FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
INFORMATICS INSTITUTE
BACHELOR OF COMPUTER SCIENCE

GUILHERME ANDRIGHETTO TEIXEIRA

A Web Interface for Accessing Option Pricer Implementations Based on the Heston Model

Graduation Thesis

Dipl. Ing. Christian de Schryver
Advisor

Prof. Dr. Leandro Krug Wives
Coadvisor

Porto Alegre, October 2013

CIP – CATALOGING-IN-PUBLICATION

Guilherme Andrighetto Teixeira,

A Web Interface for Accessing Option Pricer Implementations Based on the Heston Model /

Guilherme Andrighetto Teixeira. – Porto Alegre: Graduação em Ciência da Computação da UFRGS, 2013.

52 f.: il.

Monograph – Federal University of Rio Grande do Sul. Bachelor of Computer Science, Porto Alegre, BR–RS, 2013. Advisor: Christian de Schryver; Coadvisor: Leandro Krug Wives.

1. Financial. 2. Options. 3. Pricer. 4. Web. 5. Xml. 6. Java. 7. Gwt. 8. User interface. 9. Socket. 10. Server. I. Schryver, Christian de. II. Wives, Leandro Krug. III. Título.

November 25, 2013
gateixeira@inf.ufrgs.br
http://inf.ufrgs.br/∼gateixeira

"Don’t put off for tomorrow what you can do today because if you enjoy it today, you can do it again tomorrow." -James A. Michener

ACKNOWLEDGMENTS

I would like to thank my family for the support and for teaching me everything I know about life. Thank you father for pushing me to this university. Thank you mother for always helping me out. Thank you brother for making me happy after playing FIFA as I always win. Thank you grandparents, uncles, aunts, cousins for being comprehensive when I was not there so often as I wish I could be.

Also, I would like to thank all my professors, from the school, high school and the three universities I have studied so far: UPF, UFRGS and TU Kaiserslautern. You all are part of this. Thank you very much, UFRGS in the context of the Informatics Institute for the excellence on the art of teaching in an environment that allows the students to be really prepared either for an academic career or for the biggest companies in the world.

I would like to thank Prof. Dr. Norbert Wehn for giving me the opportunity of working in his group at the University of Kaiserslautern. It made me grow professionally, personally and made me learn many things that can be achieved only with such experience abroad. Thanks to my advisor from Germany, Dipl. Ing. Christian de Schryver, who helped me in everything since my arrival in Kaiserslautern and was patient from the beginning, even when I was back in Brazil and writing this thesis. Thank you Prof. Dr. Leandro Wives for being my advisor at UFRGS and for making helpful corrections and contributions to this work.

Finally, I want to thank all my friends and the people I met in these years of bachelor. You all have been really important and I will never forget the time we spent together. I'm sorry for being far away while cramming for exams. I'm sure it was for good.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

TCP	Transmission Control Protocol
XML	Extensible Markup Language
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPGA	Field-programmable Gate Array
XHR	XML-Http Request
GWT	Google Web Toolkit
IDE	Integrated Development Environment
GUI	Graphical User Interface
RPC	Remote Procedure Call
AJAX	Asynchronous Javascript and XML
JDBC	Java Database Connectivity
API	Application Programming Interface
SQL	Structured Query Language
JNI	Java Native Interface
JMS	Java Message System
UDP	User Datagram Protocol
IP	Internet Protocol
MVC	Model-View-Controller

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This thesis shows the implementation of a flexible web-based framework to run option pricer algorithms over different architectures in order to compare speed and energy efficiency. With the increasing complexity of the algorithms and data analyzes for the financial market, a search for faster and more economical solutions has become necessary thus leading to the development of systems that fit these growing needs. Facing this context, this on-line demonstrator has been developed with the intention of allowing fair comparisons regarding to runtime and energy consumption by integrating a FPGA-based hardware accelerator for multi-level Monte Carlo simulations that is proven to be more efficient than other common solutions with respect to both attributes. In order to make the comparisons available, we present a web user interface working together with a Java back-end that implements a multi-thread job management system, which opens a TCP socket to communicate with a C++ application over the network. It uses XML as an information pattern with the purpose of providing data compatibility between the different programming languages, which are Java and C++. Although this approach is always challenging by the need to exchange information between systems based on either different languages or architectures, our solution allows everyone worldwide to trigger computations and retrieve results, that is, giving the chance to demonstrate the efficiency of the hardware accelerator. This solution has been developed intending to provide stable references to the public for the domain of financial mathematics.

Keywords: Financial, options, pricer, web, xml, java, gwt, user interface, socket, server.

RESUMO

O presente Trabalho de Conclusão de Curso mostra a implementação de um framework web para a execução de algoritmos de precificação de opções financeiras em diferentes arquiteturas, a fim de comparar resultados no que diz respeito a velocidade de execução e eficiência energética. Com o aumento da complexidade dos algoritmos e análise de dados voltados ao mercado financeiro, uma busca por soluções mais rápidas e econômicas tornou-se indispensável, levando ao desenvolvimento de sistemas que se encaixam nessas necessidades crescentes. Frente a esse contexto, essa interface on-line foi desenvolvida com o intuito de permitir comparações justas quanto ao tempo de execução e consumo de energia, integrando um acelerador de hardware baseado em FPGA para simulações multinível de Monte Carlo, comprovadamente mais eficiente para ambos atributos que outras soluções usuais. Para que as comparações sejam possiveis, apresentamos uma interface para a internet integrada com um servidor multi-thread que implementa um sistema de gerência de tarefas, o qual conecta-se com uma aplicação C++ através de conexões de rede pelo protocolo TCP. O sistema usa XML como padrão para troca de informações, objetivando fornecer compatibilidade de dados entre as diferentes linguagens de programação, sendo elas Java e C++ . Embora tal abordagem seja sempre desafiadora perante a necessidade de troca de informações entre sistemas baseados em linguagens ou arquiteturas diferentes, nossa solução permite a qualquer pessoa no contexto global acionar as simulações e obter resultados, ou seja, possibilitando a demonstração da efetividade do acelerador de hardware. A presente solução foi desenvolvida intentando prover referências estáveis para o público no domínio de Matemática Financeira.

Palavras-chave: Financeiro, opcoes, precificador, web, xml, java, gwt, interface do usuario, socket, servidor.

1 INTRODUCTION

Nowadays, because of the increasing number of financial markets, with their stock prices that vary in the time of milliseconds, there was a growing need of approaches for behavior analysis of the different assets. Consequently, a consistent need for fast processing power and response appeared, ending up consuming too much energy. In addition, a major challenge is the complexity of existing pricing models of financial products that use approximation methods such as Monte Carlo simulations, for instance.

Any system with such a complexity level leads to a trade-off between the need for high computational power but increasingly limited resources of energy. Therefore, the Microelectronic System Design Research Group of the University Of Kaiserslautern has developed the first FPGA based accelerator for option pricing based on a Model known as Heston Model (Schryver et al. 2011) with advanced Monte Carlo simulations aimed at the European market of options. The paper (Schryver et al. 2011) has proven that this approach, compared with an 8-core Intel Xeon Server running at 3.07GHz, saves up to 89% power and provides around twice the speed. Compared to a GPU Nvidia Tesla C2050, the solution consumes about 40% less energy.

Although achieving an enviable performance both with respect to the processing speed of these algorithms and the energy consumption, there is the difficult of running on various architectures, largely due to the fact that there are numerous decimal parameters (to be explained), which comes to complicate the manual setting of the simulations. Moreover, there was not a tool to unify the simulations in only one interface, making it harder to compare the results from the different analyses.

In this context, this thesis aims to present a solution of a web-based framework for integrating financial market simulations, selecting different sets of parameters for option pricing and comparing results such as the runtime and efficiency. Through this interface the user can easily select the desired architectures as well as seeking results previously calculated. Besides, by placing it on the web, it shall provide external access, facilitating the evidence of the achieved efficiency with the presented solution for that purpose.

1.1 Motivation

The application that contains the option pricer is being developed by the group in C++, which requires compatibility with different architectures where multiple simulations will be performed at once. Facing this context, the web interface needs multi-user support, providing compatibility and communication without hanging the system. This solution

ends up as a challenge due to the diversity of architectures to be run at, such as CPU, GPU and FPGA, the complexity of algorithms that may take weeks to perform as well as the need for a quick and easy way to check the results, presenting them in an intuitive way with ease of understanding. While the infrastructure of the architectures is not available to connect with this web-based framework, a C++ library called "LibHeston" that runs the simulations locally is used in order to test this implementation and to prove that the system is fully able to create the environment for the simulations.

Also, due to server support issues and use limitation, it was decided that the interface would be implemented using the Google Web Toolkit, an open-source tool for web programming in Java that will be presented later. Hence, once the interface has been developed in a programming language that differs from the one used in the destination application, there is the communication challenge (i.e., interoperability) and data transfer between these components in both directions, from the user interface to the server which communicates with the C++ application of simulations and vice-versa. A TCP socket has been implemented to allow this communication, being constantly open and ready to receive new parameters as well as to return results of simulations with an indefinite ending. This system relies on synchronization of sending data to new simulations and receiving results, that is, it is necessary to have independence between executions, so a multi-thread system was designed to handle that amount of requests from both sides of the project.

For this transfer to be performed and the data to be understood we have decided to use XML as a standard for implementing the sending and receiving of the set of parameters and results.

We have introduced a comprehensive verification stage for the data inputs, preventing the system to run new simulations if the data are incorrect.

As it can be seen in the above description, this thesis presents a project that involves several computational concepts, encompassing different areas which ends up increasing the complexity and challenge, involving a web server, the communication between different languages and the need for large data exchanges, controlled by threads.

1.2 Structure

In this document, we first present the basic concepts regarding the algorithms to be executed from the web interface. Then, we show the concepts regarding to the system requirements. The user interface itself will be presented afterwards, followed by the back-end of the web part, embracing databases and data serialization. After, the defined data pattern will be shown, as the tools used in order to make it possible. By the end of the thesis, the communication between the systems will be presented. After, its corresponding simulations as well as the associated results and the way the system treats them. Finally, a section regarding final remarks and future work is included in order to conclude the thesis and to stress what remains to be done until we are able to have the system running openly on the internet.

2 FINANCIAL MATHEMATICS

Currently, the financial market is buzzing with listed companies trading stocks day by day, varying their capital and influencing investors around the world. Moreover, speculations are part of the business, involving the creation of various ways of manipulating assets between the involved parties. In this scenario, it is increasingly common to apply heavy, complex and long lasting mathematical calculations making it possible to draw predictions concerning assets and leading to the fundamental importance of financial mathematics, an area that was not so common in the near past. These calculations eventually consume a lot of energy and time due to the large amount of data that they have to treat. More than ever, these two factors are sought to be reduced front of the competitive global world. As previously mentioned, this project is based on the integration with the solution developed by the Group of the Microelectronics of the University of Kaiserslautern for the presented scenario and, for a better understanding of the general context, we will glimpse on some basic concepts of financial market in the following items.

