Calculation of abort thresholds for the Beam Loss Monitoring System of the Large Hadron Collider at CERN

Martin Nemcic

Faculty of Environment and Technology

University of the West of England, Bristol

A dissertation submitted for the degree of

BSc (Hons) Software Engineering

April 2012

I declare that the work in this dissertation was carried out in accordance with the Regulations of the University of the West of England. The work is original except where indicated by special reference in the text and no part of the dissertation has been submitted for any other degree. Any views expressed in the dissertation are those of the author and in no way represent those of the University of the West of England. The dissertation has not been presented to any other University for examination either in the United Kingdom or overseas.

SIGNED:.....

Contents

Abstract

The Beam Loss Monitoring (BLM) System is one of the most critical machine protection systems for the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN), Switzerland. Its main purpose is to protect the superconducting magnets from quenches and other equipment from damage by requesting a beam abort when the measured losses exceed any of the predefined threshold levels. The system consist of circa 4000 ionization chambers which are installed around the 27 kilometres ring (LHC).

This study aims to choose a technical platform and produce a system that addresses all of the limitations with the current system that is used for the calculation of the LHC BLM abort threshold values. To achieve this, a comparison and benchmarking of the Java and .NET technical platforms is performed in order to establish the most suitable solution.

To establish which technical platform is a successful replacement of the current abort threshold calculator, comparable prototype systems in Java and .NET were developed. Both systems were benchmarked on performance of the computation of abort threshold values and speed of database storage.

Findings of the study have shown that .NET platform is 12% faster than Java in the computation of the abort threshold values. Furthermore .NET is 20 times faster than Java in the database storage, which is statistically highly significant. However the study also found downfalls of the .NET, interoperability. Java is a cross-platform that may run on Microsoft Windows, Linux and Mac OS, whereas .NET can run only on Microsoft Windows.

The study found that both technical platforms have their strengths and limitations. Additionally, the environment in which the technical platform will be used was found to be important because, in some cases, performance is not the most vital part of the system.

This study has addressed all the limitations of the CERN's calculator approach. The system will be ported into the BLM LHC architecture to help protect the most powerful particle accelerator in the world.

Acknowledgements

I would like to thank Dr. Chris Simons, who supervised me, and helped me for the entire period of the project work.

Eduardo Nebot Del Busto, employee of CERN, who assisted me for the entire period with the BLM LHC abort threshold calculator algorithm.

Bernd Dehning, Annika Nordt, Christos Zamantzas and Stephen Jackson, employees of CERN, who gave me an opportunity and set up a project topic for me.

They were all very kind and helpful.

1. Introduction

European Organization for Nuclear Research (Conseil Europeen pour la Recherche Nucleaire; CERN) is an international organisation which operates the world's largest particle physics laboratory, based in Geneva-Switzerland. According to CERN (2008), "*The organisation was established in 1954 with 20 European member states. Today it employs around 2400 full-time workers, 600 students as well as some 8000 scientist and engineers representing 608 universities and research facilities across 113 nationalities*". CERN's employees conduct micro analysis in many areas such as exploring the structure of atoms to discovering antimatter. CERN conducts a vast number of experiments, the experiments use beams of energetic particles produced in the accelerator complex *(Appendix A; page 71).*

Detectors are used in the experiments in order to document what happens when a beam of particles hits either a fixed target or particles in a similar beam, coming from the opposite direction.

At the lowest energies, the Proton-Synchrotron Booster supplies beams to On-Line Isotope Mass Separator (ISOLDE), a unique source of radioactive isotopes for experiments with applications that range from astrophysics to industry and medicine. The Proton Synchrotron (PS) delivers higher energy protons to two contrasting experiments; Dimension Relativistic Atomic Complex (DIRAC) and Cosmics Leaving Outdoor Droplets (CLOUD). DIRAC tests ideas about the strong fundamental force, while CLOUD finds out how natural high-energy particles might influence the formation of clouds. The PS also provides the protons that create the antiprotons for the Antiproton Decelerator (AD). Physicists are learning more about antimatter in Antihydrogen Laser Physics Apparatus (ALPHA), Atomic Spectroscopy and Collisions using Slow Antiprotons (ASACUSA) and Antihydrogen TRAP (ATRAP) experiments. The Antiproton Cell Experiment (ACE) also uses antiprotons; in this case it assesses their suitability for cancer therapy. CERN (2008)

The Super Proton Synchrotron (SPS) is the second largest particle accelerator at CERN. Hadrons are particles made of quarks, and include nucleons, which are protons and neutrons, of ordinary matter. The SPS primarily investigates hadrons using Common Muon Proton Apparatus for Structure and Spectroscopy (COMPASS).

The Large Hadron Collider (LHC) is CERN's most powerful accelerator consisting of a

27km long ring, 100 meters underground, between the Switzerland and France border *(Appendix B; page 72)*. According to CERN (2008), "*It is a particle accelerator which is used by physicists to study the smallest known particles; the fundamental building blocks of all things*". Two beams of subatomic particles called "hadrons"; protons or lead ions move in opposite directions in the circular accelerator, accumulating more and more energy with each orbit. A number of experiments designed to investigate how particles of matter behave at a new high energy frontier are conducted in the accelerator. These LHC experiments are located at points where the LHC's two beams collide head on.

The experiments at the LHC are all run by international collaborations, bringing together scientists from institutes all over the world. Each experiment is distinct, characterised by its unique particle detector. Two large experiments, A Toroidal LHC Apparatus (ATLAS) and Compact Muon Solenoid (CMS), are based on the general-purpose detectors to analyse the myriad of particles produced by the collisions in the accelerator. They are designed to investigate the largest range of physics possible. Having two independently designed detectors is vital for cross-confirmation of any new discoveries made. Two medium-size experiments, A Large Ion Collider Experiment (ALICE) and Large Hadron Collider beauty (LHCb), have specialised detectors for analysing the LHC collisions in relation to specific phenomena. Two further experiments, TOTal Elastic and diffractive cross section Measurement (TOTEM) and Large Hadron Collider forward (LHCf), are of a much smaller size. They are designed to focus on "forward particles" (protons or heavy ions). CERN (2008)

According to CERN (2008), "*The ATLAS, CMS, ALICE and LHCb detectors are installed in four huge underground caverns located around the ring of the LHC. The detectors used by the TOTEM experiment are positioned near the CMS detector, whereas those used by LHCf are near the ATLAS detector*" *(Appendix C; page 73)*.

The LHC Beam Loss Monitoring (BLM) system is one of the most critical machine protection systems for the LHC. Its main purpose is to protect the superconducting magnets from quenches as well as protecting equipment from damage induced by beam losses. A quench is an abnormal termination of a magnet operation. The system consists of circa 4000 ionization chambers which are installed around the 27km long ring. The signals of the 4000 detectors are integrated over

12 different time intervals. In the case of the measured signal of a single detector exceeding its predefined threshold, the beam will be extracted from the LHC machine immediately. In addition to the 12 different time intervals the signals and thresholds are categorized over 32 beam energy levels, leading to ~1.5 million beam abort threshold values. The BLM threshold is a signal level in the Beam Loss Monitor at which the beam should be dumped (extracted from the LHC) in order to protect the machine element. Dehning et al (2002)

In order to estimate this signal in the BLM it can be decomposed into three parameters, which are obtained from different sources:

• maximum allowed energy density deposits in the protected elements ($\Delta Q_{critical}$); which can be conditioned by the maximum allowed element temperature or absorbed radiation dose. For steady state losses it is typically the power density which is critical.

• the energy density deposited by a single lost proton or ion ($\mathcal{E}$); this energy can be calculated over various volumes, adequate to heat transfer mechanisms and loss time scale. The maximum number.

of lost protons can be expressed by $N_p = \Delta Q_{critical}/\mathcal{E}$.

• the BLM signal generated by a single lost proton (Q_{BLM}).

Using the above parameters the threshold $T(E_{beam}, \Delta t)$ can be expressed according to Dehning et al (2011), *"by Equation 1"*. E_{beam} is the beam energy and Δt is BLM signal integration time or loss time scale.

Equation 1:
$$T(E_{beam}, \Delta t) = Q_{BLM}(E_{beam}) \cdot \frac{\Delta Q_{critical}(E_{beam}, \Delta t)}{\mathcal{E}(E_{beam}, \Delta t)}$$

Often the input to the calculation is just the maximum number of lost protons; therefore the Equation 1 is reduced to:

Equation 2:
$$T(E_{beam}, \Delta t) = Q_{BLM}(E_{beam}) \cdot N_p(E_{beam}, \Delta t)$$

However, the current set-up and procedure for the calculation of LHC BLM abort thresholds is assumed not to be safe and maintainable over the next 20 years due to a number of practical and technical limitations.

Firstly this calculation is written in C++, which is not supported by the BLM LHC architecture. In addition, the software is very difficult to deploy into the live environment, because it needs to be run from the root. A further problem is that, currently, all those people who have access to the subversion (SVN) have the freedom to carry out the calculations and modifications. This therefore means it is hard to keep track of what is being modified and by whom. This is know as a problem of change management.

One reason for this approach not being safe is because it executes root Macros, which are not compiled, and need to be generated every time which poses safety risks. A further problem is that this calculator is not straightforward to operate due to it requiring a manipulation of the code in order to produce the new threshold tables. This therefore limits the number of people who are able to use this calculator, as only those individuals with a clear understanding of the calculator's code structure are able to operate it.

All computations are performed on CERN's PCs and stored together with the application code on the SVN; this highlights a handling limitation of the data as it is not secure. Furthermore this approach is running offline which could result in a future downfall. The calculation may need to be run online in the future so it is always accessible.

Further limitations include the performance and also the storage of the input parameters and computed abort threshold values. Computing the abort threshold values takes too long. One reason for this is that the C++ code is not optimized for high performance, and the code has many bottlenecks and memory leaks. Another reason is that the algorithm is operated from the console and the user is required to manually input all of the information which is very time-consuming. All the input parameters and computed threshold values are stored in files on the server. These files are unorganised and take up valuable storage space on the server which could be used for something else. This storage problem is likely to be addressed in the near future because each year new input data files and computed abort threshold files are created.

Due to the limitations mentioned above, an assessment of the state-of-the-art, with respect to the capabilities of various technical infrastructures relevant to the problem context, is provided in the following chapters.

2. Background

Two technical platforms that are currently the most used for software development are Sun's Java Virtual Machine (JVM) and Microsoft's .NET Framework. They provide programming language that is used for developing software applications. Sun's Java uses Java programming language and .NET uses cross-language operability (for example; C++, C#), however it is only able to run on Windows. Java on the other hand is supported on many different operating systems (for example; Windows, Linux, Mac OS, Android). Java and .NET enable the application deployment process. The general concept of these platforms is to compile source code to the machine-independent intermediate representations. Java compiles byte code and .NET compiles Microsoft Intermediate Language (MSIL). The respective intermediate code is subsequently delivered to the end-user and using either JVM or Just in Time (JIT), it is interpreted or compiled at run-time.

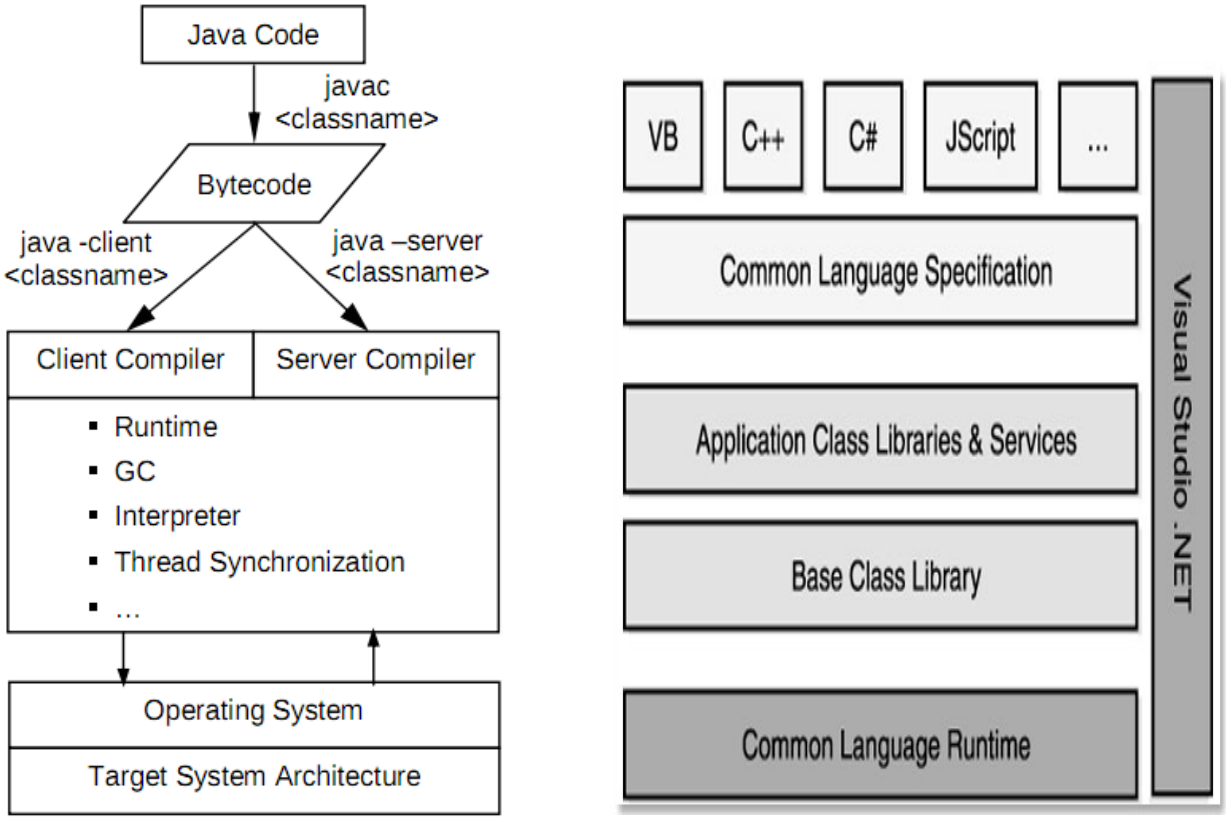


Figure 1: View of the JVM (left) and .NET Framework (right) (Plavec 2005, p.2).

2.1 Java Platform

Java is a technical platform which uses JVM and so can run on different operating systems. Sabharwal (1998) believes that, "*The JVM interprets the byte code into the machine code depending upon the underlying operating system and hardware combination. JVM interprets the input byte code*". Code interpretation is a strategy for performing the code. This is carried out by distinctly executing every instruction of input code. Thiruvathukal (2002) stated that, "*Just-in-time compilers (JITs) are virtual machines used for the compilation of JIT*". JIT compilers accumulate code method by method, as opposed to translating and executing the programme instruction, which is what interpreters do. The JVM is a virtual machine for the Java 2 platform Standard Edition. It consists of two parts; compiler and runtime. Java Runtime Environment (JRE) is a set of tools for programming that consists of a code (byte) interpreter, garbage collector, memory management, machinery for dealing with thread synchronization and other useful tools. Java HotSpot VM compiler translates byte code into native code, which is different to the compiler which translates Java code into byte code. Two editions of JVM are on the market; Java HotSpot VM Server, Java HotSpot VM Client. Plavec (2005) stated that, "*These two versions share a lot of code and are very comparable to one another*" *(Figure 1; page 5)*.

2.2 .NET Platform

Microsoft .NET is a technical platform that supports many programming languages (for example C++, C#, ASP.NET) and runs only on a Microsoft Windows operating system. According to Chengyun Chu (2008), *"Common Language Runtime (CLR) and .NET Framework Class Library (FCL) are the two components that make up .NET Framework platform"*. .NET Framework Class Library is a set of interfaces, reusable classes and also value types, that provide the most important functionality in the development of software application. According to Chengyun Chu (2008), "*The source code is compiled to Microsoft Intermediate Language (MSIL), a machine-independent intermediate language. Instructions for loading, storing, initializing and calling methods on objects are included in MSIL, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handing and other low-level operations*". Support for MSIL code execution is provided by CLR which carries out code checking and compilation, thread management, enhanced security, cross-language integration memory management, cross-language exception handling, profiling services and debugging.

2.3 Java and .NET

2.3.1 Benchmark

In order to examine/compare Java and .NET platforms, the benchmark tool is needed. According to Bull et al (2000), "*The Java Grande Benchmark Suite is a tool for comparing the performance of alternative Java execution environments, in ways which are important to so-called Grande applications*". An application which has large requirements for either memory, bandwidth, processing power, or all of these, is called a Grande application. Grande applications also include computational science and engineering codes, large scale database applications and business and financial models. In order to compare the performance of Java and .NET, Java Grande Benchmark Suite had to be re-written to a programming language that can be compiled to MSIL. In this test, performed by Plavec (2005), "*C# programming language was selected, because of the similarities with Java language*". The Benchmarking of these technical platforms was performed on the system with Intel Pentium 4, 2 GHz processor, 1 GB of RAM, running Windows 2000 Professional (SP 3) operating system. "*Java programs were compiled and run on Java 2 SDK 1.4.1 platform, while .NET Framework SDK v1.0.3705 was used for C# programs*", Plavec (2005).

Plavec (2005) performed 2 types of benchmarking:

- Benchmark programs that measure the performance of low level operations.
- Benchmark programs consisting of short codes, often called kernels, which carry out specific operations frequently used in Grande applications.

Only the first benchmark is relevant to the research question and this is discussed in the following paragraph.

According to Bull et al (2000), "*This benchmark measures the performance of low level operations (arithmetic and maths library operations)*". These benchmarks run for a set amount of time: "*the number of operations executed in that time is recorded and the performance is reported as operations per second*", Plavec (2005).

The benchmarks are:

- Arith - measurement of the performance of arithmetic operations (add, multiply and divide) on the primitive data types (int, long, float and double) where performance units are adds, divides or multiplications per second.
- Cast - tests the performance of casting between different primitive types (int, long, float, double), where performance units are casts per second.
- Create - tests the performance of creating objects and arrays (arrays of different sizes; ints, longs, floats, objects), where performance units are arrays or objects per second.
- Math - measurement of the performance of all the methods in the java.lang.Math class, where performance units are operations per second.
- Serial - measurement of the performance of serialization, both writing and reading of objects from and to a file, where types are arrays, vectors, binary trees and linked lists.

Figure 2; page 9, shows the average performance of low level operations. The first column shows the performance of Java HotSpot Server VM. The second and third columns show the performance of the .NET platform using the JIT compilation and running native, install-time compiled code. All the values are relative to the performance of Java HotSpot Client VM.

The result shows that .NET is, on average, faster than Java for all operations, except casts. Plavec (2005) believes that, "*The reason for poor performance on casts is the internal representation of simple types in .NET platform. Simple types are internally represented as structures, which allows them to be treated as any other objects, but that approach introduces some overhead*". Creating objects in Java is, on average, 5 times slower than in .NET. More detailed examination of performance *(Appendix D; page 74)* shows that Java creates base object (class object) faster than .NET. Plavec (2005) stated that, "*This means that the internal organization of base object class is more complicated in .NET, but overhead pays out when it comes to creating user objects*". The math result shows that Java has a poor implementation of Math library compared to .NET. On average, .NET turns out to be 2.3 times faster than Java HotSpot Client VM.

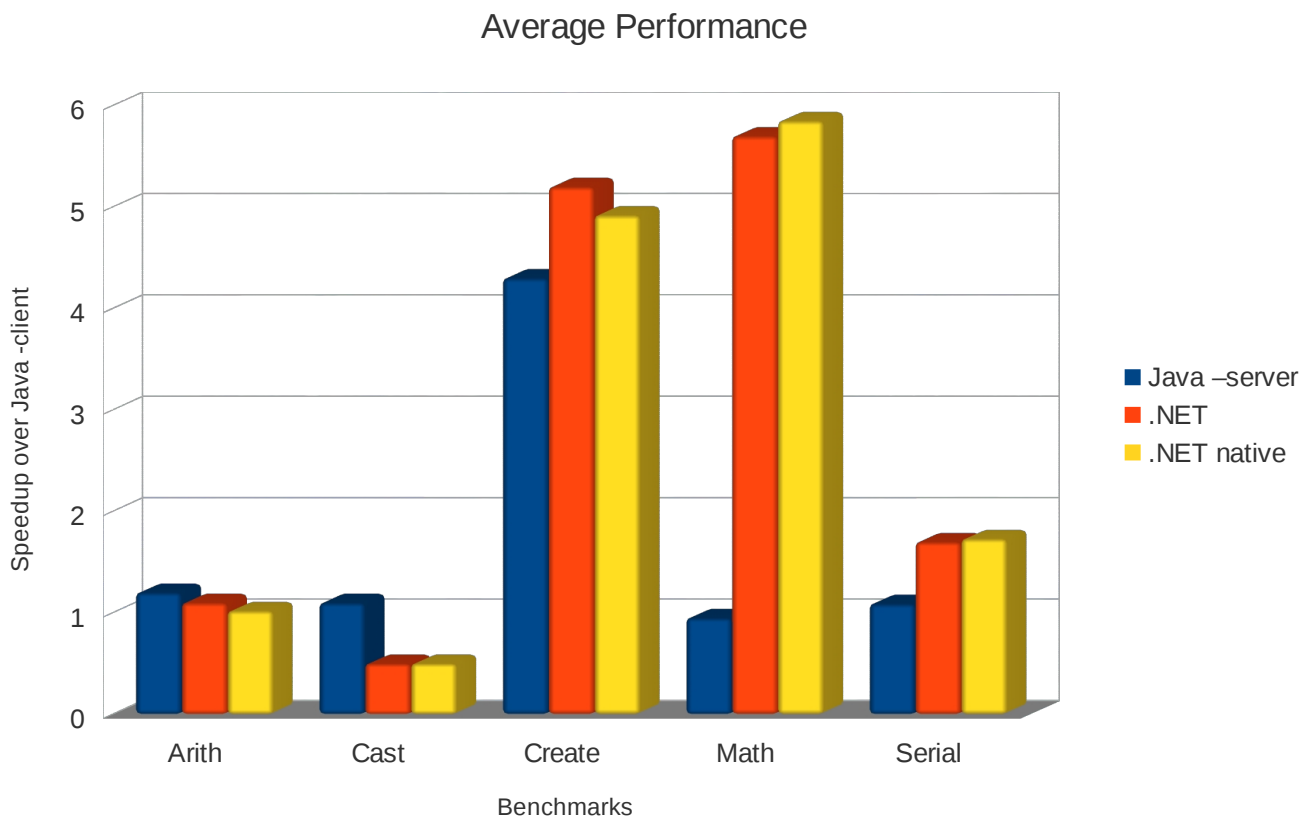


Figure 2: Benchmarks of low level operations (Plavec 2005, p.5).

Performance, in some cases, is one of the most important factors in the technical platforms. Java Grande Benchmark Suite was used to evaluate performance. The benchmarks show that the .NET platform is faster than Java. Plavec (2005) stated that, *".NET platform is 16% faster than Java"*. On the other hand, Java can run on different operating systems (for example Microsoft Windows, Linux, Mac OS) whereas .NET Framework is able to run only on Microsoft's operating system. This fact is a significant limitation of the .NET platform.

Although the material discussed here is from 2005 and technical platforms have moved on since, this does not have any impact on the study that is performed in the current project. Unfortunately, further benchmark research is not readily available in the literature.

2.3.2 Build & Deployment

Partington (2010) stated that, "*Java and .Net platforms are both very easy to build and deploy*". The deployment process is a set of activities that include; release, install and activate, deactivate, adapt, update, built-in, version tracking, uninstall and retire. All these activities help to maintain the software and keep track of what version was last deployed, who it was deployed by, who it was modified by, and when. These tools are extremely useful during the software development process.

2.3.3 Change Management

Both platforms support change management systems. The system is called Version Control System (VCS) but is also known as Subversion (SVN). This system is particularly required when more than one person works on a single project in order to track all changes and avoid mistakes.

The key advantages of SVN are:

- Backup and Restore – Files are stored and can be restored from any date.
- Track Changes – Files can be stored with relevant comments.
- Synchronization – People can share the same files in order to stay up to date with the latest version.

SVN functionality is a great advantage in .NET and Java platforms.

