JOHANNES KEPLER UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche Fakultät

Remotely Controlling a Mechanical Laboratory via the Internet

MASTER THESIS

for obtaining the academic title

Master of Science

in

INTERNATIONALER UNIVERSITÄTSLEHRGANG INFORMATICS: ENGINEERING & MANAGEMENT

composed at ISI-Hagenberg, Austria

Handed in by:

Ahmad Mohamed Hisham Ismail

Finished on:

5th of July, 2010

Scientific Advisor(s):

A.Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Schreiner

Hagenberg, July 2010

Acknowledgements

I would like to thank Prof. Bruno Buchberger (Dr.phil., Dr.h.c.mult., the initiator of the JKU International Masters Program in Informatics) for accepting me in this program and giving me the opportunity to achieve my Master Degree. Also, I would like to thank Prof. Dr. Slim Abdennadher (the Study Dean of Engineering Faculties in German University in Cairo) for recommending this program to me.

I would like also to appreciate all the academic effort by my advisor Prof. Dr. Wolfgang Schreiner. Starting with refining the project requirements, recommending a set of courses for me to attend and precisely reviewing my thesis (many times), his effort is very obvious in achieving my Master Degree. My sincere appreciation to Mr. Wernick Helmut (the PROFACTOR company representative to me) for his continuous technical support (hardware) to me.

As for the software technical support, I am grateful to Dipl.-Ing. Michael Schaffler (the instructor of "Programming Java EE6" course). In this course, he presented an overview of the Java EE technology which I chose for the implementation part of my Master Degree. Also, I am extremely grateful for Mr. Mostafa Abdel Moniem El Kady (my former team leader) for choosing me as one of his team members for developing a large scale project. His continuous support to me taught me many technological and managerial concepts.

Mrs. Betina Curtis and Mrs. Heba Fouad (the CAO and her assistant) really helped me a lot with respect to my accommodation in Austria. I am deeply thankful for their support to me.

I would like to dedicate my thesis to my family who were continuously encouraging me to achieve this Master Degree.

Abstract

A laboratory is the place where scientific investigations and experiments are conducted. It has to be equipped with up-to-date devices to allow researchers to perform experiments for testing new ideas. However, having access to such devices is somehow difficult, not only in terms of cost, but also in terms of the complexity of building and configuring them appropriately. Therefore, a new trend in computer engineering (Virtual Laboratory) has evolved which uses the Internet to be able to use devices in remote places. Using this approach, this thesis investigates a way of controlling a specific device using MATLAB/Simulink. The usage of MATLAB/Simulink arose from the existence of a pre-built Simulink model that can control the real device. In this thesis, a web interface is created to allow remote users to control the device over the Internet. The backend connection to MATLAB is also discussed in this thesis. This web interface can be used to evaluate the experiments performed over the device and automatically generate reports accordingly.

Contents

1 Introduction **1**

2 The Profactor Laboratory **4**

2.1 Terminology 5

2.2 Current Situation 5

2.3 Problem 8

2.4 Expected Results 9

3 State of the Art **11**

3.1 Remote Laboratory 11

3.2 GeT 14

3.3 RECOLAB 16

3.4 Java-Based Distance Education 19

3.5 Web Application Frameworks 23

3.6 Conclusion 24

4 Website Backend **26**

4.1 General Deployment Architecture 26

4.2 Backend Components 29

4.2.1 Simulink Scheme 30

4.2.1.1 Java Native Interface Framework 31

4.2.2 Persistence Layer 33

4.2.2.1 Java Persistence API 34

4.2.3 Synchronization Mechanism 35

4.2.3.1 Java Message Service API 36

4.2.4 Email Reporting 39

4.2.4.1 JavaMail API 39

4.3 Business Tier Architecture 40

4.3.1 Session Beans 40
4.3.2 Entity Manager 44
4.3.3 Business Tier Implementation 47
4.3.3.1 Users Services 48
4.3.3.2 MATLAB Services 49

5 Website User Interface **52**
5.1 The View Layer 52
5.1.1 PrimeFaces 55
5.2 The Controller Layer 55
5.2.1 Managed Beans 56
5.2.2 Bean Validation 58
5.2.3 Controller Layer Implementation 59
5.2.3.1 Users Functions 60
5.2.3.2 MATLAB Functions 62

6 Additional Features **64**
6.1 Administrator Interface 65
6.2 Dynamic Configuration 67
6.2.1 MATLAB Code 67
6.2.1.1 Evaluation Function 70
6.2.1.2 Email Attachments 71
6.2.2 Experiment Description Page 72

7 Typical Usage Scenario **73**
7.1 Public Web Pages 73
7.2 Access Restricted Web Pages 76

8 Conclusion **81**

References **83**

List of Figures

2.1 Profactor Vibration Control Device 6
2.2 Disturbance Patch Unit 7
2.3 Actuator Patch and Sensor Units 7
2.4 dSPACE 1104 Device 8

3.1 Remote Laboratory Architecture 13
3.2 Remote Laboratory User Interface 14
3.3 The Architecture of GeT 15
3.4 The GeT graphical interface 15
3.5 The simulation window of GeT 16
3.6 General Architecture Diagram 17
3.7 User Interface 18
3.8 Communication Structure 20
3.9 Administrator and Schedular Views 21
3.10 User Interface 22

4.1 Deployment Diagram 27
4.2 Simulink Scheme 30
4.3 Handler Class Diagram 32
4.4 Database Schema Diagram 33
4.5 JPA Entities Classes 35
4.6 JMS point-to-point model 36
4.7 JMS publish and subscribe model 36
4.8 JMS configuration file 38
4.9 JMS Message Driven Bean 38
4.10 Entity Class States 44
4.11 Business Tier Package Diagram 47
4.12 Users Management Services 48
4.13 MATLAB Services 50

5.1 Pages Flow 53
5.2 Experiment Page 54
5.3 Controller Layer Package Diagram 60
5.4 Users Management Functions 61
5.5 Experiment Functions 63

6.1 Administrator Interface - User Accounts 65
6.2 Administrator Interface - System Usage 66
6.3 Experiment Description Page 71

7.1 Login Page 74
7.2 About Page 74
7.3 Registration Page 75
7.4 Reset Password Page 75
7.5 Experiment Page 77
7.6 Experiment Page 77
7.7 Top Researchers Page 78
7.8 Profile Page 79
7.9 Change Password Page 80

Chapter 1

Introduction

Laboratories are the main places in which experiments for new research ideas are performed. They represent a fundamental part in the educational process as well. They can be used by students to achieve practical experience on already known theories. From this perspective, universities and educational institutes continuously update their laboratories with up-to-date devices and capabilities. However, the problem here is that up-to-date devices are (nowadays) costly due to their degree of precision. Costly here not only refers to money, but also to the technical difficulties of constructing them appropriately. On the other hand, the evolving technologies in computer science ease the communication between distant computers over the Internet. Therefore, the solution of building *Virtual Laboratories* is feasible now. This means that, while the actual device is in one laboratory, it is world wide accessible by researchers over the Internet.

Real laboratories require that the researchers have to be inside the laboratory themselves to perform the experiment. Also, they have to interact directly with the device (which might be difficult or tedious task). On the other hand, virtual laboratories overcome these limitations and expose to researchers an interface to experiment with the device through it. In this way, the researchers can remotely access and use the device and many automation techniques can be applied to minimize the difficult or tedious tasks.

PROFACTOR GmbH [42] is a research company that mainly focuses on increasing the productivity and the flexibility of manufacturing enterprises and also on reducing the costs and the risks involved. It provides innovative solutions and products to its partners. Among its services is the ability to advance research and technology transfer for small and medium size enterprises. Also, it can provide measurements and analysis with state of the

art laboratory equipment and systems. PROFACTOR had constructed a device to be used in noise control experiments. The construction of this device needed sophisticated technical experience.

PROFACTOR decided to build a virtual laboratory to enable world wide researchers experimenting with their device. As the purpose behind the experiment is education/research, students or researchers are allowed to experiment with the virtual laboratory free of charge. This thesis is mainly concerned with building the virtual laboratory for PROFACTOR. The virtual laboratory makes use of the Internet as a communication channel between researchers and the real device.

The results achieved by this thesis are basically the server side controller that communicates with the real device (input and output) and a web interface that enables researchers to experiment easily. The project is concerned with delivering experiment's data to and from the real device. These data define how the noise is controlled by the experiment. This thesis discusses how the virtual laboratory was designed and implemented and not how noise is controlled by the device nor how the device itself operates.

The project enables world wide researchers to experiment with the real device as if they were in the laboratory themselves. The web interface implemented as part in this project allows the researchers to input the experiment's data (different input methods are presented and the researcher selects the one he wants to use) and start the experiment. Since the physical nature of the real device does not permit experiments to be performed simultaneously, the experiments are queued and executed one-by-one in the order of arrival. After a successful experiment, a report is generated that details the device usage and the experiment outcome and is sent to the researcher via email. For privacy reasons, the data submitted by the researcher are deleted after the experiment has finished.

This thesis consists of eight chapters (including this introduction chapter and a conclusion one). Chapter 2 "The Profactor Laboratory" discusses the real device in more details. The aim of this discussion is to formally specify the requirements of the virtual laboratory and the expectation on the results of the project. Chapter 3 "State of the Art" is a literature review about similar projects already implemented previously. It shows a detailed view on four virtual laboratories and concludes with a discussion about the similarities between them and how this project can benefit from their experiences. Also, a discussion about the most popular web development frameworks is presented. In Chapters 4 and 5, the exact architecture implemented in this project is documented. These chapters are self-contained, that is all the technologies used are briefly discussed before the presentation of how are

they applied. Chapter 4 is concerned with the backend while Chapter 5 is concerned with the user interface. Chapter 6 "Additional Features" introduces some of the additional features that were not part of the original requirements specification but were implemented to have a more beneficial system. Finally, Chapter 7 "Typical Usage Scenario" highlights a complete system usage scenario in which each page in the web interface is visited and its usage is shown.

Chapter 2

The Profactor Laboratory

The department "Mechatronic Systems and Components" in the company Profactor [42] focuses on "Active Systems". *Active Systems* are defined as systems that contain sensors and actuators and react to changing conditions to enhance the overall performance of the mechanical construction. Active systems are used in many application domains such as noise and vibration control, structural health monitoring and energy harvesting. This thesis is concerned with active systems used in noise and vibration control.

Experimentally, it was shown that the main source of audible noise in a car (or in a passenger cabin in general) comes from vibrations in the mechanical structure in the body of the car. These (low frequency) vibrations are normally excited by the engine itself and then get amplified by the mechanical structure. Therefore, in order to provide a noise-free environment, it was suggested to use a counter vibration produced in a controlled way to overcome those coming from the engine and subsequently provide the desired noise-free environment.

To achieve this goal, the solution is composed of four components. The first is some Piezo-electric devices to actually produce the vibrations (such as Piezo-Stacks and Piezo-Patches). The second is some sensors to measure the current vibrations at a specific positions. The third component is the control algorithm, which can be seen as the function that maps between the readings got from the sensors and the signals sent to the Piezo-electric devices to achieve the minimum noise possible. The fourth component is the electronics part which includes the DSP boards, signal conditioning electronics and power electronics. The main challenging component in this solution is the third one (the control algorithm).

2.1 Terminology

Here is a list that defines common keywords to be used directly afterwards.

- **Disturbance Signal:**

  This is the signal that is generated by the device (e.g. car engine). The main goal of the experiment is to overcome this kind of signals.

- **S-Function:**

  It is a computer language description of a Simulink block written in MATLAB, C, C++, or Fortran. The MATLAB interpreter can automatically load and execute it.

2.2 Current Situation

Profactor has already built a testing environment with a sample mechanical structure. It resides in the company's laboratory (Figure 2.1). At the hardware level, this device is composed of a disturbance patch unit (Figure 2.2) that provides some vibrations (resembling the engine of a car) and an actuator patch unit that provides the counter vibrations required to eliminate the noise resulted by the vibrations produced by the disturbance patch. These units are actually piezo-electric patches. The testing environment also has a sensor to be used to read the current vibrations in the mechanical structure. The actuator patch unit and the sensor are shown in Figure 2.3. They are connected through a mecha-control system to the server computer by a USB link. The *dSPACE* [6] 1104 board is used here as the hardware controller level to connect the testing environment to the server computer (Figure 2.4). At the software level, Profactor uses *MATLAB/Simulink* [12, 13] to experiment with the device. A Simulink scheme was designed and implemented that makes use of ControlDesk to interact with the dSPACE board. This scheme contains an s-function block which actually is responsible for the controlling algorithm.