2.1 Options Market

The options market is where financial instruments called options are traded. These options confer the right - not obligation - to buy or sell a particular amount of assets, which can be stock, bond or any good by a predetermined value while the other party involved, it being seller or buyer, is obliged to complete the transaction (Stock Option Basics).

The options are mainly characterized by its predetermined price, that is, the exercise price of that option in case the option is exercised. Also linked to the price, there is the date of the exercise, a.k.a "Time to Maturity". Given the different market configurations, that day can be the deadline for exercising the option or the only possible date to make it happen.

2.1.1 Option Types

As respect to the exercise date, the options can be either American or European. The differences are as follows:

- **American Option:** an American option provides the right that can be exercised at any time until the deadline.
- **European Option:** an European option, however, provides the right to exercise

only on the pre-specified date.

As the pricer library has been developed with focus on the European market, this project considers only European options, although it would be able to handle American options without any changes.

2.1.2 Barriers

In addition to the above definitions, it is important to characterize options with barriers. In this case, barriers can be added above and below the fixed price, resulting in new characteristics that are as follows (Cheng 2003; Educating the world about finance):

- **Knock-out:** A knock-out barrier defines that, if at any time the value of the option reaches the barrier, then the rights and obligations related to this option are not valid anymore.
- **Knock-in:** In contrast, the knock-in barrier defines that, if at any time the value of the option reaches the barrier, then the right comes into existence but not characterized as an obligation.

For these options to receive an acceptable and accurate value, according to the current state and prognosis of the market, various pricing methods were developed and the Black-Scholes model is the most famous of them (Andersen 2007). Given the library which this project is communicating with, the chosen method is the Heston Model, a variation of the Black-Scholes that will be presented below.

2.2 Heston Model

The Heston Model is a mathematical model to price options written by Steven Heston (Heston 1993) and it describes the evolution of the volatility in an asset, which is nothing more than a version to solve a shortcoming of the Black-Scholes model.

The following formula defines the model, where "S" is the option price to be determined.

$$dS_t = \mu S_t \cdot dt + \sqrt{v_t} \cdot S_t \cdot dW_t^s \tag{2.1}$$

In addition, the value of "V" is the volatility of the market and provided by:

$$dv_t = \kappa_t \cdot (\Theta_t - v_t) \cdot dt + \xi_t \cdot \sqrt{v_t} \cdot dW_t^v \tag{2.2}$$

The parameters for the above definitions are (according to their names in Greek):

Θ : the long-term volatility drift.

κ : the speed of reversion.

ξ : the volatility of volatility.

μ : the rate of return of an asset.

Besides these parameters, some others are needed but it is unnecessary to give deep explanations on the context of this thesis.

All concepts, the ones presented in this section and the ones needed to define the simulation environment will be shown and stated in the following chapter.

These mathematical equations are applied together with Monte Carlo methods.

2.3 Monte Carlo Simulations

Monte Carlo simulation is a computerized mathematical technique based on the generation of suitable random numbers that allow risk measurement in quantitative analysis and decision making. The method relies on repeated randomized samples to obtain numerical approximations and it is useful for obtaining numerical solutions to problems which are too complicated to solve analytically. The simulation is named after the city in Monaco, where multiple Gambling Games exhibit random behavior (Heinrich 2001).

Monte Carlo methods are used to calculate option prices based on random number generation (Schryver et al. 2010; Schryver et al. 2011; 9; Heinrich 2001).

The algorithm implementation of the method works by applying thousands of possible random paths to price an asset and then, it iteratively calculates the associated value - using the Heston Model - for the option in each one of the paths. These values are evaluated and the average will be the final price of the option.

It is important to notice that the efficiency of the Monte Carlo method relies on the generation of the random numbers. These generators are also included in the initial project as seen in (Schryver et al. 2010).

Some terms of the method, such interest rate and correlation will be introduced later.

3 PROJECT

This chapter focuses on some concepts of Software Engineering and modeling that the present project relies on to make an easier implementation and a better understanding of the code. Here, we first discuss the requirements and later the base architecture for the implementation as well as the structure of the database.

3.1 Requirements

According to Sommerville (2011), the requirements for a system are the descriptions of what the system should do - the services that it provides and the constraints on its operation. The requirements are a simple reflection of what the customers expect from the functionalities of the system and the process of finding out, documenting and checking is called Requirements Engineering. There are differences between the requirement made by the customer and the system specification and that is why some problems may arise during the process as a result of a fail while making a separation between the different levels of descriptions. Because of this, it is necessary to have a good overview before the implementation starts.

The Requirements can be divided into two different groups: functional and non-functional requirements.

3.1.1 Functional Requirements

Functional Requirements are statements of services the system should provide, that is, its behavior in some situations and it may explicitly states what the system should (or should not) do (Sommerville 2011, p. 84).

The functional requirements depend on the type of software being developed, its users and the considered environment when writing the requirements. They can be either described in an abstract way to be easily understood by users or with more details, thus being easier for developers. Also, the requirements have to be carefully defined as inaccuracies may lead to many software engineering problems, causing delays on the delivery and increasing costs.

The present project can be defined by its functional requirements based on the following statements:

1. Each user has to be identified by a combination of the name, e-mail address and an

unique identification number that relates to this tuple.

2. Each simulation has to be performed to calculate the option price based on the given parameters.
3. The system shall save the parameters and the results to be retrieved later.
4. The system has to communicate with a library in C++ to allow multiple simulations.
5. As the simulations can take a while, the system needs to inform the user when it is over so the results can be retrieved at anytime.
6. Before storing the results, the system shall calculate the values for energy consumption based on the power of the system and the runtime.
7. The user shall check the results by searching the simulation identified by its unique identification number.

3.1.2 Non-functional Requirements

Non-functional requirements are those which are not concerned with the functionalities the system provides to its users. They are related to properties such as consistency, correctness, reliability and availability.

These requirements can be seen as more critical than the functional's as when a system fails the user can not make it useful, meanwhile a system that does not match completely his needs can be used in some situations anyway.

In what regards to implementing, non-nunctional requirements are harder as it is not developed in a method only and might be spread all over the code. Again, according to Sommerville (2011), there are two reasons for this:

- Non-functional requirements may affect the whole system instead of a single module.
- A single non-functional requirement may create an amount of functional requirements to implement it.

The implementation of this project cares about some non-functional requirements. As follows:

1. The framework has to present an intuitive user interface for data input, thus avoiding mistakes.
2. The system has to keep data consistency even after a misunderstood input value.
3. As a flexible framwork, the system shall be easily managed in order to allow future changes.
4. In the sense of a web system, it has to deal with multiple connections without hanging up the server.
5. The system shall be able to run multiple simulations at the same time.

6. As the back-end has to exchange data between two different programming languages, it is necessary to create a data pattern to be readable by both.

7. This system has to be easily combined with the application that forwards data to the architectures.

8. The system shall keep the database consistent.

9. An intuitive way to compare results shall be presented to the user.

3.2 Use Cases

Use cases, as defined by Sommerville (2011, p 106), are a fundamental feature of the modeling language. Among the functionalities of this model, it identifies the actors involved in the action, who may be human or other systems, labels what happens as its consequences and may describe the interaction with the system. Use cases can be either graphical or textual but both shall give a high level understanding of the scenario. In this context, as there are only few situations where the user interacts with the system a textual use case can give a detailed description. Also, the use cases may help to understand some of the system requirements.

The tables 3.1 and 3.2 show we built two different use cases that follow the actions of the user based on the functional requirements. The first one describes the first use of the system when the user inputs values and presses the button to start a new simulation while the second one explains the process of fetching results when a simulation is over.

Use Case 01	Starting new simulation
Author:	User
Precondition:	-
Normal Flow:	1. User access the page and inputs his name and e-mail address.
	2. User selects all parameters even manually or based in one of the predefined benchmarks.
	3. The simulation starts and the system returns the identification number of the simulation.
Alternative Flow:	3a. System returns an error either because of irregular values or missing values
	3b. The user corrects the values from the given information.
	3c. Return to step 3 of the normal flow

Table 3.1: Use Case 01: Starting new simulation.

3.3 System Flow

In a complex system, where applications need to be developed on different platforms and each one has its own particularities of implementation, infrastructure and performance, it is necessary to spend a certain amount of effort on several fronts. This work considers a part of the complexity of the whole project, exactly the one responsible for bringing the choices made by the user in the graphical interface to the application that

Use Case 02	Fetching results
Author:	User
Precondition:	The simulation to be finished
Normal Flow:	1. User access the page and opens the results tab.
	2. User inputs the identification number of the simulation
	3. System searches for the values and shows the results as a comparative chart for runtime and energy consumption at the chosen architectures.
Alternative Flow:	3a. System warns the user that the results were not found.
	3b. The user either searches for another simulation or waits for the e-mail telling that the desired simulation is over.
	3c. Return to step 1 of the normal flow

Table 3.2: Use Case 02: Fetching the results.

decides at which architectures the option pricer algorithms will run, based on the chosen parameters. This work cares about the integration to the other parts. For example, currently the project considers only twelve benchmarks that contain predefined values for the expected outcome of their executions, but later, from a "Data Feed Handler" more values of reference can be easily added in the interface.

The image 3.1 explicit an overview of the project from the first specification provided. It shows the programming languages used in the parts already developed and what remains to be done. The question mark indicates that the way to implement them and/or language were not defined so far. The red area shows the three parts implemented in the present work, in its original form.

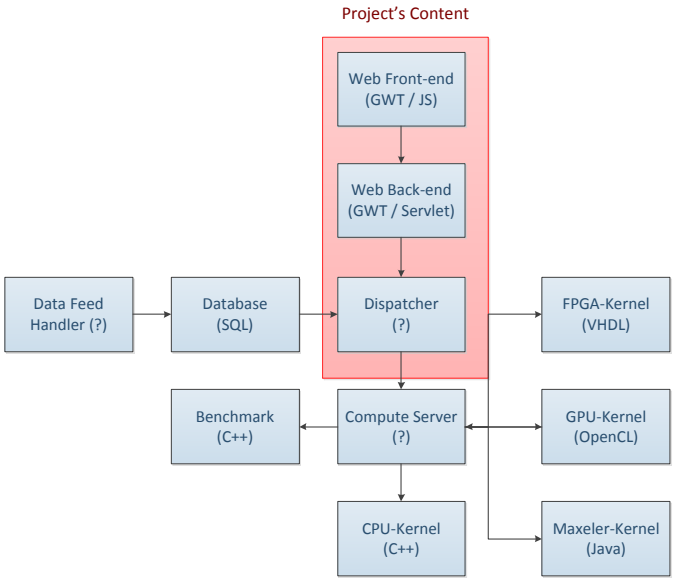

Figure 3.1: An overview of the whole project and this implementation highlighted as initially proposed.