2.3.4 Interoperability

Interoperability is the capability of the system to work with another system. Java does not have any limitations in this integration; it can run on any system in any environment. .NET framework was developed by Microsoft. Microsoft only supports their own products, which run only on their own platforms; Microsoft Windows. .NET is not able to run on different operating systems such as Linux. This is a very significant limitation of .NET, and the main reason why many systems are developed in Java.

2.3.5 Security

Java and .NET share similar design and implementation qualities; however there are crucial differences that consequently also influence their security. Both Java and .Net execute unsafe programs with security restrictions and use VM to load the policies onto the software *(Figure 3)*.

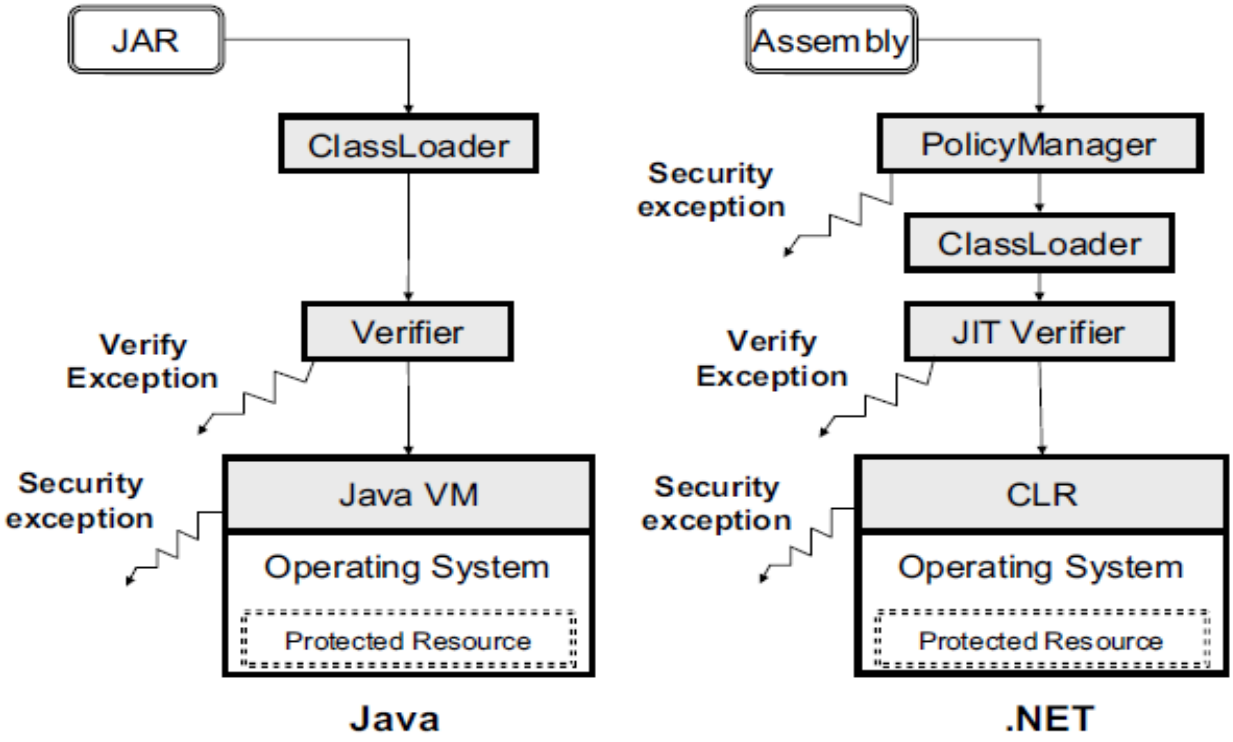


Figure 3: Architecture overview of Java and .NET (Paul and Evans 2004, p.1).

Paul and Evans (2004) stated that, "*The only security provided against untrusted code is what the platform provides.*" *Appendix E; page 75*, shows a summary of Java security vulnerabilities over the past eight years. The Java and .NET framework uses a combination of static analysis and dynamic checking to enforce policies on executing programs. Java uses byte code as a verifier whereas .NET uses JIT; these validate the low-level code properties which are fundamental in terms of memory safety, type safety and control flow safety. It is then possible to execute the programs. All the programs which pass the verifier in JVM or .NET CLR are executed.

According to Paul and Evans (2004), "*Java and .NET achieve low-level code safety through static verification and run-time checks.*" At the load time in Java, static verification is done by byte

code. On the other hand, JIT compilation in .NET performs some parts of the verification.

JVMLL and MSIL are small programs for verifying the code. Java and .NET support static permissions and dynamic permissions. Static permissions are recognised and given at load time. Dynamic permissions remain unidentified until runtime. Attributes of the executing code are authorised, depending on them being allowed by both platforms.

Paul and Evans (2004) believe that, "*.NET does not provide the same level of extensibility as Java in the policy implementation, a developer creating a new permission must still be careful to avoid errors*".

Both platforms have similar mechanisms and security goals. Paul and Evans stated that, "*Several of the specific complexities that proved to be problematic in Java have been avoided in the .NET design*".

Overall, both platforms have security issues, and their goal is to develop more secure systems.

3. Hypothesis

Research has shown that choosing a technical platform is a difficult, multi-objective trade-off. Every technical platform has its strengths and limitations. It depends on what the system is used for, what the system is expected to achieve, what environment is used and also by what users.

It is hypothesised that the benchmarking of .NET and Java platforms will result in the successful and informed choice of replacement of the current BLM abort threshold calculator by the new approach.

It is further hypothesised that the new approach will resolve the most important issues.

4. Project Goals

This project aims to choose a prototype system based on full validation of performance in comparison with the results achieved with the current approach for the BLM abort threshold calculation.

Further project aims include improving and investigating;

- Performance
- Interoperability
- Deployment issues
- Change management issues
- Security of the system
- Graphical User Interface (GUI)

Two comparable prototype systems will be developed. For this to be achieved, a system will be built with the most suitable programming languages which are Java and C#. The prototypes will be developed on a stand-alone basis but it will be easily portable to the BLM LHC software architecture at CERN. The system will be driven by the secure database system, which is used to store input parameters for the computations and computed abort threshold values.

The Java prototype system will be built on hardware configuration with Linux Operating System:
Intel(R) Core(TM) i5 CPU, M 520@2.4GHz processor,
4096 MB of RAM,
Linux Ubuntu 11.10 (Oneric) 64bit Operating System,
Eclipse Platform for Linux 3.7.0,
Java version 1.6.0_23,
OpenJDK runtime Environment (IcedTea6 1.11pre) (6b23~pre11-0ubuntu1.11.10),
OpenJDK 64-bit Server VM (build 20.0-b11, mixed mode),
MySQL 5.5.16.

The C# prototype system will be built on hardware configuration with Microsoft Windows Operating System:
Intel(R) Core(TM) 2 CPU T5200@1.6GHz processor,
1024 MB of RAM,
Microsoft Windows XP SP3 32bit Operating System,
Eclipse SDK Platform for Windows 4.2.0,
Microsoft Visual Studio 2010 Ultimate 10.0.30319.1,
Java version 1.7.0_03,
Java(TM) SE runtime Environment (Build 1.7.0_03-b05),
Java HotSpot(TM) Client VM (Build 22.1-b02, mixed mode, sharing),
MySQL 5.5.20.

A suite of test cases will be produced in order to establish whether the system has made the improvements.

5. Requirements Analysis

The following requirements were established in order to produce a successful system and to avoid all the limitations with the current approach. To perform the abort threshold computation, the system requires configuration files and card files.

Configuration files :

- Element Configuration
- BLM Configuration
- Loss Configuration

These configuration files hold formulae and all the input values needed for performing computations of the abort threshold values.
Card files are the only reference with the name for the 3 configuration files needed for the computation for the specific Element type. Also Card files can hold corrections which might be applied to computed abort threshold values.

Element Type:

- Collimators
- Cold Magnets
- MD Special Cases
- Warm Magnets

Due to the time scale of this project, the following requirements are based only on one Element Type – Cold Magnets. Threshold computation of the Cold Magnets is based on Equation 1:

$$\mathrm{T}(\mathrm{E_{beam}}, \Delta\mathrm{t}) = \mathrm{Q_{BLM}}(\mathrm{E_{beam}}) \cdot \frac{\Delta \mathrm{Q_{critical}}(\mathrm{E_{beam}}, \Delta\mathrm{t})|}{\mathcal{E}(\mathrm{E_{beam}}, \Delta\mathrm{t})}$$

Two prototypes of the system will be developed in order to compare how well each system meets its requirements.

5.1 Feature List - MoSCoW Prioritisation

- MUST have

  - Secure user authentication
  - Graphical User Interface
  - Configuration files management (Add, Delete and View records)
  - Card files management (Add, Delete and View records)
  - Perform abort thresholds computation
  - Database connection

- SHOULD have

  - Log window

- COULD have

  - Editing functionality of the Configuration files and Card files records
  - Library of all computed abort threshold values

- WON'T have (But would like in the future)

  - Perform computation of all Element Types, not only Cold Magnets
  - Plotting data functionality
  - Perform abort thresholds computation on a database level (PL/SQL)

5.2 Functional Requirements

5.2.1 Use Case: Card Files Administration

Perform BLM abort threshold calculator Card files management activities:

- Add Card file
- View/Delete Card file

5.2.1.1 Brief Description

This use-case describes the tasks the system should do in order to maintain the records of the card files in the database. The tasks that are involved allow adding, deleting and viewing all Card files.

5.2.1.2 Actors

Administrator.

5.2.1.3 Flow of Events

1. Basic Flow

When the Administrator executes this use-case, then he/she will choose and perform one of the tasks below:

If the Administrator wants to add a new Card file to the database then:
The system executes Add Card file in extension point 6.1.2.

If the Administrator wants to view/delete a Card file:
The system executes View/Delete Card file in extension point 6.1.3.

Else the use-case ends.

2. Alternative Flows

I. Invalid Data Entered

If the Administrator enters invalid data in any of the fields then:
The system generates an informative message stating that there is an error in the

data entered and encourages the user to re-enter the data.

The Administrator re-enters the data in specified field:

The system returns to a ready state.

5.2.1.4 Pre-Condition

I. Administrator authentication

User must be logged in in order for the successful execution of execution points: 6.1.2 and 6.1.3.

The user will be asked to log in before performing any activity.

II. Adding records

Configuration files must be previously stored in the database in order to successfully execute the execution point 6.1.2.

III. Viewing/Deleting records

Card files must be previously stored in the database in order to successfully execute the execution point 6.1.3.

5.2.1.5 Post-Condition

I. New Card file added

Card file will be added to the database as a result of successful execution of execution point 6.1.2.

II. Viewed/Deleted Card file

Card file will be displayed/deleted from the database as a result of successful execution of execution point 6.1.3.

5.2.1.6 Extension Points

6.1 Maintain records of the Card files

6.1.2 Add new Card file

The user selects 'Card files' tab from the main tab menu. He/she then selects 'Add' tab:

The system displays the Card file details which contains the following fields:

Card Name

Family name

Element Type
Element Configuration
BLM Configuration
Loss Configuration

The user then saves the details entered.
The system saves the data entered in the database fulfilling all validation rules.
The system returns to the main flow of the use-case.

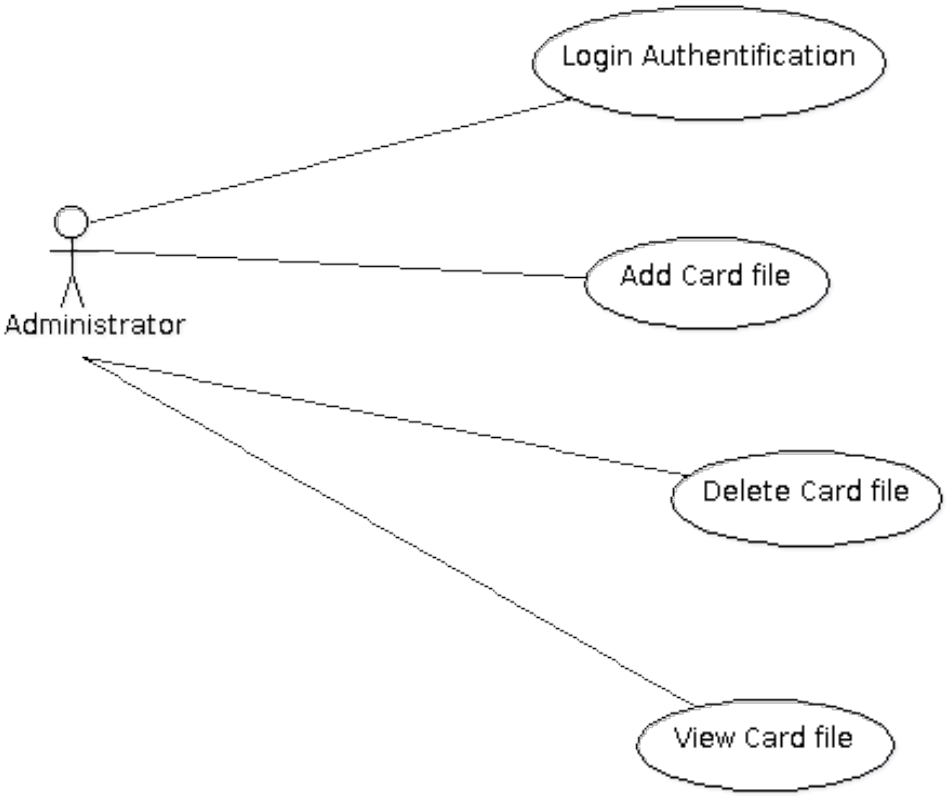
6.1.3 View/Delete Card files

The user selects 'Card files' tab from the main tab menu. He/she then selects 'Library' tab:
The system displays the table of all the Card files with details which contains the following fields:

Card Name
Family name
Element Type
Element Configuration
BLM Configuration
Loss Configuration

The user then selects from the table Card file for removal and clicks the 'Delete' button.
The system deletes the selected Card file from the database and refreshes the table of records.
The system returns to the main flow of the use-case.

5.2.1.7. Use Case Diagram



5.2.2 Use Case: Configuration Files Administration

Perform BLM abort threshold calculator Configuration files management activities

- Add Configuration file
- View/Delete Configuration files

5.2.2.1 Brief Description

This use-case describes the tasks the system should do in order to maintain the records of Configuration files in the database. The tasks that are involved allow adding, deleting and viewing all Configuration files.

5.2.2.2 Actors

Administrator.

5.2.2.3 Flow of Events

1. Basic Flow

When the Administrator executes this use-case, then he/she will choose and perform one of the tasks below:

If the Administrator wants to add a new Configuration file to the database then:
The system executes Add Configuration file in extension point 6.1.2.

If the Administrator wants to view/delete a Configuration file:
The system executes View/Delete Configuration files in extension point 6.1.3.

Else the use-case ends.

2. Alternative Flows

I. Invalid Data Entered

If the Administrator enters invalid data in any of the fields then:
The system generates an informative message stating that there is an error in the data entered and encourages the user to re-enter the data.

The Administrator re-enters the data in specified field:

The system returns to a ready state.

5.2.2.4 Pre-Condition

I. Administrator authentication

User must be logged in in order for the successful execution of execution points: 6.1.2 and 6.1.3.

The user will be asked to log in before performing any activity.

II. Viewing/Deleting records

Configuration files must be previously stored in the database in order to successfully execute the execution point 6.1.3.

5.2.2.5 Post-Condition

I. New Configuration file added

Configuration file will be added to the database as a result of successful execution of execution point 6.1.2.

II. Configuration file viewed/deleted

Configuration files will be viewed/deleted from the database as a result of successful execution of execution point 6.1.3.

5.2.2.6 Extension Points

6.1 Maintain records of the Configuration files.

6.1.2 Add new Configuration file

The user selects 'Configuration files' tab from the main tab menu. He/she then selects 'Add' tab:

The system displays the Configuration files details which contains the following fields:

- **ElementConf**:

BLMConf	**LossConf**
BLM.Name	LOSS.Name
BLM.Type	LOSS.EdepMax_FORM
BLM.RespType	LOSS.EdepMax_NPAR
BLM.RespForm	LOSS.EdepMax_PAR0
MG.Name	
MG.EnLi_FORM	
MG.EnLi_NPAR	
MG.EnLi_PAR0	

MG.EnLi_PAR1	BLM.RespNPAR	LOSS.EdepMax_PAR1
MG.QuLi_FORM	BLM.RespPAR0	LOSS.EdepThermal_FORM
MG.QuLi_NPAR	BLM.RespPAR1	LOSS.EdepThermal_NPAR
MG.QuLi_PAR0	BLM.ConvGy2C	LOSS.EdepThermal_PAR0
MG.QuLi_PAR1	BLM.ConvBit2Gy	LOSS.EdepThermal_PAR1
MG.tauMe_FORM	BLM.Corr_RS01	
MG.tauMe_NPAR	BLM.Corr_RS02	
MG.tauMe_PAR0	BLM.Corr_RS03	
MG.tauMe_PAR1	BLM.Corr_RS04	
MG.tauHe_FORM	BLM.Corr_RS05	
MG.tauHe_NPAR	BLM.Corr_RS06	
MG.tauHe_PAR0	BLM.Corr_RS07	
MG.tauHe_PAR1	BLM.Corr_RS08	
MG.He_frac	BLM.Corr_RS09	
	BLM.Corr_RS10	
	BLM.Corr_RS11	
	BLM.Corr_RS12	

The user then saves the details entered.

The system saves the data entered to the database fulfilling all validation rules.

The system returns to the main flow of the use-case.

6.1.3 View/Delete Configuration file

The user selects 'Configuration files' tab from the main tab menu. He/she then selects 'Library' tab:

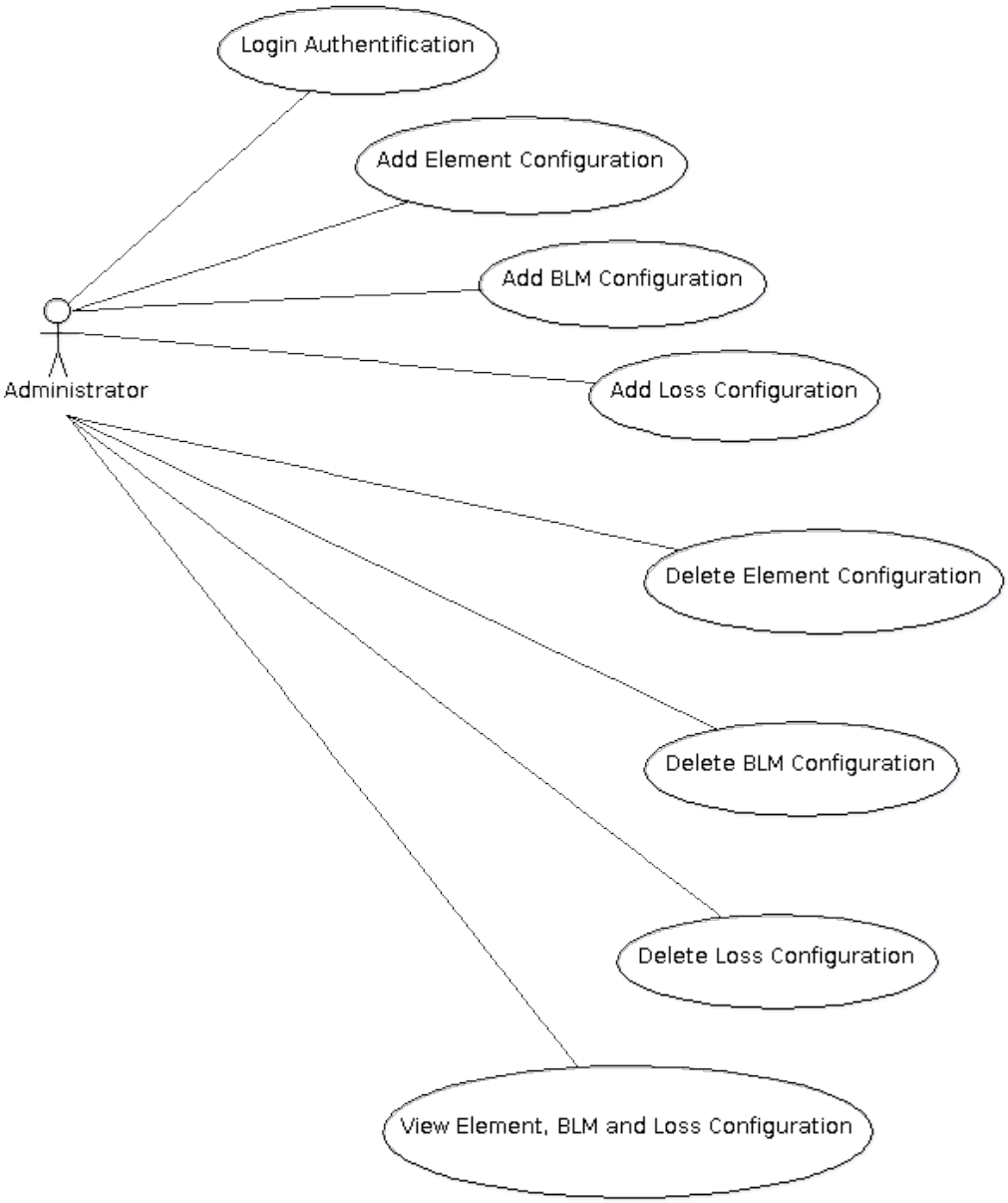
The system displays three tables of the Configuration files (Element Configuration, BLM Configuration and Loss Configuration) with details which contains the following fields:

• **ElementConf**:	**BLMConf**	**LossConf**
MG.Name	BLM.Name	LOSS.Name
MG.EnLi_FORM	BLM.Type	LOSS.EdepMax_FORM
MG.EnLi_NPAR	BLM.RespType	LOSS.EdepMax_NPAR

MG.EnLi_PAR0	BLM.RespForm	LOSS.EdepMax_PAR0
MG.EnLi_PAR1	BLM.RespNPAR	LOSS.EdepMax_PAR1
MG.QuLi_FORM	BLM.RespPAR0	LOSS.EdepThermal_FORM
MG.QuLi_NPAR	BLM.RespPAR1	LOSS.EdepThermal_NPAR
MG.QuLi_PAR0	BLM.ConvGy2C	LOSS.EdepThermal_PAR0
MG.QuLi_PAR1	BLM.ConvBit2Gy	LOSS.EdepThermal_PAR1
MG.tauMe_FORM	BLM.Corr_RS01	
MG.tauMe_NPAR	BLM.Corr_RS02	
MG.tauMe_PAR0	BLM.Corr_RS03	
MG.tauMe_PAR1	BLM.Corr_RS04	
MG.tauHe_FORM	BLM.Corr_RS05	
MG.tauHe_NPAR	BLM.Corr_RS06	
MG.tauHe_PAR0	BLM.Corr_RS07	
MG.tauHe_PAR1	BLM.Corr_RS08	
MG.He_frac	BLM.Corr_RS09	
	BLM.Corr_RS10	
	BLM.Corr_RS11	
	BLM.Corr_RS12	

The user then selects the Configuration file (Element Configuration, BLM Configuration or Loss Configuration) for removal from the table and clicks the 'Delete' button.
The system deletes the selected Configuration file from the database and refreshes the table of records.
The system returns to the main flow of the use-case.

5.2.2.7 Use Case Diagram



5.2.3 Computation of abort threshold values

This describes the tasks the user should do in order to perform the computation of threshold values.

5.2.3.1 Flow of Events

1. Basic Flow

- User selects Element Type from the list.
- System automatically loads Family Name based on the Element Type chosen.
- System automatically loads Card Name and Configuration files, based on the Family Name chosen.
- User clicks 'Compute' button.
- System performs computation of the abort threshold values and displays result.
- User sends values to the database.
- Database stores abort threshold values.

2. Alternative Flows

I. Invalid Data Entered

If the user enters invalid data in any of the fields then:

The system generates an informative message stating that there is an error in the data entered and encourages the user to re-enter the data.

The user re-enters the data in specified field:

The system returns to a ready state.

5.2.3.2 Pre-Condition

I. User authentication

User must be logged in in order to perform computations.

The user will be asked to log in before performing any activity.

