized the EVA for this condition from table 5.4, in which is possible to notice the variation in the result between each run of the fast food restaurant example.

- As a consistency check, a fast food restaurant example without a PID Controller, using a constant price determined by a PS algorithm that maximizes the EVA was developed to compare the results with the fast food example with the PID controller, running scenario 1 for a turn away factor of 2.5. From table 5.2, for these given conditions, the EVA was approximately 10000 for the PID controller and returned a negative optimized EVA for all four trials for the constant price condition, in table 5.7.

- For a big turn away penalty factor, the best case for a system is to set the price for 2, serving as many customers, as the price is equal to the unit cost. In this case, the best

| Run | EVA |
|---|---|
| 1 | 83975 |
| 2 | 83321 |
| 3 | 84316 |
| 4 | 83901 |
| 5 | 85627 |
| 6 | 85305 |
| 7 | 84595 |
| 8 | 83909 |
| 9 | 83468 |
| 10 | 85340 |

Table 5.6: 10 sequential runs for the scenario 1 with Turn Away Penalty Factor 0.5 and using the PID's parameter that maximized the EVA for this condition

| Price | EVA |
|---|---|
| 5.234 | -1483 |
| 4.688 | -1727 |
| 5.317 | -265 |
| 5.566 | -2107 |

Table 5.7: Table of resultant constant prices that maximized the EVA for scenario 1 for the model without a PID controller and a turn away penalty factor of 2.5

EVA possible is, calculated from equation 5.1:

$$EVA = -FixedCost(1 - TaxRate) - (ExpectedReturn * CapitalInvested) \quad (5.1)$$

For scenario 1, a big turn away penalty factor can be considered as 5 or above. This behaviour can be seen for the price and reneging customers in figures 5.25 and 5.26, respectively:

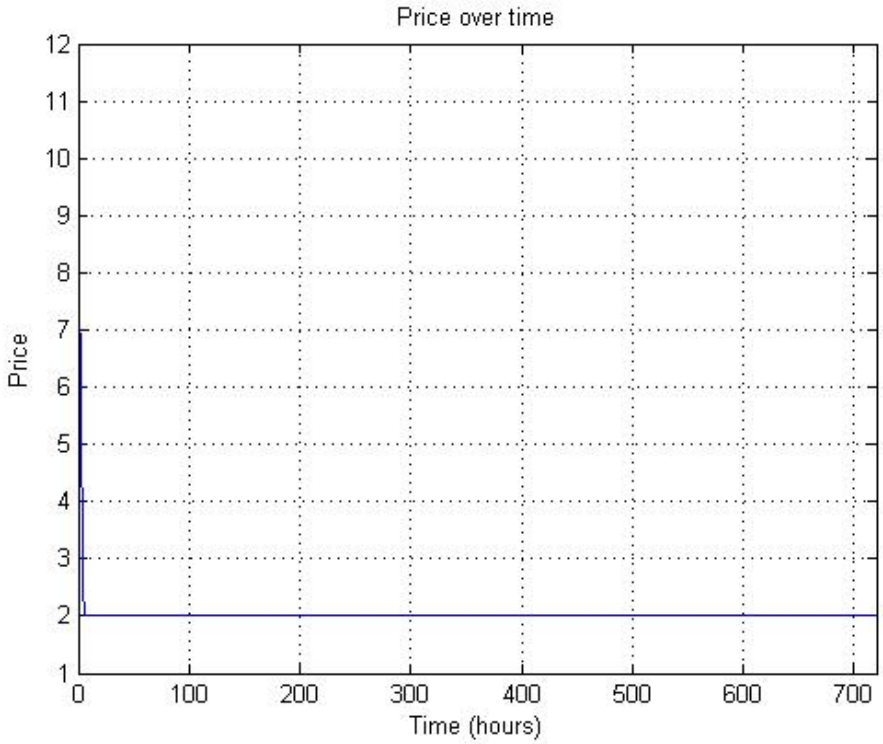


Figure 5.25: The price over time for the scenario 1 with a turn away penalty factor of 5