The image 3.2 shows precisely the work made in this implementation. It differs from the red area in the image 3.1 in the sense that the "dispatcher" needs to provide communication between different systems for multiple instances. Wherefore, divided into two

parts, both "Sockets" are part of the "Dispatcher" from the original model. They communicate directly to the library "LibHeston" to run the algorithms locally to simulate the "Compute Server" from the original definition while it is not implemented by the group. The "Compute Server" will be responsible for forwarding each simulation to the respective architectures and our implementation is fully able to be integrated to it as soon as it is done. Later, we explain in details how this process is made.

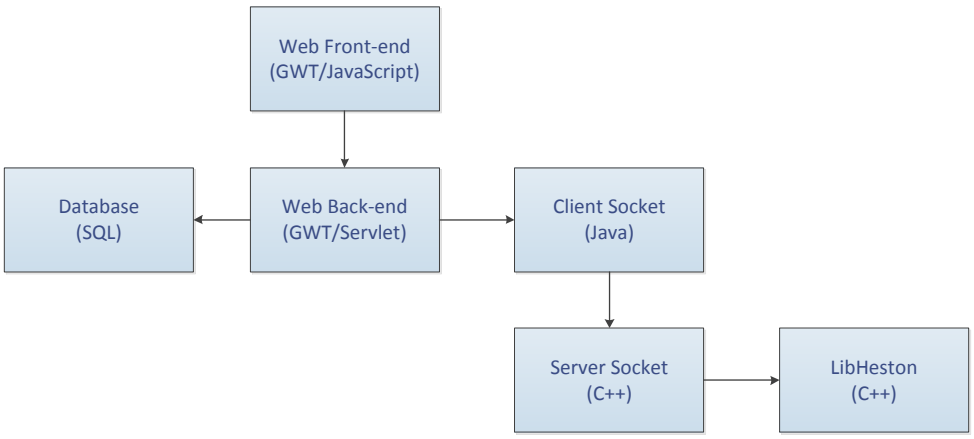

Figure 3.2: The workflow of this project as it was implemented.

3.4 Modeling the Implementation

This section provides information on respect to the implementation of concepts and definitions of software engineering. The tools responsible for putting such models into practice will be presented in the chapter 4.

3.4.1 MVC Architecture

Model-View-Controller is an architectural pattern to separate presentation, interaction and data and there is one logical component defined for each one of these main parts of the implementation. The Model is defined as the information and the rules to manipulate data and it gives the capacity to access the functionalities of the application. The View is the layer that interacts with the user, presenting the interface and reflecting the state of the model according to its data. In the meantime, the Controller is the intermediary between the View and the Model. It defines the behavior of the application and it is responsible for interpreting the requests (a click on a button, for example) made by users. From these requests, the Controller communicates with the model and updates the view based on what has been changed. Using such approach, the code becomes reusable and it is possible to the developer to create libraries, add interfaces easily and develop in parallel without interfering with other parts of the project (Sommerville 2011).

The image 3.3 from (Sommerville 2011, p. 156) can be seen as an abstraction of the MVC.

By using a framework web that works fine with the MVC pattern (to be presented in the section 4.1.1) this project has been modelled as follows:

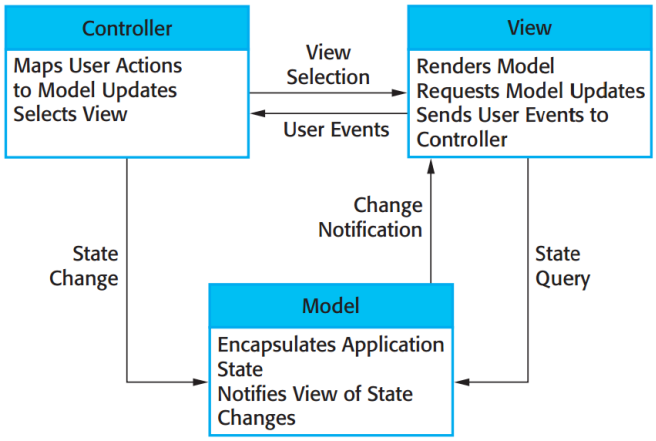

Figure 3.3: The organization of the MVC.

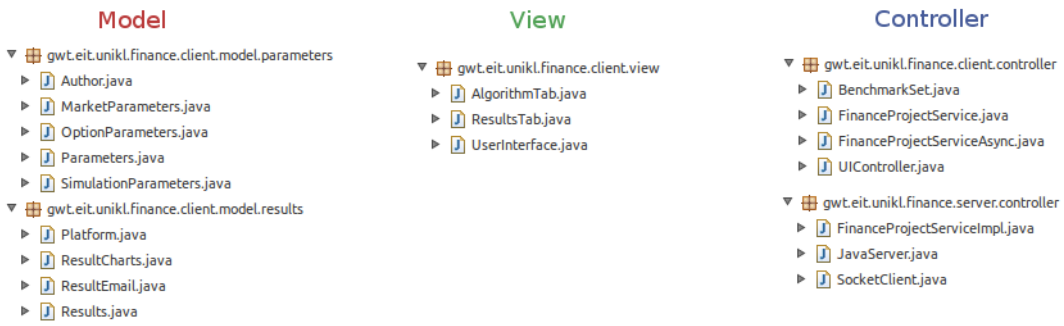

Figure 3.4: The project's classes following an MVC architecture.

The structure shows that the View is divided into two different packages, "parameters" and "results", while the Controller is defined into packages "client" and "server" in order to provide a better understanding of the data modelling.

3.4.2 Data Storage

While running the system, there are two situations where it is necessary to store data.

First, to start a new simulation, all the parameters that has been selected by the user for computing the price of an option must be saved in order to identify the simulation, linking it to the user who has created the new job. At this time, the database receives the unique index of identification as well as all the values that make the setting.

The second situation in which storage is required is just after the end of a simulation. At that moment, the value obtained for pricing, the ultimate precision as well as the values of runtime and energy consumption must be linked to their simulation by having these results to be saved and related to the identification number.

3.4.2.1 Relational Database

Given the need for storage, we chose to use a relational database to implement such functionality.

Relational databases are the most used database model since the 80's mostly due to its

scalability and its arrangement of data that allows the information to be quickly compared. These databases are created using a special language, Structured Query Language (See section 4.2.2) and it is based on a group of tables, each one representing a relation of values that have the same attributes. These tables have an unique value that specifies a set of values in a table, called primary key. Each primary key of a table is used to differ each entry of the table from the others but it is also used to cross-reference between tables. For this to be done, there is the field called foreign key that matches the primary key of another table (Heuser 2008).

3.4.2.2 Structure

The structure of the database used in this project can be seen in the entity-relationship diagram 3.5, a data model for describing a database. It shows the data divided into five different tables, one for each set of values. As explained, the foreign keys link the tables by using the unique identification number of a simulation, that is, the primary key of the table "job".

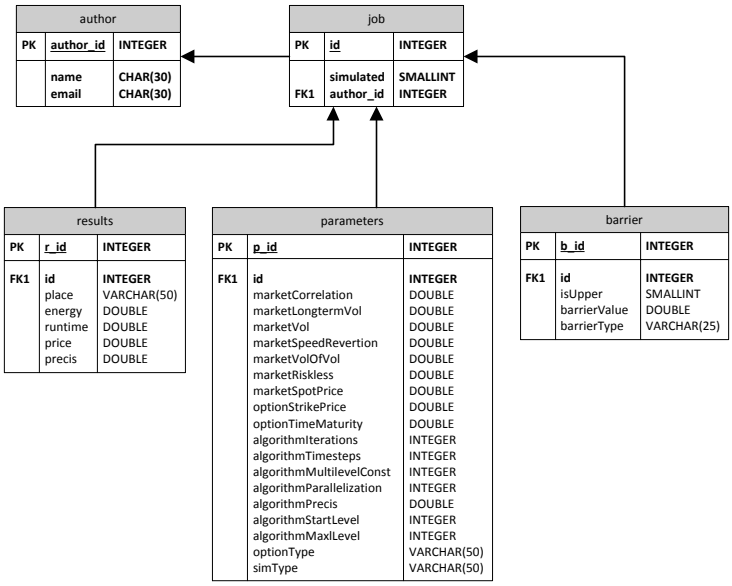

Figure 3.5: Entity-relationship diagram - How the database is structured.

It is noteworthy that, after introducing the abstraction of how data is stored, we need to show the tools that are used to implement it. They will appear on the following chapter, which deals specifically with all stages of the implementation.

4 IMPLEMENTATION

From now on we present how the system works and all the tools that were used to implement it. It is divided into different sections. First, we introduce the client side, that is, the user interface. Then, how the system works on the server is explained. Finally, how the simulations were performed and how the system retrieves the results is shown.

4.1 Client Side

The system presented in this thesis is based on a web user interface that has been developed in order to facilitate the execution of the simulation and display of results (1). Therefore, this section introduces the structure and concepts about how the system interacts with the users. The first section presents an introduction to the used tool, Google Web Toolkit. Later on, it will be shown how the parameters are presented and how the system treats them considering that it aims at the communication with a C++ application to run the required simulations.

4.1.1 Google Web Toolkit

Nowadays writing web applications based on front and back-end for multiple browsers can take time, being complex and error prone hard to solve. Furthermore, the communication between browser and server requires knowledge about low-level calls XHR (XMLHttpRequest), whose implementations are not trivial.

Against this background, Google Web Toolkit (GWT) is a tool that fits in perfectly with the proposed work. Created by Google, the tool has the basic idea of facilitating the development of front-end applications in JavaScript with Ajax integration for asynchronous server requests. GWT emphasizes reusability and efficiency and provides, besides components for dynamic user interface, a simple mechanism for remote procedure calls, very useful for the ultimate goal of the project presented in this thesis. GWT is an open-source tool therefore it has several libraries available developed by Google and third parties.

The tool works focused on the model of object orientation since the code is developed in Java instead of JavaScript, which also allows the use of an IDE of your choice. For this project, the IDE "Eclipse" seemed to be the best option as it has plugins for integration with GWT, as a builder GUI-based drag-and-drop and the GWT compiler. The compiler, when the implementation is deployed to the server, translates

the Java application to JavaScript files that are (optionally) obfuscated and highly optimized so the application gains in performance with respect to a JavaScript code written manually. The compilation safely discards dead code - aggressively cutting of classes, methods, fields and unused parameters - to ensure that the compiled code is minimized (Google Web Toolkit Developer's Guide 2012).