II. Performing computations

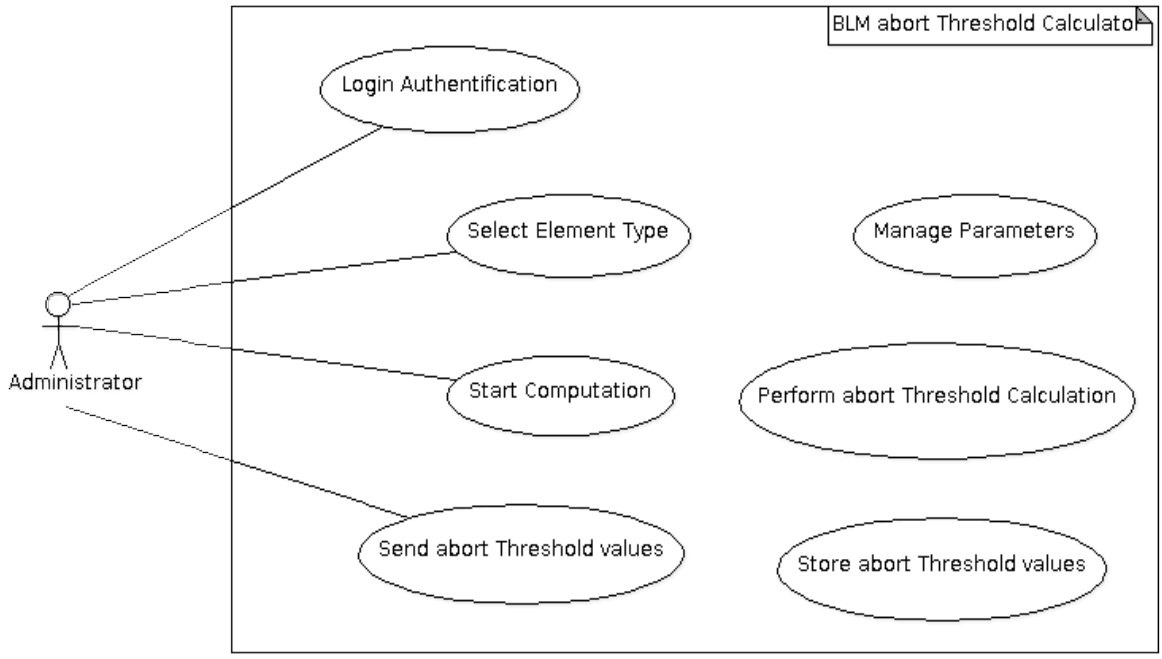
Configuration files and Card files must be previously stored in the database in order to successfully perform the computations.

5.2.3.3 Post-Condition

Successful computation

Abort **t**hreshold values table will be stored in the database.

5.2.3.4 Use Case Diagram



5.3 Non-Functional Requirements

- System will be used by 5 users.
- System allows only one user to be logged in per session.
- Database holds 180 Family Names.
- 180 Card files will be stored in the database.
- Increase of the card files per year is 10.
- 61 Configuration files will be stored in the database.
- Increase of the configuration files is 7 per year.
- Computed abort threshold table consists of 384 integer values.

- **Product**

  a. Reliability of the system 24/7 in order to add, view or delete configuration files and card files from/to the database and perform abort threshold computations.

  b. System needs to be reliable 24/7 to avoid any malfunctions which may cause data loss.

  c. System must allow 24/7 backup and recovery facilities.

  d. The database should be available 24/7 at all times.

- **Security**

  System must be highly secured 24/7 in order to prevent fraud and theft. This can be done by prompting the administrator to enter his/her staff login in order to have access to the system. Back up data every 24 hours in order to avoid loss of data in case of system crash.

- **Accuracy**

  System must be accurate 24/7 and there should not be any discrepancies in the data. Use of validations can avoid the administrator from entering incorrect data.

- **Interoperability**

  System must be able to integrate with 3 systems; MySQL, MS Windows and Linux.

- **Availability**

  System must be available 24/7 in order to perform management activities or computations of BLM abort threshold values.

5.4 Architecture Class Model

Architecture class model *(Figure 4)* shows the relationship between packages of the BLM abort threshold calculator system.

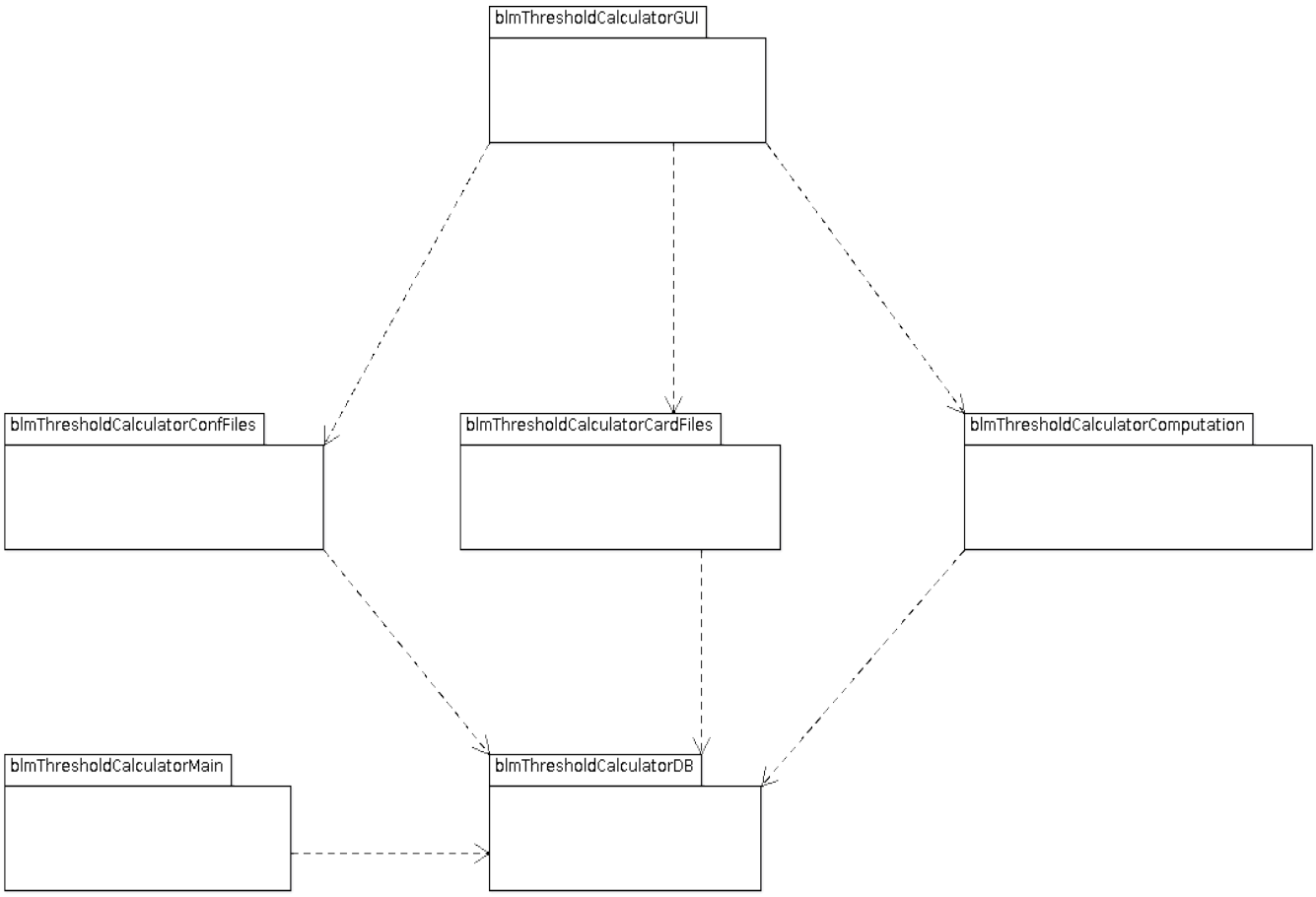


Figure 4: Architecture class model of BLM abort threshold calculator system.

5.5 Logical Class Model

This view (*Figure 5*) addresses the functional requirements of the abort thresholds calculator prototype system. It demonstrates what the system needs to provide in terms of the service to its user. It illustrates key use-case realizations, packages and classes which make up the system. It also illustrate the packages and their logical relationships.

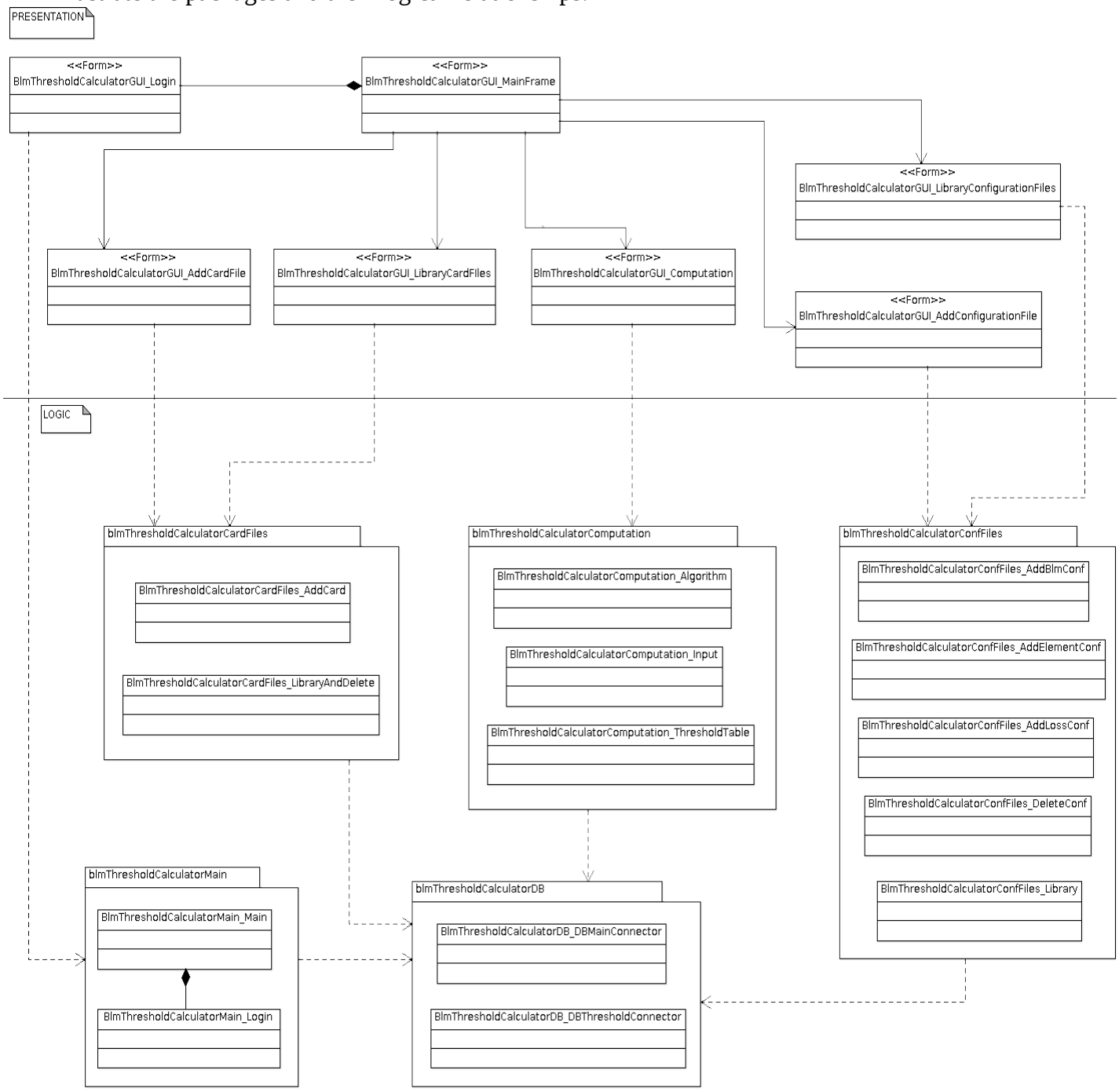


Figure 5: Logical class model of BLM abort threshold calculator system.

5.6 Finite State Machine Diagram

The diagram *(Figure 6; page 33)* shows and describe states of the abort threshold calculator system. The behaviour is analysed and represented in a series of stages:

Start program

Start – start the BLM abort threshold calculator (main Graphical User Interface).

User Input

Login – user is requested to enter relevant user name and password.

Connect to the Database

Connection – the system makes a new connection to the database via JDBC connector.

Login Authentication

Authentication – a request is sent to the database, via the system, which checks whether the data entered matches.

Start Main Menu

Running – system is in running stage and is able to process the following operations:

- Perform abort threshold computation
- Add/Delete/View Configuration files
- Add/Delete/View Card files

Exit System

Logging out – user presses the button to log-out and the system exits.

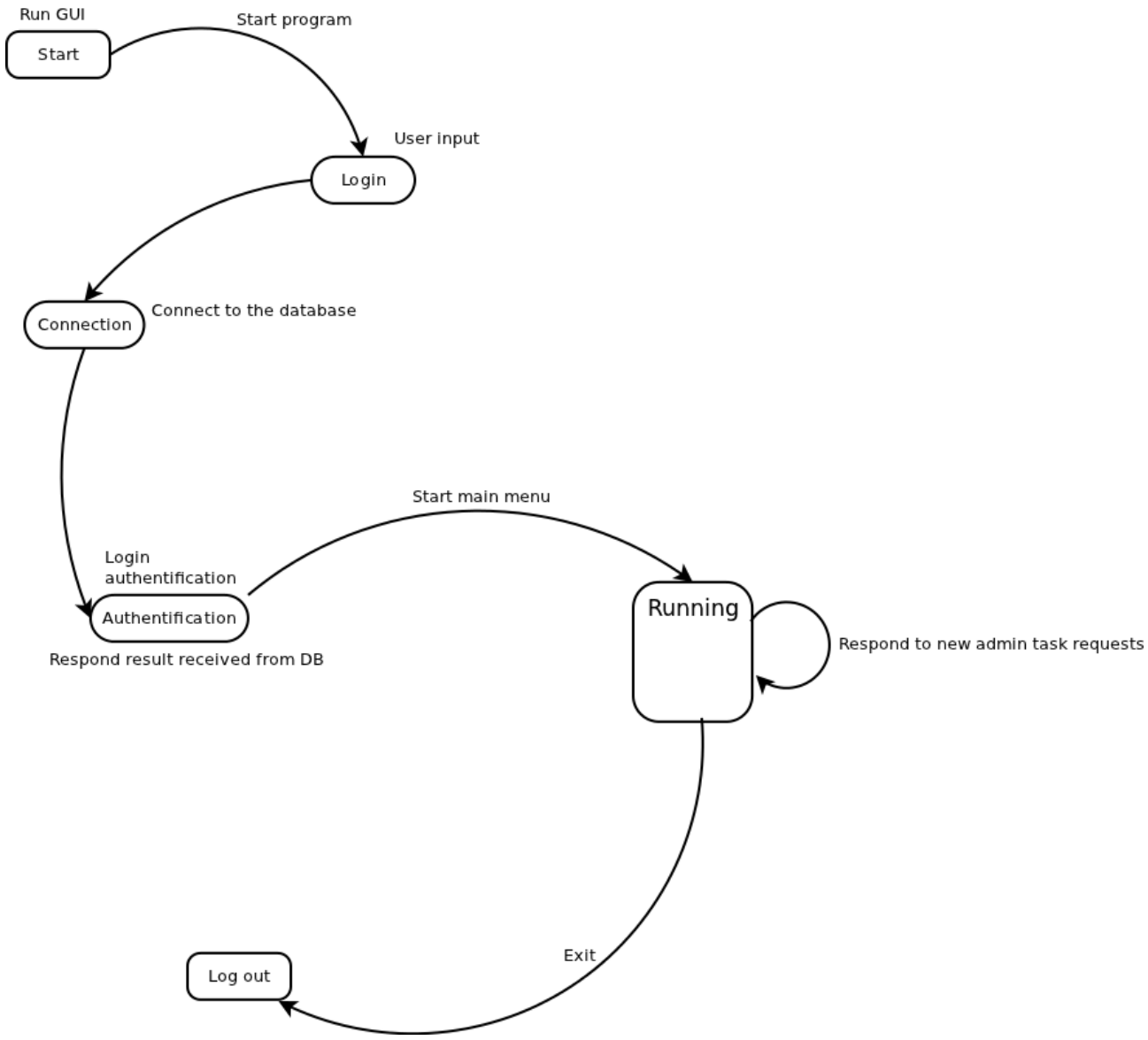


Figure 6: Finite State Machine Diagram of BLM abort threshold calculator system.

5.7 Test Strategy and Benchmarking

In order to establish the necessary level of confidence in the software, testing will be performed. The test will consist of comparing benchmarks achieved from the Java and .NET(C#) prototype systems. A benchmark tool for the comparison of the .NET and Java platforms within the same programming environment is not available off-the-shelf.

Profiler is a tool for measuring performance, performance bottlenecks and also for identifying methods with high execution time in technical platforms. This tool is available in Java's Eclipse Framework and also in Microsoft's .NET Visual Studio 2010. Profiling will be used in order to compare performance of the Java and .NET prototype systems by examining each method of the algorithm for computing BLM abort threshold values.

Another test will be based on the time taken for four different tasks, performed in the Java and .NET prototypes. The four different tests are:

- Computation of the abort threshold algorithm without corrections applied
- Computation of the abort threshold algorithm with corrections applied
- Sending results (abort threshold values) without corrections applied into to the database (write)
- Sending results (abort threshold values) with corrections applied into to the database (write)

For these four measurements, Eclipse's and Visual's studio measurement time library will be used to measure how long each task takes in both platforms. Measurements will be taken in nano seconds to get the most accurate results and then converted to milliseconds to 3 decimal places. The obtained results from both platforms will be compared and displayed in tables, and graphs will also be provided.

Black box testing will also be performed to established if the prototype systems have met all the hypothesised functionality.

The Java prototype system will be developed on a Linux machine and the .NET on a Windows machine (mentioned in "Project Goals" chapter). .NET does not run on a different environment than Microsoft Windows. For these reasons benchmarking of the C# and Java prototype systems will be performed on a Windows machine with configuration:

Intel(R) Core(TM) 2 CPU T5200@1.6GHz processor,
1024 MB of RAM,
Microsoft Windows XP SP3 32bit Operating System,
Eclipse SDK Platform for Windows 4.2.0,
Microsoft Visual Studio 2010 Ultimate 10.0.30319.1,
Java version 1.7.0_03,
Java(TM) SE runtime Environment (Build 1.7.0_03-b05),
Java HotSpot(TM) Client VM (Build 22.1-b02, mixed mode, sharing),
MySQL 5.5.20.

Benchmark Test Case - .NET(C#) and Java prototype systems:

Test	Test Name	Test Description	Test Input
1.	**Performance**	Measurement of the performance (examine each method of the algorithm) in Java and C# prototype systems.	Computed abort threshold values *** *****
2.	**Computation without corrections applied**	50 measurements of the time needed for calculation of BLM beam abort threshold values without corrections applied.	50 times computed abort threshold values without corrections applied *** *****
3.	**Computation with corrections applied**	50 measurements of the time needed for calculation of BLM beam abort threshold values with corrections applied.	50 times computed abort threshold values with corrections applied *** ** *****
4.	**Database - write**	50 measurements of the time needed for storing calculated abort threshold values without corrections applied into the database.	50 times computed abort threshold values without corrections applied *******
5.	**Database - write**	50 measurements of the time needed for storing calculated abort threshold values with corrections applied into the database.	50 times computed abort threshold values with corrections applied *******

***** *Appendix F; page 76* – Input test data for the abort threshold algorithm.

****** *Appendix G; page 77* – Abort threshold algorithm corrections input test data.

******* *Appendix H; page 78* – Computed abort threshold table.

Black-box Test Cases – Java and .NET(C#) prototype systems:

Test	Test Description	Test Input	Test Expected Output	Test Result
1.	**User login**	Login: martin Password: 1985	Successful login	**Pass/Fail**
2.	**Storing Element Configuration file into the database**	9 double values 4 integer values 4 formulae *	Element Configuration stored into the database	**Pass/Fail**
3.	**Storing BLM Configuration file into the database**	16 double values 13 integer values 2 formulae *	BLM Configuration stored into the database	**Pass/Fail**
4.	**Storing Loss Configuration file into the database**	4 double values 2 integer values 2 formulae *	Loss Configuration stored into the database	**Pass/Fail**
5.	**Displays library of all Configuration files**	Stored Configuration files into the database	Displayed Element, Blm and Loss Configurations	**Pass/Fail**
6.	**Delete Element Configuration file from the database**	Added Element Conf. to the database	Deleted Element Conf. from the database	**Pass/Fail**
7.	**Delete BLM Configuration file from the database**	Added BLM Conf. to the database	Deleted BLM Conf. from the database	**Pass/Fail**
8.	**Delete Loss Configuration file from the database**	Added Loss Conf. to the database	Deleted Loss Conf. from the database	**Pass/Fail**
9.	**Add Card file into the database**	Card name 1 Element Conf. 1 BLM Conf. 1 Loss Conf.	Card file added to the database	**Pass/Fail**
10.	**Displays library of all Card files**	Stored Card files in the database	Displayed all the Card files from the database	**Pass/Fail**
11.	**Calculate abort threshold values without corrections applied**	78 double values 23 integer values *	32*12 computed abort threshold values ***	**Pass/Fail**
12.	**Calculate abort threshold values with all corrections applied**	82 double values 50 integer values 2 boolean values * **	32*12 computed abort threshold values ***	**Pass/Fail**
13.	**Displays preview of the computed abort threshold values**	Calculate abort threshold values *	Displayed preview of thresholds table ***	**Pass/Fail**
14.	**Stores abort threshold values without corrections into the database**	Calculate abort threshold values *	Data stored into the database	**Pass/Fail**
15.	**Stores abort threshold values with corrections into the database**	Calculate abort threshold values *	Data stored into the database	**Pass/Fail**
16.	**Log window prints all appropriate messages**	Compute/add/del operation performed	Printed appropriate informative message	**Pass/Fail**
17.	**Handling all SQL exceptions**	Empty data field input	Informative message displayed	**Pass/Fail**

* *Appendix F; page 76* – Configuration files input test data / abort threshold algorithm test data.

** *Appendix G; page 77* – Abort threshold algorithm corrections input test data.

*** *Appendix H; page 78* – Computed abort threshold table.

6. Design

The BLM abort threshold calculator prototype systems are designed in Java and C#. Java's Eclipse Framework and Microsoft's Visual Studio Framework is used with MySQL database server.
JDBC is an Application Programming Interface (API) for the Java and .NET platforms, which provides and defines how the client accesses a database. This API will be used as it provides connectivity between the Java, .NET and SQL databases in the prototype systems.

6.1 User interface

Graphical User Interface (GUI), also sometimes called human-computer interface, uses icons, windows, menus and buttons which are operated by a mouse. GUI helps users to interact with the system in a very simple way.
The abort threshold calculator user interface is developed on the following principles, Galitz (2007):

- **The structure principle**

  User interface (UI) is arranged in a way that associated items are grouped together and unrelated items are separated.

- **The simplicity principle**

  UI is very simplified and easy to operate. In case of problems, a help facility is provided.

- **The visibility principle**

  All the functionality of the abort threshold calculator is accessible through the user interface.

- **The feedback principle**

  The user is informed through a log window about successful operations or errors.

- **The reuse principle**

  Design is established with the same naming for the same operations with different objects in order to reduce ambiguity.

6.1.1 Structure

A “Login” screen is displayed, followed by the main menu. The main menu is separated into a “Calculation” panel and two tab menus: “Card Files” and “Configuration Files”. Both tab menus contain 2 sub-menus; “Library” and “Add”.

All the menus cover the main functionality of the system. A very easy and simple navigation of the system is provided. The user can constantly access any part of the menu through the main menu located at the top *(Figure 7)*.