Currently, the researchers write their control algorithms directly on the PC, and use MATLAB/Simulink to compile and transfer them to the dSPACE board. The results are returned back to MATLAB/Simulink and shown to the researcher.

Figure 2.1: Profactor Vibration Control Device

Figure 2.2: Disturbance Patch Unit

Figure 2.3: Actuator Patch and Sensor Units

Figure 2.4: dSPACE 1104 Device

2.3 Problem

In the course of enhancing the recognition of the Profactor company within the scientific community of smart and adaptive structures, it was decided to export their testing environment, i.e. to make it available worldwide for the purpose of evaluating advanced control algorithms for vibration control. The main goal to be achieved is to develop an optimal algorithm that overcome the audible noise.

This thesis is concerned with the creation of a web interface that provides the functionality of the built testing environment to researchers all over the world via the Internet. Researchers will be allowed to upload their control algorithms and evaluate the achieved performance without having direct access to the environment itself. The lowest level of concern in this project is the software level MATLAB/Simulink.

2.4 Expected Results

The outcome of this thesis is a fully functional web interface that exports the functionality of the built testing environment to remote researchers. The expected functionalities to be available in the web interface are as follows:

1. Manage user accounts (register and login/logout) and keep track of their profiles (name, address, institution and email).

2. Upload a control algorithm (in the form of a s-function or a compiled file) or input the equation directly to the website.

   The control algorithms to be used can be specified in a form of equation (to be used by non-MATLAB users). The exact format in both cases (s-function or direct input) is to be defined.

3. Input the running time interval of the execution (scale of seconds).

   The time limit is a configurable item for each user by the administrator. A default value will be given to all new users.

4. Choose the disturbance signal (already available signals or newly defined ones).

   Examples of already available signals are "white noise" and "harmonic" signals. The format for the user-defined signals will be an ASCII file that consists of two columns "Time vector" and "Data vector". It describes the data value at each time instant.

5. Ability to start and stop the execution of the control algorithm.

   The device is to be scheduled among multiple users so that only one user has access to the system at a time. This is to be done using an automatic queue that receives the algorithms to be executed and executes them one-by-one sequentially.

6. Generate a report about the execution.

   The report will be in the form of three columns "Time", "Sensor reading" and "Actuator value". It describes the values for the sensor and the actuator at each time instant. Also a visual display will be generated (the exact format is to be defined).

7. Send the report back to the user by email.

8. Benchmark the algorithms and keep track of the best performance value so far.

   The performance value here is defined as the amount of reduction of the noise generated by the disturbance signal measured in decibels.

9. Log the use of the system.

   The events to be logged by the system are:

   - New user registered.
   - New algorithm uploaded.
   - Algorithm execution started.
   - Algorithm execution ended.
   - Execution report emailed to the user.

The rules of data privacy have to be obeyed and therefore the algorithms must not be stored on the server (they get executed, evaluated and then deleted). Only the resulting performance measure have to be saved to keep track of the best algorithm executed so far.

Chapter 3

State of the Art

In this chapter, we briefly describe four projects that were implemented previously and that are in the same scope of the project of this thesis. In every project, the motivation to the project, the requirements of the software system, the architecture of software communication, and the hardware specifications will be highlighted. Also, we discuss about some of the currently popular web application frameworks. Finally, a conclusion section is presented to show how can we benefit from these previous projects in the current project.

3.1 Remote Laboratory

The Remote Library is a project that is widely used in the academic field [4]. The goal of the project is to give remote students full control to the laboratory equipments as if they were actually directly using it. The Remote Library saves the equipment used (since there is no direct interaction), the money and time of remote users (since they don't have to be present in the physical laboratory). Furthermore, it should protect the equipment from malicious code. Mainly, the system is composed of three main blocks, the laboratory equipment, communication software and teaching software.

The requirements of this project were formulated from different points of views as follows:

- **Technical Requirements:**

  The equipment has to be reliable so that the maintained time and cost is minimized. The server software should be able to communicate with

the equipment through its data acquisition card and it should provide full control to the users. Also, the communication between the user and the server has to be minimized to reduce network traffic.

- **Security Requirements:**

  The software has to synchronize the usage of the equipment between multiple users. Also, it has to protect the equipment from wrong usage by the students. Therefore, it was decided not to accept compiled code so that the software can analyze the submitted algorithms for different threats.

- **Educational Requirements:**

  Every student is granted a private directory on the server to be able to save the data files and previous solutions. No other student has access to this directory. Professors have access to all directories to be able to mark the solutions.

- **Financial Requirements:**

  The remote students should be able to use the system without buying any new software. Typically, a web browser should be sufficient to use the system.

The software for this project was implemented by the authors according to their requirements. This is because they didn't find an available commercial product that fulfils all their requirements. They had three choices, *MATLAB/Simulink* [12, 13], *LabVIEW* [11] and *Quanser* [45].

- **MATLAB/Simulink:**

  The MATLAB Web Server Toolbox was used to allow the execution through network. However, it can not execute remotely written software. Therefore the students will not be able to upload their own algorithms to be executed.

- **LabVIEW:**

  LabVIEW allows the student to remotely configure the parameters for the algorithm to be executed. However, it also does not accept remotely written software.

- **Quanser:**

  Quanser was a more powerful solution that uses MATLAB/Simulink as backend. The problem with it is that it only accepts compiled files to be executed directly which as shown in the paragraph "Security

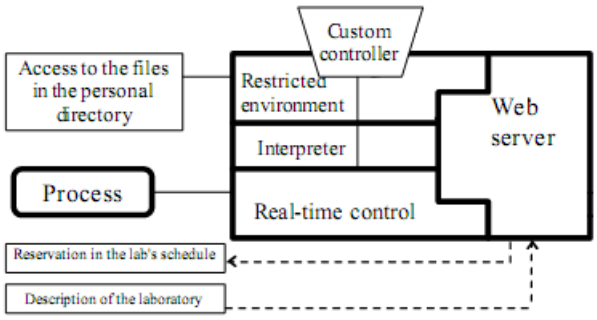

Figure 3.1: Remote Laboratory Architecture (from [4] with permission)

Requirements" above is not allowed. In addition to that, it requires a license for the server and for each client, which makes it an expensive solution (breaking the requirements stated in the paragraph "Financial Requirements").

The architecture of this system is shown in Figure 3.1. The software server is responsible for interpreting the user source code (after passing through a filter to detect malicious code), executing it, sending the results back. As for the communication between the server and client, the most efficient way is to develop a protocol based on *TCP/IP* [50]. However, it would require special programs running on both server and client. Another solution is to develop an *HTTP* [48] based protocol so that all web browsers would be used. The authors prefer *Java applets* [20] as a way of communication since they only require a compatible browser; therefore it is a portable low-cost solution. On the other end, the client is presented by a graphical user interface inside his local web browser upon starting a new session (Figure 3.2). The interface is divided into three frames. The top left frame is where the user writes his own algorithm to be executed and the bottom left frame is where the parameters are shown to be edited by the user. The right frame contains the results of the execution.

The users write their algorithms to be executed in *Python* [44]. Python was selected by the authors since it is free and allows to build a restrictive environment that is able to automatically filter the user code. This feature solved most of the security problems. Currently, the project restricts the user code in accessing the equipments and the file system (the private directory of each student). Another feature of Python is that it is very simple language to be learnt by the students.

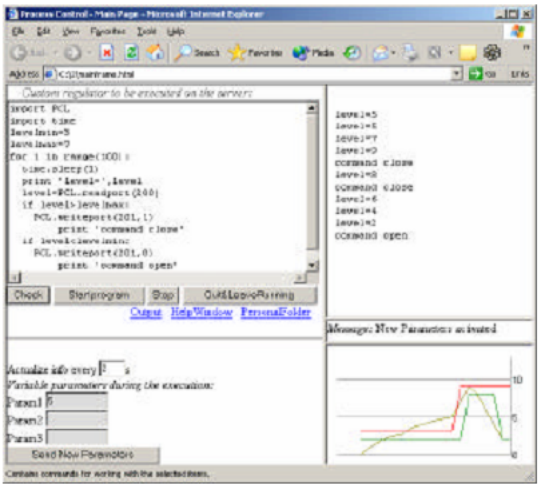

Figure 3.2: Remote Laboratory User Interface (from [4] with permission)

3.2 GeT

The Graphic Environment Tools [3] is a project used to teach robot programming to students. It provides high level interface to many robots. The authors developed an intuitive, visual, object-oriented, multi-robot programming language to be used by the students. The student writes his own algorithm using this common language, chooses a robot in the laboratory and compiles his algorithm accordingly. The project allows programming, debugging, simulating and executing algorithms on the available robots in the laboratory. The execution can be directly seen through a webcam.

The architecture of GeT (Figure 3.3) is simply composed of a web server (Intel Celeron 400 MHz - 64 MB RAM - 6.4 GB HD) and many servers where each server is directly connected (via a RS232 link) to a robot using CGI scripts. The core of the project is written in *Java* [28]. Therefore, users only need a web browser that supports Java Applets. The graphical interface presented to users (Figure 3.4) is divided into three parts, *The Program Score* (top area) is where the user can insert the commands for his algorithm, *The Workspace Description* (bottom left) includes a graphical description of the robot and its environment, *The Command Icon Repository* (bottom right) contains a set of commands to manage the robot movements. The web camera is connected on the parallel port and controlled by the *Microsoft NetMeeting* Software [17].

Once the user has finished implementing his algorithm using the common language, he can validate it on a specific robot to be sure that this robot

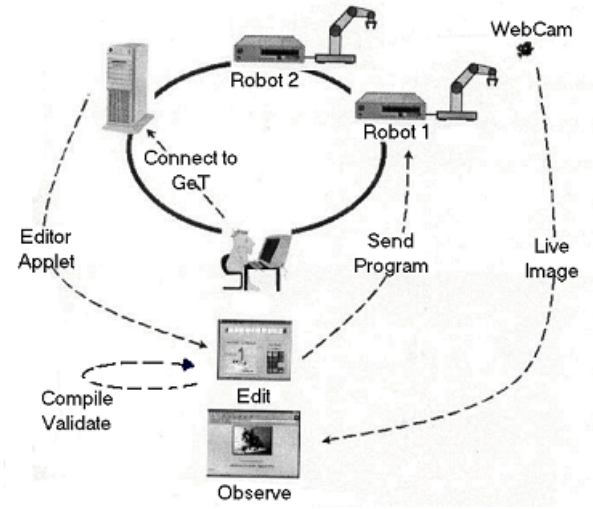

Figure 3.3: The Architecture of GeT (from [3] with permission)

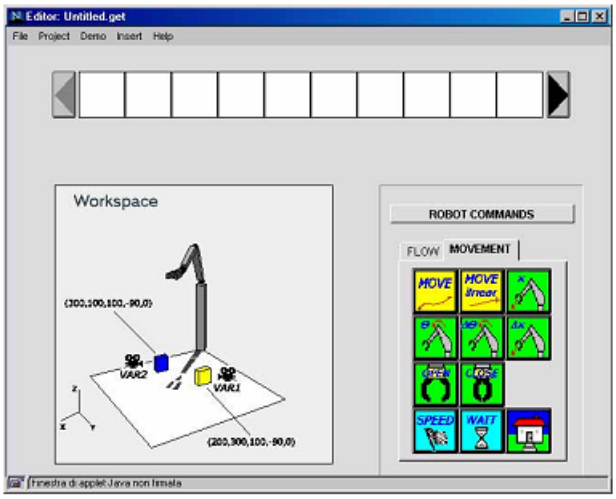

Figure 3.4: The GeT graphical interface (from [3] with permission)

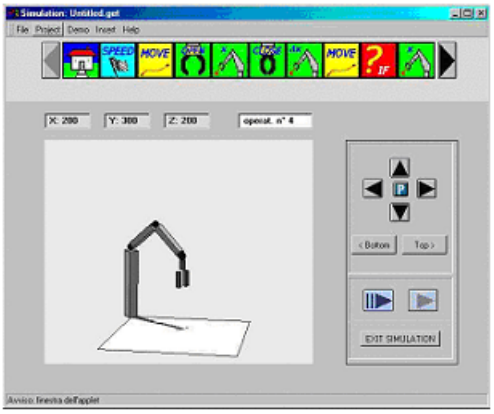

Figure 3.5: The simulation window of GeT (from [3] with permission)

is capable of executing the algorithm. After the validation process, the user can simulate the execution of the algorithm in the simulation window (Figure 3.5). The simulation window allows the user to step in the algorithm or continuously executing it. Finally, the user compiles the algorithm to the robot specific language and executes it directly. Accessing the physical robot is synchronized by a queuing mechanism in the server directly attached to the robot.