4.1.2 User Interface

In Internet systems based on user interaction, it is essential to think first on the interface that will make this communication possible. For this, it is necessary to take into account the clarity, proper arrangement of items, ease of access to the various sections and portability to various platforms.

In this case, the interface operates according to the set of values used to run the algorithms introduced in the chapter 2. The different parts of this interface as well as the way to deal with the fields will be explained below.

Parameters Tab

As said in the first chapter, a group of parameters is organized in different datasets to be determined prior to making possible the simulation.

Once it was decided to use the Google Web Toolkit the structure of the page had to be defined in such a way that facilitates understanding of the various data types. Thus, the user interface is divided by two main parts which include the fields to select parameters for new simulations and the tab to retrieve results of finished jobs. The image 4.1 shows how the set of parameters are divided.

As it is shown in the Figure 4.1, the Parameters Tab is composed of several groups, containing different sets of values required in the execution of the algorithms. These different groups are described in the following sections.

Market Parameters

4.1.2.1 Correlation

Correlation refers to the way that the values of two sets of numbers vary together. The value of a correlation coefficient is always between -1 and 1, where 0 represents no correlation between the sets, 1 represents a positive correlation and -1 represents a negative correlation. For example, considering two stock prices, if their correlation is 1, then both prices are rising.

- Correlation must be between -1 and 1.

4.1.2.2 Spot Price

Spot Price defines the current price at which an underlying asset can be sold or bought. Basically, the future price of an option depends on its Spot Price, the Riskless Rate and the Time to Maturity which will be introduced soon.

- Spot Price must be greater than 0.

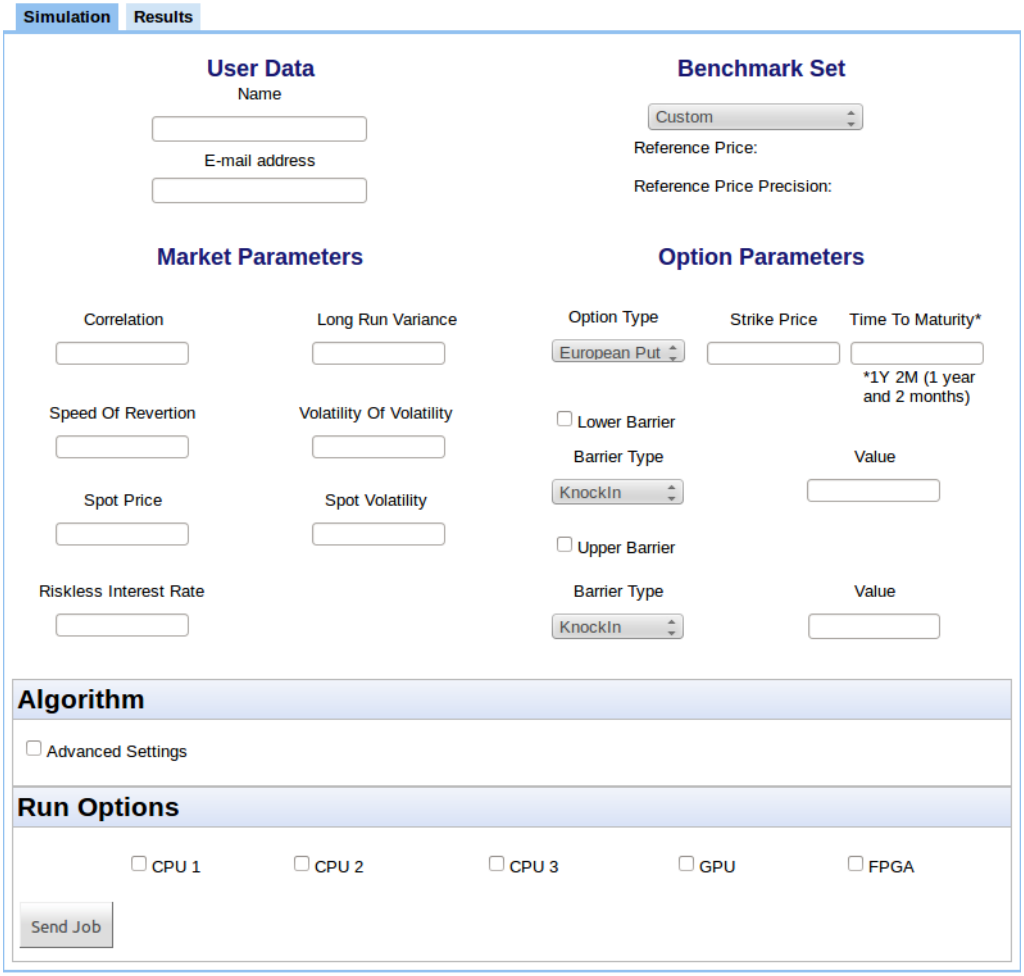

Figure 4.1: The Parameters Tab - How the parameters are disposed in the user interface.

4.1.2.3 Spot Volatility

Volatility measures the variation of a price during a certain period of time. The higher the volatility is, higher the risk when buying or selling an option because under these circumstances it is difficult to predict the variation, which can change faster over a short period in either direction. Thus, volatility is considered a risk measure.

- Spot Volatility must be greater than 0.

4.1.2.4 Volatility of Volatility

Considering the definition of Spot Volatility, Volatility of Volatility is the same but applied to the measurement of the variation of the first one.

- Volatility of Volatility must be equal or greater than 0.

4.1.2.5 Riskless Interest Rate

The Risk-less Interest Rate (sometimes Risk-Free Rate of Return) is a rate of return of any investment with no risk of financial loss. In practice, however, this rate does not exist

because all investments carry at least a very small risk, but, in case of having a potential rate of return greater than the risk-free rate, the operation is safe enough.

- Risk-Less Interest Rate must be equal or greater than 0.

4.1.2.6 Long-Run Variance

Long-Run Variance is defined in an stochastic process as being the the sum of all autocovariances that this process may apply.

- Long-Run Variance must be greater than 0.

4.1.2.7 Speed Of Reversion

Speed of Reversion describes how the interest rate evolves during the measured time.

- Speed of Reversion must be greater than 0.

Option Parameters

4.1.2.8 Option Type

As said before, this project only considers European Option and all its kind of parameters are included in the user interface. The types are as follows:

- European Put
- European Call
- European Digital Put
- European Digital Call

4.1.2.9 Strike Price

The Strike Price is the specified price on an option. In case the option is exercised, the strike price will be the final price that the owner of an option can purchase or sell the underlying asset.

- Strike Price must be greater than 0 and between the barriers (in case that they exist).

4.1.2.10 Time to Maturity

Time to Maturity is the lifetime of an option contract. Again, as it treats European Options only, the maturity date indicates when the option can be exercised.

- Time to Maturity must follow the format of “1Y 2M” which means “one year and two months”

4.1.2.11 Barriers

As already stated, Barrier Option is an exotic derivative which consists of an option containing a barrier that can be upper or lower the Strike Price, where the "Knock In" activates the option while the "Knock Out" deactivates it.

Obviously, the value of the lower barrier has to be below the Strike Price and the upper barrier, above.

- Both barriers must be greater than 0 and they have to follow the rule explained above.

Simulation Parameters

All the needed parameters have to be set due to run correctly the pricer algorithm and it depends on the selected Monte Carlo type. In case that the Checkbox "Advanced Settings" is not selected the algorithm will be a Classic Single Level Monte Carlo filled out with default values that will be explained later on, otherwise the parameters are as follows:

4.1.2.12 Classic Monte Carlo

Classic Monte Carlo is a single level simulation of the Monte Carlo method that considers only Number of Simulated Paths and Number of Discretization Steps per Path. Their default values are 1000 and 1024, respectively,

- Number of Simulated Paths and Number of Discretization Steps per Path must be greater than 0.

4.1.2.13 Adaptive Multi-Level Monte Carlo

This option of Multi-Level Monte Carlo is placed between Single-Level and the Multi-Level, due to its option of decision only over the precision, using the default values for all the other parameters such as start and final level. The field called "Heuristic Optimization" is not implemented yet in the simulation algorithm and, consequently, it is not used so far.

- Precision must be greater than 0.

4.1.2.14 Multi-Level Monte Carlo

Multi-Level Monte Carlo is the most complex execution of Monte Carlo methods. Here it is able to define the precision (Statistical Error) which is the most important parameter once it defines how close to a reliable value the calculation gets. Usually, the higher is the required precision, the higher is the time to finish the simulation. In addition, the start level and the final level can be defined. For example, considering a start level zero and a final level 5, in case the simulation does not reach the precision until the fifth level, the simulation fails. For the Fully Manual Checkbox the number of paths by each level is defined, where the number of levels is given by:

$$Number of Levels = Final Level - Start Level + 1 \tag{4.1}$$

Therefore, the value in Number of Paths on Levels is a set of values containing exactly the $Numberof Levels$ separated by a semicolon.

- Start Level and Final Level must be integer and greater than 0.
- Statistical Error must be greater than 0
- Level Refinement Constant must be different from 0 and 1.
- In case of checking "Fully Manual", it must contain a set of integers per each of $Numberof Levels$ separated by semicolon.

Benchmark Set

Due to the necessity of finding the right set of values to run a valid simulation and the difficulty of predicting the prices, a set of values for Market and Option parameters which contains a reference for the price and precision were created. So far, 12 sets are available for the choice of the user and they will provide an easy comparison over the required precision in case of Multi-Level simulation.

The simulation will take place and run each level until it reaches a price value that, subtracting from the reference price gives a results smaller than the precision. In this case, it is a successful simulation. The simulation fails when this result is bigger. For the single level simulations the reference precision will not be considered but the simulation is still valid.

The implementation provides an easy way to insert more sets of values as soon as it has more results for reference.

Architectures

It is aforementioned that this project is mainly focused on showing how the option pricer implementation behaves over different platforms, comparing runtime and energy consumption. Thus, the web interface has five checkbox to define where to perform the simulations between three CPU, one GPU and one FPGA where more than one can be selected to allow results comparison.

Although the implementation in different platforms has not been finished since the group have not prepared the necessary infrastructure yet, the code is ready to launch the simulations and get the results back as soon as the platforms are ready to it.

4.1.3 Correctness Of Values

In any situation where there is a need for implementation of algorithms aiming at high accuracy, as here, it is essential that the correct value be obeyed. As seen, the system has several sets containing groups of parameters that can be preset or manually entered by the user, which, in case of a little knowledge, can lead to incertitude. Also, if the algorithm is executed with non-valid values, it would not be trivial to prove this error by analyzing the obtained result. It happens because it is difficult to have a reference and the only way could be doing a manual inspection of the parameters that were selected and stored in the database with the intention of finding out the value that gets out of the required range.