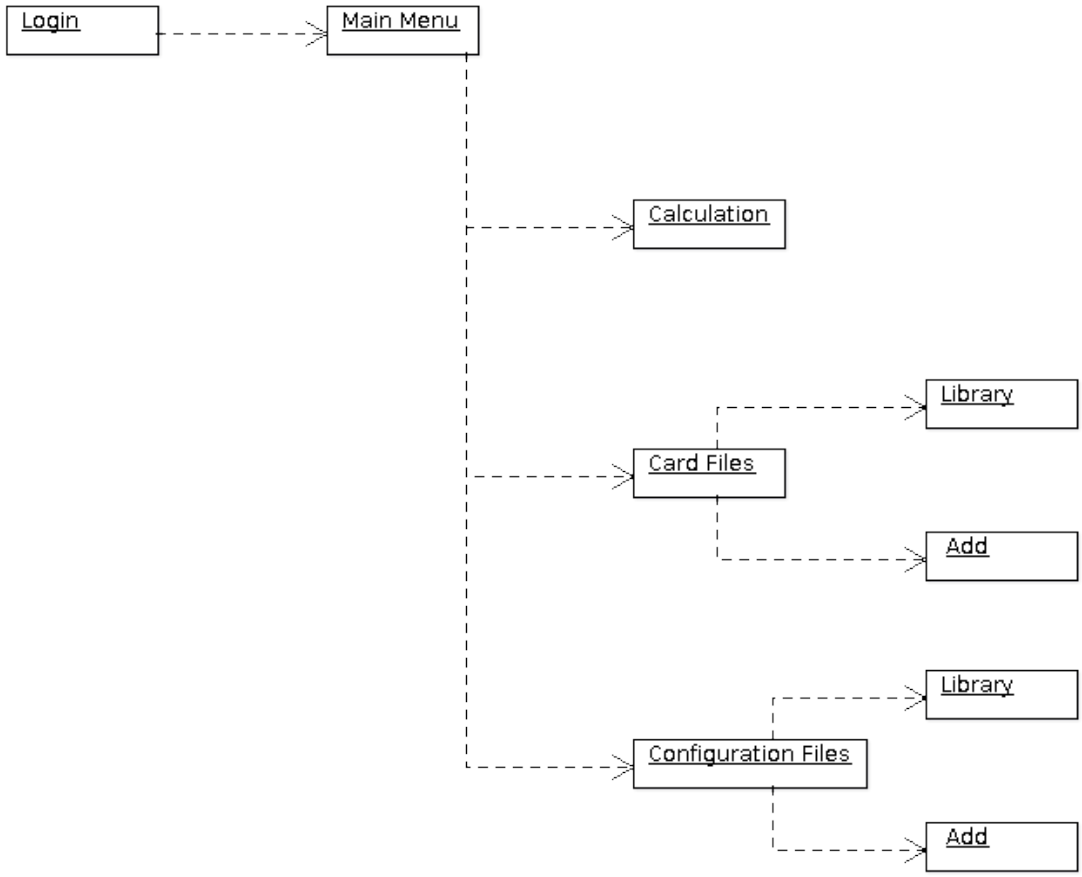


Figure 7: BLM abort threshold calculator system Graphical User Interface menu structure.

6.1.1.1 Description and Actions

- **“Login”**

  Requires user login and password. After successful login, the application starts.

- **“Main Menu”**

  Is constantly presented at the top of the screen and covers all the menus that provide the main functionality of abort threshold calculator system.

- **“Calculation”**

  This panel has a core functionality for computation of abort threshold values. It also has functionality for sending computed results to the database.

- **“Card Files”**

  This menu has another two sub-menus for management of the card files.

- **“Card Files – Library”**

  Displays tables with all details, which contains all the card files stored in the database. User can also perform the delete action.

- **“Card Files – Add”**

  Contains all required operations for performing adding card files.

- **“Configuration Files”**

  This menu has another two sub-menus for the management of the configuration files.

- **“Configuration Files – Library”**

  Displays three different tables with all details, that contains all configuration files stored in the database. User can also perform the delete action.

- **“Configuration Files – Add”**

  Contains all required operations for performing adding configuration files.

6.2 Class Diagram

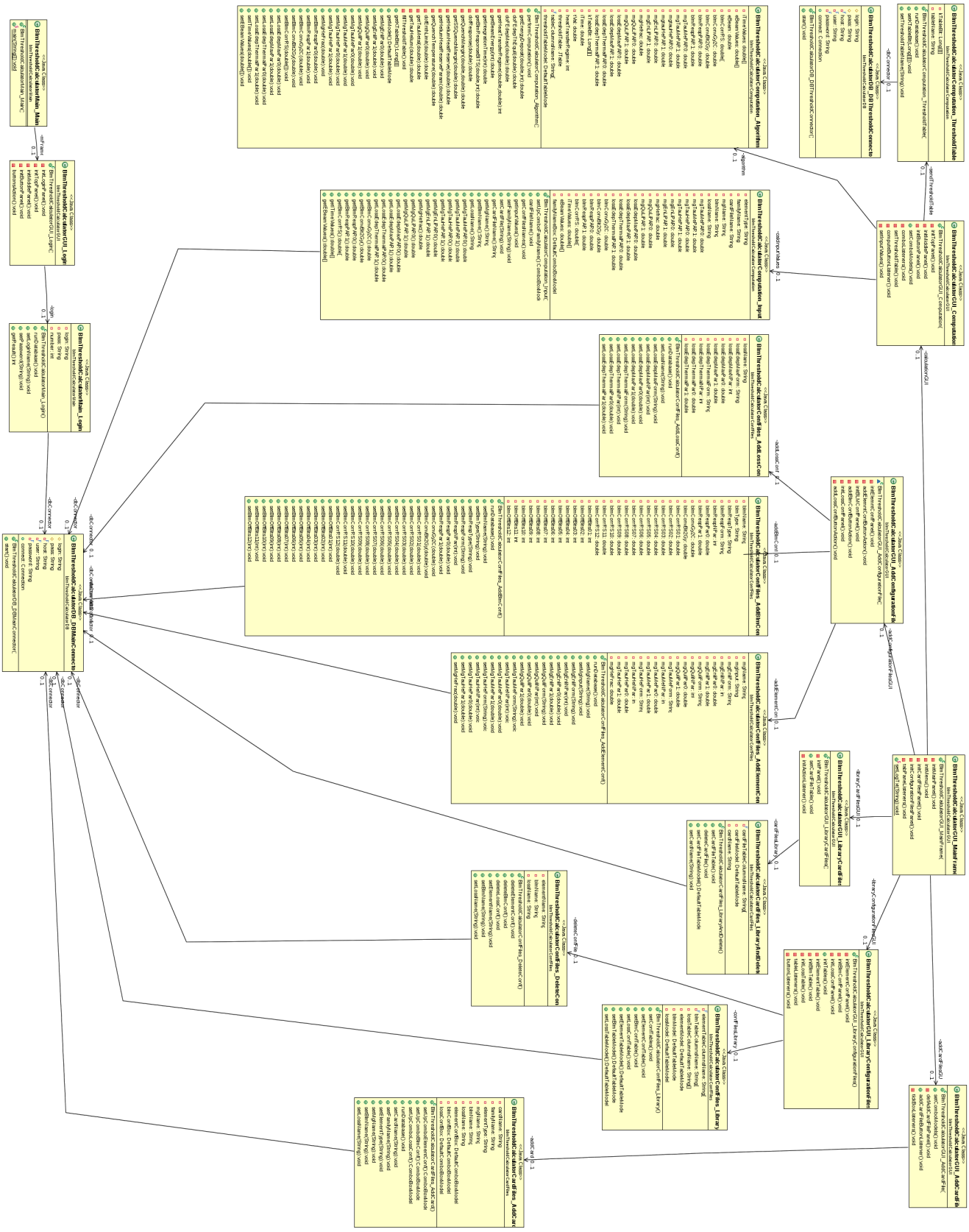


Figure 8: Class diagram of the BLM abort threshold calculator system.

The class diagram *(Figure 8; page 40)* shows the BLM abort threshold calculator design.

The classes have a unique name for easy manipulation. The class name starts with the package name and is followed by an underscore and the name of the class.

- **Package name:** blmThresholdCalculatorComputation
- **Class name:** BlmThresholdCalculatorComputation_Algorithm

This is to show that parts of a module are grouped together because they all contribute to a single well-defined task of the module. These classes are classified as highly cohesive.

Classes are designed to reduce the independences between components and also to ensure all the variables are properly encapsulated. The system is very maintainable and very stable.

6.3 Sequence Diagrams

The diagrams represent the flow of messages, events and actions between the components of the BLM abort threshold calculator system when the user performs Configuration Files Administration *(Figure 9)* and Card Files Administration *(Figure 10; page 43)*.

6.3.1 Configuration Files Administration

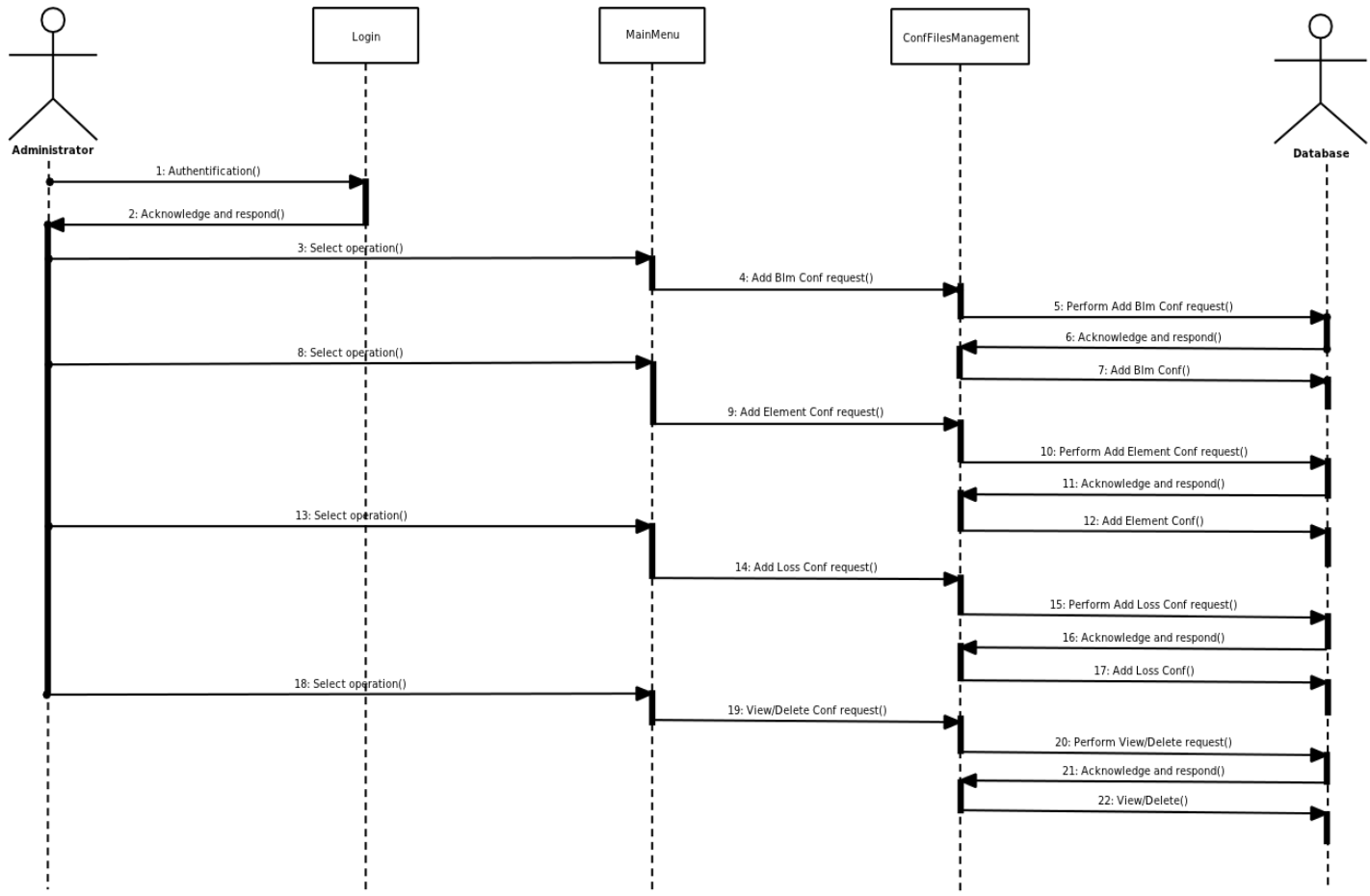


Figure 9: Configuration files administration sequence diagram.

6.3.2 Card Files Administration

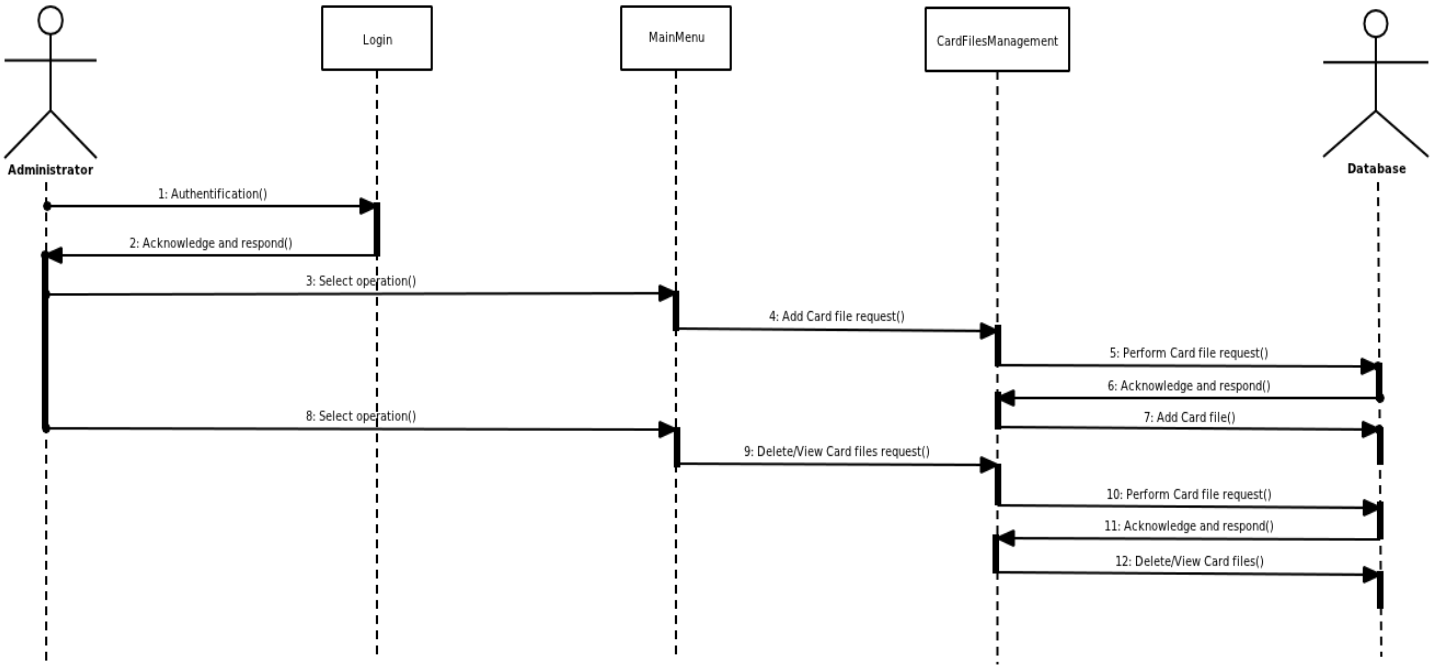


Figure 10: Card files administration sequence diagram.

6.4 Flowchart Diagram

This diagram *(Figure 11)* defines and analyses the processes and shows a step-by-step picture of how the abort threshold calculator prototype system works.

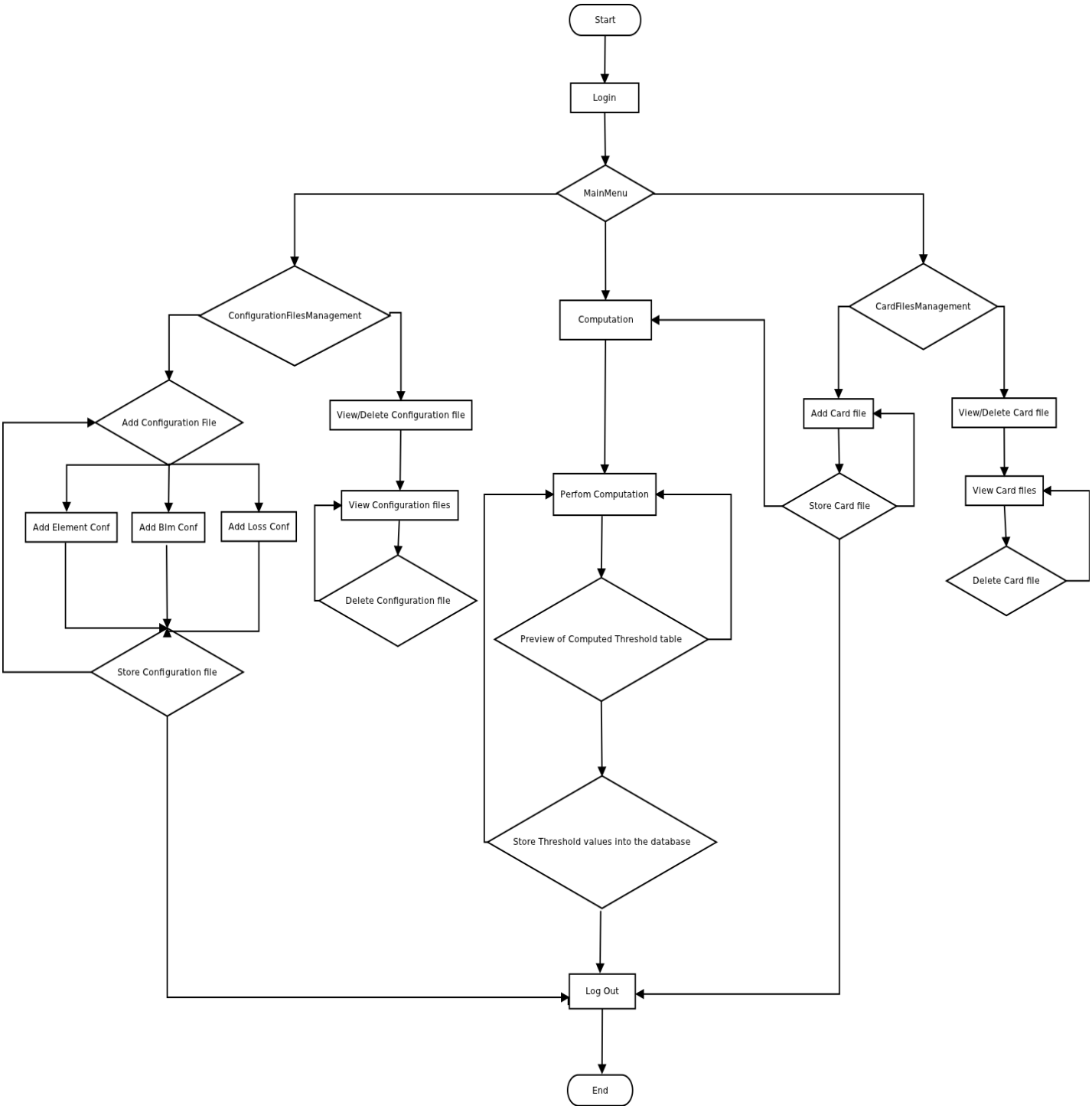


Figure 11: BLM abort threshold calculator flowchart diagram.

6.5 Entity Relationship Diagram

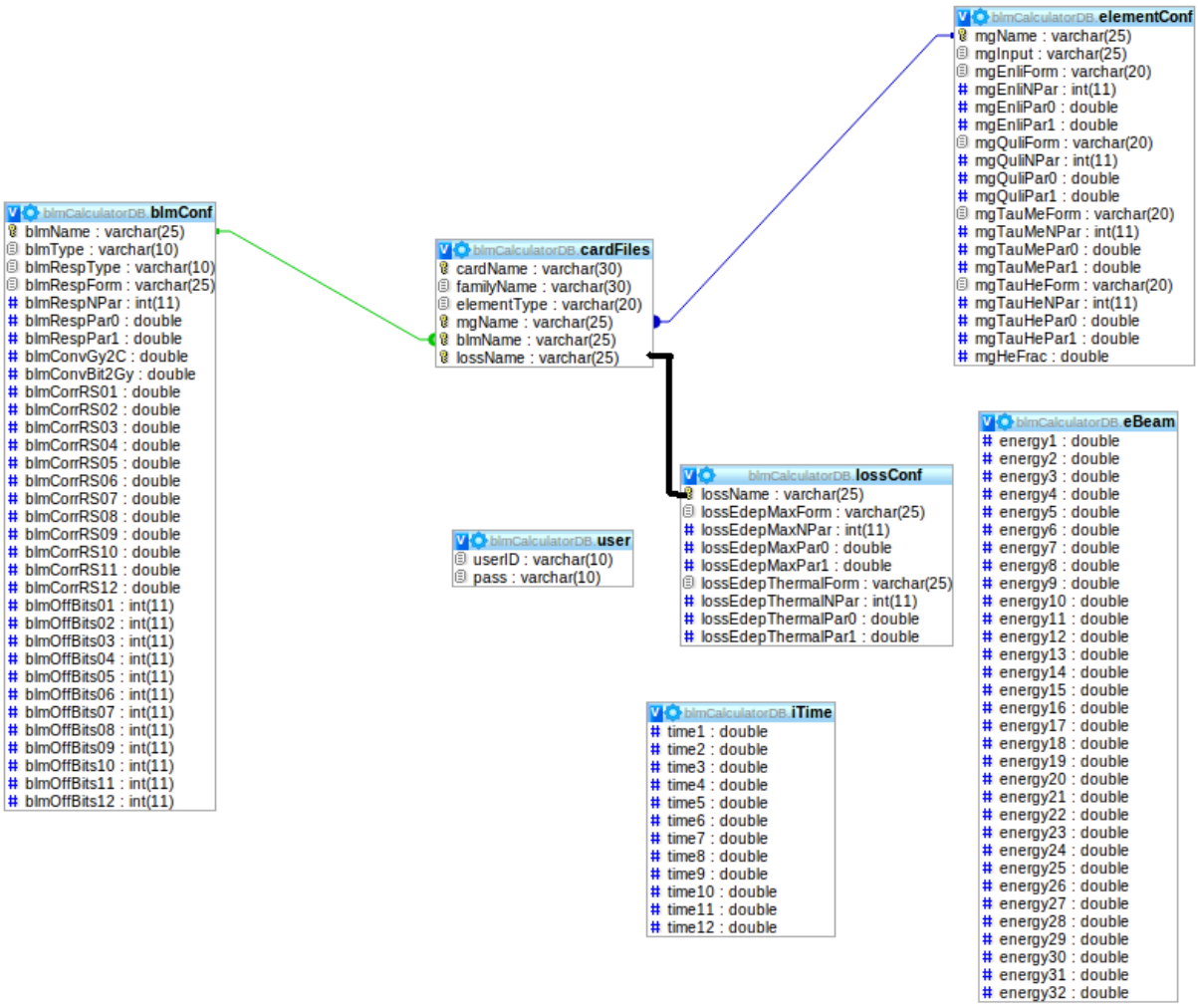


Figure 12: BLM abort threshold calculator entity relationship diagram.

The BLM abort threshold calculator database architecture *(Figure 12)* of the tables is produced with high cohesion and low coupling to reduce high dependency among each other. It holds 7 tables:

Table Name	Table Description
cardFiles	card name, family name, element type and foreign keys (names) to configuration files: mgName (Element conf), lossName and blmName.
blmConf	configuration file name as a primary key and input values for a computation.
elementConf	
lossConf	
eBeam	energy values used for a computation
iTime	integration time values for a computation
user	user name and password.

6.6 Test Data

In order to establish if the software design has met its requirements, specific test data were provided in chapter 5.6 (Test Strategy and Benchmarking) to ensure that the systems works correctly. This test data will be used for 50 tests performed to get the most accurate results. The computed abort threshold values from both prototype systems will be compared with the results obtained from CERN to ensure that the algorithm was implemented correctly. Below is the analysis of the input data for the abort threshold algorithm.

Specification of configuration files data types *(Appendix F; page 76)*:

Configuration file name	Double	Integer	Formulae
Element	9	4	4
BLM	16	13	2
Loss	4	2	2

Specification of card files data types:

	String values	Element conf	BLM conf	Loss conf
Card File	4	1	1	1

Specification of algorithm corrections data types *(Appendix G; page 77)*:

Correction	String	Double	Integer	Big Integer	Boolean
IL	-	2	-	-	-
RC	-	1	-	-	-
Correct Decrease	-	-	-	-	2
Min Bits	-	-	12	-	-
Max Bits	-	-	-	12	-
Ad Hoc	1	12	-	-	-
Scale	-	1	-	-	-

Calculation of the abort threshold values can be done with or without applying a correction. Below are the discussed input data types needed for a computation:

	Double	Integer	Big Integer	Boolean
Calculation without corrections	78	23	-	-
Calculation with corrections	82	38	12	2

The computed abort threshold table consists of 384 integer values *(Appendix H; page 78)*.