3.3 RECOLAB

The main idea of this project [43] is to provide a general architecture for an application that is used to control the execution of a physical process. The chosen platform for this project was MATLAB/Simulink (with some additional toolboxes). This choice was based on the following reasons:

- It is reliable, well-known and commonly used in teaching control courses.
- It allows real-time execution on a physical system, using a specific control algorithm.
- Researchers use it as development tool for simulations and real applications.

The architecture of the project is mainly divided into 2 blocks (Figure 3.6). The user block (where the users are working) and the remote block (where

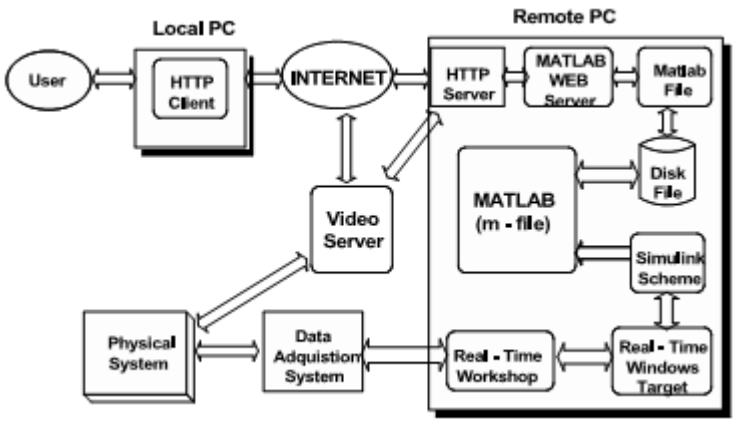

Figure 3.6: General Architecture Diagram (from [43] with permission)

the actual system is). On the user block, it is only required to have a web browser and an internet connection. The server used here is PC Pentium II (800 MHz) with 256 MB of RAM. The building blocks of the remote part are:

- **HTTP server** Apache with PHP configured to use MATLAB Web Server.
- **MATLAB web server** toolbox to use the mathematical and graphic capacities of MATLAB in a web page.
- **MATLAB/Simulink** that actually allows the control of the physical system and production of results.
- **Real-Time windows target** toolbox that provides necessary blocks for using the data acquisition system in real-time.
- **Real-Time workshop** toolbox that generates C code to be executed in real-time.
- **Data acquisition system** board with analog and digital I/O.
- **Physical system** that consists of DC motor, power amplifier, tachometer, absolute encoder, potentiometer and magnetic break.

The user interface is built using dynamically generated HTML code from *PHP* [41] (Figure 3.7). A *CGI* [49] application is developed to be in charge of communication between the web server and MATLAB. The user is allowed

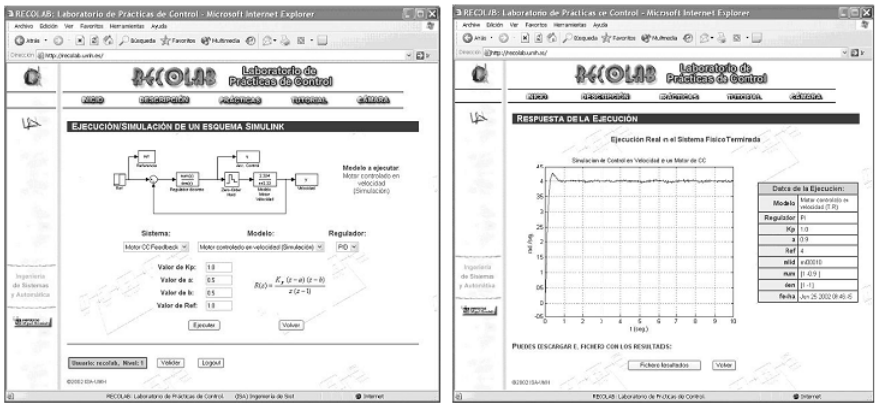

Figure 3.7: User Interface (from [43] with permission)

to choose which Simulink scheme is executed and whether the execution will be just simulation or in realtime over the physical system. Also the user choose the control strategy and its parameters.

The project is open to all users to be used for simulation only. It is considered as an open teaching tool to be used by any student. However, controlling the physical system is restricted to validated users. The validation process is simply implemented in PHP based on username/password authentication. However, the access to the physical system is done based on a priority assigned with the username.

The CGI application is used to pass the data from the user interface to MATLAB and also returns the results to the user interface. This can be done using the MATLAB web server toolbox. The toolbox is capable of executing MATLAB functions and directly generate the results in web pages. However, it can not communicate with the realtime toolboxes to control the physical system. So it is only used to get the data from the user interface and pass it to MATLAB.

The application life cycle is as follows:

1. The user connects to the application, chooses whether it is simulation or real-time execution and adjusts the parameters.
2. The MATLAB web server collects all the input data from user.
3. The MATLAB web server stores the data in a .mat file which is synchronized between multiple concurrent users.

4. The Main CGI application executes the request and monitors any problem that might happen. It also synchronizes the running MATLAB sessions (simulation or real-time execution).

5. PHP collects all the results and produce HTML page to be sent to user. The .mat file is also available for the user to download and analyze.

A final note on this project is that it can only have one realtime execution at a time (since it is connected to one physical system). However, it can run multiple simulation executions concurrently. Therefore, users are encouraged to use the simulation for experimenting before using the real-time execution.

3.4 Java-Based Distance Education

The "Java-Based Distance Education" project [46] was developed for the "University of Hagen" in Germany. This university heavily depends on distance teaching methods. The courseware is provided through the Internet using simulation interfaces, animations and support from other online learners. Video-conferencing is used for seminars and examinations. However, the laboratory experimentations were inconvenient as the physical presence of the student in the laboratory was required to perform the experiment. Two solutions were proposed for this problem, the use of virtual experiments (simulation techniques) and the remote control of actual devices used in the experiment. The second solution was preferred since expensive devices can be shared between universities using this way and thus increase the laboratory resources.

An important requirement in this project is that the remote user should get the feeling that he is in the real laboratory performing the experiment. So, it was decided to provide a video and audio broadcast so that the user can monitor the execution of his algorithm live. As expected, the real devices can only be used by one student at a time. Therefore a synchronization system have to be developed. Beside this, a booking system should allow the students to book the devices for a certain time period that is most suitable for them. On the server side, it is required that the real laboratory is secure. Therefore, all the user commands sent to the server should be filtered before actual execution to exclude all the commands that can cause damage. For evaluation purposes, all the communication between user and server will be saved on the server. On the client side, the users should be able to use the project without requiring additional software to be installed. Thus, the project should allow the use of free client software to use it.

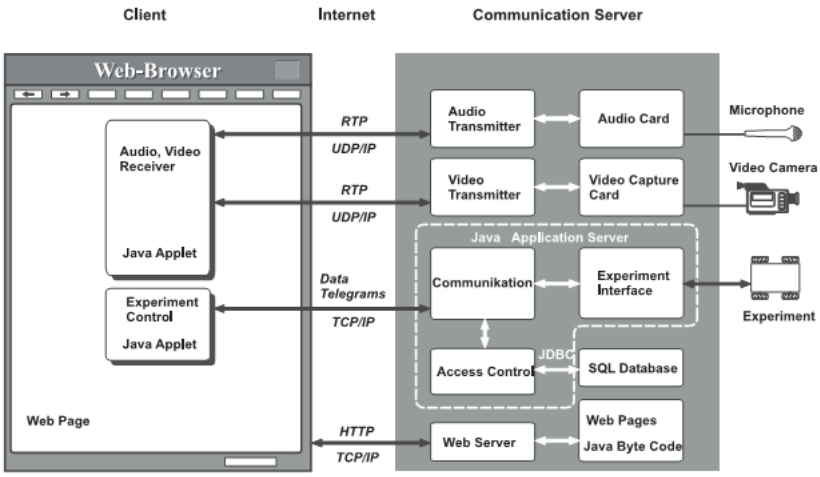

Figure 3.8: Communication Structure (from [46] with permission)

The project is designed as client/server architecture that is mainly developed in Java (Figure 3.8). Students can use any web browser (on any platform) that supports Java. Thus, there is no need for new software on the client side. The browser requests Java applets from the server and starts executing them. A main feature of the system is that it is modular such that additions of new features can be easily implemented and plugged into the system. Also, using the Java applets allowed easy upgrades of the controller software without interruption of the user. This is because the user browser is requesting the last version of the applet on the server and executes it. Finally, the project is designed to be generic, that is it can be used with any experiment (independently of the device).

The access management system is also implemented in Java and can be executed in the web browser. It consists of an administrator view and a schedular view for the students (Figure 3.9). The administrator can easily manage user accounts (create and/or delete them). He can also manage time quotas for different experiments. Logging messages can be easily interpreted in the administrator view. For each experiment, some preparations might be necessary before actual experiment. The student can easily download and do these preparations. These preparations are then submitted to the teacher who reviews them and accordingly opens the schedular view for the student to choose a suitable time to perform the actual experiment.

To start the experiment, the student connects to the server through the web browser. The server then replies with two Java applets (Figure 3.10). The applets in the left side receives the video and audio stream from the labo-

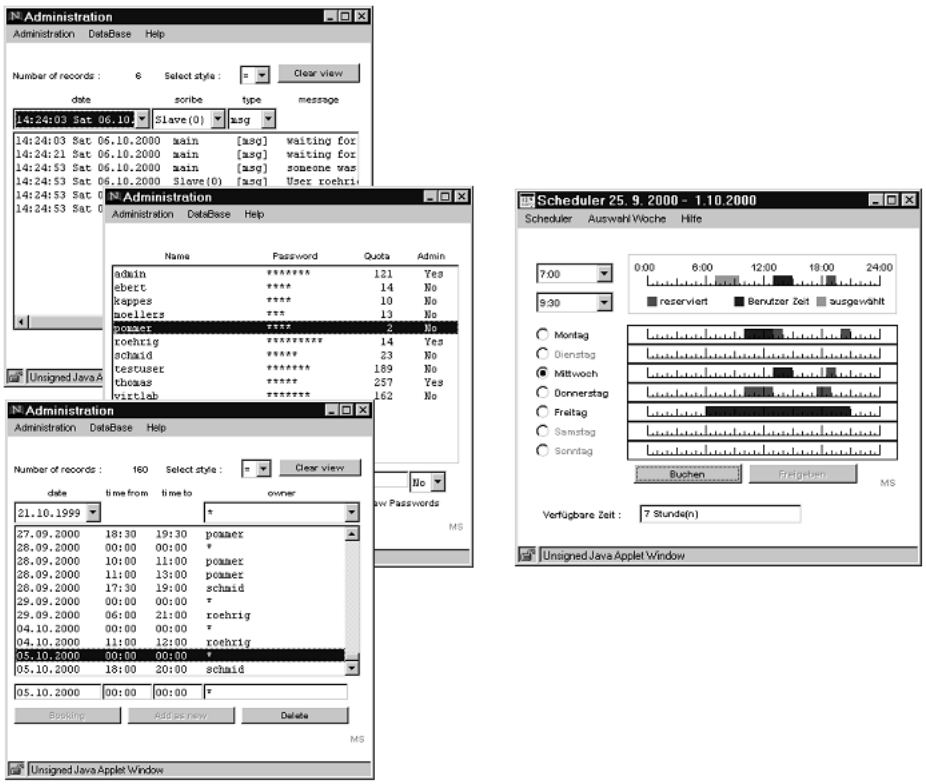

Figure 3.9: Administrator and Schedular Views (from [46] with permission)

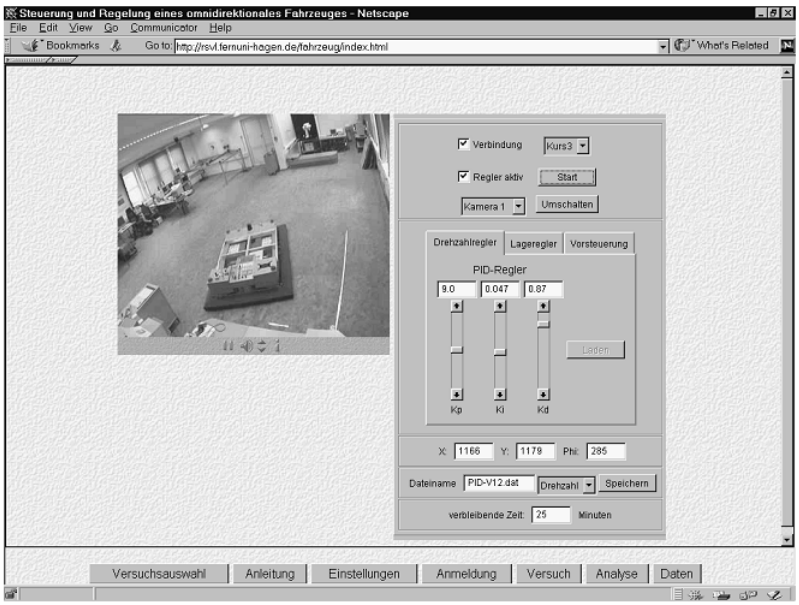

Figure 3.10: User Interface (from [46] with permission)

ratory. This is done using the *JMF (Java Media Framework) API* [36] for using multimedia in Java applets. The other applet controls the experiment itself. This applet simply start an authentication mechanism to validate the user (using username and password). After that, the applet shows the remaining time allocated for this experiment (after which the connection is terminated). Now, the user can send his commands and configuration to the server through data telegrams to be executed. The actual design of the applet is specific to the experiment, that is every experiment has its own design but all of the applets share a common framework.