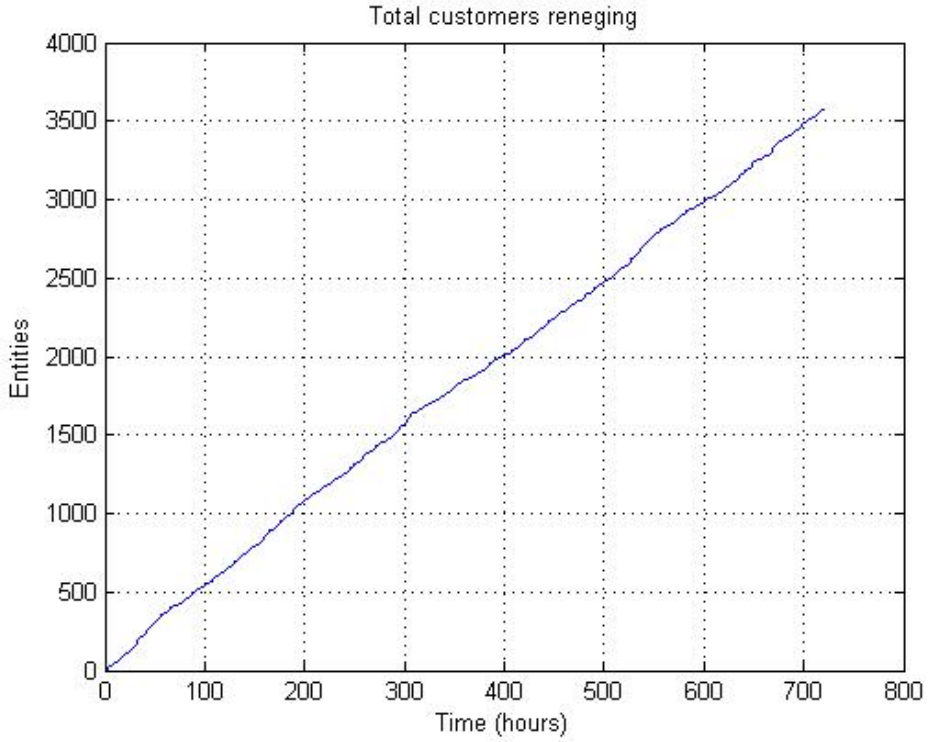


Figure 5.26: The reneging customers for the scenario 1 with a turn away penalty factor of 5

The feasibility of the restaurant operation is highly dependent on the turn away penalty factor chosen from the study presented above. Thus, the choice of this parameter depends on the strategy that the manager wants to implement.

A recently opened restaurant's manager or a manager that is targeting an increase in the restaurant's market share, for example, could set a high turn away penalty factor, in order to be able to serve more customers, which could potentially lead to a larger loyal clientèle. As already mentioned in chapter 4, the benefits of a loyal clientèle is justified in [31]. However, a high turn away penalty factor would reduce the EVA in the short run.

It is evident then that there is a trade-off between a bigger EVA in the short run that could possibly decrease over time and a small EVA in the short run with the possibility of retaining more customer which could lead to an even bigger EVA in the future. There is no free lunch.

For the given set of parameters, a fast food restaurant without shared resources (scenario 2) was the best option, as this condition yields a greater EVA from tables 5.3 and 5.2, nevertheless one might find infeasible to invest the amount necessary to implement a restaurant without shared resources or would prefer to diversify the investment into other options. Moreover, a fast food restaurant with a PID controller for price controlling yields better results, i.e., a bigger EVA, than one with a constant price charged that maximizes the EVA for this condition.

## 5.3 Sensitivity Analysis

A Sensitivity analysis investigates the influence of the time parameters, i.e., service time and interarrival time, for both scenarios presented in section 5.2. The service time is the same for all the servers in the model, varying together in the sensitivity analysis.