7. Implementation

Implementation was carried out in Java and the prototype system was developed. The intention of this project was to produce a comparable prototype system in .NET(C#). Unfortunately this was not possible due to a lack of prior experience with C# and the limited time scale of the project. However, this does not affect the project or the project's results in any way. The prototype system in C# was developed as a console based application with the most important functionality; computation of abort threshold values and storage of these results to the database.

7.1 Software Version Control

Iterative and rapid prototyping supported with Software Version Control (SVC) was used. SVC allows an examination of the changes that have been made in case there is an occurrence of unstable system behaviour after implementing new functionality.

Version	Implemented functionality
1.0	• Threshold computation database created, “user” table added • JDBC connection established • Login window designed • Login functionality implemented • Main frame designed and created
1.1	• Log window added to the main frame • Main menu designed • Add Configuration files panel designed and implemented • Element, Blm and Loss database tables created • View Configuration files panel designed and implemented
1.2	• Add Card file panel designed and implemented • Card file database table created • View Card file panel designed and implemented • Log window functionality implemented
1.3	• Calculation algorithm without corrections applied implemented • Energy and Integration tables added to the database • Threshold results database created • Delete functionality for Card files and Configuration files implemented
1.4	• Calculation functionality implemented • Preview Threshold results functionality implemented • Send results to the database functionality implemented

7.2 Algorithm

Computation of the abort threshold values in the prototype systems is based on Equation 1:

$$T(E_{beam}, \Delta t) = Q_{BLM}(E_{beam}) \cdot \frac{\Delta Q_{critical}(E_{beam}, \Delta t)}{\mathcal{E}(E_{beam}, \Delta t)}$$

The implemented algorithm consists of two "for" loops; 32 Energy values and 12 Integration Time values.

$Q_{BLM\ (Ebeam)}$ - is expressed as: tVal = getBlmResponseBITS(eBeam, it);

$\Delta Q_{critical(Ebeam,\ \Delta t)}$ - is expressed as: tVal *= getQuenchMargin(eBeam, iTime);

$\mathcal{E}_{(Ebeam,\ \Delta t)}$ – is expressed as: tVal /= getEnergyDeposit(eBeam, heatTransferRegime);

T(Ebeam, Δt) - is expressed as: kTableBit[ie][it] *(Appendix H; page 78)*

----------**BLM Abort Threshold Algorithm**----------

```
//for loop; Energy start
for (int ie = 0; ie < 32; ie++) {
        eBeam = getBeamEnergy(ie); //get Energy values from the DB

        //for loop; Integration Time start
        for (int it = 0; it < 12; it++) {

            iTime = getIntegrationTime(it); //get Integration Time values from the DB
            tVal = getBlmResponseBITS(eBeam, it); //QBLM (Ebeam)
            tVal *= getQuenchMargin(eBeam, iTime); //ΔQcritical(Ebeam, Δt)
            heatTransferRegime = getHeatTransferRegime(eBeam,iTime);
            tVal /= getEnergyDeposit(eBeam, heatTransferRegime); //ℰ(Ebeam, Δt)
            tVal *= 3;

        kTableBit[ie][it] = (new Double(tVal)).longValue(); //T(Ebeam, Δt)

        }//for loop;  Integration Time end
    }//for loop; Energy end
```

----------**BLM Abort Threshold Algorithm**----------

7.3 Implementation Screen Shots

7.3.1 Login

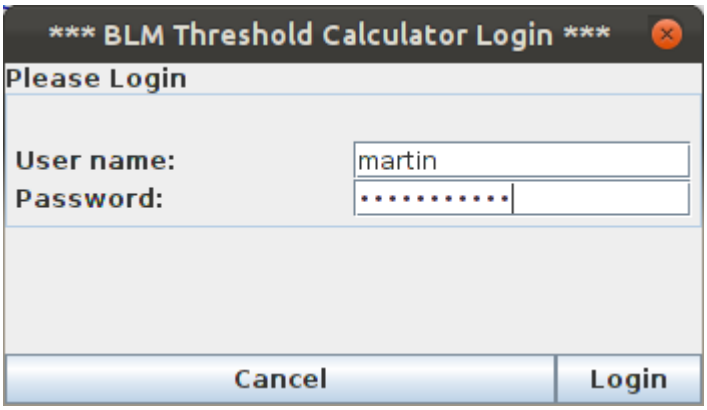


Figure 13: BLM abort threshold calculator login screen.

The login screen *(Figure 13)* appears on the start of the application. The user is required to enter their user name and password. When the 'Cancel' button is pressed, the application closes. When the 'Login' button is pressed, an SQL connection to the database is created and the data are checked against the login data stored in the database. If the data are correct, the application starts. If the data are incorrect, an informative window with a message is displayed *(Figure 14)*.

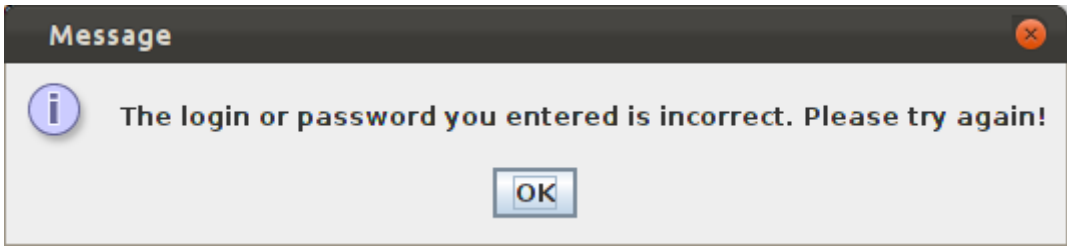


Figure 14: Incorrect name or password window.

7.3.2 Add Configuration File

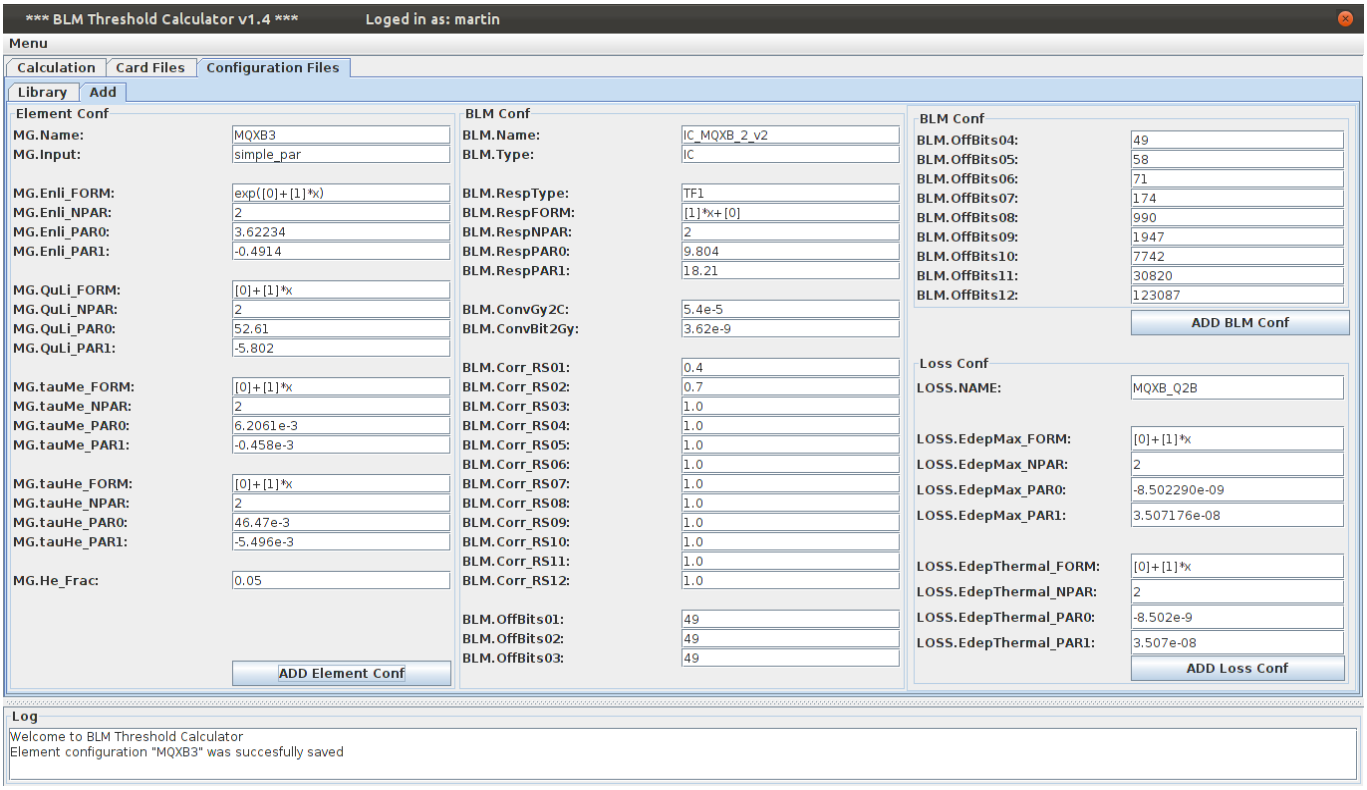


Figure 15: Add Configuration File panel.

"Add" Configuration file *(Figure 15)* allows 3 different configuration files to be added (Element, Blm and Loss). When any of the 'ADD' buttons is pressed, the SQL connection is created and data are sent and stored in the database. The log window displays an informative message *(Figure 16)*.

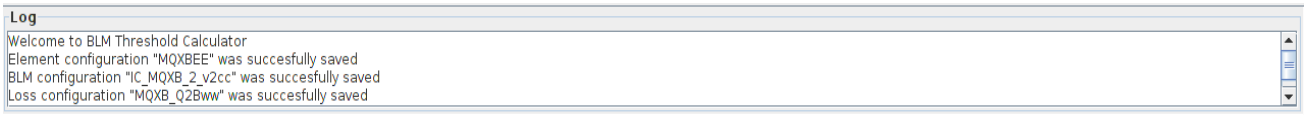


Figure 16: Log window.

7.3.3 Configuration Files Library

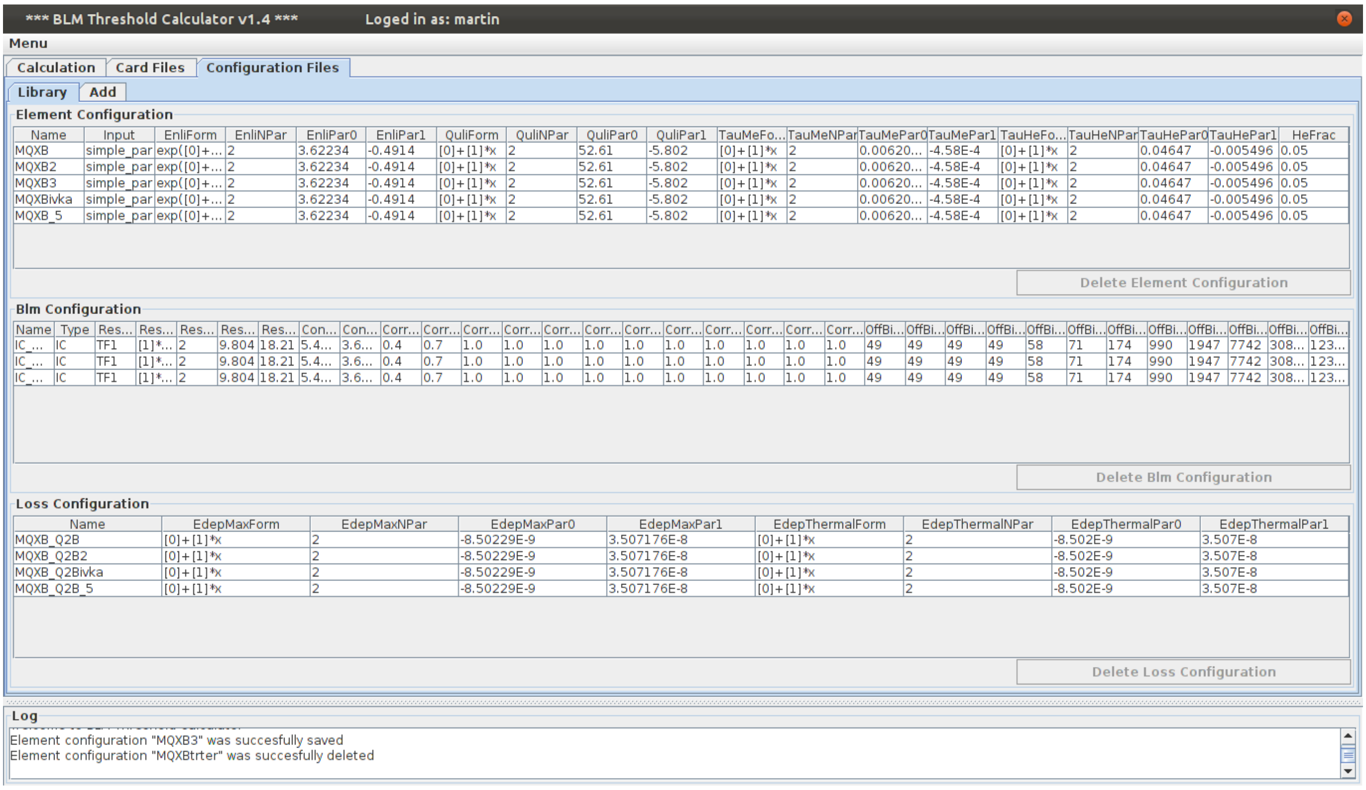


Figure 17: Configuration Files Library panel.

The Configuration Files Library *(Figure 17)* shows all the configuration files with all the details that are stored in the database. The user is able to perform the delete function by selecting the appropriate configuration file and pressing the appropriate delete button. When the delete button is pressed, the selected record is deleted from the database and the table of records is refreshed. An informative message about the performed operation is displayed on the log window.

7.3.4 Add Card File

7.3.4.1 Without Corrections

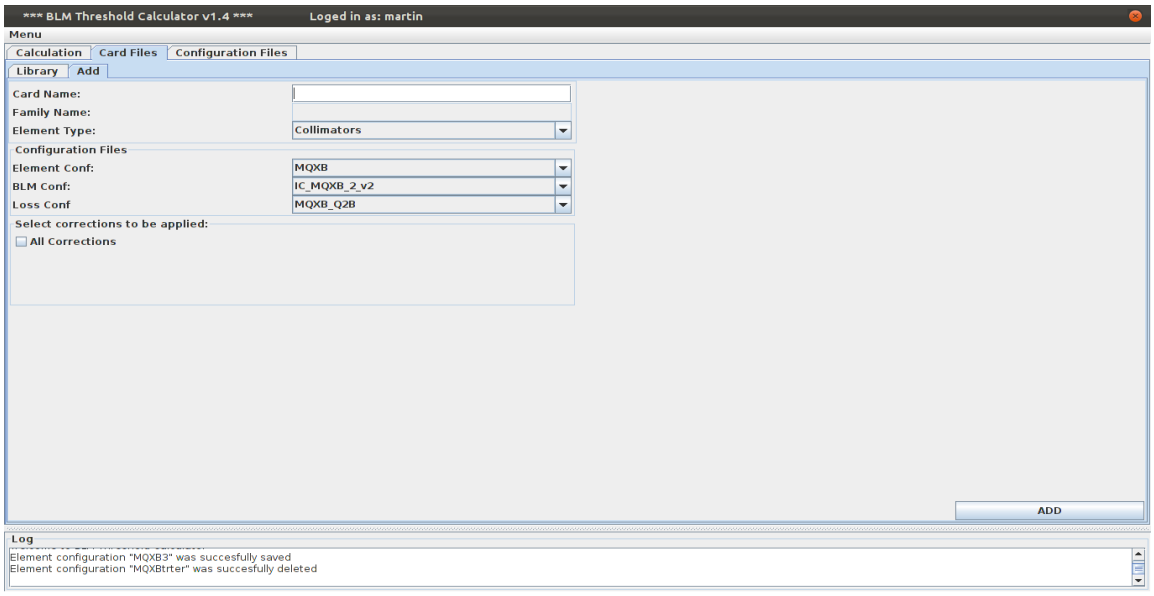


Figure 18: Add Card file with no corrections panel.

Add Card file with no corrections *(Figure 18)* allows the user to select the appropriate configuration files, Element type and give a card name. When the 'ADD' button is pressed, all the entered data are stored in the database and an informative message is displayed on the log window.

7.3.4.2 With Corrections

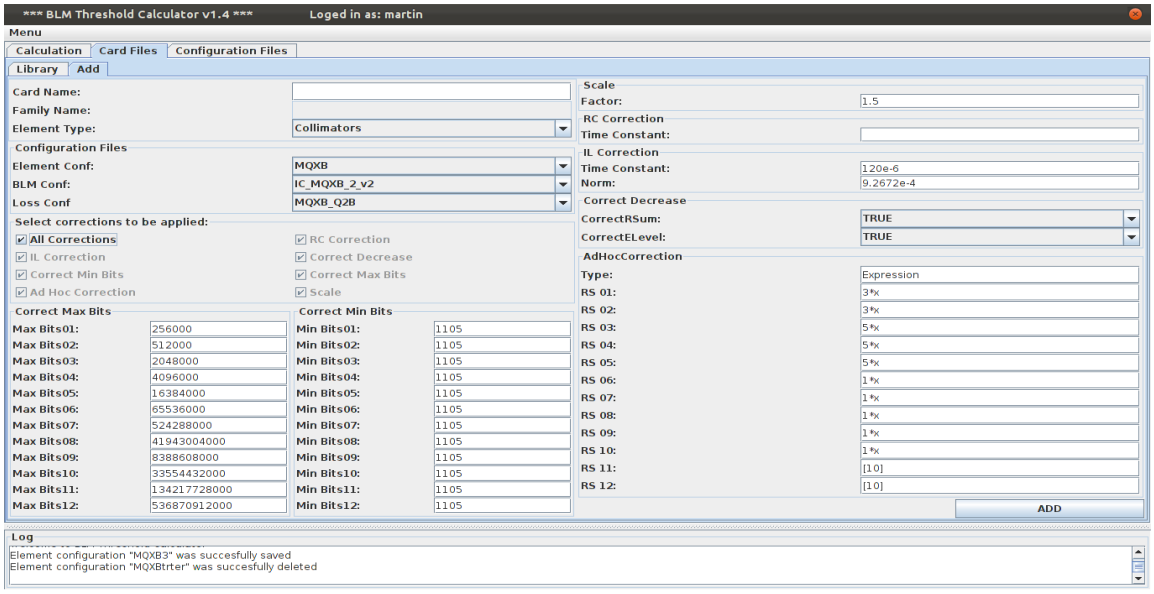


Figure 19: Add Card file with corrections panel.

Due to the time scale of this project, Add Card file with corrections *(Figure 19; page 52)* functionality is not implemented. The abort threshold algorithm with corrections applied was built only as a console based application in the Java and .NET prototypes for testing and benchmarking purposes.

7.3.5 Card Files Library

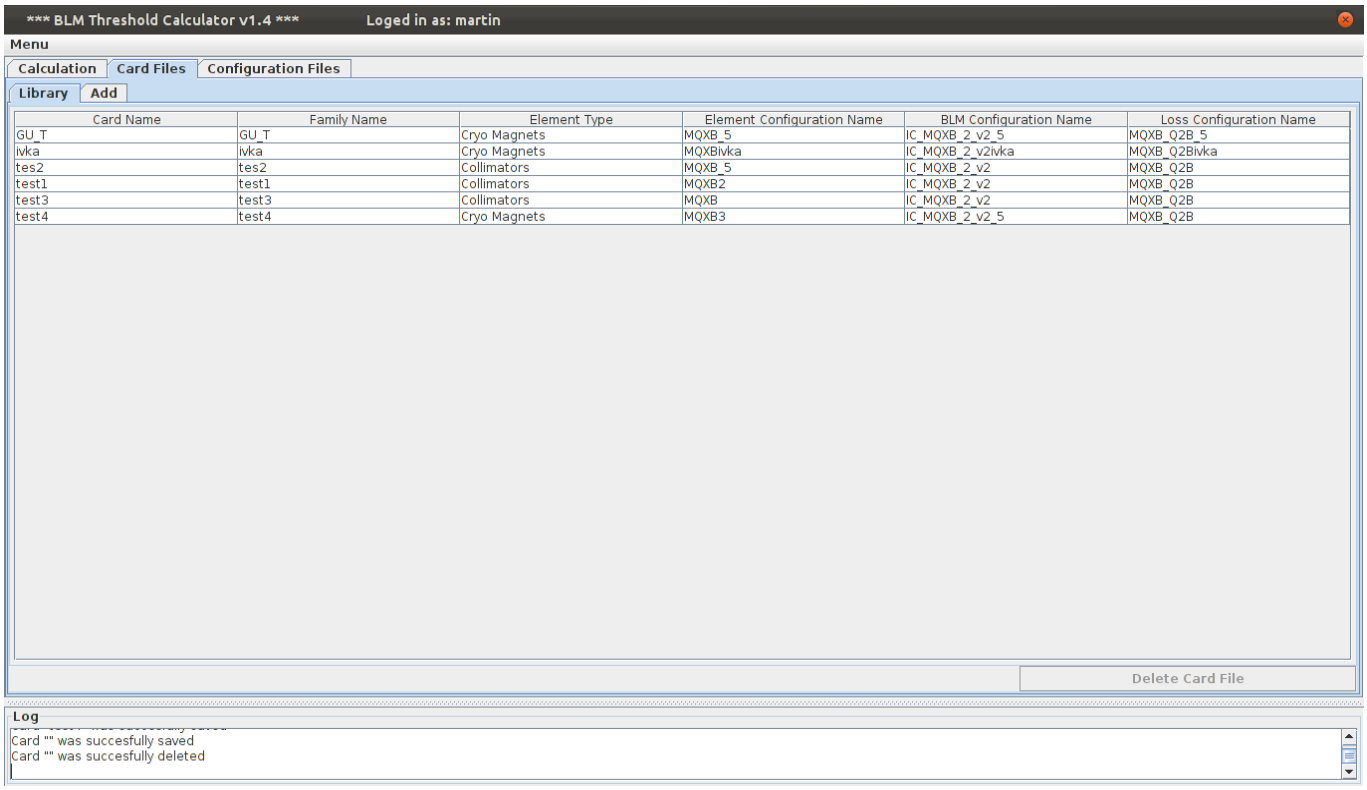


Figure 20: Card files Library panel.

The Card files Library *(Figure 20)* displays all the card file records with all the details that are stored in the database. The user can perform the delete action and the selected item is deleted from the database and the table with records is refreshed. The log window displays an informative message.