The server software written in Java uses the *JDBC* (Java DataBase Connectivity) [29] standard to access the database. *mSQL* [51] is used as the database engine. mSQL and its access API mSQL-JDBC are freely available for non-commercial use. The database schema is simply composed of three tables: *The accounts* table to store username, password, time quota and an administrator flag. *The logging* table stores the messages that describes the usage of the project, time of occupance and id of source module. The last table is *The schedule* table that stores the bookings of the students to the actual devices in the laboratory to have time to perform the experiments.

3.5 Web Application Frameworks

Web Application Frameworks are the tools to build a web interface. They aim to facilitate the common tasks associated in web development. Common tasks include accessing databases, software components (for code re-use) and session management. The frameworks differ in the functionalities that can be implemented and in the ease of developing. This is always a compromise for each framework to decide upon.

The most widely used framework in business domain is *Java EE (Enterprise Edition)* [24]. This is because it achieved a good balance with the two extremes discussed previously. From one side, the functionalities to be implemented are not restricted since the developing language used is pure Java. Any desired function can be either implemented directly in Java or be implemented in C/C++ and be imported using *JNI (Java Native Interface)* [26]. Of course, this results in much more code to be written which decreases the ease of development. However, Java EE is used in a modular way, that is the architecture is usually based on the concept of re-usable components (modules). Another reason for the wide spread of Java EE is that it is designed according to standard specifications organized by *JCP* [35]. Furthermore, the fact that Java uses a virtual machine results in a platform independent solution. The last two reasons result in allowing Java EE to be compatible with many other frameworks making it ideal for integrating large projects together to form one enterprise solution.

Another widely used framework is ASP.NET by *Microsoft* [16]. Its popularity come from the relatively ease of development. Many functionalities can be implemented easily by following wizards in the development environment. Of course, this is very convenient for un-experienced developers. Also, professional developers can use ASP.NET to produce web applications with much less effort and time. On the other hand, due to this simplicity of development, some of executable code is hidden from the developer and therefore can not be controlled. This might be a problem if the developer is a professional and knows how to do a certain task but is unable to do it because he has no control over some of the code. Another advantage of ASP.NET is that it is built on *CLR (Common Language Runtime)* [15]. This allows ASP.NET to use code written in many languages (C++, C#, J#, VB, etc).

PHP [41] is also widely used as a general purpose scripting language. It is used to produce dynamic web pages by embedding PHP code inside HTML documents. The resultant mix of PHP and HTML is interpreted by a PHP processor module hosted by a web server to produce a web page. The main advantage of PHP is its simplicity of learning and code reading. Also, it does

not require any sophisticated environment to execute in (just a processor module). However, a big problem with PHP is that it is weakly typed, that is no explicit variable type declaration is required. This usually increases the probability of software bugs and unexpected behaviors. Also, PHP allows security policies to be configured. However, this is not straightforward and needs experts in security field. This is usually underestimated by the obvious simplicity of PHP code.

Many more web application frameworks are used in different projects. Among these frameworks, *Ruby on Rails* [47] which is mainly used in rapid developments, *Django* [5] is written in Python and aims at simplifying the development of large database driven web interfaces and *Flex* [1] which targets RIA (Rich Internet Applications). It is very difficult to claim that a particular framework is absolutely better than all other frameworks. This is because every framework has its own advantages and its own power. It is the role of the software designers to chose (and may be combine) the suitable frameworks according to the requirements of the project to be developed.

3.6 Conclusion

For the presented projects, it is clear that all share some basic requirements. For example, all of them require that safety of the real device is of extreme importance. Also, synchronization between multiple users is an issue to be considered once the device will be controlled over the Internet. On the client side, it is of advantage if the user can directly use the project without installing any further software (that is it should be of no cost to the users). Not surprisingly, all these requirements have been discussed in the envisioned system to be implemented. In addition to that, some requirements are project-specific.

From the architecture point of view, it is also clear that all of the presented projects have nearly the same architecture, that is, they are composed of two main parts, a server side and a client side. The server side should handle the requests and through some logical layer format a proper response. A database is used to persist the data (user accounts, logging information, ...) by different users. There is a controller with data acquisition board used to directly control the physical device. On the client side, a standard web browser is used to control the experiment. The user is presented with an easy to use interface to configure the necessary parameters and control the experiment. To get the feeling of actual presence in the laboratory, a video and audio stream might be used. After finishing the experiment, the user is then presented with the results data gathered during the experiment. The

format of the output varies between simple reports and complex graphs that analyze the execution.

Chapter 4

Website Backend

In this chapter, we discuss the architecture of the system backend. The lowest level of concern to this project is the Simulink scheme used to control the real device. The architectural layers starting from this level and up to the software level that provides the functionality services composes the system backend. All the software components required in this system are presented here along with a brief description of the applied technologies. The chapter starts with the deployment architecture diagram for the whole system. Next, we present the software components that were used as building blocks for the backend. Finally, a discussion about the exact implementation and the connection of these building blocks is presented.

4.1 General Deployment Architecture

The main aim of this project is to control a single device over the Internet. Therefore, the architecture design of the project is centralized in a dedicated server computer (Figure 4.1). This server computer is connected to the Internet to be able to accept user requests. The connection to the Internet is established through a firewall to limit the unauthorized access to the server. On the other end, the server is directly connected to the dSPACE 1104 Board which is capable of controlling input/output ports of the experimenting device.

The server computer is mainly composed of three building blocks:

- **Apache Web Server:**

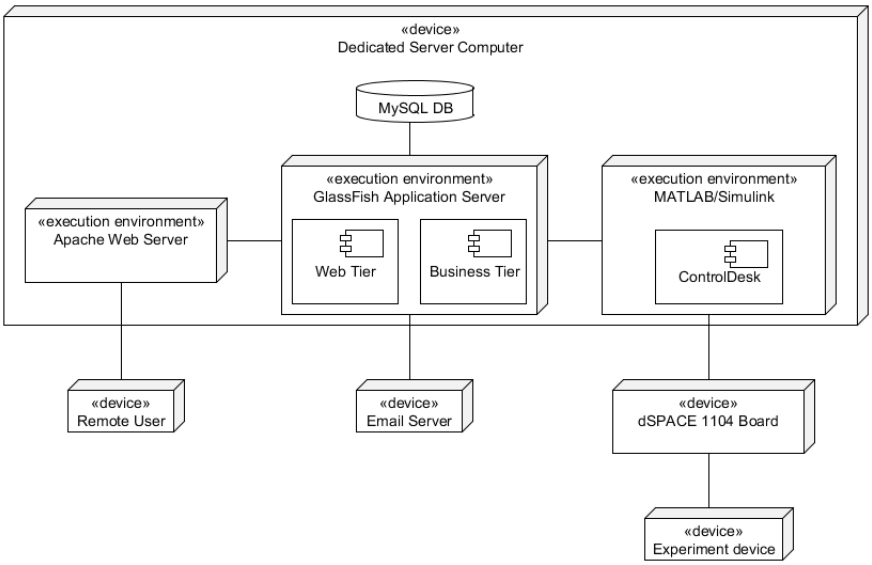

Figure 4.1: Deployment Diagram

The *Apache server* [2] is mainly used in serving static content and dynamic web pages in the Internet. It accepts HTTP requests and generates the appropriate responses. A response can consist of a static page (a single file to be transmitted back to the user) or a dynamic page (e.g. a PHP page). Apache is an open source software that can be installed on many operating systems. Its functionality can be extended using external modules. Examples of these modules are "mod_ssl" and "mod_proxy" for SSL security and proxy support. Requests can be delegated to another server for being processed and forming the results (most often HTML pages) to be sent back to the user. In this project, requests are delegated to the second building block, the GlassFish Application Server.

- **GlassFish Application Server:**

  GlassFish [22] is also an open source Java EE (Enterprise Edition) application server. The version used in this project is GlassFish V3 which is Java EE 6 compliant, that is it fully supports all Java EE related JSRs (Java Specification Requests). Abstractly, it is responsible for handling the business logic requirements of the project. In most cases and in this project, the EJB (Enterprise JavaBean) technology is used to expose the implementation of these business logic requirements.

  Basically, the GlassFish application server hosts two software components (namely, "Web Tier" and "Business Tier"). Those two compo-

nents are considered the frontend and the backend (respectively) of the system. The "Web Tier" component is the higher level responsible for the visual interface the user interacts with (it is described in details in Chapter 5). On the other hand, the "Business Tier" is the lower level that provides all the logical functionality in the system (it is described in details in the following sections in this chapter). The two components are connected to each other using a "controller" sub-component in the "Web Tier" component (this design pattern is called "MVC" which is described in detail in section 4.2).

Another important feature of GlassFish is the support for clustering. This means that in projects that are expected to have high traffic (a large number of requests per unit time), the application server can be installed on many computers and still behave exactly as if it is only one server (so no request can be lost or delayed). This is usually done by using load balancers to balance the traffic between servers. This feature is not used by this project since there is only one device to be controlled and therefore all the requests have to be queued for execution. However, it is possible to extend the current architecture (to have more real devices) and use the installed GlassFish in a clustered environment without modification to the software design.

- **MATLAB/Simulink:**

  MATLAB/Simulink is used in this project to be able to control the hardware device. A Simulink model is designed such that it can control the execution of a controller algorithm on the device. The parameters of this controller algorithm (or the algorithm itself) is sent from the second building block, the GlassFish Application Server. Next, MATLAB/Simulink uses the ControlDesk to execute the controller algorithm over the dSPACE 1104 board. Finally, the results are gathered from the dSPACE board and sent back to MATLAB/Simulink which can be inquired by GlassFish for the results.

A typical scenario in this architecture is as follows:

1. A remote user uses his internet browser to make a request to the server.
2. The Apache web server accepts this request and delegates it to the GlassFish application server.
3. The GlassFish application server accepts the request using its top component "Web Tier". The appropriate business logic is then executed in the lower component "Business Tier".

4. The request is then enqueued in the synchronization queue to be executed; a response message is generated and sent back to the web server which in turn sends it to the remote user.

5. In the synchronization queue, a single request is dispatched and sent to MATLAB/Simulink to be executed.

6. Upon successful execution, the results are inquired from MATLAB and sent by email to the user.

4.2 Backend Components

The software architecture of this project follows the *MVC (Model View Controller)* [23] architecture pattern. The advantage of this architecture is that it totally isolates the business logic functionalities from the interface. This allows for independent implementation for both layers and also eases the injection of new interfaces for the same implementation of the business logic.

- **Model:**

  The Model layer is responsible for handling all the business logic functionalities for the project. In most cases, it also handles the data inside databases through a data access layer. The functions implemented in the Model layer accept input parameters and according to the current state of the model produce output (and/or modify the state).

- **View:**

  The View Layer is considered as the graphical representation of the model state. This layer is actually what the user interacts with. It should keep synchronization with the model state.

- **Controller:**

  The Controller layer is the in-between layer of the Model and the View layers. It is responsible for accepting events from the View layer (denoting user interaction) and handling these events by calling the appropriate methods in the Model layer. This distinction is the actual cause of why it is possible for MVC pattern to have many views. There is one model which can be used in various ways according to the events of the working view.