From table 5.8, which presents the results for scenario 1 with the turn away penalty factor set as 2, the EVA increased for a bigger IIT, which indicates that, for the default scenario 1, the restaurant was not able to satisfy as many customers tolerances as for the model running with a bigger IIT. This increase in the EVA for this altered condition can be explained as the influence of a bigger excess capacity in the model. When reducing the IIT, the number of extra customers

| IIT | Service Time | EVA |
|---|---|---|
| 1.5385 | 1 | 26159.0837 |
| 1.6923 | 1 | 34296.1625 |
| 1.3846 | 1 | -11991.7402 |
| 1.5385 | 1.1 | 11396.9024 |
| 1.5385 | 0.9 | 41959.692 |

Table 5.8: Table for the sensitivity analysis for scenario 1 and turn away penalty factor of 2.0

is unmanageable for the controller, evidenced by the EVA becoming negative, failing to take advantage of this potentially more profitable situation, by serving the extra incoming customers and, thus, generating a bigger EVA.

The conclusions for this analysis in relation to the service time are quite straight forward, in which a smaller service time, led to more customers being served and therefore a bigger EVA and a bigger service time, led to a reduction in the EVA. A bigger service time, on the other hand, led to a smaller EVA, as less customers could be served in the same time interval.

| IIT | Service Time | EVA |
|---|---|---|
| 1.5385 | 1 | 55780.7514 |
| 1.6923 | 1 | 57472.0993 |
| 1.3846 | 1 | 50838.7001 |
| 1.5385 | 1.1 | 47354.5575 |
| 1.5385 | 0.9 | 66347.9377 |

Table 5.9: Table for the sensitivity analysis for scenario 2 and turn away penalty factor of 2.0

As scenario 2 has a bigger excess capacity, as the model does not have to share resources in this condition, the results presented in table 5.9, are not as sensitive as in scenario 1. However, the same direct relation between the EVA and the IIT is present in a smaller scale. Again, a smaller service time led to a bigger EVA and a bigger service time led to a smaller EVA.

This study showed some operating conditions for the fast food restaurant example with and without shared resources. In the manager position, in the case of a bigger IIT, it would be worth considering an increase in the restaurant's capacity, by hiring more employees and adding extra resources and servers, including the cost for this alteration in the restaurants configuration, as it could allow the service of more customers, which could lead to a bigger EVA. Another possibility to the manager is to reduce the service time, by improving the machinery available or replacing by a newer model or even training more the employees, including the cost for such

improvement.

After the investigation and selection of the possible approaches for improving the fast food restaurant, the manager should modify the model, calculating a new expected EVA, that can be compared with the results of the implementation.

# Chapter 6

# Conclusions

The purposes of this study was, initially, to investigate possible software platforms to facilitate the development and control of models with shared resources, allowing the user to create and investigate the model to better understand the dynamics involved in this process. The chosen platform was then used to build a model, in this case, of the fast food restaurant example, which involved two shared resources between two parallel sets of sequential servers, with random IIT in the queue and service times.

In the second stage of this project, financial goals, namely, profit and EVA, were included in the fast food restaurant model, as well as a price controller, using an intrinsic measurement of the process evaluated (average WIP) and a PID control law, the parameters of which were found through pattern search optimization that maximized the EVA for the scenario analysed. The whole methodology to perform the financial optimization of the model was defined in such a way that it could be reproduced for other models.

The resulting fast food restaurant model (in SimEvents/Simulink) can be regarded as a "flight simulator" for fast food restaurant's managers, in which they can create a custom instance of the model for their own settings and gather simulation results for this set. This is particularly interesting in order to carry out virtual experiments on improvements to a real-life restaurant or even to decide on the feasibility of a new enterprise. It is also possible to validate the hypothesis made for the model by comparing the results of the execution to real-life fast food restaurant data.

During the course of this work, it became evident that the chosen platform for modelling and control of models with Shared Resources still needs some refinements, specially in the

integration between the model and the control part. This is because the Simulink environment uses a different representation of the signals in each of the parts of the program: the model has event-based signals, which is the default type of signals for the SimEvents library, while the controller has time-based signals, which is the default for all other Simulink Libraries. Another difficulty using this approach is replicating the models and parameters, as the programs is not in a text based language, which could be just copied and pasted from the text into the software and executed.

Despite this, this project showed that it was possible to create a basic framework to perform Process modelling and control with adjustable parameters in SimEvents. Of course, the developed model is still quite basic and does not represent all the necessary complexity of a real-life model. On attempting to include other real-life aspects in a more complex model that would be presented as part of this work, it became clear that the scalability of the models using this approach could become a significant problem.