The first step in dealing with this situation was to name the parameters that appear in the interface. In contrast to what you may think, following the same nomenclature used

in the implementation of the algorithms might not be the best approach since it contains highly technical terms that may not be accessible to a large number of potential users because they have been developed by people with emphasis on microelectronics or financial mathematics. After some discussions between the involved groups, the nomenclature has been defined and the final is the one presented in this document.

With this first step already defined, it was necessary to create a method to undertake the range of values, according to the definition mentioned in the section above, and returning to the user the exact location of the value that goes out of what was defined enabling a quick fix. After meetings that finally defined this range for the values, this method has been implemented, returning a message displayed to the user as a Javascript "alert", according to the following format:

"***CORRELATION*** should be in the range from -1 to 1"

"***STRIKE PRICE*** should be between
LOWER BARRIER and ***UPPER BARRIER*** "

In the above situation, the incorrect values were those of Correlation and Strike Price. It is important to realize that the parameters will be checked again every time the user presses the "Send Job" button just because after the last check, a value that was previously correct may have been changed, being invalid now.

Besides those already mentioned standardizations, perhaps one that was led to spend more time and further discussions was the "Time to Maturity". As it comes to a future date, it is necessary to define the precision first, that is, if it takes into consideration years, months and weeks. After some discussions, it has been concluded that having a weekly accuracy would be unnecessary once it would not cause a huge effect on the final result. Thus, the maturation time would be given in months and years and now the pending question was how to input that value.

From that, among the various forms of date representation, an approach of a calendar marking the final date of the period might not be the best solution, since it hinders the conversion to create the period of time from the present day to the chosen date. In addition, an idea of decimal numbers to represent years could create an ambiguity, similar to that observed when working with decimal hours where the fraction of seconds is different from a hundred, leading to confusion. The same goes for the amount of months as the decimal part, i.e. "1.5" does not mean 1 year and 5 months. The found solution to get around this was to create a parser for the date following a prespecified format, informing it on the interface before the user enters the value. It was defined as follows:

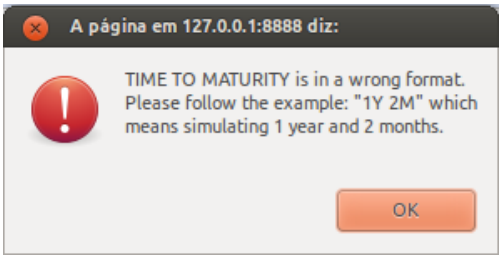

Figure 4.2: Format of Time To Maturity - The warning message of an invalid value.

Where the integer number before "Y" represents the number of years and the one before "M", the quantity of months. It is noteworthy that some extra care had to be taken,

for example, being careful to invalid alphanumeric characters to be not accepted, being necessary to explicitly follow the required format by using 2 integers interspersed and followed by the letters Y and M. The parser is not case-sensitive.

At the end of the definition, the user has to select at least one "Architecture" to have a valid set of values and it is quite obvious to say that the correctness considers blank fields as invalid values.

Finally, having all the parameters set with correct values, there is nothing remaining and the interface is ready to start the next step which will be presented soon and that is used to prepare and initiate the simulation. Before, the second but not less important part of this user interface will be shown in the next section.

4.1.4 Results Tab

On the other side of the user interface the Results Tab takes place. This contains a field to input the unique id of a simulation - given by the code in every new simulation - and gets the results presented as a graphic, allowing the user to compare the energy consumption, the runtime of the simulation, as well as the final value for the option price. All the explanations regarding to the Results Tab will be provided ahead. The image 4.3 shows how the Results Tab looks like:

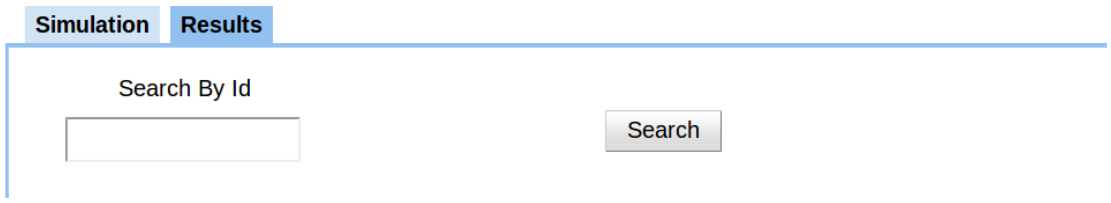

Figure 4.3: The Results Tab - An input field to search for the id of the sought result.

4.2 Server Side

This section will introduce the GWT main concepts used to create the server side that controls the user interface and everything that is necessary for the simulation to take place.

Considering that the system had taken all the selected parameters and checked their correctness it needs now to find a way to communicate with the server to handle this data, being able to store it and start the simulation. As said before, GWT provides an easy way to reach the main goal here by using its RPC methods at which it is not necessary to care either about communication or security questions, thereby, the implementation becomes easier.

In addition, the data storage will be presented, capable of making our application able to fetch results later on and giving the power to handle a huge amount of data as soon as the system has a big quantity of done simulations. With the system based on the server and connected to a database among other benefits, it gives the ability to create statistics based on the selected data, to check the correctness of the results in case of having an algorithm of difficult verification, as well as the ease of searching and displaying data that could be easily confusingly in other form of storage, such as files, for example.

In the end of this section a brief introduction about how the simulation will be orga-

nized before being sent to the C++ code responsible for starting the simulations is given, detailing the communication over languages and the way it is made up to create a consistent multi-languages pattern.

4.2.1 Remote Procedure Calls

Remote Procedure Call is a protocol that allow the user to request a service remotely through network without having to understand low-level technologies. RPC works carrying data between different programs that need some kind of parameters exchange for any reasons, and in this case, to receive the set of parameters from the client by a web browser to be treated in a specific manner shown below, according to our needs. This service can be easily managed by the RPC technology provided by GWT.

In order to clarify the operation of RPC methods and to know how GWT handles with the relationship between the client-side and the server-side code we must, first, rely on a tip provided by the documentation, which says:

> "Although GWT translates Java into JavaScript for client-side code, GWT does not meddle with your ability to run Java bytecode on your server whatsoever. Server-side code doesn't need to be translatable, so you're free to use any Java library you find useful."

This means that RPC methods allows the system to be developed using native java code, including any Java library, whether it has web support or not. Due to this, RPC methods are the most important feature provided by the tool, being really helpful to the greater need of this project: the necessity of running in a web browser but with the ability to communicate with an external code running on servers.

To understand deeply how the GWT RPC works a brief explanation of AJAX is provided. AJAX is an acronym for "Asynchronous JavaScript and XML" and a group of techniques on the client-side to create web applications. With Ajax, the application is able to send data to, and receive data from a server without interfering with the display and behavior of the page. Our goal here is to call the desired methods, running libraries and hence simulations, giving back to the user just a simple information that says whether the simulation will run or not, but also allowing the user to continue using the page on the same screen, which means that no refreshes on the page are required. GWT takes this best part of using Remote procedure calls and creates an easy way to work with. A class implementing "RemoteService" is enough to provide the referred calls with no worries on the part of the developer. Furthermore, GWT RPC abstracts out lots of information and wraps Ajax to allow the developer to code in JAVA without worrying about different browsers.

4.2.1.1 Implementation

Remote Procedure Calls, as explained above, are required in every request to the server. The question here is "when are these requests made?". The first one is clearly when the user sends the parameters to start a new simulation passing the chosen set through the network. The second one is when the results are required and in this case only the id of the simulation is needed. Every time that the system needs to send or receive something

through the network it is essential to implement the class RemoteService and generate two different interfaces: the service itself, which contains the methods declarations, and the service asynchronous, allowing the system to keep running on the background. The interfaces are called "FinanceProjectService" and "FinanceProjectServiceAsync". Also, one class needs to be created to implement the declared methods. This class, "FinanceProjectServiceImpl", extends RemoteService but implements "FinanceProjectService" as well to serve as the first class to handle data on the server side.

The image 4.4 illustrates the implementation of a RPC with all the classes and interfaces implemented by the developer as they interact with those created by the GWT after compiling.

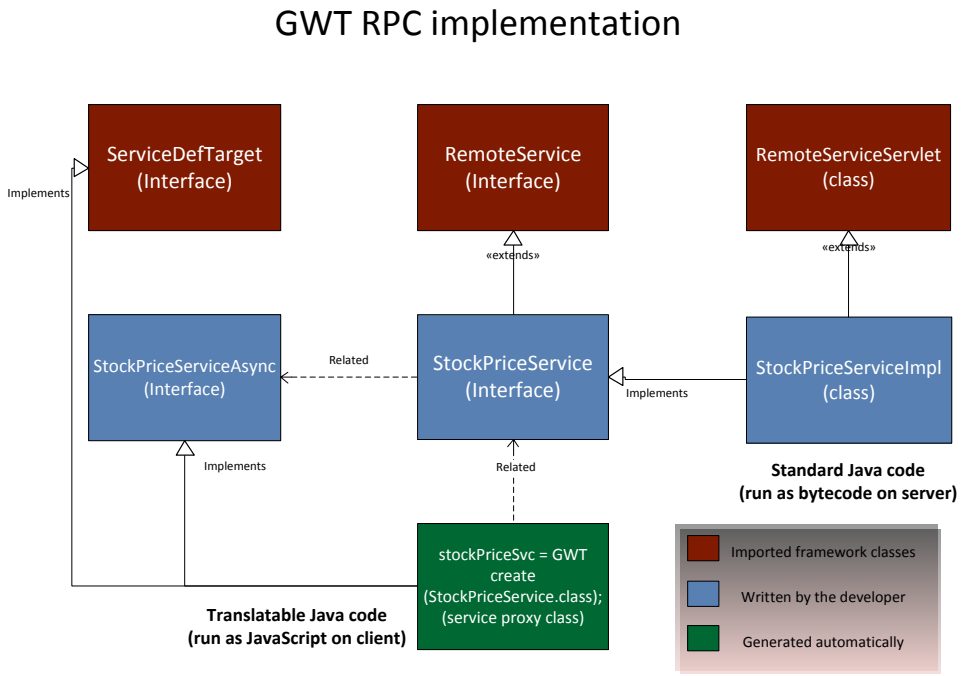

Figure 4.4: How RPC works on GWT - From GWT Docs with sample project "StockPrice" (Google Web Toolkit Developer's Guide 2012).

It is important to say that, since the parameters selected by the user are in JavaScript on the client-side and shall be received by the server-side in Java, the system must recognize them in both sides. Therefore, each of the classes that have objects passed through the network need to implement "Serializable" - as it is done in all classes in the package "Parameters" -, ensuring success in converting between the languages. This serialization is similar to the one implemented in the project to transfer data between Java and C++, explained in the next section.