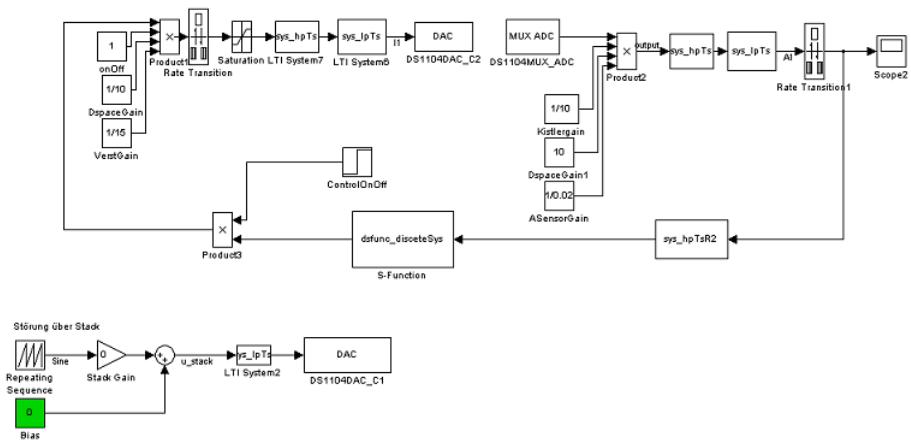

Figure 4.2: Simulink Scheme

As it was shown previously in Figure 4.1, the GlassFish Application Server contains mainly two software components (namely: "Business Tier" and "Web Tier"). The Business Tier component is the implementation of the Model layer in the previously discussed MVC pattern. The Web Tier component implements the View layer as well as the controller layer. Since the View layer is totally independent from the Model layer and the controller layer is responsible for connecting them, it is usual to implement the View and the Controller layers in one software component. This facilitates the process of updating the interface without changing the business logic functionality which is the main reason of using the MVC architecture pattern.

4.2.1 Simulink Scheme

The virtual laboratory implemented in this project uses MathWorks *MATLAB/Simulink* [12, 13] to be able to control the real device. Figure 4.2 shows the Simulink scheme used in this project. Mainly, it consists of two parts, specifically the disturbance signal generator and the controller loop. The lower part of the scheme shows that a "Repeating Sequence" unit is used to generate a disturbance signal. It can be configured by providing a list of time units and a list of corresponding disturbance value at each time unit. The lists will be repeated as long the experiment is running (i.e. a periodic signal is generated). This signal is directly sent to the disturbance patch unit through the dSPACE 1104 board. On the other hand, the upper part of the scheme shows the controller loop. The main part here is the "S-Function" unit (placed at the center of the bottom part of the loop) which

can be used to insert the external algorithm to be executed by the model. The output of this block is the signal to the actuator patch unit. On the right hand side we have the input to the algorithm as the sensor output values.

From the user point of view, this model can be configured by providing the controller algorithm (or only parameters to pre-defined algorithms), the disturbance signal and the simulation time for the experiment. This is the aim of the project, to provide a web interface to easily configure the model and generate an execution report after the simulation finishes. Thus, the web interface backend should be able to send commands to MATLAB and receive the results back from it. This is achieved using the "Java Native Interface Framework" discussed below.

4.2.1.1 Java Native Interface Framework

The *JNI (Java Native Interface (JNI))* [26] is a fundamental technology that is defined in the standard Java Edition [28] (i.e it is not specific to Java Enterprise Edition). Java applications run inside a virtual machine which is responsible of translating the Java bytecode into real instructions (according to the running platform) to be executed by the real machine. However, there exist some well-known, reliable and ready to use libraries written in other languages (mainly C, C++ and assembly). There are also some functions which are performance critical, such that they should be written in low level language (e.g. assembly) to be optimized according to usage. For these reasons, the Java applications require the ability to have a way of using functions written in other languages and calling them as if they were written in Java. This is exactly the main aim of the JNI technology, which defines the interface between Java and other languages. At runtime, upon detecting a call to a native function, the Java virtual machine is able to directly execute the required function on the real machine. After native execution, the control is then returned to the virtual machine to continue interpreting and executing the remaining Java code.

MathWorks provides a library that can be used to access MATLAB engine directly [14]. This library exposes an API for C/C++ and Fortran [10] programs to use MATLAB. Therefore, to use MATLAB from a Java application, a component should be implemented that uses the JNI technology to access the library as C application. To implement a Java interface to this library (which consists of around ten functions), we use JMatLink, a Java open source project that uses the JNI technology to wrap the MATLAB API in three Java classes [18]. Using this project (along with the exposed

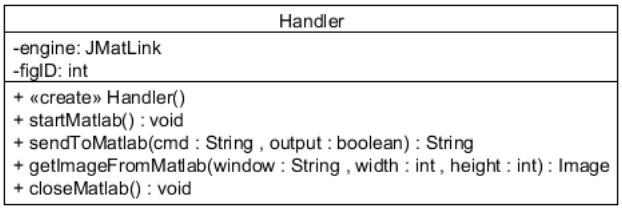

Figure 4.3: Handler Class Diagram

library), it is now possible to control MATLAB from a Java application. This is a list of C functions exposed by the MATLAB library:

- **engOpen:** Open MATLAB engine
- **engClose:** Close MATLAB engine
- **engGetVariable:** Get a variable from the engine workspace
- **engPutVariable:** Send a variable to the engine workspace
- **engEvalString:** Evaluate a MATLAB command
- **engOutputBuffer:** Create a buffer to save MATLAB output
- **engOpenSingleUse:** Open a MATLAB engine session for single, nonshared use
- **engGetVisible:** Get visibility status of MATLAB engine session
- **engSetVisible:** Set visibility status of MATLAB engine session

In this project, another layer of abstraction was implemented. This layer is represented by a Java class "Handler" that manages the session between the web interface backend and JMatLink (Figure 4.3). All MATLAB code sent to its engine passes this layer first. It contains a constructor that is called only one time at the server startup and never got destroyed after that. Also, it has four methods, two of which are concerned with starting and closing the MATLAB engine. Another function is used to send a command to MATLAB; it can be configured to capture the MATLAB's output of that command. Finally, the last function handles the request for images from MATLAB. It keeps track of how many figures are there in MATLAB and asks MATLAB to send an image for requested window with the given width and height.

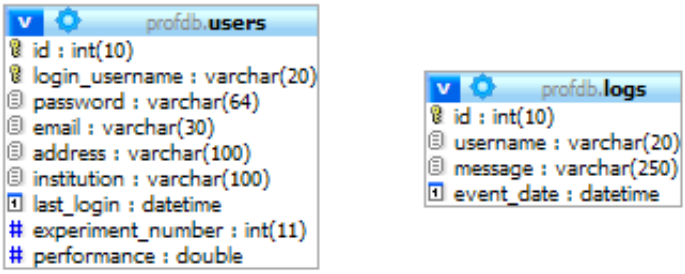

Figure 4.4: Database Schema Diagram

4.2.2 Persistence Layer

The database used in this project is the *MySQL Relational Database Management System (RDBMS)* [34]. It is a stable open source project that is very popular in the field of web applications. Many large scale applications (e.g. Wikipedia and Facebook) use it for data storage.

The database scheme of this project is extremely simple (Figure 4.4) because of the privacy requirements of the project. It is not allowed to store any data related to the algorithms submitted by the users in the persistence layer; rather the algorithm is submitted, executed, and then deleted from the server. According to this, the data in the persistence layer are for managing purposes only. The scheme mainly consists of two tables "users" and "logs" explained below.

The "users" table keeps track from the registered users in the system. The details saved for each user are "username", "email", "address" and "institution". The password for each user is also saved in this table. However, the password is encrypted using the *SHA-2 (Secure Hash Algorithm)* [19]. This algorithm is used to create a message digest which is a condensed representation of the original message. It is computationally infeasible to retrieve the original message given the digest one (that is one can not learn the password from the saved encryption). It is also not possible to find two messages that produces the same digest (that is with high probability, no two different passwords result in the same encryption). For each user, the last login date and the number of performed experiments are saved. These values are used by the administrator to track the usage to the system. Finally, the best performance value for each user is saved to keep track of the best performance values achieved by researchers so far.

The "logs" table is used to keep track of important events in the system.

It keeps track of the event name, the occurrence date and the username responsible of it. The list of important events is:

- New User Registration
- Account Password Reset
- New Experiment Request
- Starting Simulation
- Simulation Finish
- Sending Email
- Email Sent

If there is any problem with any function in the system, then its corresponding event will not be logged in this table. Therefore, it can be easily checked which function has caused the problem. Only the administrator can see this table and delete any events he finds old or irrelevant.

4.2.2.1 Java Persistence API

The *Java Persistence API (JPA)* [27] is a Java framework that facilitates the usage of a relational database from Java applications. It introduced the concept of "Entity Classes" that is considered as the transition between Java language and the database schema. Java is an object-oriented language, so it would deal with "Entity Classes" as ordinary Java classes. Using appropriate configuration, it is possible to map the "Entity Classes" to the database schema. Therefore, instance objects of these "Entity Classes" are automatically mapped to database rows.

JPA also defined *Java Persistence Query Language (JPQL)* [37] which is a query language for the "Entity Classes". It is a platform-independent query language. Implementations for this technology have to convert the query in JPQL syntax to the appropriate syntax according to which database vendor is used. As a result, the application is database-independent since it uses a unified query language (as long as the used implementation supports the database vendor).

This project has two entity classes (namely, "PersonEntity" and "LogEntity" as shown in Figure 4.5) that simply map to the two tables in the database. JPA allows the use of Java annotations to properly configure the

```java
@Entity
@Table(name = "users")
public class PersonEntity implements Serializable {
    @Id
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Basic(optional = false)
    @Column(name = "login_username")
    private String loginUsername;
    @Basic(optional = false)
    @Column(name = "password")
    private String password;
    @Basic(optional = false)
    @Column(name = "email")
    private String email;
    @Column(name = "address")
    private String address;
    @Column(name = "institution")
    private String institution;
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "last_login")
    private Date lastLogin;
    @Basic(optional = false)
    @Column(name = "experiment_number")
    private int experimentNumber;
    @Basic(optional = false)
    @Column(name = "performance")
    private double performance;
```

```java
@Entity
@Table(name = "logs")
public class LogEntity implements Serializable {
    @Id
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Basic(optional = false)
    @Column(name = "username")
    private String username;
    @Basic(optional = false)
    @Column(name = "message")
    private String message;
    @Basic(optional = false)
    @Column(name = "event_date")
    @Temporal(TemporalType.TIMESTAMP)
    private Date eventDate;
```

Figure 4.5: JPA Entities Classes

mapping between the entity class and the database table (that is the configuration doesn't have to be added in external file as it was the case with earlier versions of JPA). These two Java classes are normal Java classes that are annotated with "@Entity" annotation. This annotation declares a class as an "Entity Class" which is mapped to a database table named according to the "@Table" annotations. Each class defines its own set of fields according to the columns in the corresponding database table. The primary key of each table is mapped to a field using the annotation "@Id". Every column in the table is also mapped to a field in the entity class using the annotation "@Column". Two additional annotations were used in this project are: "@Basic(optional=false)" to declare an optional field and "@Temporal(TemporalType.TIMESTAMP)" to declare a date/time field.

4.2.3 Synchronization Mechanism

The requirements of this project are that there is only one device to be controlled and that no two experiments can be performed simultaneously on the same device. Therefore, there is a need for a synchronization component to able to queue all the experiments requests and then process them one after another. The designed synchronization component in this project depends on the "Java Message Service API" discussed in the next section.

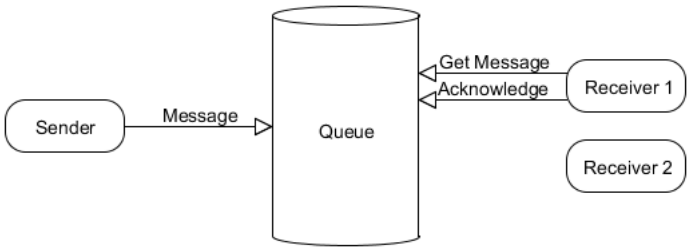

Figure 4.6: JMS point-to-point model

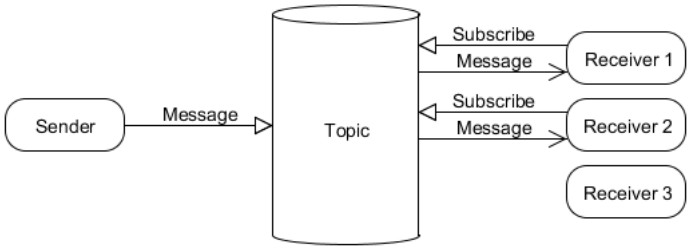

Figure 4.7: JMS publish and subscribe model

4.2.3.1 Java Message Service API

As part of Java Enterprise Edition technology, the *Java Message Service (JMS)* [25] API is defined. The main aim of JMS is to enable Java applications to create, send, receive, and read messages. This ability eases the design of distributed systems. It also introduced the idea of asynchronous communication between software components. Typically, components communicate with each other through an interface, i.e. a set of pre-defined functions. One component would call a function in the interface of another component and then hold on waiting for the reply from that function. This way of communication is synchronized. On the other hand, JMS comes with the ability of unsynchronized communication. That is, one software component can create a message and send it to a pre-defined location and then continue execution without waiting for a reply. Software components that listen to this pre-defined location get notified by the existence of a new message and accordingly receive, read and analyze its contents and finally call the appropriate function on their side.