7.3.6 Computation

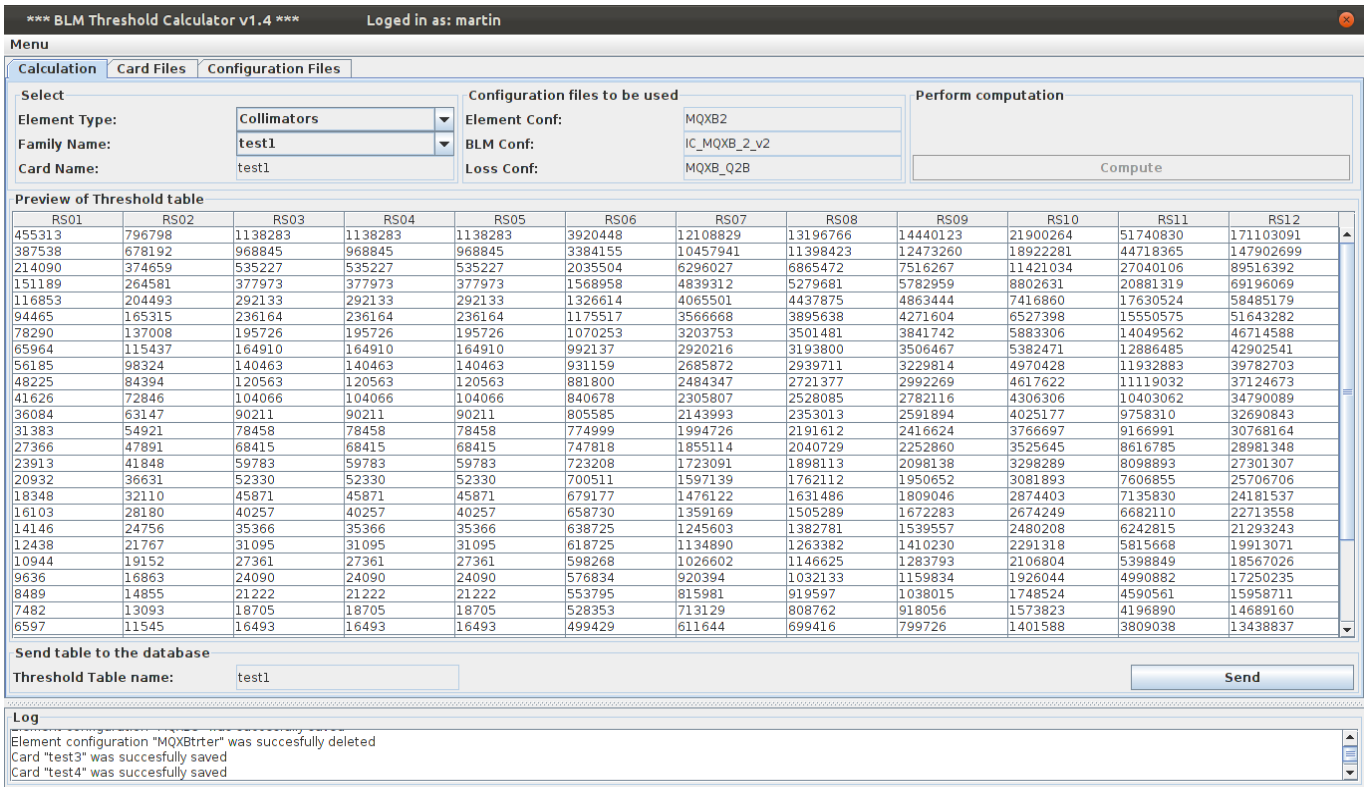


Figure 21: BLM abort threshold calculation panel.

Calculation *(Figure 21)* is the main functionality of the system. The user has to select the appropriate options for calculating threshold values. When the “Compute” button is pressed, the threshold algorithm is fired and it computes values based on the input values from the configuration files selected. Computed results are displayed on the screen. By pressing the “Send” button, SQL query creates a new database table (with 12 columns of integration time; see *Appendix H; page 78*) with the name of the card file used, and the results are sent and stored in the database. An informative message is displayed on the log window.

8. Results and Benchmarks

The computed abort threshold results from both prototype systems were compared with the results obtained from CERN's abort threshold calculator. Results were identical; the algorithm was implemented correctly.

8.1 Black-Box Testing

8.1.1 Java Prototype System

Test	Test Description	Test Input	Test Expected Output	Test Result
1.	**User login**	Login: martin Password: 1985	Successful login	**Pass**
2.	**Storing Element Configuration file into the database**	9 double values 4 integer values 4 formulae	Element Configuration stored into the database	**Pass**
3.	**Storing BLM Configuration file into the database**	16 double values 13 integer values 2 formulae	BLM Configuration stored into the database	**Pass**
4.	**Storing Loss Configuration file into the database**	4 double values 2 integer values 2 formulae	Loss Configuration stored into the database	**Pass**
5.	**Displays library of all Configuration files**	Stored Configuration files into the database	Displayed Element, Blm and Loss Configurations	**Pass**
6.	**Delete Element Configuration file from the database**	Added Element Conf. to the database	Deleted Element Conf. from the database	**Pass**
7.	**Delete BLM Configuration file from the database**	Added BLM Conf. to the database	Deleted BLM Conf. from the database	**Pass**
8.	**Delete Loss Configuration file from the database**	Added Loss Conf. to the database	Deleted Loss Conf. from the database	**Pass**
9.	**Add Card file into the database**	Card name 1 Element Conf. 1 BLM Conf. 1 Loss Conf.	Card file added to the database	**Pass**
10.	**Displays library of all Card files**	Stored Card files in the database	Displayed all the Card files from the database	**Pass**
11.	**Calculate abort threshold values without corrections applied**	78 double values 23 integer values	32*12 computed abort threshold values	**Pass**
12.	**Calculate abort threshold values with all corrections applied**	82 double values 50 integer values 2 boolean values	32*12 computed abort threshold values	**Pass**
13.	**Displays preview of the computed abort threshold values**	Calculate abort threshold values	Displayed preview of abort thresholds table	**Pass**
14.	**Stores abort threshold values without corrections into the database**	Calculate abort threshold values	Data stored into the database	**Pass**
15.	**Stores abort threshold values with corrections into the database**	Calculate abort threshold values	Data stored into the database	**Pass**
16.	**Log window prints all appropriate messages**	Compute/add/del operation performed	Printed appropriate informative message	**Pass**
17.	**Handling all SQL exceptions**	Empty data field input	Informative message displayed	**Pass**

Black-box testing has shown that the Java system works correctly and passed all the test cases.

8.1.2 .NET(C#) Prototype System

Test	Test Description	Test Input	Test Expected Output	Test Result
1.	**User login**	Login: martin Password: 1985	Successful login	**Fail**
2.	**Storing Element Configuration file into the database**	9 double values 4 integer values 4 formulae	Element Configuration stored into the database	**Fail**
3.	**Storing BLM Configuration file into the database**	16 double values 13 integer values 2 formulae	BLM Configuration stored into the database	**Fail**
4.	**Storing Loss Configuration file into the database**	4 double values 2 integer values 2 formulae	Loss Configuration stored into the database	**Fail**
5.	**Displays library of all Configuration files**	Stored Configuration files into the database	Displayed Element, Blm and Loss Configurations	**Fail**
6.	**Delete Element Configuration file from the database**	Added Element Conf. to the database	Deleted Element Conf. from the database	**Fail**
7.	**Delete BLM Configuration file from the database**	Added BLM Conf. to the database	Deleted BLM Conf. from the database	**Fail**
8.	**Delete Loss Configuration file from the database**	Added Loss Conf. to the database	Deleted Loss Conf. from the database	**Fail**
9.	**Add Card file into the database**	Card name 1 Element Conf. 1 BLM Conf. 1 Loss Conf.	Card file added to the database	**Fail**
10.	**Displays library of all Card files**	Stored Card files in the database	Displayed all the Card files from the database	**Fail**
11.	**Calculate abort threshold values without corrections applied**	78 double values 23 integer values	32*12 computed abort threshold values	**Pass**
12.	**Calculate abort Threshold values with all corrections applied**	82 double values 50 integer values 2 boolean values	32*12 computed abort threshold values	**Pass**
13.	**Displays preview of the computed abort threshold values**	Calculate abort threshold values	Displayed preview of abort thresholds table	**Fail**
14.	**Stores abort Threshold values without corrections into the database**	Calculate abort Threshold values	Data stored into the database	**Pass**
15.	**Stores abort Threshold values with corrections into the database**	Calculate abort Threshold values	Data stored into the database	**Pass**
16.	**Log window prints all appropriate messages**	Compute/add/del operation performed	Printed appropriate informative message	**Fail**
17.	**Handling all SQL exceptions**	Empty data field input	Informative message displayed	**Fail**

Black-box testing has shown that the software works as expected in terms of the computation and storing of abort threshold values in the database. Although additional functionality failed, this does not have any impact on the benchmarking purposes with the Java prototype system.

8.2 Benchmarking

The following results are based on the four different tests performed in the Java and C# prototype systems; each of them consist of examining the same task 50 times in both platforms:

- Calculation of abort threshold values without corrections applied
- Calculation of abort threshold values with corrections applied
- Storing calculated abort threshold values without corrections applied into the database
- Storing calculated abort threshold values with corrections applied into the database

From the achieved results, probability was computed using a 1 Tailed statistical T-Test.

8.2.1 Calculation of Abort Threshold Values

8.2.1.1 Without Algorithm Corrections

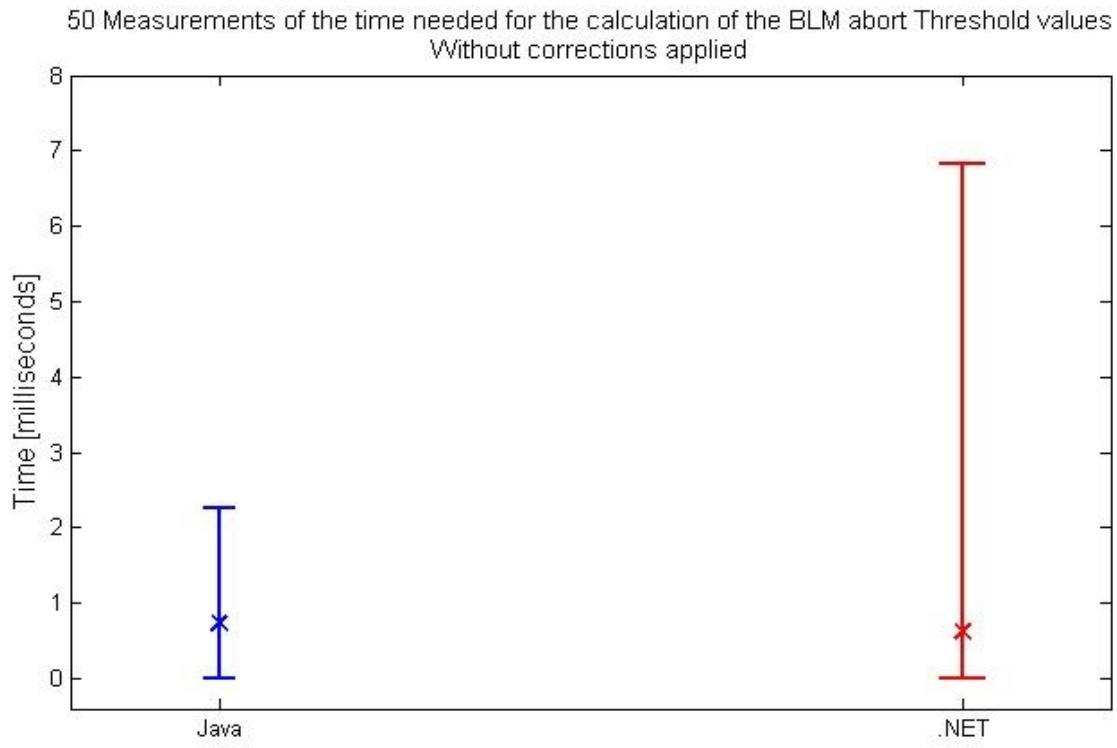


Figure 22: BLM abort threshold algorithm without corrections applied.

Figure 22; page 57, shows the plot with Standart Deviation and Mean. The results demonstrate that although the mean time required for the calculation of the BLM abort threshold values does not differ significantly, (p=0.41032) there is much more variance in the .NET prototype system, whereas Java appears to be a lot more consistent.

The table below shows a summary of the 50 measurements of the time needed for a calculation of the BLM abort threshold values without corrections applied, in each prototype system:

Threshold algorithm [no corrections]		
	Java	**.NET(C#)**
Highest value	5.385ms	15.625ms
Lowest value	0.048ms	0ms
Mean	0.727ms	0.625ms
Standart Deviation	0.769	3.093
Probability	0.41032	

The 50 measurements of the calculation with and without corrections is shown in further detail (see *Appendix I; page 79)*.

Figure 23 and *Figure 24; page 59*, demonstrate how much time (in %) was spent on a single execution, on each method of abort threshold algorithm without corrections, in Java and .NET prototype systems.

Thread name	Percent Per Thread	Cumulative Time (seconds)	Min Time	Avg Time	Max Time
main[9706]	100.00%	0.503865			
main(java.lang.String[]) void	79.75%	0.401853	0.401853	0.401853	0.401853
Calculation_NoCorrections()	79.74%	0.401759	0.401759	0.401759	0.401759
performComputation() void	77.74%	0.391707	0.391707	0.391707	0.391707
getBeamEnergy(int) double	0.22%	0.001087	0.000020	0.000034	0.000057
getEnergyDeposit(double, int) double	12.75%	0.064262	0.000097	0.000167	0.001363
doFEdepThEquil(double) double	2.46%	0.012405	0.000019	0.000032	0.000167
doFEdepMax(double) double	2.52%	0.012683	0.000019	0.000033	0.000229
getHeatTransferRegime(double, double) int	12.75%	0.064264	0.000099	0.000167	0.000511
getTauMetal(double) double	2.48%	0.012490	0.000019	0.000033	0.000066
getTauHelium(double) double	2.51%	0.012624	0.000019	0.000033	0.000140
getQuenchMargin(double, double) double	27.31%	0.137619	0.000139	0.000358	0.003117
dofEnthLimit(double) double	1.85%	0.009301	0.000020	0.000050	0.002934
getSSQuenchMargin(double) double	1.28%	0.006431	0.000019	0.000032	0.000066
getHeliumHeatReserve(double) double	7.52%	0.037915	0.000098	0.000169	0.001069
getHeliumHeatReserveParam(double) double	1.65%	0.008330	0.000019	0.000037	0.000877
getQuenchTemperature(double) double	1.42%	0.007178	0.000019	0.000032	0.000051
getTauMetal(double) double	2.47%	0.012468	0.000019	0.000032	0.000079
getTauHelium(double) double	2.46%	0.012395	0.000019	0.000032	0.000065
getBlmResponseBITS(double, int) double	7.99%	0.040262	0.000059	0.000105	0.001336
dofResponse(double) double	2.50%	0.012618	0.000019	0.000033	0.000082
getIntegrationTime(int) double	2.55%	0.012870	0.000019	0.000034	0.000605

Figure 23: Methods of abort threshold algorithm without corrections in Java prototype.

Function Name	Application Inclusive Time %	Application Inclusive Time	Number of Calls
blmAlgorythm.Calculation_NoCorrections..ctor()	99,60	1,71	1
blmAlgorythm.Calculation_NoCorrections.doFEdepMax(float64)	1,58	0,03	384
blmAlgorythm.Calculation_NoCorrections.doFEdepThEquil(float64)	1,61	0,03	384
blmAlgorythm.Calculation_NoCorrections.dofEnthLimit(float64)	3,33	0,06	186
blmAlgorythm.Calculation_NoCorrections.dofResponse(float64)	1,65	0,03	384
blmAlgorythm.Calculation_NoCorrections.getBlmResponseBITS(float64,int32)	6,87	0,12	384
blmAlgorythm.Calculation_NoCorrections.getEnergyDeposit(float64,int32)	9,95	0,17	384
blmAlgorythm.Calculation_NoCorrections.getHeatTransferRegime(float64,float64)	10,30	0,18	384
blmAlgorythm.Calculation_NoCorrections.getHeliumHeatReserve(float64)	5,61	0,10	224
blmAlgorythm.Calculation_NoCorrections.getHeliumHeatReserveParam(float64)	0,95	0,02	224
blmAlgorythm.Calculation_NoCorrections.getQuenchMargin(float64,float64)	25,16	0,43	384
blmAlgorythm.Calculation_NoCorrections.getQuenchTemperature(float64)	0,90	0,02	224
blmAlgorythm.Calculation_NoCorrections.getSSQuenchMargin(float64)	0,80	0,01	198
blmAlgorythm.Calculation_NoCorrections.getTauHelium(float64)	3,01	0,05	768
blmAlgorythm.Calculation_NoCorrections.getTauMetal(float64)	3,17	0,05	768
blmAlgorythm.Calculation_NoCorrections.InitBlock()	0,00	0,00	1
blmAlgorythm.Calculation_NoCorrections.Main(string[])	100,00	1,71	1
blmAlgorythm.Calculation_NoCorrections.performComputation()	63,07	1,08	1

Figure 24: Methods of abort threshold algorithm without corrections in .NET prototype.

8.2.1.2 With Algorithm Corrections

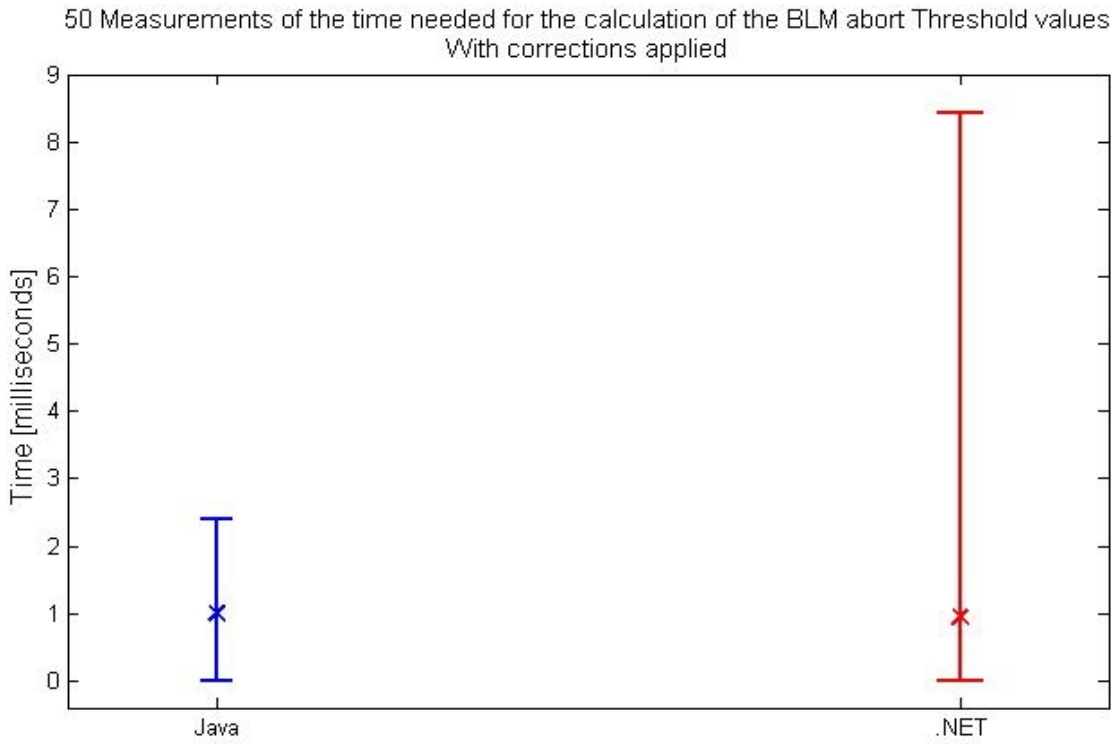


Figure 25: BLM abort threshold algorithm with corrections applied.

Figure 25 shows the plot with Standart Deviation and Mean. The results are very similar to the results of the abort threshold algorithm without corrections applied. Java again appears to be a lot more consistent and there is much more variance in .NET. The results are not statistically significant (p = 0.44988), indicating that the speed of calculating abort threshold values is similar in both prototype systems.

The table below shows a summary of the 50 measurements for the time needed for a calculation of the BLM abort threshold values with corrections applied, in each prototype system:

Threshold algorithm [with corrections]		
	Java	**.NET(C#)**
Highest value	3.132ms	15.625ms
Lowest value	0.106ms	0ms
Mean	1.006ms	0.938ms
Standart Deviation	0.699	3.748ms
Probability	0.44988	

Figure 26 and *Figure 27* show how much time (in %) was spent on a single execution, on each method of abort threshold algorithm with corrections, in the Java and .NET prototype systems.

Thread name	<Percent Per Thread	Cumulative Time (seconds)	Min Time	Avg Time	Max Time
main[9900]	100.00%	0.602484			
main(java.lang.String[]) void	82.59%	0.497568	0.497568	0.497568	0.497568
Calculaction_WithCorrections()	82.57%	0.497462	0.497462	0.497462	0.497462
performComputation() void	79.54%	0.479205	0.479205	0.479205	0.479205
getQuenchMargin(double, double) double	16.69%	0.100553	0.000136	0.000262	0.003392
getHeliumHeatReserve(double) double	4.52%	0.027261	0.000095	0.000122	0.000854
getQuenchTemperature(double) double	0.87%	0.005264	0.000018	0.000023	0.000064
getHeliumHeatReserveParam(double) dou	0.87%	0.005243	0.000019	0.000023	0.000045
getTauHelium(double) double	1.54%	0.009305	0.000019	0.000024	0.000356
getTauMetal(double) double	1.49%	0.008979	0.000018	0.000023	0.000091
dofEnthLimit(double) double	1.27%	0.007634	0.000019	0.000041	0.003195
getSSQuenchMargin(double) double	0.79%	0.004785	0.000019	0.000024	0.000121
correctMaxBits() void	14.11%	0.085011	0.032587	0.042506	0.052424
rsCorrectMaxBits(double, int) java.lang.Lon	4.23%	0.025474	0.000021	0.000033	0.000079
correctDecrease(boolean, boolean) void	12.71%	0.076557	0.035019	0.038278	0.041538
getIntegrationTime(int) double	6.03%	0.036358	0.000020	0.000025	0.000358
getEnergyDeposit(double, int) double	7.85%	0.047322	0.000095	0.000123	0.001701
doFEdepThEquil(double) double	1.63%	0.009811	0.000018	0.000026	0.000878
doFEdepMax(double) double	1.60%	0.009644	0.000018	0.000025	0.000685
getHeatTransferRegime(double, double) int	7.70%	0.046371	0.000095	0.000121	0.000525
getTauMetal(double) double	1.52%	0.009177	0.000018	0.000024	0.000197
getTauHelium(double) double	1.51%	0.009094	0.000018	0.000024	0.000088
correctMinBits() void	5.18%	0.031228	0.031228	0.031228	0.031228
rsCorrMinBITS(double, int) java.lang.Long	2.61%	0.015723	0.000033	0.000041	0.000127
getBlmResponseBITS(double, int) double	4.55%	0.027401	0.000057	0.000071	0.000217
dofResponse(double) double	1.51%	0.009094	0.000018	0.000024	0.000072
getIntegrationTime(int) double	1.50%	0.009053	0.000018	0.000024	0.000066
ilCorrection(double, double) void	0.20%	0.001235	0.001235	0.001235	0.001235
getIntegrationTime(int) double	0.09%	0.000544	0.000021	0.000022	0.000024
getBeamEnergy(int) double	0.13%	0.000794	0.000019	0.000025	0.000054
scaleCorrection(double) void	0.10%	0.000603	0.000231	0.000301	0.000372

Figure 26: Methods of abort threshold algorithm with corrections in Java prototype.

Function Name	Application Inclusive Time %	Application Inclusive Time	Number of Calls
blmAlgorythm.Calculaction_WithCorrections..ctor()	99,99	2,06	1
blmAlgorythm.Calculaction_WithCorrections.correctDecrease(bool,bool)	2,82	0,06	2
blmAlgorythm.Calculaction_WithCorrections.correctMaxBits()	8,17	0,17	2
blmAlgorythm.Calculaction_WithCorrections.correctMinBits()	3,84	0,08	1
blmAlgorythm.Calculaction_WithCorrections.doFEdepMax(float64)	1,20	0,02	384
blmAlgorythm.Calculaction_WithCorrections.doFEdepThEquil(float64)	1,20	0,02	384
blmAlgorythm.Calculaction_WithCorrections.dofEnthLimit(float64)	2,35	0,05	186
blmAlgorythm.Calculaction_WithCorrections.dofResponse(float64)	1,30	0,03	384
blmAlgorythm.Calculaction_WithCorrections.getBlmResponseBITS(float64,int32)	5,55	0,11	384
blmAlgorythm.Calculaction_WithCorrections.getEnergyDeposit(float64,int32)	8,39	0,17	384
blmAlgorythm.Calculaction_WithCorrections.getHeatTransferRegime(float64,float64)	8,37	0,17	384
blmAlgorythm.Calculaction_WithCorrections.getHeliumHeatReserve(float64)	4,72	0,10	224
blmAlgorythm.Calculaction_WithCorrections.getHeliumHeatReserveParam(float64)	0,81	0,02	224
blmAlgorythm.Calculaction_WithCorrections.getQuenchMargin(float64,float64)	20,92	0,43	384
blmAlgorythm.Calculaction_WithCorrections.getQuenchTemperature(float64)	0,74	0,02	224
blmAlgorythm.Calculaction_WithCorrections.getSSQuenchMargin(float64)	0,67	0,01	198
blmAlgorythm.Calculaction_WithCorrections.getTauHelium(float64)	2,42	0,05	768
blmAlgorythm.Calculaction_WithCorrections.getTauMetal(float64)	2,50	0,05	768
blmAlgorythm.Calculaction_WithCorrections.ilCorrection(float64,float64)	0,43	0,01	1
blmAlgorythm.Calculaction_WithCorrections.InitBlock()	0,00	0,00	1
blmAlgorythm.Calculaction_WithCorrections.Main(string[])	100,00	2,06	1
blmAlgorythm.Calculaction_WithCorrections.performComputation()	69,10	1,42	1
blmAlgorythm.Calculaction_WithCorrections.rsCorrectMaxBits(float64,int32)	4,11	0,08	768
blmAlgorythm.Calculaction_WithCorrections.rsCorrMinBITS(float64,int32)	1,87	0,04	384
blmAlgorythm.Calculaction_WithCorrections.scaleCorrection(float64)	1,93	0,04	2

Figure 27: Methods of abort threshold algorithm with corrections in **.NET prototype**.