## 6.1 Future work

Future work could explore other systems in the field of shared resources, which has a variety of possible other examples that could be studied in detail, such as the models of the traffic light system and the logistic system presented in chapter 3 of this work.

Another possibility would be to refine the modified fast food restaurant model proposed in this work to include other phenomena to the system, such as:

- Allow the customers to leave the waiting queue in case the service rate does not correspond to the customers tolerance;
- Allowing customers to choose their meal;
- Add more cashiers and include a separate queue for each of the cashiers;
- Increase the set of control variables, allowing the controller, which represents the

establishment's manager, to set opening and closing hours, hire and fire employees, specifying behaviours and learning curves for each employee;

- Specify other optimization goals for this process, which could involve minimising the price variation or the number of customers exiting the restaurant without ordering;
- Alter the IIT of customers to represent peak and off-peak hours;
- Sensitivity analysis of all model parameters.

Finally the outcomes of this work could be used in the future as a base for modelling and control of a real-life fast food restaurant, using the methods presented in chapter 4 to drive the production and management decisions of the store, using specific methods to acquire real-life parameters to be used in the models proposed.

# Bibliography


[1] C. Audet and J. E. Dennis Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889 – 903, 2003.

[2] R. Basak, A. Sanyal, S. Kumar Nath, and R. Goswami. Comparative view of genetic algoithm and pattern search for global optimization. *International Journal Of Engineering And Science*, 3:09–12, 2013.

[3] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Events Systems*. Springer, 2nd edition, 2008.

[4] A. M. Castro and T. S. Calmon. Study of inventory control strategies with genetic algorith optimization. Technical report, Federal University of Rio de Janeiro, 2013. Undergraduate project - `http://monografias.poli.ufrj.br/monografias/monopoli10005877.pdf`.

[5] C. Chou and H. Liu. Simulation study on the queuing system in a fast-food restaurant. *Journal of Restaurant & Foodservice Marketing*, 3(2):23–36, 1999.

[6] M.I. Clune, P.J. Mosterman, and C.G. Cassandras. Discrete event and hybrid system simulation with simevents. In *Discrete Event Systems, 2006 8th International Workshop on*, pages 386–387, 2006.

[7] F. Danuso. Semola. `http://www.dpvta.uniud.it/danuso/docs/Semola/homep.htm`. [Online; accessed 26-02-2015].

[8] F. Danuso. Semola 5 - revision history. `http://www.dpvta.uniud.it/danuso/docs/Semola/general/history.php`. [Online; accessed 26-02-2015].

[9] F. Danuso and A. Rocca. Semola: A simple and easy modelling language. *Ecological Modelling*, 285:54 – 77, 2014.

[10] R. C. Dorf and R. H. Bishop. *Modern Control Systems*. Pearson Prentice Hall, 11th edition, 2008.

[11] K. Farahmand, A. Francisco, and G. Martinez. Simulation and animation of the operation of a fast food restaurant. In *Simulation Conference, 1996. Proceedings. Winter*, pages 1264–1271, 1996.

[12] T. N. Ferreira and D. M. Martins. Study of inventory management using a predictive control strategy. Technical report, Federal University of Rio de Janeiro, 2015. Undergraduate project - to be published in `http://monografias.poli.ufrj.br/`.

[13] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, 4th edition, 2002.

[14] G. F. Franklin, J. D. Powell, and M. L. Workman. *Discrete-Time Control Systems*. Addison-Wesley, 3rd edition, 1998.

[15] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Boston, MA, USA, 1st edition, 1989.

[16] B. Hannon and B. McGarvey. *Dynamic Modeling For Business Management: An Introduction*. Springer, New York, 2004.

[17] The MathWorks Inc. Optimization of non-smooth objective function. `http://www.mathworks.com/examples/global-optimization/381-optimization-of-non-smooth-objective-function`. [Online; accessed 03-03-2015].

[18] The MathWorks Inc. Subsystem, atomic subsystem, nonvirtual subsystem, codereuse subsystem. `http://www.mathworks.com/help/simulink/slref/atomicsubsystem.html`. [Online; accessed 24-02-2015].