JMS supports two design models, namely, ”point-to-point” and ”publish

and subscribe". The point-to-point model (Figure 4.6) uses the one sender one receiver principle. That is, one sender creates and sends a message to the queue. Upon receiving the message at the queue, it notifies one of its receivers clients. At the receiver side, there is a pool of receivers from which the queue notify one to handle the message. It is not guaranteed which instance of receivers get notified (it is the task of the used implementation to optimize the usage). However, it is guaranteed that only one receiver handles the message and that it is the responsibility of that receiver to acknowledge the successful handling of the message.

The other model "publish and subscribe" (Figure 4.7) uses the "one sender multiple receivers" principle, which means that the sender (called publisher) publishes his message on a certain topic. At the same time, the receivers (called subscribers) subscribe to that topic, declaring that they want to get messages published in that topic. At the topic side, once it gets a message, it looks for the subscribed receivers (and who are ready to receive new message) and sends that message to all of them. An extra feature is that subscribers that were not ready at the time of publishing a message can configure the topic to notify them later whenever they are ready.

One final note here is that a receiver (in any of the two models) is a *Message Driven Bean (MDB)* [39], that is a Java bean class that is driven by messages. Usually, MDBs have a very short life cycle. Initially, the instances do not exist and then whenever needed they are constructed. By definition, MDBs have to define one method called "onMessage" which is called upon receiving a message. Finally, after finishing handling the message, the instance can wait for some time to handle another message or be destroyed, going back to the initial state "do not exist". The exact behavior is defined in a configuration file.

As per the requirements of this project (Section 4.2.3), the JMS point-to-point design model was chosen to be implemented. So, whenever there is an experiment request submitted on the web interface, it would be validated and then submitted to the queue. At this point, a reply message would be sent to the user interface telling that the request was submitted successfully.

On the server side, the queue should notify one instance of MDBs to handle the request. However, there is a problem that, if there exists two instances in the pool and the queue got two requests, it would give each request to one instance and then allow them to execute in parallel. This contradicts the requirements of the project that there would be only one execution at a time. As a solution for this problem, the pool is configured to have a maximum size of only one instance. Now, if the queue got two requests, it would supply the first to the MDB instance and is not allowed to create another

```
<ejb>
    <ejb-name>JobsQueue</ejb-name>
    <jndi-name>***</jndi-name>
    <bean-pool>
        <steady-pool-size>0</steady-pool-size>
        <resize-quantity>1</resize-quantity>
        <max-pool-size>1</max-pool-size>
        <pool-idle-timeout-in-seconds>300</pool-idle-timeout-in-seconds>
    </bean-pool>
</ejb>
```

Figure 4.8: JMS configuration file

```
@MessageDriven(mappedName = "***", activationConfig = {
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue")
})
public class JobsQueue implements MessageListener {

    public void onMessage(Message message) {

    }

    @PreDestroy
    public void uploadClean() {

    }
```

Figure 4.9: JMS Message Driven Bean

instance to handle the second request. As a result, the second request would be held in the queue till the instance finishes handling the first request and then got notified by the second request. If for some time the queue did not receive any message, then the MDB instance is allowed to be destroyed to free unused resources. By the next request, it should be created again.

Figure 4.8 shows the current configuration for the queue. It typically defines that the pool is allowed to be empty at some time when there are no incoming requests. Then it defines how many instances can be created at once to be ready for incoming requests (this is configured to be only one instance at a time). The maximum pool size (that is the maximum number of instances to live simultaneously) is set to one for the reasons discussed earlier. Finally, if there are no requests for more than five minutes, that instance should be destroyed to free some space. It will be created again upon the next request.

Figure 4.9 shows the abstract implementation (more details on implementation in section 4.3.3.2) of the message driven bean that is used by the queue to handle requests. It is a normal Java class that is annotated as "@MessageDriven" to declare that it is a MDB class. It also implements the optional "MessageListener" interface to show that the type messages it is listening to is the JMS messages. As discussed earlier, the MDB class

should provide a method called "onMessage" that is called upon receiving a message. If the queue did not receive any message for a certain amount of time (configured to be five minutes), it destroys the MDB instance. However, just before destroying the instance, it calls a method annotated with "@PreDestroy" (if it exists) to perform some clean up. It is used here to clean any temporary uploaded file, if they still exists (sections 4.3.3.2 and 5.1.1). Finally, some optional annotations properties are also used in this MDB, namely "acknowledgeMode" which declares that the container should do the receiver acknowledgement automatically and the "destinationType" which specifies the exact type of destination to connect to.

4.2.4 Email Reporting

As per the requirements of this project, the user submits his algorithm for execution on the server and waits for the results back. The results are mainly, the MATLAB output for the commands used in execution along with a detailed report of execution from the device controller. Optionally, the detailed report may ba visualized in a graph. Finally, the results are structured in an email and sent to user by email. The JavaMail API (discussed in next section) is used to send the email.

4.2.4.1 JavaMail API

The *JavaMail API* [30] is an open-source project provided by Sun. It is very simple API and very powerful at the same time. The JavaMail API is not specific to Java Enterprise Edition but is distributed as a standalone library that can be included in the standard Java. A default implementation of the well-known protocols: *Post Office Protocol (POP3)* [8], *Simple Mail Transfer Protocol (SMTP)* [9] and *Internet Message Access Protocol (IMAP)* [7] are currently supported. In addition to that, the modular design of the JavaMail guarantees that new protocols can be easily supported by addition of their implementations as plugins.

The "Session" class in JavaMail API is considered as its entry point. Initially, an instance of this class should be obtained. This instance can be configured to use a specific SMTP provider through its constructor. Also, if authentication is required (that is the mail provider requires authentication before usage), it can be passed also to the session instance. Another feature of the session is that it can be configured in "debug" mode in which it prints the communication messages in details. At this point, it is possible to create an instance of the "Message" class, i.e. the email itself. It can be

configured to have recipients of each type ("TO", "CC" and "BCC"); also the "FROM" field is also configurable. As for the email body, it can vary between a simple text message to multipart message (with attachments) to a HTML message. Finally, the "Transport" class can be used to send the email. Using this easy to use API, robust email sending is now possible. Sample 1 shows a basic example code to send an email with attachment.

4.3 Business Tier Architecture

In this section, we discuss the design of the Business Tier Architecture for this project. Basically, we detail here how the software components discussed in Section 4.2 were used in this project. From a general perspective view, we can see the "Business Tier" as a provider for the set of services of the project. The services are provided using the concept of "Session Beans". Next, we discuss the "Entity Manager" that is used by the "Session Beans" to query database (Section 4.2.2). Finally, a discussion about the exact implementation of the provided services is presented.

4.3.1 Session Beans

A fundamental part of the business tier component is the concept of *Session Beans* [40]. As it is the case with already discussed beans types, session beans are normal Java classes with the appropriate annotation. They are used to implement the business logic functions required by the project. However, they are implemented as a services; that is a service provided by the project backend component (business tier). It should be wisely implemented so that each function in a session bean implements exactly one service provided by the project and returns the appropriate response. The services provided by a session bean can be accessed from other components on the server (locally) and/or from external users (remotely). To define exactly the appropriate way of calling a function, two interfaces can be defined annotated by "@Local" and "@Remote". Methods signatures defined in an interface allows those methods to be used either locally, remotely or both ways.

There are mainly three types of session beans:

- **Stateless Session Beans:**

  The server keeps track of a pool of these session beans instances. When a user asks for a service from these beans, the server is responsible for

Sample 1 Simple send email using JavaMail API.

```
/*
 * Properties for the Session object initialization.
 */
Properties props = System.getProperties();
props.put("mail.smtp.host", ***);

/*
 * Construct a Session object
 */
Session session = Session.getInstance(props, null);
session.setDebug(true);

/*
 * Create the message.
 */
Message msg = new MimeMessage(session);
msg.setFrom(new InternetAddress(***));

msg.setRecipients(Message.RecipientType.TO,
            InternetAddress.parse(***, false));
msg.setRecipients(Message.RecipientType.CC,
            InternetAddress.parse(***, false));

msg.setSubject("Email Subject");

/*
 * Add attachment file.
 */
MimeBodyPart mbp1 = new MimeBodyPart();
mbp1.setText("Email Text: See attachment");
MimeBodyPart mbp2 = new MimeBodyPart();
mbp2.attachFile("report.txt");
MimeMultipart mp = new MimeMultipart();
mp.addBodyPart(mbp1);
mp.addBodyPart(mbp2);
msg.setContent(mp);

// send the email
Transport.send(msg);
```

picking a free instance bean and calls the required service from it, finally return the response to the user and returning the instance back to the pool. Upon the next request (either from the same user or from other user), the same process is executed again which picks a free instance, calls the service, returns the response to user and returns the instance to the pool. Thus, it is not guaranteed that calling two services consecutively result in the same instance bean executing them. Thus, from design point of-view, no user data should be saved with the instance. Typically, it is used in simple services that do not depend on each other. Most session beans used in this project are of type "Stateless". Sample 2 shows an example stateless session bean that implements the service of login of just one user (for simplicity).

- **Stateful Session Beans:**

  This type of session beans keeps track the of communication with the caller. This is accomplished by returning the reference of the instance used with the user's first request. This reference (the same used bean instance) is used in all subsequent requests coming from that user. Once, the user terminates the communication, the bean is re-initialized and returned to the pool to be ready to server subsequent users. A disadvantage of this type is that it can not be called as a web service, using the *Simple Object Access Protocol (SOAP)* [53] protocol for example. This is mainly because SOAP is based on XML to format the messages and the concept of object references is not defined in it.

- **Singleton Session Beans:**

  Another type of session beans is the singleton session bean. This type is very similar to the stateless session beans in the way it provides the services. However, as is implied from the name, only one instance is allowed to be created per application. Therefore, the pool size is just one instance that is created at the server startup (using the annotation "@Startup"). Since it is only one instance, it is allowed to share state between calls (as the stateful session beans). Also, it can be called as a web service (as the stateless session beans).

In general, session beans are not method reentrant. This means that, no two threads can execute one method simultaneously. Thread-Safety is guaranteed by the server and is totally abstracted from the programmer concerns. However, this raises a problem with the singleton session beans. Since it is only one instance, it would become a bottleneck between requests. This is solved by declaring the required behavior through the annotation "@ConcurrencyManagement". If the specified management is "CONTAINER", then it is delegated to the server to handle thread-safety. However, the programmer can use a "@Lock" annotation to improve the performance. This

Sample 2 Simple Stateless Session Bean.

```
@Stateless
public class LogInBean implements LogInBeanLocal {

    public String logIn(String username, String password) {
        String result = "";

        if(username.equals("Admin") {
            result = "success";
        } else {
            result = "fail";
        }

        return result;
    }
}

@Local
public interface LogInBeanLocal {
    public String logIn(String username, String password);
}
```

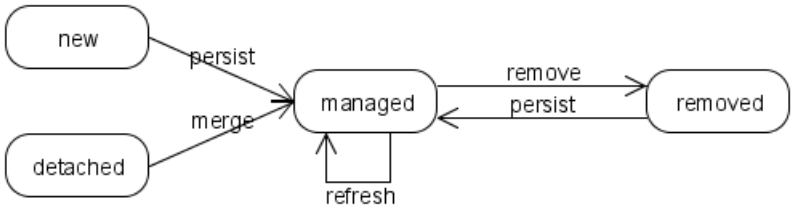

Figure 4.10: Entity Class States

annotation can have value "READ" which means that threads are allowed to reenter this method concurrently. The annotation can also have value "WRITE" which means that no other thread should enter any method in the bean until this (the annotated method) finishes. On the other hand, another management policy is "BEAN" which specifies that the programmer will handle thread-safety himself (using the "synchronized" keyword for example).