4.2.2 The Data Storage

Currently, web applications are strongly based on data storage dealing with huge amounts of values that can to be related to each other. Even though bigger companies may have big data centers spread all over the world, there are times it can be saved in one server only, that is, being easy to manage and assist the database.

4.2.2.1 MySQL and JDBC

As the system works with pure Java native code and it has to be capable of storing data on the server, any database together with a java-compatible technology might be able to access it. MySQL has been the chosen database as it is the world's most widely used open source relational database management system that provides multi-user access to databases, together with JDBC, an Oracle's API to communicate Java code and MySQL, providing ease and correctness of use to the tool.

Using both technologies, a database to store the data has been created and every time that it receives a new simulation, an unique number of identification is given. This unique ID provides the option to retrieve the results later on and it is returned to the user as the result of the RPC call to be shown as an "alert window". Image 4.5.

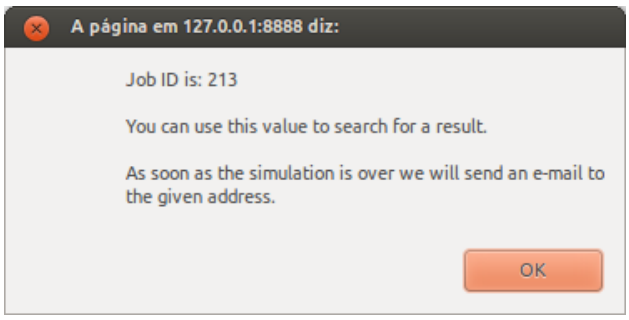

Figure 4.5: The alert containing the unique id of the new simulation.

4.3 Data Pattern

In the last section it is shown that the project has a specification determining the need to implement Serializable to establish a data pattern between the Java code compiled to JavaScript and the Java back-end when data is transferred from the client to the server-side. The same needs to be done on the server-side before communicating Java and C++ to run the pricer algorithms. However, unlike before, now there is no native implementation responsible for this data standardization. To solve this problem and allow the communication XML was used.

4.3.1 XML

XML (acronym for Extensible Markup Language) is a set of rules to represent documents. Due to its simple syntax and structure, XML has been created to be readable and interpretable for both humans and machines becoming a commonly used way to represent data structures instead of proper programming language, thus serving as a bridge for communication between different languages and platforms. XML is free of platform, extensible and it emphasizes simplicity (Bray et al. 1996).

Next, we will introduce a short explanation of its usage as well as the tools used to format the data from the server-side in XML.

4.3.1.1 *Usage*

It is important to know that every document has to follow a set of rules. For XML, the main rules are as follows:

1. Every document begins with a tag showing that it is a XML and specifying its version.
$$<?xml version = "1.0"?> \tag{4.2}$$
2. An XML file must have one root element
3. The tags have to be opened and closed following the rule of
$$< tag > data < /tag > \tag{4.3}$$
4. The programmer can define his own set of tags, being able to create a specific document according to his necessity

```
<?xml version="1.0"?>
<exchange>
      <currency>
            <name>American Dollar</name>
            <acronym>USD</acronym>
            <price>1</price>
      </currency>
      <currency>
            <name>Euro</name>
            <acronym>EUR</acronym>
            <price>1.3081</price>
      </currency>
      <currency>
            <name>British Pound</name>
            <acronym>GBP</acronym>
            <price>1.5157</price>
      </currency>
      <currency>
            <name>Brazilian Real</name>
            <acronym>BRL</acronym>
            <price>0.5041</price>
      </currency>
</exchange>
```

Figure 4.6: An example of XML file.

The example 4.6 illustrates how the tags can be distributed to be readable and have their data extracted. Moreover, it gives a glimpse of what would be an example of currency exchange, taking "exchange" as a main tag containing the different currencies, each one having a name, acronym and its price based on American Dollar. It is clear that the file begins with the aforementioned tag containing its version. Following this format, the data is ready to be read and manipulated.

Now that it has been showed how the structure has to be, the focus goes to the data. In order to transform the set of parameters from a Java class to an XML document a free library called "XStream" is used. Later, to work with the C++ code, a serializer implemented by the group does the work. Both are explained from now on.

4.3.1.2 *XStream Serializer*

According to the XStream Serializer Developer's Guide, XStream is "a simple library to serialize objects to XML and back again". It works in the java code by simply importing the library and calling its method "toXML" for serialization and "fromXML" for deserialization. Its important features, again as it is written in the official documentation, are:

1. Ease of use.
2. No mappings are required when the user calls its serialization method.
3. Performance. Speed and low memory footprint are a crucial part of the design.
4. Clean XML. No information is duplicated, turning the XML easier to read than native Java serialization.

For its usage it is necessary to specify a Java class that the generated XML will be based on and also a class to receive the data while deserializing, with the attributes nominated by the same than the XML tags in order to match deserialized data.

```
//instantiate the object from the XStream library which will serialize the object and create the xml
XStream xstream = new XStream (new DomDriver());
xstream.alias("barrier", Barrier.class);
xstream.alias("parameters", Parameters.class);

//the header and the parameters to serialize
String xml = "<?xml version=\"1.0\"?>\n" + xstream.toXML(parameters);
```

Figure 4.7: An example of XStream serialization.

```
///the deserialize into Results
XStream xstream = new XStream (new DomDriver());
xstream.alias("Result", Results.class);
Results result = (Results) xstream.fromXML(fileXml);
```

Figure 4.8: An example of XStream deserialization.

For the needs, XStream has proven to be the perfect library, combining performance with ease of use and its way of construction ensures no duplication of data.

The XStream is the tool to be used in the Java code on the server-side. As already said, a different one must be used on the C++ code.

4.3.1.3 *C++ Serializer*

Due to the necessity of communication over different projects, the group has developed a C++ library to data serialization and deserialization, which is of paramount importance in this project to handle the upcoming data from Java code. This library has been created by Francisco Borja (2011) and it provides the option of working with files to transfer data standardizing both the serialization and deserialization.

The library accepts the following data types:

time_t, double, float, std::string, __int8, __int16, __int32(int), unsigned __int8, unsigned, __int16 and unsigned __int32(unsigned int).

In order to adapt the XML to be read by the present deserializer the system has to treat equivalent Java types only. Thus, it uses integer set with 1 or 0 instead of boolean, for example. Also, as __int8, __int16 and __int32 are compatible with windows only and the server runs on Linux, equivalent recognizable types were defined in order to avoid compilation errors.

The mapping is given as follows:

1. __int8 to int8_t
2. __int16 to int16_t
3. __int32 to int32_t
4. unsigned __int8 to uint8_t
5. unsigned __int16 to uint16_t
6. unsigned __int32 to uint32_t

In what follows, an illustration of how the XML was assembled and how libraries treat serialization data is showed.

4.3.2 Structure

As shown, an XML string is easily created by instantiating an object from the XStream library and calling its method "toXML". The willingness to data classes is configured in a such a way that the structure of the XML follows the required standards and becomes interpretable by whatever library used for deserialization.

Once the system has the XML string already created and well defined based on the rules, it is now able to send it to the C++ side which will treat the data. This communication will be made by a TCP Socket - explained in the next section.

The image 4.9 clearly explains the process of creating the XML, illustrating the proper equivalence between classes and objects with their XML tags.

4.4 Communication

Sometimes when working on huge systems performing all of the application development with a single language may be ideal, but it is not always practical. There are many times when it may be necessary to integrate a new application with a legacy one, as here, for example, where the application for pricing is developed in C++ and a web application that works according and synchronized with it is needed. This communication between both parts can be an issue.

The idea while facing this situation was to isolate both applications so that the design of the web interface together with its sever side in Java is not compromised, while the

```
public class Author implements Serializable {

      private String name;
      private String email;
      …
public class MarketParameters implements Serializable {

      private double correlation;
      private double longRunVariance;
      private double spotVolatility;
      private double speedOfRevertion;
      private double volatilityOfVolatility;
      private double risklessRate;
      private double spotPrice;
      private double referencePrice;
      private double referencePricePrecision;
      …
public class Parameters implements Serializable {
      private Author author;
      private MarketParameters mktParameters;
      private OptionParameters optParameters;
      private SimulationParameters simParameters;

      private int CPU1;
      private int CPU2;
      private int CPU3;
      private int GPU;
      private int FPGA;

      private int id;
      …
```

```
- <parameters>
  - <author>
      <name>Guilherme Teieira</name>
      <email>gui.andrighetto@gmail.com</email>
    </author>
  - <mktParameters>
      <correlation>0.0</correlation>
      <longRunVariance>0.04</longRunVariance>
      <spotVolatility>0.04</spotVolatility>
      <speedOfRevertion>0.5</speedOfRevertion>
      <volatilityOfVolatility>1.0</volatilityOfVolatility>
      <risklessRate>0.0</risklessRate>
      <spotPrice>100.0</spotPrice>
      <referencePrice>0.7487</referencePrice>
      <referencePricePrecision>1.0E-5</referencePricePrecision>
    </mktParameters>
    <optParameters> ... </optParameters>
    <simParameters> ... </simParameters>
    <CPU1>0</CPU1>
    <CPU2>1</CPU2>
    <CPU3>0</CPU3>
    <GPU>1</GPU>
    <FPGA>1</FPGA>
    <id>206</id>
  </parameters>
```

Figure 4.9: The Java Class as an XML file. The left side shows the main class Parameters and its subclasses MarketParameters and Author while the right side is the same but in the XML format. Option and Simulation parameters were hidden for a better understanding.

older one can be upgraded whenever it is necessary without impacting the Java system. However, the correlation between the parameters of both systems must be preserved.

In order to find a way to create a usual communication that respected the project needs, some distributed computing solutions integrating Java and C++ applications have been explored. Some thoughts were given first to Java Native Interface (JNI), then to Java Message System (JMS) and finally web services. Although these approaches can be good and useful in the right situations, they were not the best options to use in this system. For instance, calling into native code from Java via JNI can be complex, time-consuming and error-prone. Using JMS requires a JMS provider be licensed, installed and configured, while a web service requires significant development and web-based infrastructure.

Another approach is to use a socket-based network communication directly between Java and C++. For this project, although this is a relatively low-level approach, it is still an effective solution that combines performance with scalability that fits perfectly due to the need to control multiple simulations simultaneously. After some research, it has been found out that combining serialized XML as the message protocol with TCP Socket a degree of platform and language independence is preserved.

The image 4.10 illustrates the communication in a very simple way. Here it is shown that a C++ application can communicate with a Java application easily.