[19] The MathWorks Inc. Writing objective functions. `http://www.mathworks.com/help/optim/ug/writing-objective-functions.html`. [Online; accessed 03-03-2015].

[20] Q. Iqbal, L. E. Whitman, and D. Malzahn. Reducing customer wait time at a fast food restaurant on campus. *Journal of Foodservice Business Research*, 15:319–334, 2012.

[21] L. Leemis. Standard uniform distribution. `http://www.math.wm.edu/~leemis/chart/UDR/PDFs/Standarduniform.pdf`. [Online; accessed 25-02-2015].

[22] J. D.C. Little and S. C. Graves. Little's law. `http://web.mit.edu/sgraves/www/papers/Little%27s%20Law-Published.pdf`. [Online; accessed 25-02-2015].

[23] Automation Supplies Ltd. Type120 heavy duty belt conveyors. `http://www.automation-supplies.com/Type120-Belt.html`. [Online; accessed 26-02-2015].

[24] N. Matloff. Introduction to discrete-event simulation and the simpy language. `http://web.cs.ucdavis.edu/~matloff/matloff/public_html/SimCourse/PLN/DESimIntro.pdf`, feb 2008. [Online; accessed 26-02-2015].

[25] F. V. Monteiro. Comparative study of inventory management policies designed by control techniques. Technical report, Federal University of Rio de Janeiro, 2014. Undergraduate project - `http://monografias.poli.ufrj.br/monografias/monopoli10011694.pdf`.

[26] M. J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.

[27] National Institute of Standards and Technology. Exponential distribution. `http://www.itl.nist.gov/div898/handbook/eda/section3/eda3667.htm`. [Online; accessed 25-02-2015].

[28] K. Ogata. *Discrete-Time Control Systems*. Prentice Hall, 2nd edition, 1995.

[29] C. Parkan. Simulation of a fast-food operation where dissatisfied customers renege. *The Journal of the Operational Research Society*, 38(2):137–148, 1987.

[30] The Statistics Portal. Revenue of the united states fast food restaurant industry from 2002 to 2018 (in billion u.s. dollars). `http://www.statista.com/statistics/196614/revenue-of-the-us-fast-food-restaurant-industry-since-2002/`. [Online; accessed 01-03-2015].

[31] F. F. Reichheld and W. E. Sasser Jr. Zero defections: Quality comes to services. *Harvard Business Review*, 68(5):105–111, 1990.

[32] T. J. Richards and L. Mancino. Demand for food-away-from-home: a multiple-discrete–continuous extreme value model. *European Review of Agricultural Economics*, page jbt008, 2013.

[33] K. Sigman. Inverse transform method. `http://www.columbia.edu/~ks20/4404-Sigman/4404-Notes-ITM.pdf`, 2010. [Online; accessed 25-02-2015].

[34] Team SimPy. Welcome to simpy. `https://simpy.readthedocs.org/en/latest/`. [Online; accessed 26-02-2015].

[35] J. Sloman and A. Wride. *Economics*. Pearson Prentice Hall, 7th edition, 2009.

[36] J. D. Sterman. *Business Dynamics - Systems Thinking and Modeling for a Complex World*. McGraw-Hill, 2000.

[37] ISEE Systems. ithink: Systems thinking business. `http://www.iseesystems.com/softwares/Business/ithinkSoftware.aspx`. [Online; accessed 23-02-2015].

[38] G.D. Tsiotras and H. Badr. Modeling the blocking phenomenon of service facilities. *Mathematical and Computer Modelling*, 12(6):641 – 649, 1989.

[39] W. L. Winston. *Operations Research: Applications and Algorithms*. Cengage Learning, 4th edition, 2003.

# Appendices

# Appendix A

# Exponential Distribution

The exponential distribution is often used to describe the random variation presented in queueing systems, for modelling arrival and departure events. This usage is justified for two reasons: (i) it leads to simple closed-form equations and (ii) under certain conditions, this distribution is expected for real-life processes [16].