4.3.2 Entity Manager

In Section 4.2.2.1 "Java Persistence API", the concept of "Entity Classes" was discussed (Figure 4.5). It was shown that these classes are used to represent the database tables as Java objects. At this layer, the entity classes are managed by the *Entity Manager* [33] which is responsible of creating, persisting or removing entities from the database. It can also synchronize entities with the database.

From the point of view of the Entity Manager, an entity class is one of exactly four states (Figure 4.10):

- **new:**

  Initially, to store a new object in the database, this object should be created through the regular constructor. After construction, we get an instance of an entity class in the state "new", that is not managed by the entity manager yet.

- **managed:**

  For an instance in the state "new" to be managed by the entity manager, it should be persisted in the database. Thus, "persist" method in the entity manager should be called on this instance.

- **detached:**

  The state "detached" means that we have a reference to an instance of an un-managed entity class. However, this instance is not new (it is persisted in the database). For this reason, it is not possible to call "persist" method to make a managed instance (would result in the insertion of new instance). Therefore, for this case, one should use "merge" method that takes the un-managed instance and returns the managed one from the database.

- **removed:**

  An instance in the state "managed" can be totally removed from the database by calling the "remove" method. This results in removing the instance from the database and declaring the instance to be "removed". At this moment, a "removed" instance can be re-inserted into the database as if it is "new" by calling the "persist" method.

Sample 3 shows an example usage for the Entity Manager. An instance of the Entity Manager is achieved using the injection mechanism using the "@PersistenceContext" annotation. Afterwards, the manager is initialized and ready to be used directly. In this sample it is first used to add a new user to the database. The user is constructed normally and is declared to be in "new" state. Its fields can be handled normally. Calling the persist method causes it to be saved in the database and the instance itself is now in "managed" state. The second method is used to delete a method from the database. The user to be deleted is passed as an argument (in the "detached" state since it is not "managed" and exists in the database). Therefore, it should be "managed" before being deleted. This is the job of the "merge" method. After that, we have a "managed" instance on which we call "remove" method to delete it from the database. The third method shows how can one search in the database easily using the entity manager. Assuming that the user name is his primary key, by simply calling the "find" method and passing the class type of the instance and the primary key value, we receive the "managed" user we were looking for. Finally, the entity manager can execute any query and return the results. In this example, it simply returns all users from the database.

A final note about the Entity Manager is that its ability to handle transactions. Transaction deals with some changes to be done in the database. However, all the changed should be performed or none of them. It is not allowed to have part of the changes performed and not the rest. This is exactly what the entity manager perform for the programmers automatically. By starting any service in the session bean, the transaction begins. All the changes within this method to the database are performed in the scope of

Sample 3 Simple Usage for the Entity Manager.

```
@Stateless
public class UserBean implements UserBeanLocal {

    @PersistenceContext
    EntityManager em;

    public void addUser(){
        User user = new User();
        user.setName("Admin");

        em.persist(user);
    }

    public void deleteUser(User user){
        User managedUser = em.merge(user);
        em.remove(managedUser);
    }

    public User getUser(String name){
        User user = em.find(User.class, name);
        return user;
    }

    public List<User> getAllUsers() {
        Query query = em.createQuery("SELECT u FROM User u");
        return query.getResultList();
    }
}
```

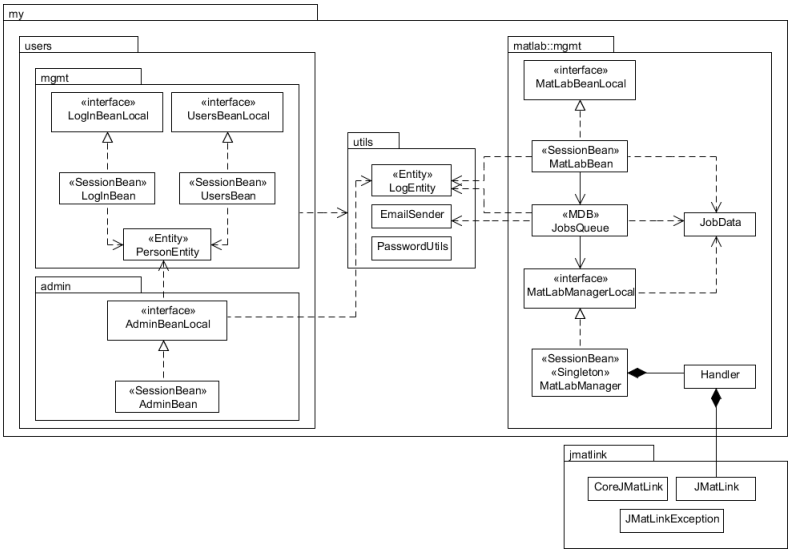

Figure 4.11: Business Tier Package Diagram

the begun transaction. However, if any error happened, the transaction is automatically rolled back to the initial state. Of course, the programmer is allowed to force his own transaction management policy using the "@TransactionAttribute" annotation.

4.3.3 Business Tier Implementation

In this section, the detailed services presented by the business tier are discussed. The package diagram (Figure 4.11) shows the general design of the business tier in this project. At its core, there are five sub-packages. Two of the packages are related to user management services (namely "mgmt" and "admin" sub-packages), and thus are placed in one bigger package. Another package deals with the experiment request services. The fourth sub-package "utils" contains three classes (namely "LogEntity", "EmailSender" and "PasswordUtils"). The three classes were already discussed before in sections 4.2.2.1, 4.2.4.1 and 4.2.2 respectively. The last sub-package is the "JMatLink" which is also discussed in Section 4.2.1.1. The following sections will discuss the three sub-packages.

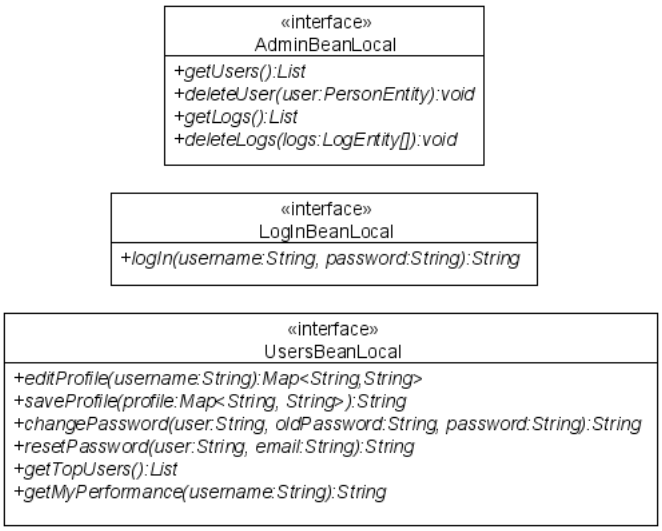

Figure 4.12: Users Management Services

4.3.3.1 Users Services

The users management services are divided into services special to the administrator, services for all other users and login service for both. The administrator is able to show all the registered users and delete the account of any user he wants. These two services are performed by using the methods "getUsers" and "deleteUser" respectively. In the same manner, the administrator is able to see the logged events and delete any of them. These other two services are performed by using the methods "getLogs" and "deleteLogs" respectively.

On the other hand, a normal user can request the following services:

- **"editProfile":**

  The user passes his username to this service and ask for his profile. Since the username is unique value among users, it is allowed to search by username to retrieve the user profile.

- **"saveProfile":**

  The "saveProfile" service performs two different tasks according to the caller context. It is either used to register a new user account or to update a user profile. In both cases, the user passes his new profile data to the service. In case of a user registration, a new "PersonEntity" is

created with the passed profile data. The password field is encrypted (section 4.2.2) before being saved to the database. On the other hand, in case of an update profile, the old user profile is fetched from the database. Then, the new profile data replaces the old data. Finally, since the profile was already fetched from database, it is in "managed" state. Therefore, the modifications to it are directly persisted in the database.

- **"changePassword":**

  This service is used to modify the user password. The user should pass his username, old password and the new password. The username is used to fetch the user account from the database. The old password is encrypted and compared to the saved value. Upon successful match, the new password is encrypted and replaces the old value.

- **"resetPassword":**

  This is similar to the "changePassword" service. However, the passed parameters are the username and the email address. The username is used to fetch the user account from the database. The passed email address is compared to the saved value in the account. Upon successful match, a new password is randomly generated and encrypted to replace the old value. The unencrypted new password is sent to the user via email.

- **"getTopUsers":**

  This service is used to show the researchers sorted according to their performance value. It executes a custom query to the database to fetch the first five results out of all users having positive performance value ordered descendingly.

- **"getMyPerformance":**

  This is a simple service that given the passed username, it fetches his performance value only.

Finally, the login service is common to all users. The user passes his username and password to this method. The username is used to fetch the user account from the database. The password is encrypted and compared to the saved value. Upon successful match, the method authorizes the supplied username.

4.3.3.2 MATLAB Services

The "matlab::mgmt" sub-package is concerned with the experiment services. The main entry point to the experiment is through the service "submitJob"

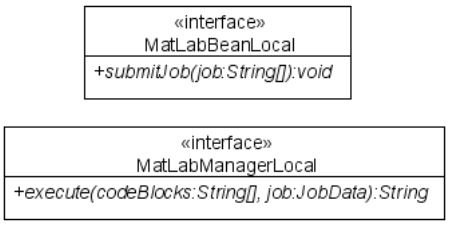

Figure 4.13: MATLAB Services

in "MatLabBean". This service main job is to locate the queue MDB (Section 4.2.3) and construct a message for the passed job data. The message is then send to be enqueued and immediately return reply to the user that his request was enqueued successfully.

Sample 4 Processing Experiment Request

```
public void onMessage(Message message) {

    // get job data from the message

    // create temporary folder to be used in simulation

    // if the user uploads a file, move it to temporary folder

    // read the external commands file

    String output = matLabManagerLocal.execute(code, job);
    doReport(attachments, username, userEmail, output);

    // delete temporary folder with all its contents
}
```

On the queue side, all messages are processed one after another. Sample 4 shows abstractly what is done for each experiment request. First, it dispatches the job data from the current message. After this, it starts building a private environment to be used while simulating the experiment. At this point, the external commands file is parsed for the MATLAB code to be executed for this job. The main aim of this file is the ability to modify the behavior of the experiment at runtime without the need to re-compile or re-install the web interface (6.2.1). Next, the collected commands along

with the job data fetched earlier from the message itself are sent to the "MatLabManager" to be processed. The result from "MatLabManager" is the MATLAB reply for each command sent to it. This is then used to form the report and send it to the user via email. Finally, the private folder (that is generated per request and contains all the files used in execution by request e.g. uploaded files and results images) is deleted along with all its contents. If some configurable amount of time has passed without having any messages on the queue, the MDB is deleted. But just before that, it cleans up any private folder or temporary upload folder (details on upload folder can be found in Section 5.1.1).

The "MatLabManager" is a singleton session bean (section 4.3.1). It is constructed right after the server start-up. Initially, a MATLAB session is opened. Then according to the job data, the relevant code blocks are processed. A command in a code block contains variable names which should be replaced by real values coming from the job data. A buffer is used to store every MATLAB reply to generate a cumulative reply for the whole job. Furthermore, if the external commands file specifies a scope screen to be saved, it is the responsibility of the "MatLabManager" to ask the handler for the an image from MATLAB (Section 4.2.1.1). The incoming images are saved in the private folder created especially for this job. Finally, the MATLAB session is closed.

Chapter 5

Website User Interface

In the previous chapter, we discussed the MVC model in details and presented the architecture of the "Model" layer. In this chapter, we continue to describe the remaining two layers (namely, "View" and "Controller"). All the technologies used in the implementation of the two layers are also discussed in this chapter. Finally, the chapter concludes with the "Controller Layer Implementation" which presents the actual mapping of functionalities between the user interface "View" and the backend "Model" layers.

5.1 The View Layer

The view layer is concerned with the interface that the user interacts with. In general, the view layer can vary between desktop application, applets or other technologies. However, since this project is concerned with building a web interface, the chosen interface to be built consists of web pages.

The technology used in this project is the *JavaServer Faces 2.0 (JSF)* [32]. This is a new technology introduced in the Java Enterprise Edition 6. This project uses the *Facelets* [31] framework which is the preferred presentation technology for JSF projects. It uses "xhtml" documents [54], i.e. it requires valid XML documents to work correctly. It also supports the *Unified Expression Language* [38]. This enables the web pages to be easily connected to managed beans (Section 5.2.1) in the controller layer.