8.2.2 Abort Threshold Values Database Storage

8.2.2.1 Without Algorithm Corrections

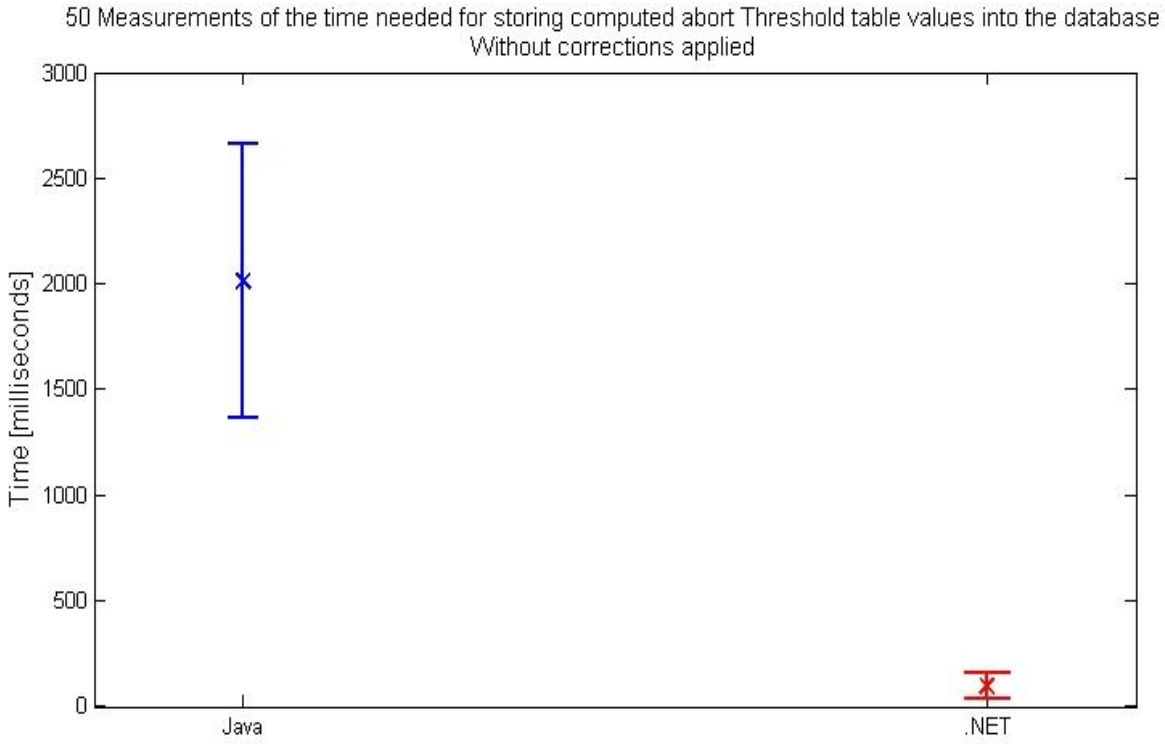


Figure 28: Storage of computed abort threshold values without corrections into the database.

Figure 28 shows the plot with Standart Deviation and Mean. The results demonstrate that the mean time required for the storage of the BLM abort threshold values in the database differs highly significantly (p=5.56E-38). Java has much more variance and takes 20 times longer than .NET. .NET is a lot more consistent and much faster.

The table below shows a summary of the 50 measurements for the time needed for storing the computed abort threshold values without corrections applied, in each prototype system:

Storing Thresholds into the DB [no corrections]		
	Java	**.NET(C#)**
Highest value	3992.357ms	500ms
Lowest value	1173.345ms	62.5ms
Mean	2012.258ms	97.5ms
Standart Deviation	649	62.7
Probability	5.56E-038	

A more detailed table of the 50 measurements of the time needed for storing computed abort threshold values with and without corrections applied, is provided (see *Appendix J; page 80)*.

8.2.2.2 With Algorithm Corrections

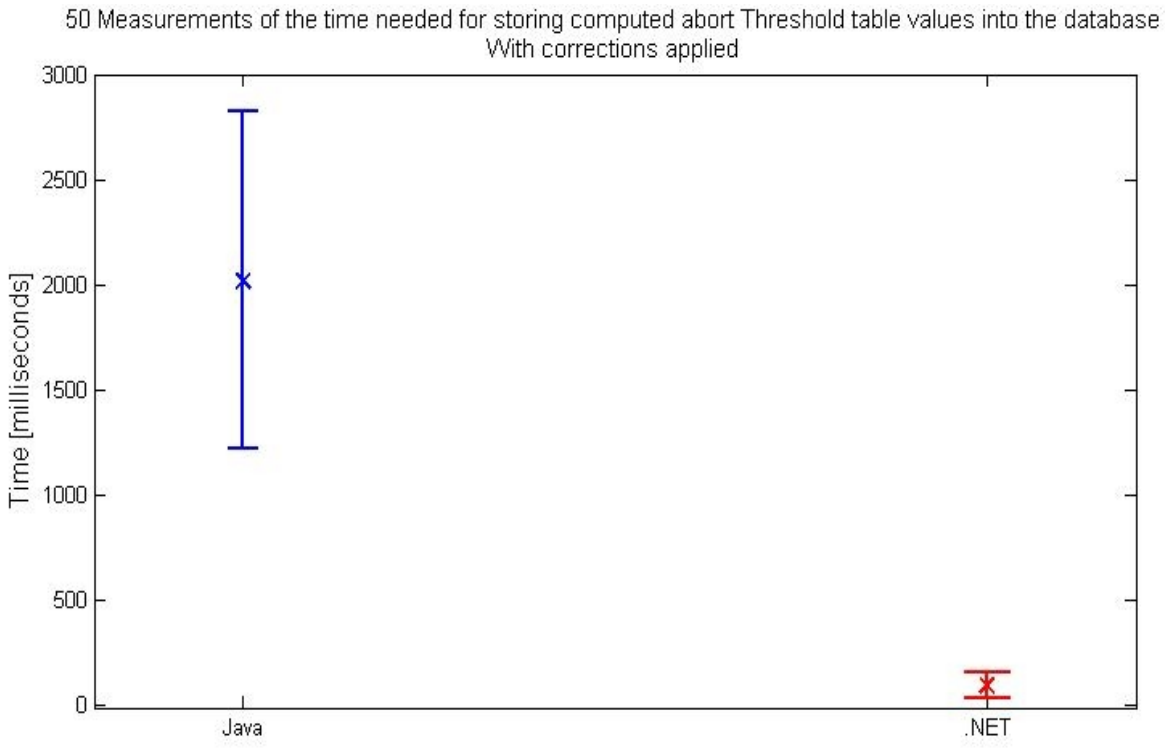


Figure 29: Storage of computed abort threshold values with corrections into the database.

Figure 29 shows the plot with Standart Deviation and Mean. The results are very similar to the results of the storage of abort threshold values without corrections applied. Results are statistically highly significant (p=3.38E-31). .NET is much faster and has much less variance than Java.

The table below shows a summary of the 50 measurements of the time needed for storing computed abort threshold values with corrections applied, in each prototype system:

Storing Thresholds into the DB [with corrections]		
	Java	**.NET(C#)**
Highest value	4815.768ms	484.375ms
Lowest value	1160.349ms	62.5ms
Mean	2020.429ms	94.7ms
Standart Deviation	801	60.7
Probability	3.38E-031	

8.3 Results Analysis

The results from the benchmarks of the .NET and Java prototype systems confirmed state-of-the-art findings; the .NET platform is overall faster in all aspects, than the Java platform.

Statistical analysis has shown that .NET is 12% faster than Java in the computation of BLM abort threshold values, with and without the algorithm corrections applied.

Results also show that .NET is 20 times faster than the Java platform in storing computed values in the database. This finding is statistically highly significant.

However, benchmarking of the 50 measurements of the time needed for a calculation of the BLM abort threshold values, with and without corrections applied, produced unexpected results in the .NET platform (*Appendix I; page 79)*. In the measurements of the calculation without corrections applied; 48 measurements were 0 milliseconds and 2 measurements were 15.625 milliseconds. In the case of the calculation with corrections applied; 47 measurements were 0 milliseconds and 3 measurements were again 15.625 milliseconds. All measurements performed in the .NET prototype system used library, that is included in the framework, for timing.

Investigation of these unexpected results, obtained from the .NET prototype system, did not find any reasonable explanation for the findings.

9. Conclusion

This study focuses on the critical evaluation and investigation of a suitable solution that can address limitations with CERN's LHC BLM abort threshold calculator system. Based on the research, in order to support this study, two technical platforms have been chosen; Java and .NET. The research also aims to find the strengths and limitations of Microsoft's .NET and Sun's Java technical platforms. The investigation compares and benchmarks both technical platforms in the same technical environment. Comparable prototype systems for the computation abort threshold values have been developed and tested for the performance and speed of the database storage access.

It was proven, based on comparing and benchmarking technical platforms, that the developed system is a successful replacement of CERN's current approach. Benchmarked results from the prototype systems supported initial research and also Plavec's (2005) conclusion, *".NET platform is 16% faster than Java*". Study results have shown that the .NET prototype system is 12% faster than the Java prototype system in the computation of the abort threshold values. This is based on the state-of-the-art, and is because .NET runs at native speed whereas Java is interpreted. However, Java uses Just In Time (JIT) compilation, whereas .NET is not limited to this and uses Ahead Of Time compilation to avoid start-up delays. In the database storage, benchmarks show that Java is 20 times slower than the .NET platform, which is statistically highly significant. There is no available explanation regarding these findings about Java's database storage process. Furthermore, research provided an insight into the platforms, relating to interoperability. Java is a programming language that uses a common language that can be compiled and run on different technical platforms (for example; Microsoft Windows, Linux, Mac OS). .NET allows programming in different languages (for example; C#, C++) and has compilers that generate a platform specific code (i.e. Microsoft Windows). Therefore, .NET is used to make any programming language into a Microsoft Windows program and Java is used to write programs for different operating systems. Java is superior to .NET as it can run on a different technical platform environment, although .NET has been shown to be statistically significantly faster than Java.

The Eclipse Framework was used for the development of the Java prototype system. This choice provided all the tools needed to support the declared hypothesis. The framework supports many different programming libraries that have been used during the development of the software.

The project aims to develop two comparable prototype systems. However regarding the time scale of the project it is not possible to build a Graphical User Interface application in C# with Microsoft Visual Studio. C# and Java are both class-based object-orientated, with runtime compilation and use of garbage-collection. Their programming syntax is very similar but not the same. It would be unrealistic to develop a comparable system to Java's prototype, due to a lack of prior experience with C#'s syntax and the time scale of the project. However the console base system, of which the most important function is to calculate the abort threshold algorithm, was successfully implemented in C# and this is the aim of the current study. A further limitation is that the environment which is used for the benchmarking is not completely accurate. The benchmarking has been performed with no professional tools. The Profilers (tools that act as helpful features) are included in the Eclipse and Visual Studio frameworks, and are useful for the programmers. These features produce only an informative message and do not perform a very deep analysis with external factors that might affect the results. The .NET prototype is profiled on Microsoft’s Visual Studio and Java's prototype is profiled on the Eclipse Framework. Tools for the comparison and benchmarking of both prototype systems within the identical technical environment are not available off-the-shelf for use in this study.

Overall, this project meets its goals. The Java prototype system, with the Graphical User Interface and basic functionality of the abort threshold algorithm, has been successfully developed. Furthermore, the prototype system addresses all of the limitations which have been discussed. The .NET prototype, with proof of its capability as a console based application for benchmarking purposes, has also been successfully developed.

The Java prototype system is now ready to be ported for testing purposes in the Beam Loss Monitoring System. Co-operation with CERN will continue and implementation of the new functionality and features of the abort threshold calculator application will be carried out until the system is ready for operational use.

10. References

BULL, J.M., DAVEY, R.A., HENTY, D.S., SMITH, L.A., WESTHEAD, M.D. “A Benchmark Suite for High Performance Java“, *Concurrency: Practice and Experience, 2000.* 12, 375-388.

BULL, J.M., FREEMAN, R., POTTAGE, L., SMITH, L.A., “*Benchmarking Java against C and Fortran for Scientic Applications* “, in Proceedings of ACM Java Grande/ISCOPE Conference, June 2001.

CERN, 2008-last update, The Large Hadron Collider [Homepage of CERN], [Online]. Available: http://public.web.cern.ch/public/en/LHC/LHC-en.html [11/29, 2011].

CERN, 2008-last update, Research at CERN [Homepage of CERN], [Online]. Available: http://public.web.cern.ch/public/en/Research/Research-en.html [11/29, 2011].

CHENGYUN CHU, 2008. Introduction to Microsoft .NET Security. *Security & Privacy, IEEE*, 6(6), pp. 73-78.

DEHNING, B., FERIOLI, G., FRIESENBICHLER, W., GSCHWENDTNER, E. and KOOPMAN, J., 2002-last update, LHC Beam Loss Monitor System Design [Homepage of CERN], [Online]. Available: http://ab-div-bdi-bl-blm.web.cern.ch/ab-div-bdi-bl-blm/literature/229_1.PDF [11/29, 2011].

DEHNING, B., HOLZER, E.B., JACKSON, S., KRUK, G., NEBOT, E., NEMCIC, M., NORDT, A., ORECKA, A., SAPINSKI, M., SKAUGEN, A. and ZAMANTZAS, C., 2011-last update, Handling of BLM abort thresholds in the LHC [Homepage of CERN], [Online]. Available: http://cdsweb.cern.ch/record/1379461/files/CERN-ATS-2011-060.pdf [11/28, 2011].

GALITZ, W.O., 2007. *The Essential Guide to User Interface Design: An Introduction to GUI DesignPrinciples and Techniques.* Third Edition edn. Canada: Wiley Publishing, Inc.

PARTINGTON, V., 13.10, 2010-last update, Deployment Automation vs. Build Automation [Homepage of JAVALOBBY], [Online].
Available: http://java.dzone.com/articles/deployment-automation-vs-build [03/10, 2012].

PAUL, N. and EVANS, D., 2004. .NET security: lessons learned and missed from Java, *Computer Security Applications Conference, 2004. 20th Annual* 2004, pp. 272-281.

PLAVEC, F., 2005-last update, .NET to Java Comparison.
Available: www.eecg.toronto.edu/plavec/pub/net_java.pdf [12/27, 2011].

SABHARWAL, C.L., 1998. Java, Java, Java. *Potentials, IEEE*, 17(3; Java's simplicity and potential power. Java is programming language, and a language for the intranet and the World Wide Web (WWW). Java is a Write On one platform and Run on Many platforms (WORM) language. For network-friendly, platform-indep(TRUNCATED)), pp. 33-37.

SIDDIQUE, S., SHERIFF, S.D., WIJESURIYA, H., WICKRAMARATNE, C. and MAKALANDA, J., 2006. ILJc: Porting Microsoft.NET IL (Intermediate Language) to Java, *Industrial and Information Systems, First International Conference on* 2006, pp. 119-121.

11. Bibliography

DAWSON, C.W., 2009. *Projects in Computing and Information Systems, A Student's Guide* Second edn. Addison Wesley.

FOWLER, M., 2009. *UML Distilled, A Brief Guide to the Standart Object Modelling Language.* Third Edition edn. Boston, United States: Pearson Education, Inc.

FREEMAN, H., 2002. Software testing. *Instrumentation & Measurement Magazine, IEEE,* 5(3), pp. 48-50.

GEER, D., 2005. Eclipse becomes the dominant Java IDE. *Computer,* 38(7), pp. 16-18.

LANDUA, R. and RAU, M., 10/12, 2008-last update, The LHC: A Step Closer To The Big Bang [Homepage of Science in School], [Online].
Available: http://www.scienceinschool.org/2008/issue10/lhcwhy [12/19, 2011].

LO, C.-.D., CHANG, M., FRIEDER, O. and GROSSMAN, D., 2002. The object behaviour of Java object-oriented database management systems, *Information Technology: Coding and Computing, 2002. Proceedings. International Conference on* 2002, pp. 247-252.

LUTZ, M.H. and LAPLANTE, P.A., 2003. C# and the .NET framework: ready for real time? *Software, IEEE,* 20(1), pp. 74-80.

MARTIN, J. and MULLER, H.A., 2001. Strategies for migration from C to Java, *Software Maintenance and Reengineering, 2001. Fifth European Conference on* 2001, pp. 200-209.

PONT, M.J., 1998. Why Java is dangerous. *Software, IEEE,* 15(1), pp. 20-22.

RASCHKE, A., 2009. Translation of UML 2 Activity Diagrams into Finite State Machines for Model Checking, *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on* 2009, pp. 149-154.

ROSENBERG, D., 1998. Bringing Java to the enterprise: Oracle on its Java server strategy. *Internet Computing, IEEE*, 2(2), pp. 52-59.

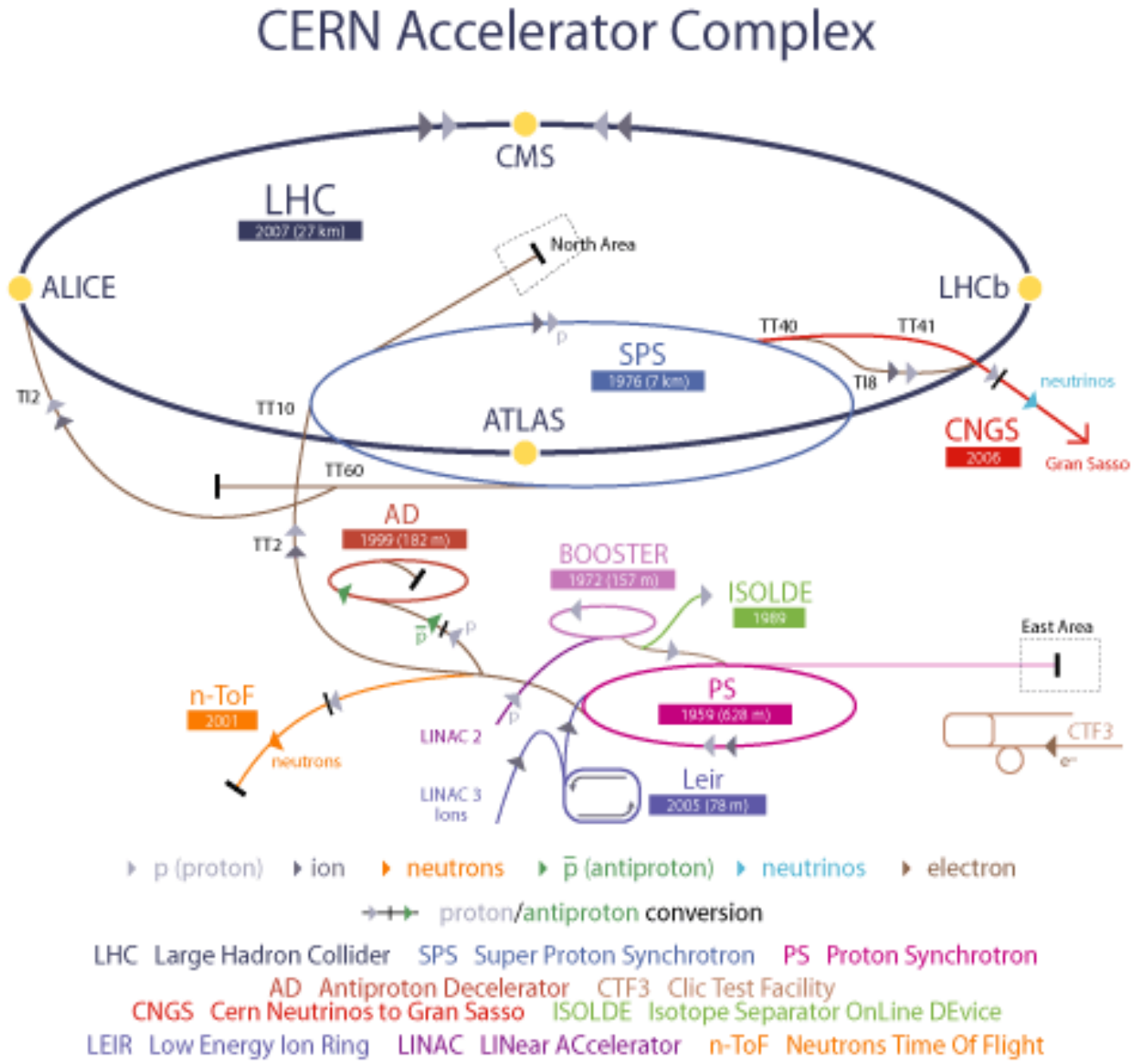
SOMMERVILLE, I., 2007. Software Testing. *Software Engineering.* Eighth edn. United States of America: Addison-Wesley, pp. 538-561.

SWAIN, M., ANDERSON, J.A., KORRAPATI, R. and SWAIN, N.K., 2002. Database programming using Java, *SoutheastCon, 2002. Proceedings IEEE* 2002, pp. 220-225.

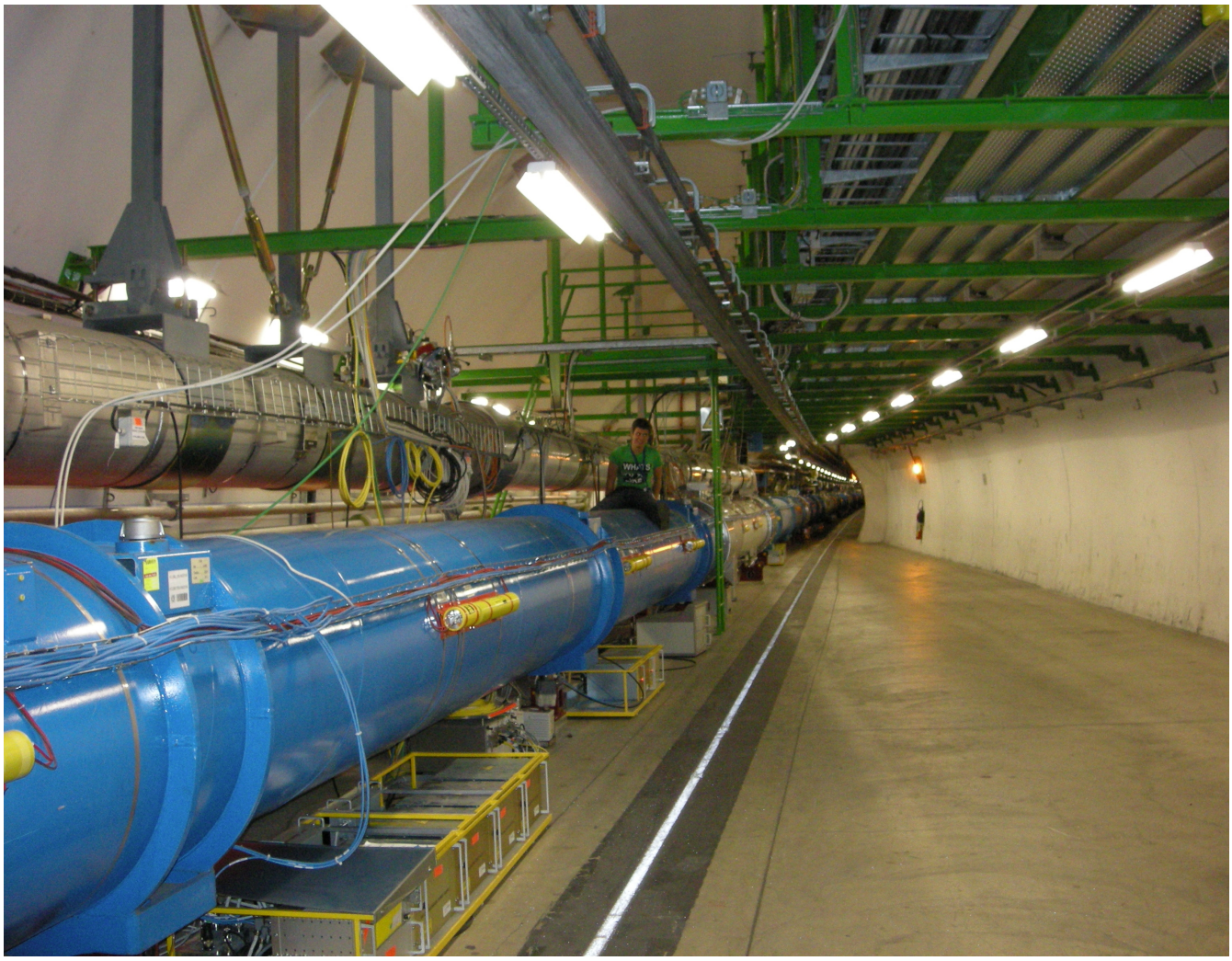
THIRUVATHUKAL, G.K., 2002. Java at middle age: enabling Java for computational science. *Computing in Science & Engineering*, 4(1), pp. 74-84.