The probability density function $f(x)$ or $P(x)$ and the cumulative distribution function $F(x)$ for the exponential distribution is shown below:

$$f(x) = \frac{1}{\lambda} e^{\frac{-x}{\lambda}} \tag{A.1}$$

For $\lambda$ = 1, the probability density function for the exponential distribution is the following:

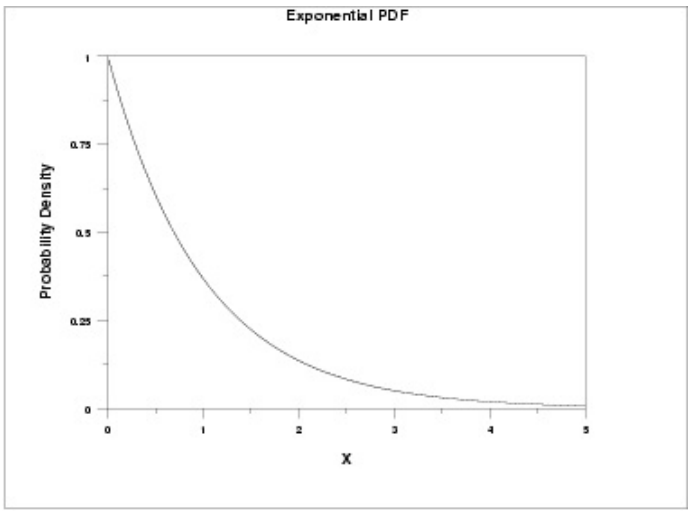


Figure A.1: The probability density function for $\lambda$ = 1. Figure source: [27]

$$F(x) = \begin{cases} 1 - e^{\frac{-x}{\lambda}}, & \text{for } x \geq 0, \\ 0 \text{ otherwise} \end{cases} \tag{A.2}$$

For $\lambda$ = 1, the cumulative distribution function for the exponential distribution is the following:

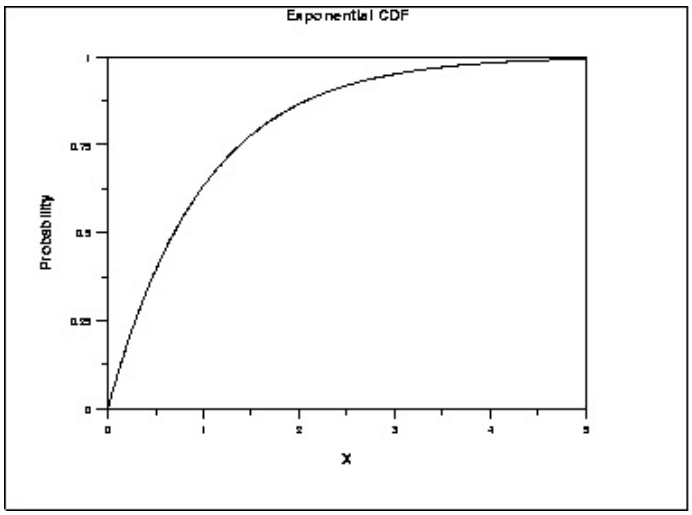


Figure A.2: The cumulative distribution function for $\lambda$ = 1. Figure source: [27]

For such distribution, the expected value of the variable $x$ known as $E[x]$ and the standard deviation, $\sigma^2$ are given as follows:

$$E[x] = \lambda \quad \text{and} \quad \sigma^2 = \lambda^2 \tag{A.3}$$

The reason for this choice of distribution to model a generator is its no-memory property. See [39] for a proof of the lemma below:

If $A$ has an exponential distribution, then for all non-negative values of $t$ and $h$,

$$P(A > t + h | A \geq t) = P(A > h) \tag{A.4}$$

A method for generating the exponential distribution is given below, utilizing the Inverse Transform Sampling method and a continuous uniform distribution $U$ over the interval (0,1), also know as the standard uniform distribution:

$$f(x) = \begin{cases} 1, & \text{for } 0 \leq x \leq 1 \\ 0, & \text{otherwise} \end{cases} \tag{A.5}$$