4.4.1 Socket TCP

TCP Socket is "one end-point of a two-way communication link between two programs running on the network" (Lesson: All About Sockets). In other words, sockets are an abstraction to addresses through which processes communicate and in order to make this possible there is a typical flow of events. First, there is an application creating a socket server and another one that implements what is called a client - not the same one that is used in the user interface as it is also targeted to run on the server. This socket on

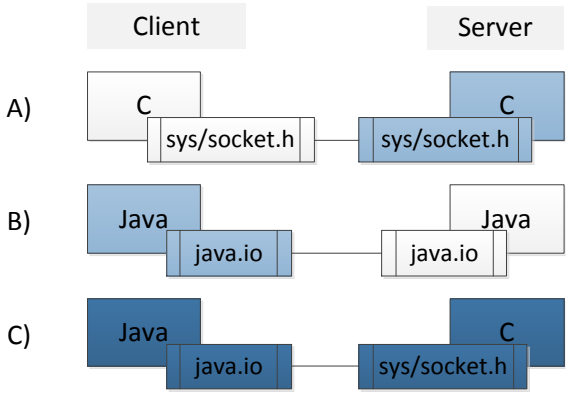

Figure 4.10: Different ways to communicate code using TCP Sockets. The option "c" illustrates the intersection that is the basic concept of this implementation.

the server waits for requests from the client by calling a method "bind" which establishes an address to be used by the client to find the server. After binding, the server creates a listener to be waiting for the client to request a service. When the connection comes from the client side, the listener perceives it, the request is performed and a reply is sent back to the client. The image 4.11 explains the flow of a generic socket system.

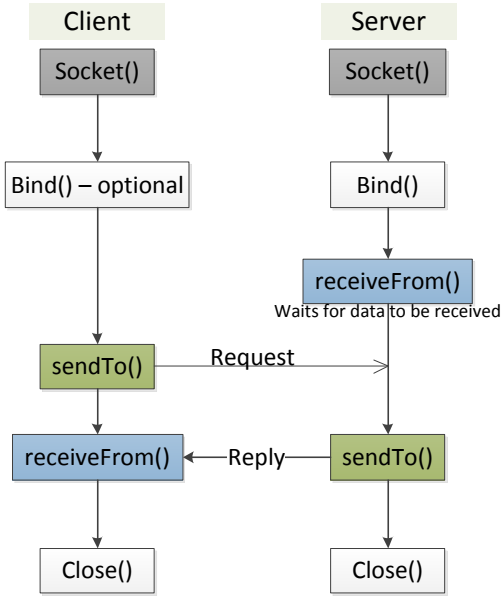

Figure 4.11: Socket's flow.

Moreover, a socket system needs three basic things to be configured, first, the IP address both for local and remote addresses, a port number (8080, for example) and a transport protocol can be used. Here, the chosen protocol was TCP instead of UDP since it is extremely necessary to guarantee delivery of data on both sides.

4.4.1.1 Transmission Control Protocol

The Transmission Control Protocol, or simply TCP, is one of the most important protocol of the transport layer from TCP/IP model and its main characteristics are:

1. The delivery of data in accordance with the order of entry.
2. The amount of data is checked and controlled, thereby avoiding saturation of the network.
3. it allows communication from multiple clients to a single server.

These and other features enable the application to work safely regardless of data traffic, an important issue for the proper functioning of this project.

In the following, it shows how the communication is established, based on the flow of operation of this socket. From here, it is important to consider that the C++ code responsible for the socket server must always be running in the background, listening and able to connect every time a new simulation is requested if it does not have any inaccuracy in the data, as already mentioned.

The Server

The implementation responsible for running the pricer algorithm based on Heston Model is contained in a C++ application, as already mentioned.

In order to start the simulation running this algorithm and using as a parameter the data set selected by the user in the web interface a class responsible for data deserialization (explained in 4.3.1.3) but also responsible for communicating to the application that runs the simulations was implemented. This class implements the methods for the socket server as well and once running, it waits for the connection from the client socket.

The socket server creates one thread for each connection received at its listener so, the simulations can run separately, providing the results as they finish and not necessarily in the same order that they started.

The Client

For the client, creating the socket occurs almost the same as the server. First the socket is created in the chosen port and IP (so far we used "localhost" for testing the simulations) and then the listener is instantiated. It is important to know that for each remote call from the user interface the system creates a different thread containing one listener each, fit to receive the results. After the creation of these threads, a sender method will be called to use the opened connection to send data to be received by the listener in the server side (Java to C++).

4.4.2 Connectivity and Data Transfer

To transfer the string XML created after data serialization from the user interface it has been chosen to make use of the memory buffer. This works in such a way that once created

the socket, if there is communication between the server (C++) and the client (Java), the client inserts the string in the buffer leading the server side listener to recognize the arrival of data. This listener reads the string buffer and saves it in a C++ string until it finds the end of the buffer, indicating the end of the XML.

However, this is not the only time that this socket is required. After some simulations finish it is primordial to return the results. On this time the origin is the server (C++) and the results are sent to the client (Java). The client will be listening in as many threads as needed for each simulation and they will be closed as the results are received. Here, the data rather than being passed via buffer is saved in a ".xml" file that will be named "results" concatenated with the unique id of the simulation. Only this name is passed via buffer to be read by the Java code, which first deserializes the generated string using again XStream and afterwards saves the results in the database (to be discussed in the section 4.5.3). It was made this way since, as stated in the previous section, the serializer works with files as standard, therefore, having a better usage and not interfering in performance once the returned amount of results is always a small set of values.

4.5 Simulations

Now that we have shown the process of communication between both codes works and standardization of the data for the parameters and results have been explained, it is time to focus on the simulation. Here, it will not focus on the implementations themselves but on the interaction between the applications since this implementation was responsible for the control and integration of the whole system, starting in the user interface and passing by storage, standardization and data transfer until it reaches the C++ application (or the test library) and makes it able to run simulations and get results back.

After working with the serialization library to deserialize data from the Java code the parameters are stored in a C++ class defined according to the format of the Java classes, taking into account the datatypes of both serializes. This class works as the destination to the parameters and it is organized as it is showed in the image 4.12.

As soon as this class has an instantiated object with all its attributes filled out - and it actually does, once the correctness of the data was checked just after collecting from the user interface - it has all the necessary parameters, avoiding either type errors or incorrect references. Having this dataset it is now able to handle those two different kinds of Monte-Carlo Simulations, "Single Level" and "Multi Level". Depending on the selected simulation the system reacts differently, instantiating the object for the respective simulation. Note that in case of "Single Level" the simulation will not take a long time until it finishes and the parameter referred as a precision is ignored, being not set or set with the default values. The same process is done with the other non-required parameters that might contain default values used to validate the set. For "Multi Level" simulations a long runtime may be expected, which depending on the selected parameters can last longer than a week and, in this case, the system is able to handle this simulation at the same time than the new ones coming from the user interface due to the thread system introduced in the previous section.

```
DECLARE_SERIALIZABLE_STRUCT2
(
        Author                  ,
        std::string             ,       name    ,
        std::string             ,       email
);


DECLARE_SERIALIZABLE_STRUCT9
(
        MarketParameters                ,
        double                          ,       correlation             ,
        double                          ,       longRunVariance         ,
        double                          ,       spotVolatility          ,
        double                          ,       speedOfRevertion        ,
        double                          ,       volatilityOfVolatility  ,
        double                          ,       risklessRate            ,
        double                          ,       spotPrice               ,
        double                          ,       referencePrice          ,
        double                          ,       referencePricePrecision
);
DECLARE_SERIALIZABLE_STRUCT3
(
        Barrier1                        ,
        double                          ,       barrierValue            ,
        std::string                     ,       selectedBarrierType     ,
        int32_t                         ,       hasBarrier
);

DECLARE_SERIALIZABLE_STRUCT5
(
        OptionParameters                ,
        double                          ,       strikePrice             ,
        double                          ,       timeToMaturity          ,
        std::string                     ,       selectedOptionType      ,
        Barrier1                        ,       upperBarrier            ,
        Barrier1                        ,       lowerBarrier
);
```

Figure 4.12: The Parameters C++ Class - How they are organized over a different language (it is standard to the other parameters). This structure using C++ constants makes the XML to be readable by the serializer.

4.5.1 Single Level Simulations

Here we focus on the implementation for the Single Level Simulations, also called "Classic Monte Carlo".

This simulation implements the simplest Monte Carlo algorithm, as introduced in the 2, and it takes care of a few number of parameters only. Considering this case and once it is already known that the Single Level simulation does not take any precision for the calculation, this implementation selects only the strictly necessary parameters, ignoring others for filling data with default values. For example, before running a simulation, the "libheston" requires that all parameters are set with any value and in case of "null" access it returns an error of null pointer stopping its execution, which should never happen, it is then subjected to have the set of parameters lost somewhere in the database without any results once the simulation had not started yet, and causing a stop on the server side application. Thus, the fault tolerance of this system consists in filling data with some dummy parameters in order to maintain the consistency of the algorithm both to Single Level as for the MultiLevel.

By default, the values are as follows:

MultilevelConstant = 4

NumberOfIterations = 1000

NumberOfTimeSteps = 1024

Parallelization = 10

This set consists of coherent values according to the needs of the algorithms to run correctly and to be able to present considerable results. As the simulation parameters are the most complex, error prone and difficult for a normal user to select and deal with, these can be stuck in the code allowing the user to treat with higher-level parameters only. As said, it keeps the consistency of the set of simulations. For example, it is impossible to compare the results of two different simulations with almost the same parameters but completely different "number of simulated paths". The algorithm will not run enough when it is set with a low number of paths, turning into a result that does not mean so much to the final result. Furthermore, in case the user does not know what he has to do to keep its consistence, the system does this for him, almost in the same way it does for the set of benchmarks by filling out the other parameter sets.

4.5.2 Multi Level Simulations

Unlike what is done to the Single Level simulations, Multilevel simulations must consider a large, complex and accurate set of parameters. Due to this, it is not necessary to fill out the attributes with default values, considering the deserialized ones that come from the user interface. When working with this kind of simulations it is critical to use correct values mostly when related to precision. Every time the user asks for a big precision the complexity of the algorithm grows exponentially making the time for the simulation unpredictable. During the tests, most of the time the precision is based on the same as it is used in the referred benchmark, giving us one acceptable time to finish the simulation but also control its correctness. However, it has been tested with really high precisions and some simulations took days or even weeks until they were done. Also for these cases, despite the delay in their execution, it could get satisfactory results that matched the reference.

To be more exact with regard to the precision used, the system had acceptable times for testing among one, two or three decimal places. For greater accuracy, it was very difficult to predict the execution, often leading to the need to not monitor the results in real time, but checking them later in accordance with what was saved in the database.