12. Appendix

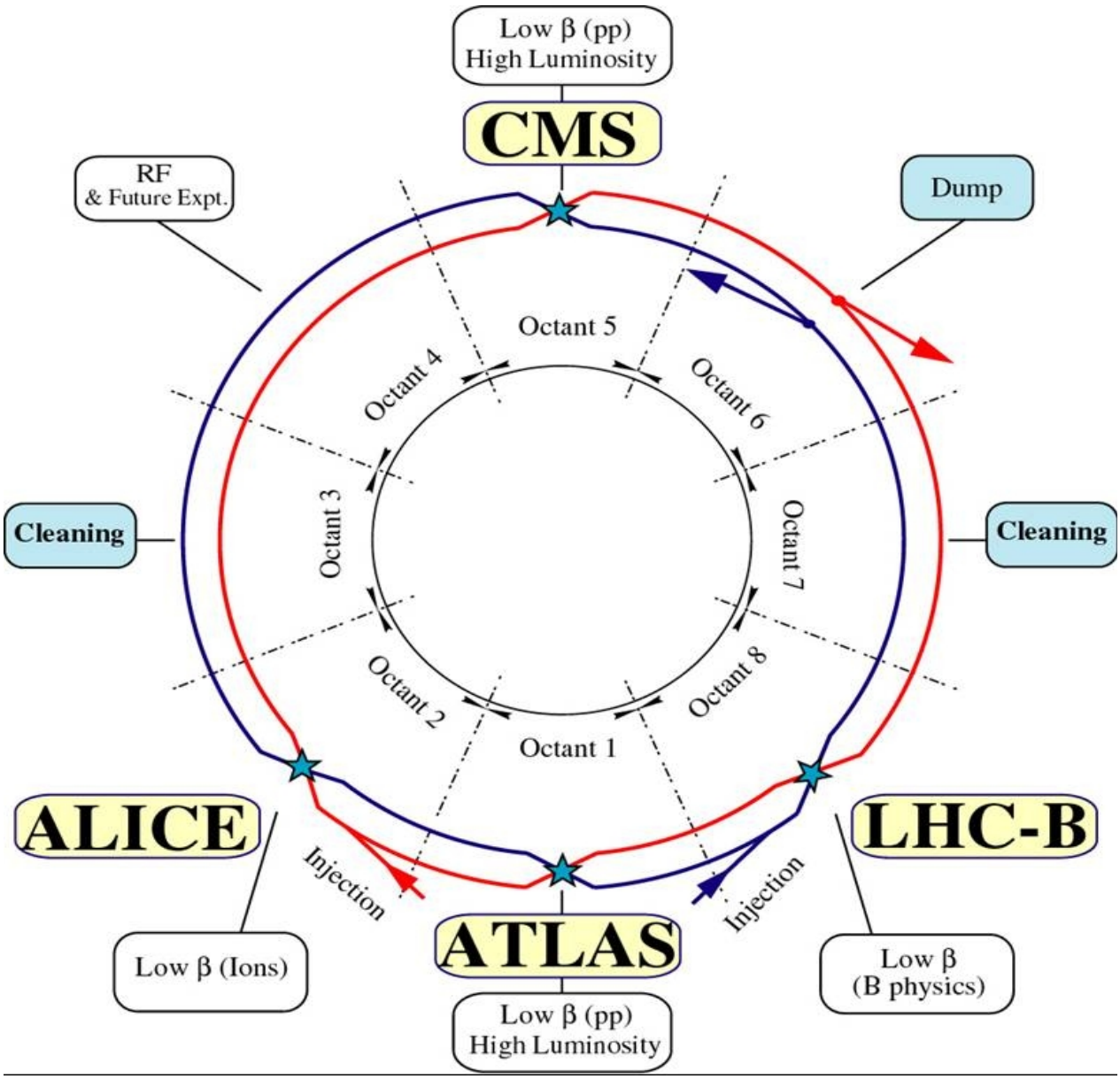
A: CERN Accelerator Complex (CERN 2008).



B: Author sitting on the LHC Accelerator (LHCb point – 90 metres underground).



C: Schematic overview shows the four main experiments and two ring structures of the LHC (CERN 2008).



D: Benchmark results for performance of low level operations (Plavec 2005, p.8).

Benchmark (Units)		Java – client	Java - server	.NET	.NET install-time generated code
Arith (adds/s, multiplies/s, divides/s)	Add:Int	3.60E+008	1.39E+009	9.00E+008	9.01E+008
	Add:Long	2.25E+008	1.25E+008	1.34E+008	1.33E+008
	Add:Float	1956251.8	1979891.8	1982959	1982959
	Add:Double	1950476.2	1976834	1982959	1985843
	Mult:Int	1.37E+008	1.40E+008	1.28E+008	1.28E+008
	Mult:Long	7.31E+007	4.50E+007	1.24E+008	1.00E+008
	Mult:Float	1968095.4	1998048.8	1989122	1991830
	Mult:Double	1894191.6	1905116.2	1916347	1919041
	Div:Int	3.48E+007	3.48E+007	3.48E+007	3.48E+007
	Div:Long	1.40E+007	1.95E+007	1.77E+007	1.77E+007
	Div:Float	1800439.5	1867080	1817859	1820445
	Div:Double	1833154.4	1902461.6	1835619	1846209
Assign (assignments/s)	Same:Scalar:Local	9.33E+008	1.34E+011	2.22E+009	2.21E+009
	Same:Scalar:Instance	1.04E+009	3.20E+010	1.04E+009	1.04E+009
	Same:Scalar:Class	9.19E+008	3.73E+010	1.06E+009	1.05E+009
	Same:Array:Local	4.80E+008	1.10E+009	3.83E+008	2.80E+008
	Same:Array:Instance	2.44E+008	5.57E+008	1.83E+008	1.81E+008
	Same:Array:Class	4.79E+008	1.09E+009	3.87E+008	3.87E+008
	Other:Scalar:Instance	3.50E+008	3.20E+010	1.03E+009	1.04E+009
	Other:Scalar:Class	3.56E+008	3.73E+010	1.06E+009	1.06E+009
	Other:Array:Instance	2.04E+008	1.10E+009	5.66E+008	6.55E+008
	Other:Array:Class	3.24E+008	1.10E+009	8.66E+008	4.79E+008
Cast (casts/s)	IntFloat	5.84E+007	5.04E+007	1.38E+007	1.38E+007
	IntDouble	5.95E+007	4.34E+007	1.70E+007	1.71E+007
	LongFloat	2.01E+007	2.98E+007	1.16E+007	1.23E+007
	LongDouble	2.01E+007	2.39E+007	1.29E+007	1.31E+007
Create (arrays/s)	Array:Int:1	3.32E+007	2.08E+007	4.25E+007	3.71E+007
	Array:Int:2	2.27E+007	1.89E+007	3.63E+007	3.47E+007
	Array:Int:4	1.71E+007	1.58E+007	3.00E+007	2.73E+007
	Array:Int:8	1.14E+007	1.15E+007	2.67E+007	2.34E+007
	Array:Int:16	6907833.5	7094483.5	1.69E+007	1.68E+007
	Array:Int:32	3826964.5	3996097.5	1.12E+007	1.03E+007
	Array:Int:64	1950476.2	3200750.2	5904144	6124860
	Array:Int:128	1016074.6	1717832.6	3666309	3256480
	Array:Long:1	2.27E+007	1.89E+007	3.75E+007	3.51E+007
	Array:Long:2	1.72E+007	1.58E+007	3.40E+007	3.02E+007
	Array:Long:4	1.15E+007	1.15E+007	2.63E+007	2.31E+007
	Array:Long:8	6916582	7114199	1.76E+007	1.63E+007
	Array:Long:16	3838081	3977857.8	1.05E+007	1.05E+007

E: Java Security Vulnerabilities (Paul and Evans 2004, p.2).

Vulnerabilities reported in Java platform in CVE database and Sun's website.

Instances of the form CVE-YEAR-NUMB are CVE entries; CAN-YEAR-NUMB are CVE candidates.

Category	Count	Instances
API bugs	10	CVE-2000-0676, CVE-2000-0711, CAN-2000-0563, CVE-2002-0865, CVE-2002-0866, CVE-2002-1260, CAN-2002-1293, CAN-2002-1290, CAN-2002-1288, CAN-2002-0979
Verification	9	http://java.sun.com/sfaq/chronology.html (4), CVE-1999-0141, CVE-1999-0440, CVE-2000-0327, CVE-2002-0076, CAN-2003-0111
Class loading	9	http://java.sun.com/sfaq/chronology.html (5), CAN-2000-1117, CVE-2002-1287, CAN-2003-0896, CAN-2004-0723
Other or unknown	5	CVE-1999-0766, CVE-2001-1008, CVE-2002-1257, CAN-2002-1286, CVE-2002-1325
Html tags	4	CAN-2001-0068, CAN-2002-1258, CAN-2002-1295, CAN-2002-1291
Missing policy checks	2	CVE-1999-0142, CVE-1999-1262
Configuration	2	CVE-1999-0162, CAN-2002-0058
DoS attacks (crash)	4	CVE-2002-0867, CAN-2002-1289, CAN-2003-0525, CAN-2004-0651
DoS attacks (consumption)	2	CAN-2002-1292, http://sunsolve.sun.com/pub-cgi/search.pl (alert 57555)

F: Test data specification of the configuration files.

BLM abort Threshold Configuration files

Element Configuration

Parameter Name	Parameter value
MG.Name	MQXB
MG.Input	simple_par
MG.Enli_FORM	exp([0]+[1]*x)
MG.Enli_NPAR	2
MG.Enli_PAR0	3.62234
MG.Enli_PAR1	-0.4914
MG.QuLi_FORM	[0]+[1]*x
MG.QuLi_NPAR	2
MG.QuLi_PAR0	52.61
MG.QuLi_PAR1	-5.802
MG.tauMe_FORM	[0]+[1]*x
MG.tauMe_NPAR	2
MG.tauMe_PAR0	0.0062061
MG.tauMe_PAR1	-0.000458
MG.tauHe_FORM	[0]+[1]*x
MG.tauHe_NPAR	2
MG.tauHe_PAR0	0.04647
MG.tauHe_PAR1	-0.005496
MG.He_Frac	0.05

Loss Configuration

Parameter Name	Parameter value
LOSS.NAME	MQXB_Q2B
LOSS.EdepMax_FORM	[0]+[1]*x
LOSS.EdepMax_NPAR	2
LOSS.EdepMax_PAR0	-8.50E-009
LOSS.EdepMax_PAR1	3.51E-008
LOSS.EdepThermal_FORM	[0]+[1]*x
LOSS.EdepThermal_NPAR	2
LOSS.EdepThermal_PAR0	-8.50E-009
LOSS.EdepThermal_PAR1	3.51E-008

BLM Configuration

Parameter Name	Parameter value
BLM.Name	IC_MQXB_2_v2
BLM.Type	IC
BLM.RespType	TF1
BLM.RespFORM	[1]*x+[0]
BLM.RespNPAR	2
BLM.RespPAR0	9.804
BLM.RespPAR1	18.21
BLM.ConvGy2C	0.000054
BLM.ConvBit2Gy	3.62E-009
BLM.Corr_RS01	0.4
BLM.Corr_RS02	0.7
BLM.Corr_RS03	1
BLM.Corr_RS04	1
BLM.Corr_RS05	1
BLM.Corr_RS06	1
BLM.Corr_RS07	1
BLM.Corr_RS08	1
BLM.Corr_RS09	1
BLM.Corr_RS10	1
BLM.Corr_RS11	1
BLM.Corr_RS12	1
BLM.OffBits01	49
BLM.OffBits02	49
BLM.OffBits03	49
BLM.OffBits04	49
BLM.OffBits05	58
BLM.OffBits06	71
BLM.OffBits07	174
BLM.OffBits08	990
BLM.OffBits09	1947
BLM.OffBits10	7742
BLM.OffBits11	30820
BLM.OffBits12	123087

G: Test data specification of the BLM abort threshold algorithm corrections.

BLM abort Threshold algorithm corrections

Correct Max Bits

Parameter Name	Parameter value
Max Bits 01	256000
Max Bits 02	512000
Max Bits 03	2048000
Max Bits 04	4096000
Max Bits 05	16384000
Max Bits 06	65536000
Max Bits 07	524288000
Max Bits 08	41943004000
Max Bits 09	8388608000
Max Bits 10	33554432000
Max Bits 11	134217728000
Max Bits 12	536870912000

Correct Min Bits

Parameter Name	Parameter value
Min Bits 01	1105
Min Bits 02	1105
Min Bits 03	1105
Min Bits 04	1105
Min Bits 05	1105
Min Bits 06	1105
Min Bits 07	1105
Min Bits 08	1105
Min Bits 09	1105
Min Bits 10	1105
Min Bits 11	1105
Min Bits 12	1105

RC Correction

Parameter Name	Parameter value
Time Constant	0.0024

AdHoc Correction

Parameter Name	Parameter value
Type	Expression
RS 01	3*x
RS 02	3*x
RS 03	5*x
RS 04	5*x
RS 05	5*x
RS 06	1*x
RS 07	1*x
RS 08	1*x
RS 09	1*x
RS 10	1*x
RS 11	[10]
RS 12	[10]

IL Correction

Parameter Name	Parameter value
Time Constant	1.20E-004
Norm	9.27E-004

Correct Decrease

Parameter Name	Parameter value
CorrectRSum	TRUE
CorrectELevel	TRUE

Scale

Parameter Name	Parameter value
Factor	1.5

H: Computed abort thresholds (32 Beam Energy * 12 Integration Time).

RS01	RS02	RS03	RS04	RS05	RS06	RS07	RS08	RS09	RS10	RS11	RS12
455313	796798	1138283	1138283	1138283	3920448	12108829	13196766	14440123	21900264	51740830	171103091
387538	678192	968845	968845	968845	3384155	10457941	11398423	12473260	18922281	44718365	147902699
214090	374659	535227	535227	535227	2035504	6296027	6865472	7516267	11421034	27040106	89516392
151189	264581	377973	377973	377973	1568958	4839312	5279681	5782959	8802631	20881319	69196069
116853	204493	292133	292133	292133	1326614	4065501	4437875	4863444	7416860	17630524	58485179
94465	165315	236164	236164	236164	1175517	3566668	3895638	4271604	6527398	15550575	51643282
78290	137008	195726	195726	195726	1070253	3203753	3501481	3841742	5883306	14049562	46714588
65964	115437	164910	164910	164910	992137	2920216	3193800	3506467	5382471	12886485	42902541
56185	98324	140463	140463	140463	931159	2685872	2939711	3229814	4970428	11932883	39782703
48225	84394	120563	120563	120563	881800	2484347	2721377	2992269	4617622	11119032	37124673
41626	72846	104066	104066	104066	840678	2305807	2528085	2782116	4306306	10403062	34790089
36084	63147	90211	90211	90211	805585	2143993	2353013	2591894	4025177	9758310	32690843
31383	54921	78458	78458	78458	774999	1994726	2191612	2416624	3766697	9166991	30768164
27366	47891	68415	68415	68415	747818	1855114	2040729	2252860	3525645	8616785	28981348
23913	41848	59783	59783	59783	723208	1723091	1898113	2098138	3298289	8098893	27301307
20932	36631	52330	52330	52330	700511	1597139	1762112	1950652	3081893	7606855	25706706
18348	32110	45871	45871	45871	679177	1476122	1631486	1809046	2874403	7135830	24181537
16103	28180	40257	40257	40257	658730	1359169	1505289	1672283	2674249	6682110	22713558
14146	24756	35366	35366	35366	638725	1245603	1382781	1539557	2480208	6242815	21293243
12438	21767	31095	31095	31095	618725	1134890	1263382	1410230	2291318	5815668	19913071
10944	19152	27361	27361	27361	598268	1026602	1146625	1283793	2106804	5398849	18567026
9636	16863	24090	24090	24090	576834	920394	1032133	1159834	1926044	4990882	17250235
8489	14855	21222	21222	21222	553795	815981	919597	1038015	1748524	4590561	15958711
7482	13093	18705	18705	18705	528353	713129	808762	918056	1573823	4196890	14689160
6597	11545	16493	16493	16493	499429	611644	699416	799726	1401588	3809038	13438837
5819	10184	14549	14549	14549	465487	511361	591380	682829	1231524	3426307	12205436
5135	8986	12838	12838	12838	403098	412143	484503	567199	1063381	3048106	10987006
4532	7932	11331	11331	11331	305772	313870	378656	452697	896944	2673932	9781883
4267	7468	10669	10669	10669	259014	266658	327809	397696	817019	2494310	9203475
4267	7468	10669	10669	10669	259014	266658	327809	397696	817019	2494310	9203475
4267	7468	10669	10669	10669	259014	266658	327809	397696	817019	2494310	9203475
4267	7468	10669	10669	10669	259014	266658	327809	397696	817019	2494310	9203475

I: 50 measurements of the time needed for the calculation of the BLM abort threshold values.

50 Measurements of the time needed for the calculation of the BLM abort Threshold values

Without corrections applied			With corrections applied		
Test	**Java [ms]**	**C# [ms]**	**Test**	**Java [ms]**	**C# [ms]**
1	1.445	15.625	**1**	3.132	15.625
2	0.994	0	**2**	2.415	0
3	0.965	0	**3**	2.381	0
4	0.952	0	**4**	1.786	0
5	1.017	0	**5**	1.784	0
6	0.967	0	**6**	1.552	0
7	0.972	0	**7**	1.463	0
8	1.079	0	**8**	1.519	0
9	1.035	0	**9**	1.499	0
10	0.967	0	**10**	1.407	0
11	1.017	0	**11**	1.445	0
12	5.385	0	**12**	1.472	0
13	1.098	0	**13**	1.531	0
14	1.106	0	**14**	1.608	0
15	0.958	0	**15**	1.615	0
16	0.975	0	**16**	1.679	0
17	0.715	0	**17**	1.456	0
18	0.701	0	**18**	1.442	0
19	0.69	0	**19**	1.428	0
20	0.788	0	**20**	1.448	0
21	0.689	0	**21**	1.185	0
22	0.694	0	**22**	1.176	15.625
23	0.688	0	**23**	0.926	0
24	0.693	0	**24**	0.899	0
25	0.777	0	**25**	0.939	0
26	0.754	0	**26**	0.886	0
27	0.817	0	**27**	0.931	0
28	0.782	0	**28**	0.887	0
29	0.709	0	**29**	0.885	0
30	0.677	0	**30**	0.884	0
31	0.67	0	**31**	0.924	0
32	0.417	0	**32**	0.772	0
33	0.348	0	**33**	0.688	0
34	0.301	0	**34**	0.655	0
35	0.276	0	**35**	0.607	0
36	0.267	0	**36**	0.399	0
37	0.273	15.625	**37**	0.362	0
38	1.151	0	**38**	0.347	0
39	0.272	0	**39**	0.261	0
40	0.242	0	**40**	0.273	0
41	0.227	0	**41**	0.292	0
42	0.226	0	**42**	0.167	0
43	0.227	0	**43**	0.108	0
44	0.05	0	**44**	0.109	0
45	0.048	0	**45**	0.112	0
46	0.05	0	**46**	0.11	0
47	0.049	0	**47**	0.106	0
48	0.05	0	**48**	0.11	0
49	0.067	0	**49**	0.108	0
50	0.056	0	**50**	0.11	15.625
High	5.385	15.625	**High**	3.132	15.625
Low	0.048	0	**Low**	0.106	0
Mean	0.727	0.625	**Mean**	1.006	0.938
SD	0.769	3.093	**SD**	0.699	3.748
P	0.41032		**P**	0.44988	
	P > 0.05			P > 0.05	

J: 50 measurements of the time needed for storing computed abort threshold values into DB.

50 Measurements of the time needed for storing computed abort Threshold table values into the database

Without corrections applied			With corrections applied		
Test	**Java [ms]**	**C# [ms]**	**Test**	**Java**	**C#**
1	1198.232	500	**1**	1431.236	484.375
2	1352.214	78.125	**2**	1197.185	78.125
3	2905.232	62.5	**3**	2328.269	78.125
4	1399.742	78.125	**4**	1182.429	78.125
5	1179.673	78.125	**5**	2337.642	78.125
6	2621.562	78.125	**6**	2759.079	78.125
7	1256.635	78.125	**7**	1561.527	78.125
8	1344.04	109.375	**8**	1184.696	93.75
9	2547.244	93.75	**9**	2860.448	125
10	1724.509	62.5	**10**	1271.902	78.125
11	1296.522	140.625	**11**	1161.774	78.125
12	2152.378	109.375	**12**	4815.768	78.125
13	2039.034	62.5	**13**	1173.042	93.75
14	1662.24	93.75	**14**	1382.307	62.5
15	1940.6	125	**15**	3261.187	125
16	1936.336	78.125	**16**	1183.693	62.5
17	1179.186	78.125	**17**	1379.17	93.75
18	2117.514	62.5	**18**	3508.279	78.125
19	2646.646	93.75	**19**	1302.104	140.625
20	1347.07	78.125	**20**	1170.994	125
21	2626.55	78.125	**21**	2172.718	62.5
22	2358.449	62.5	**22**	2204.296	109.375
23	2229.639	156.25	**23**	1383.257	78.125
24	2400.752	109.375	**24**	3226.357	78.125
25	2183.4	62.5	**25**	1884.269	62.5
26	1767.368	78.125	**26**	1302.699	109.375
27	1715.097	93.75	**27**	2193.43	78.125
28	1686.401	78.125	**28**	1939.686	62.5
29	2614.747	62.5	**29**	2615.81	140.625
30	1414.687	78.125	**30**	2408.452	78.125
31	2295.43	109.375	**31**	2494.097	62.5
32	2339.833	78.125	**32**	1169.525	156.25
33	1169.558	62.5	**33**	1305.505	62.5
34	1362.595	93.75	**34**	3419.047	78.125
35	3019.047	125	**35**	2003.399	109.375
36	1173.345	78.125	**36**	1981.599	93.75
37	1374.063	93.75	**37**	2604.965	93.75
38	3229.005	78.125	**38**	1339.095	78.125
39	1429.039	78.125	**39**	1160.349	78.125
40	1706.144	109.375	**40**	2685.242	62.5
41	3992.357	93.75	**41**	2116.149	78.125
42	2614.245	78.125	**42**	2237.97	125
43	2305.175	78.125	**43**	2553.111	78.125
44	2625.354	93.75	**44**	1606.916	62.5
45	2551.474	93.75	**45**	1716.007	78.125
46	2071.152	78.125	**46**	2594.391	78.125
47	2239.994	109.375	**47**	2695.458	78.125
48	2194.815	171.875	**48**	1616.53	78.125
49	1172.677	78.125	**49**	1241.266	78.125
50	2903.887	93.75	**50**	2697.167	78.125
High	3992.357	500	**High**	4815.768	484.375
Low	1173.345	62.5	**Low**	1160.349	62.5
Mean	2012.258	97.5	**Mean**	2020.429	94.7
SD	649	62.7	**SD**	801	60.7
P	5.56E-038		**P**	3.38E-031	
	P < 0.01			P < 0.01	