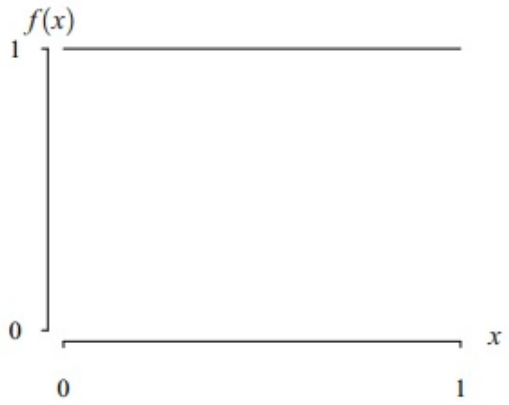


Figure A.3: The standard Uniform probability density function. Figure source: [21]

Solving the equation of the cumulative distribution function $F(x)$ for $x$, for $F(x)$ varying from 0 to 1 [33]:

$$y = F(x) = 1 - e^{\frac{-x}{\lambda}} \tag{A.6}$$

$$1 - y = e^{\frac{-x}{\lambda}} \tag{A.7}$$

$$ln(1 - y) = ln(e^{\frac{-x}{\lambda}}) \tag{A.8}$$

$$\frac{-x}{\lambda} = ln(1 - y) \tag{A.9}$$

$$x = -\lambda\, ln(1 - y) \tag{A.10}$$

$$x = F^{-1}(y) = -\lambda\, ln(1 - y) \tag{A.11}$$

Therefore,

$$exp = -\lambda\, ln(1 - U) \tag{A.12}$$

# Appendix B

# The iThink equations for figure 2.7

```
1
2  COMPLETED_WORK(t) = COMPLETED_WORK(t - dt) + (WORK_COMPLETING) * dt
3  INIT COMPLETED_WORK = 0
4  INFLOWS:
5  WORK_COMPLETING = CONVEYOR OUTFLOW
6      TRANSIT TIME = CYCLE_TIME
7  PROCESS(t) = PROCESS(t - dt) + (WORK_FLOWING_INTO_PROCESS
8                 - WORK_COMPLETING) * dt
9  INIT PROCESS = 0
10     TRANSIT TIME = varies
11     INFLOW LIMIT = INF
12     CAPACITY = INF
13 INFLOWS:
14 WORK_FLOWING_INTO_PROCESS = CONVEYOR OUTFLOW
15     TRANSIT TIME = IIT
16 OUTFLOWS:
17 WORK_COMPLETING = CONVEYOR OUTFLOW
18     TRANSIT TIME = CYCLE_TIME
19 SUPPLY_OF_WORK(t) = SUPPLY_OF_WORK(t - dt) +
20                (- FLOW_TO_WORK_CONTROL) * dt
21 INIT SUPPLY_OF_WORK = 2000
22 OUTFLOWS:
23 FLOW_TO_WORK_CONTROL = 1/DT
24 WORK_CONTROL(t) = WORK_CONTROL(t - dt) + (FLOW_TO_WORK_CONTROL
25                    - WORK_FLOWING_INTO_PROCESS) * dt
26 INIT WORK_CONTROL = 0
27     TRANSIT TIME = varies
28     INFLOW LIMIT = INF
29     CAPACITY = 1
30 INFLOWS:
31 FLOW_TO_WORK_CONTROL = 1/DT
32 OUTFLOWS:
33 WORK_FLOWING_INTO_PROCESS = CONVEYOR OUTFLOW
34     TRANSIT TIME = IIT
35 CYCLE_TIME = 100
36 IIT = 2.0
```