4.5.3 Results

After performing the required simulation, the C++ code has the computed result for the option price. In case the system has run one of the 12 standard benchmarks, this result allows a concrete analysis of the correctness of the result by subtracting of it the reference value, which results in the calculated precision that must be smaller than the sought one to have a valid result, in case of a multilevel simulation.

So far, all multilevel simulations performed with valid parameters have obtained also valid results for any runtime.

From this point on, an XML file that contains the unique id of the simulation, the calculated price and the achieved accuracy is generated, again using the C++ serializer in an inverse process that has been done in order to get the parameters. This file is saved, titled "result_10.xml" where "10" would be the id regarding the simulation of id ten. For the simulation of id twenty, the filename would be "result_20.xml" and so on, so that it can be recognized later by the Java code. The image 4.13 illustrates an XML configuration as the result.

```
- <Result>
    <price>0.744770</price>
    <precision>-0.003930</precision>
    <CPU1>1</CPU1>
    <CPU2>1</CPU2>
    <CPU3>0</CPU3>
    <GPU>0</GPU>
    <FPGA>1</FPGA>
    <id>215</id>
  </Result>
```

Figure 4.13: The Results .xml - In this case, the simulation number 215 that had run in the architectures CPU1, CPU2 and FPGA.

As the TCP Socket continues running with the Java side listening the reception of any data, the system only sends the name of the file that contains the results so that, henceforth, the file is read by Java, parsed as a string and finally deserialized by the XStream serializer as an inverse process that was performed with the initial parameters, that is, becoming a Java object.

At this point, the total simulation time is added to the parameters by means of a timestamp that was initiated when the parameters were sent to the socket and stopped on the returning of the filename. This time will serve as the basis for production and analysis of the results since, eventually, it will come from different architectures and must vary depending on the place where the simulation was executed.

With respect to energy consumption, while there is not the opportunity to run the simulation on other architectures but where the system is hosted, default values were adopted in accordance with the selected architectures for which it could be shown consistently to the functionality in an attempt to get as close as possible to the ratio of consumption. Thus, the energy is calculated using its standard formula

$$Energy = Runtime * Powerconsumption \tag{4.4}$$

having the power value hard-coded in an ascending order for FPGA, GPU and the different CPU. These values agree with what was demonstrated in the paper (Schryver et al. 2011), one of the motivational bases for the development of this project.

4.5.3.1 Mailing System

It was a challenge to find a way to bring back the user to the system after the simulation if it is not finalized immediately.

Imagine that the overall result may take any time between 1 hour and 3 weeks, it becomes really difficult to predict whether the user will return to the system to analyze the results and even if he will remember the correct id related to his respective simulation. Therefore, it was decided to implement a small e-mail system, with the intention to inform the user that the simulation is over, providing a link for easy access to the "Results Tab" and informing the id so there is no need to memorization by the user.

This system has been implemented using the Java library "mail", with the creation of the address "gwthestonsimulation@gmail.com". The fields "name" and "e-mail" have been added to the user interface in order to make this possible and are needed before the start of a new simulation.

With this feature, the user is invited and attracted to fetch the results in a simple and fast way. In addition, the access to all simulations is facilitated since the user has an easy access to all his ids. The text below shows how the e-mail goes to the user (the link provides access to internal IPs only so far).

> Dear User,
>
> Your simulation is done under the id 10 and you can search for your results accessing the following link:
>
> http://gwt.eit.uni-kl.de:8080/financeproject/FinanceProject.html

4.5.3.2 Display

The best way to compare the results of different architectures on the runtime and power consumption is through a chart that allows easy distinction between the various segments. Although it was possible to create a graphical library, it was decided to use the "Google Chart", easy to setup and use and with the ability to create various types of graphics.

Google Chart

Google Chart is a tool provided by Google, as the name says, for the creation and styling of different ways to visualize datasets in web pages.

By default, it is done with pure Javascript calls embedded in the page. These Javascript methods need to be populated with data and configured according to the desired style. Once they are configured, the library communicates with Google servers, providing the graphics because of these calls.

In order to facilitate the integration of graph systems developed with GWT, Google provides a library that implements the Javascript calls and that has been proved to be useful to the proposed objective.

So far, the only need has been to implement a traditional bar chart, as shown in the screenshot 4.14, where the FPGA, CPU1 and CPU2 were selected as architectures.

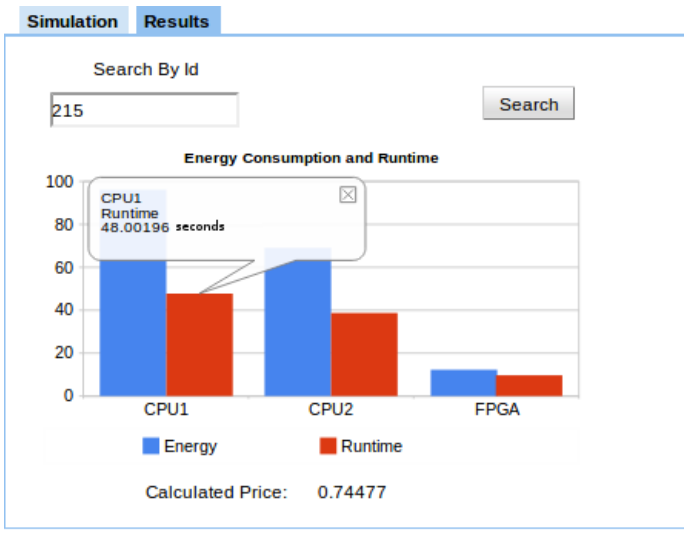

Figure 4.14: The Results - In this case, a chart that shows the calculated results for the simulation number 215.

The only restriction for using this library is a need for Internet connection, since it makes requests to the Google server for creating charts. This situation has been discussed among the group, as some data may be sent externally and also because there is the need to connect all the time. However, the conclusion was that a set of decimal numbers do not have much value in a random context, in this case, without sending the parameters and that, as a web system, it is assumed that it is constantly connected to external networks. Therefore, the library responds well to the present requirements without many side effects giving to the user the chance to check the efficiency of this approach even without real-time values.

5 CONCLUSION

Currently, the complexity of products, models and the needs for risk assessment in the field of financial mathematics are increasingly present in the major markets and stock exchanges. Furthermore, their complex and long-lasting calculations create a challenging situation regarding the runtime and energy consumption. Increasingly better solutions face these problems satisfactorily always seeking to put together cost and benefit.

In this context, this thesis has showed the implementation of a web interface for communication with a system for option pricing, implying the need to go deep into various fields of Computer Science and making use of complex and error-prone tools that led to sometimes not-trivial implementations. It was shown that the interface - and all that this implies - was satisfactory, enabling the simulations to be run simultaneously.

Finally, it became clear that this work is able to achieve the goal of communicating to an application that runs over different architectures and demonstrating the improvement provided by the solution of the group of the University of Kaiserslautern using an FPGA (Schryver et al. 2010; Schryver et al. 2011; Schryver et al. 2011).

For the platform to be on-line to external networks, some parts shall be implemented in the future, especially with respect to the structure to be simultaneously applied to all the architectures as well as the need for security before setting an internal system of the university as public. What must be done is shown below.

5.1 Future Work

This project focuses on the development of a web interface to connect and communicate with a system in a different language and it is able to perform satisfactorily what has been proposed, as illustrated throughout this thesis. But the whole project is divided into different parts as showed, some of these have already been completed:

One part to be developed is based primarily on system security and server configuration, so that it can be uploaded in an environment external to the University of Kaiserslautern, and the possibility of obtaining results in real-time that differ from those used to demonstrate the functionality of the interface.

These results must be obtained through the different architectures to be configured by the group of microelectronics of the University of Kaiserslautern and might be able to connect to the "server side" designed to be part of this project and thus exchanging data through TCP sockets and returning real values according to the energy and perfor-

mance of each one of the architectures in particular. This real-time characteristic relies on the availability of the CPUs, GPU or FPGA. Part of this work is currently being implemented by the graduate student Carolina Nogueira, also from the Federal University of Rio Grande do Sul - Brazil, in the cooperation program of the Institute of Computer Science with the University of Kaiserslautern.

With the configuration of all architectures and server, the system will be fully accredited to prove the efficiency of the solution found by the group, using an FPGA, with regard to differences in time and energy consumption for option pricer implementations based on the Heston Model.

REFERENCES

[Andersen 2007]ANDERSEN, L. Efficient simulation of the heston stochastic volatility model. New York, USA, January 2007.

[Bray et al. 1996]BRAY, T. et al. *Extensible Markup Language (XML)*. [S.l.], 1996. Available from Internet: <http://www.w3.org/TR/REC-xml>.

[Cheng 2003]CHENG, K. An overview of barrier options. Global Derivatives, September 2003.

[Educating the world about finance]EDUCATING the world about finance. (Accessed 20 august 2013). Available from Internet: <www.investopedia.com>.

[Google Web Toolkit Developer's Guide 2012]GOOGLE Web Toolkit Developer's Guide. 2012. Licensed under the Creative Commons Attribution 3.0 (Accessed 20 august 2013). Available from Internet: <www.gwtproject.org>.

[Heinrich 2001]HEINRICH, S. Multilevel monte carlo methods. *Large-Scale Scientific Computing*, p. 58–67, 2001.

[Heston 1993]HESTON, L. A closed-form solution for options with stochastic volatility with applications to bond and currency options. New Haven, USA, 1993.

[Heuser 2008]HEUSER, C. A. *Projeto de Banco de Dados*. Porto Alegre: Bookman, 2008. ISBN 978-8577803828.

[9]KORN, R.; KORN, E.; KROISTANDT, G. Monte carlo methods and models in finance and insurance. CRC Press, Boca Raton, FL., 2010.

[Lesson: All About Sockets]LESSON: All About Sockets. (Accessed 20 august 2013). Available from Internet: <docs.oracle.com/javase/tutorial/networking/sockets/>.

[Schryver et al. 2010]SCHRYVER, C. et al. A new hardware efficient inversion based random number generator for arbitrary distributions. *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, p. 190–195, December 2010.

[Schryver et al. 2011]SCHRYVER, C. de et al. Hardware accelerators for financial mathematics - methodology, results and benchmarking. *Proceedings of the 1st Young Researcher Symposium*, Kaiserslautern, Germany, p. 55–60, February 2011. ISSN 1613-0073.

[Schryver et al. 2011]SCHRYVER, C. de et al. An energy efficient fpga accelerator for monte carlo option pricing with the heston model. *Proceedings of the IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, p. 468–474, December 2011.

[Sommerville 2011]SOMMERVILLE, I. *Software Engineering*. 9. ed. Boston: Addison-Wesley, 2011. ISBN 0-13703-515-2.

[Stock Option Basics]STOCK Option Basics. (Accessed 20 august 2013). Available from Internet: <www.theoptionsguide.com/stock-option.aspx>.