# Appendix C

# The SimPy example for figure 2.7

```
import random
import simpy
import numpy as np
import matplotlib.pyplot as plt

CT = 100
SIM_TIME = 1000
dt = 0.01
SUPPLY = np.zeros((SIM_TIME*int(1/dt),1))
WIP = np.zeros((SIM_TIME*int(1/dt),1))
COMPLETED = np.zeros((SIM_TIME*int(1/dt),1))
STEP = np.zeros((SIM_TIME*int(1/dt),1))
IIT = 1

class Process(object):

    def __init__(self, env, psize):
        self.env = env
        self.machine1 = simpy.Resource(env, psize)

    def step(self, item,time):
        yield self.env.timeout(time)


def item(env, name, p):

    print('%s enters the Process at %.2f.' % (name, env.now))
    with p.machine1.request() as request:
        yield request

        print('%s enters step 1 at %.2f.' % (name, env.now))

        for k in range (int(env.now*1/dt) ,
                        int(env.now*1/dt) + int(CT*1/dt)):
            if k < int(SIM_TIME*1/dt):
                WIP[k,0] += 1
                STEP[k,0] += 1
```

```
38
39          yield env.process(p.step(name,ct))
40
41          print('%s leaves the Process at %.2f.' % (name, env.now))
42
43          for k in range (int(env.now*1/dt) ,
44                          int(SIM_TIME*1/dt)):ss
45                  COMPLETED[k,0] += 1
46
47  def setup(env, psize, ct,supply):
48
49      process = Process(env, psize)
50      i = 0
51      SUPPLY[0,0] = 2000
52
53      tnprocess = IIT
54
55      for k in range (1, int(tnprocess*1/dt)):
56              SUPPLY[k,0] = SUPPLY[k-1,0]
57
58      while (SUPPLY[int(env.now*1/dt),0]>0):
59
60          yield env.timeout(tnprocess)
61
62          SUPPLY[int(env.now*1/dt),0] = SUPPLY[int(env.now*1/dt)-1,0] - 1
63
64          tnprocess = IIT
65
66          for k in range (int(env.now*1/dt) + 1,
67                          int(env.now*1/dt) + int(tnprocess*1/dt)):
68
69              if k < int(SIM_TIME*1/dt):
70                  SUPPLY[k,0] = SUPPLY[k-1,0]
71
72          i+= 1
73          env.process(item(env, 'Item %d' % i, process))
74
75  #RANDOM_SEED = 42
76  #random.seed(RANDOM_SEED)
77
78  # Create an environment and start the setup process
79  env = simpy.Environment()
80  env.process(setup(env, 100, CT,SUPPLY))
81
82  # Execute!
83  env.run(until=SIM_TIME)
84
85  t = np.zeros((SIM_TIME*int(1/dt),1))
86
87  for k in range (0, SIM_TIME*int(1/dt)):
88      t[k,0] = k*0.01
89
90  plt.clf()
91
92  plt.subplot(3, 2, 1)
```

```
93  plt.plot(t, SUPPLY)
94  plt.grid(1)
95  plt.ylabel('Supply')
96  plt.xlabel('time')
97
98  plt.subplot(3, 2, 2)
99  plt.plot(t, COMPLETED)
100 plt.grid(1)
101 plt.ylabel('Completed')
102 plt.xlabel('time')
103
104 plt.subplot(3, 2, 3)
105 plt.plot(t, WIP)
106 plt.ylabel('WIP')
107 plt.grid(1)
108 plt.xlabel('time')
109
110
111 plt.subplot(2, 2, 4)
112 plt.plot(t, STEP)
113 plt.ylabel('STEP')
114 plt.grid(1)
115 plt.xlabel('time')
116
117 plt.show()
118
119 sumWIP = 0
120
121 for k in range (100*int(1/dt), SIM_TIME*int(1/dt)):
122     sumWIP += WIP[k,0]
123 print "Average WIP"
124 print sumWIP/(SIM_TIME*(1/dt) - 100*(1/dt))
125 print "Average Thruput"
126 print sumWIP/(CT*(SIM_TIME*(1/dt) - 100*(1/dt)))
```

# Appendix D

# Time average function

The function in this file computes the average of a signal over time, present in the fast food restaurant example for the average WIP calculation.

```
function y = timeavg(u,t)
%TIMEAVG Compute time average of input signal U.
%   Y = TIMEAVG(U,T) computes the time average of U,
%   where T is the current simulation time.

% Declare variables that must retain values between iterations.
persistent running_weighted_sum last_u last_t;

% Initialize persistent variables in the first iteration.
if isempty(last_t)
   running_weighted_sum = 0;
   last_u = 0;
   last_t = 0;
end

% Update the persistent variables.
running_weighted_sum = running_weighted_sum + last_u*(t-last_t);
last_u = u;
last_t = t;

% Compute the outputs.
if t > 0
   y = running_weighted_sum/t;
else
   y = 0;
end
```