Figure 5.1 shows the interaction between different web pages in this project. The default home page for this project is the "login.xhtml" page. It allows the user to login to the web interface if he is already a registered user. If

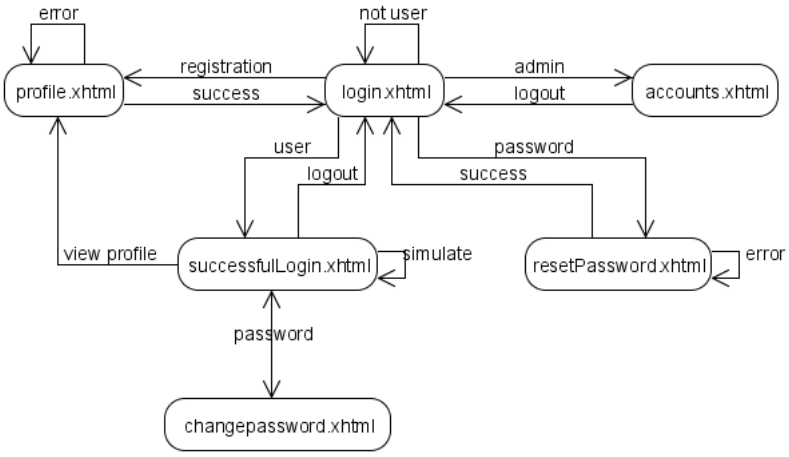

Figure 5.1: Pages Flow

the user didn't register for an account before, he is presented with a link to the registration page "profile.xhtml". The registration page either accepts the new account and returns the user to the "login.xhtml", or in case of any error, it gives him another opportunity to create an account. Also, at the "login.xhtml" page, if the user is already registered but cannot remember his account password, he is presented with a link to a page "resetPassword.xhtml" to reset the forgotten password. The "resetPassword.xhtml" page requires the user to enter a valid username along with the registered email address. In case the given information matches an account in the database, the backend component generates randomly a new password and sends it to the user via email. Finally, if the user is the project administrator, he can provide the username and password for the administrator account, and the "login.xhtml" page will directly redirect him to the administration interface.

Once a normal user passes the "login.xhtml" page, he is redirected to the "successfulLogin.xhtml" (Figure 5.2). This is considered the most important page of the web interface, because it is the page that allows the user to experiment with the device. In this page, the user can input the experiment parameters and/or upload files. Upon passing an initial screening process of validating the input data, the user is allowed to submit his algorithm for simulation. Also, the user is able to update his profile through the "profile.xhtml" which is already pre-loaded with the saved user profile. Moreover, the user can change his account password, provided that he knows the current password (this is extremely recommended to be used after reset

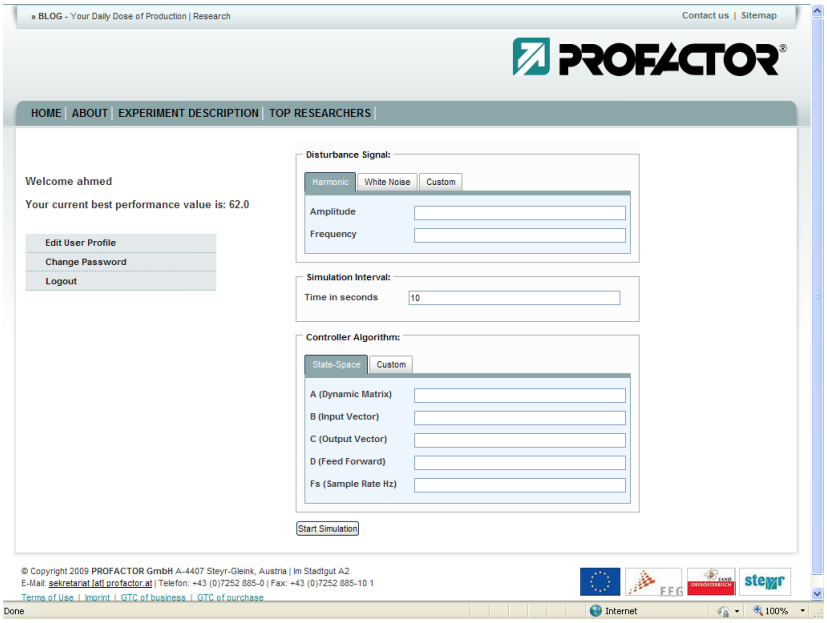

Figure 5.2: Experiment Page

password, since the new password is randomly generated and difficult to be memorized).

Some of the pages are open to everyone, i.e. it is not required to login before visiting them. These pages are "about.xhtml", "description.xhtml" and "top.xhtml". The page "about.xhtml" explains the background information of the purpose of the experiment. It also discusses the general description of the experiment device. Finally, it discusses the typical usage of the web interface with the emphasize on the privacy of algorithms submitted by users. The second open page "description.xhtml" provides a detailed scientific description to the experiment. It should be used as a reference material for researchers for this experiment. Finally, the open page "top.xhtml" shows the best five researchers according to performance values achieved by their submitted algorithms.

A final note about the View Layer is that it the project should have the same look and feel of the current running web interface. For this reason, the stylesheets that were used with the old web interface are also used in this project. This to make sure that the exact styles (colors, elements width, images, ...) are also applicable in this project. Moreover, to give exactly the same feeling when using the new interface, it was decided to use exactly the same header and footer of the old web pages. Therefore, all the web pages in this project share a common header and footer pages which were already

used by the old web interface.

5.1.1 PrimeFaces

PrimeFaces [52] is an open source project that provides an extensive set of presentation component to be used directly in Java Server Faces projects. It also supports *Asynchronous Javascript And XML (AJAX)* [55], a technique that enables developing of dynamic pages by sending asynchronous requests to the server, get a reply and update the page view without reloading the whole page. This extremely enhances the usability of the web interface. Also, the library contains an extensive documentation about every component it provides, making it simple to use or even modify a component according to one requirements.

Mainly, PrimeFaces is used to provide the tabs view in the experiment page (Figure 5.2). This gives the user the feeling that, for a given section, he has multiple options to choose one from. Upon choosing a suitable option for him, he is presented by the corresponding parameters to the selected view. Upon submitting an experiment, the selected views are used to detect exactly which parameters to be used in the experiment.

Finally, the PrimeFaces library is also used to provide a way for the user to upload a file. It provides a simple component to upload a file to the server. The component can be configured to set an upper bound for the size of uploaded files and also to upload the files to a temporary folder on the server to be used later (the uploaded files reside in server memory. If the total file size exceeds a certain configurable threshold, the files are written to disk at this temporary folder). Sample 5 shows a basic configuration for the file upload component. It describes a filter that is provided by the library to handle the multipart request (from file upload). It then configures a threshold of 51200 bytes so that any larger files should be saved in the default temporary folder. Another filter is used (after the multipart filter) to detect the open pages and those that requires logged in user. It is called "LoginFilter" and placed in the "filters" package (more details are provided in Section 5.2.3).

5.2 The Controller Layer

The Controller layer is the middle layer between the Model layer and the View layer. It connects the View layer with the corresponding services offered by the Model layer. As defined by the specifications of JSF, it is

Sample 5 PrimeFaces File Upload Configuration

```
<filter>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <filter-class>
        org.primefaces.webapp.filter.FileUploadFilter
    </filter-class>
    <init-param>
        <param-name>thresholdSize</param-name>
        <param-value>51200</param-value>
    </init-param>
    <init-param>
        <param-name>uploadDirectory</param-name>
        <param-value>***</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>
```

also responsible for mapping the values from HTML forms in web pages to fields in Java classes. This is important as it abstracts much of code that is typically used without modifications. Having, the entered values in fields in Java classes also simplifies the validation process.

5.2.1 Managed Beans

The Managed Beans are normal Java classes that are annotated with a "@Named" annotation. They are constructed and managed by the server and can be injected at anytime using the "@Inject" annotation. Using this annotation, we can get a reference to an instance of this bean. An important feature of the managed beans is their scope. Every managed bean should have a scope in which it is alive. Three scopes are defined, namely "RequestScoped", "SessionScoped" and "ApplicationScoped". It is clear from the names that "RequestScoped" lives only during a request. That is, a user submits a form, the values are mapped to the corresponding managed bean. A response is generated and sent to the user. The managed bean is now useless and should be destroyed. It is then re-constructed with the next request. In the second scope type "SessionScoped" beans live longer than in "RequestScoped". This scope type keeps track of the user session

and keeps the submitted values stored in it. If the user re-visits the form he already submitted, he gets the form pre-loaded with his original values (this also enhances the usability of the web interface, especially in long forms). Finally the "ApplicationScoped" beans are similar to the "SessionScoped" but they live for the whole lifetime of the application. They also share data between users.

Sample 6 Simple Usage to Managed Beans.

```
@Named("user")
@SessionScoped
public class User {

    private String firstName;
    private String lastName;

    @Inject
    private Order order;

    @EJB
    UsersBeanLocal usersBean;

}

<h:outputLabel value="First Name"/>
<h:inputText value="#{user.firstName}"/>
<h:outputLabel  value="Last Name"/>
<h:inputText value="#{user.lastName}"/>
```

Sample 6 shows a simple example of using a managed bean to map the values of input elements in the HTML forms to the fields in Java classes. Another managed bean is injected here using the "@Inject" annotation. The sample also shows how the managed bean is able to call the session beans in the Model Layer (Section 4.3.1). This is done using the annotation "@EJB" which can inject an instance of the required interface of the session bean. This instance can be then called normally to perform its service and return back to the managed bean.

5.2.2 Bean Validation

Another important feature defined in the Java Enterprise Edition is the *Bean Validation Framework* [21]. The framework supports validation at different layers. At the presentation layer (View layer), the elements of web pages can be validated using custom validation elements. Another level of validation can be at the managed bean itself. Sample 7 shows examples of both levels of validation. The first example shows the input element should have a value of length between five and twenty characters to be accepted. The second example shows the validations at the managed beans. Basically, it has the same meaning of the first example but at different level. Moreover, it constrains the age to be within 10 and 70.

Sample 7 Simple Usage to Bean Validation.

```
<h:inputText id="firstName" value="#{user.firstName}">
    <f:validateLength minimum="5" maximum="20" />
</h:inputText>

public class User {
    @NotNull
    @Size(min=5, max=20)
    private String firstName;

    @NotNull
    @Min(value = 10)
    @Max(value = 70)
    private Integer age;
}
```

In addition to the already defined constraints, the framework defines an interface "Validator" to be implemented by any user-defined constraints. At time of validation, the framework will call the "validate" method and, if the method doesn't throw exception, the input is considered valid. This custom validation technique is used heavily by this project as follows:

- **"HarmonicValidator":** Constrains the values of the Harmonic disturbance signal to be numeric values.
- **"NoiseValidator":** Constrains the values of the White Noise disturbance signal to be numeric values.

- **"ParamAValidator":** Constrains the input of Parameter A of the state-space equations to be a square matrix.
- **"ParamBValidator":** Constrains the input of Parameter B of the state-space equations to be a column vector.
- **"ParamCValidator":** Constrains the input of Parameter C of the state-space equations to be a row vector.
- **"ParamDValidator":** Constrains the input of Parameter D of the state-space equations to be a numeric value.
- **"PasswordValidator":** Makes sure that the value of the password field is the same as the value of the confirm password field.
- **"SamplingValidator":** Constrains the input of the Sampling Rate of the state-space equations to be a numeric value that can divide six.
- **"TimeValidator":** Constraints the input of the Simulation Interval field to be an integer value less than one minute.

All the validations techniques supported by the framework are able to provide an appropriate suitable error message to the user. All error messages are written in one file and can be easily translated to other languages. In addition to that, every technique has a way of adding a custom message to the user. The framework also provides a "message" element to be added in the web pages that displays the error messages from any level. The "message" element is used to display error messages for a specific input element. However, the "messages" element is used to collect all other error messages. Finally, the framework supports accessing the "message" or "messages" elements programmatically and adding any message to them (from managed beans methods for example).

5.2.3 Controller Layer Implementation

This section details the implementation of the controller layer for this project. The package diagram (Figure 5.3) shows the general design of the controller layer. At its core, there are five sub-packages. Two of the packages are related to user management functionalities (namely "mgmt" and "admin" sub-packages), and thus are placed in one bigger package "users". Another package deals with the experiment functionalities. The forth sub-package is just the "utils" package and it contains nine classes (the detailed usage of each class were discussed in Section 5.2.2). The last sub-package is the "filters" which is also discussed in Section 5.1.1. The following sections will discuss the three sub-packages.

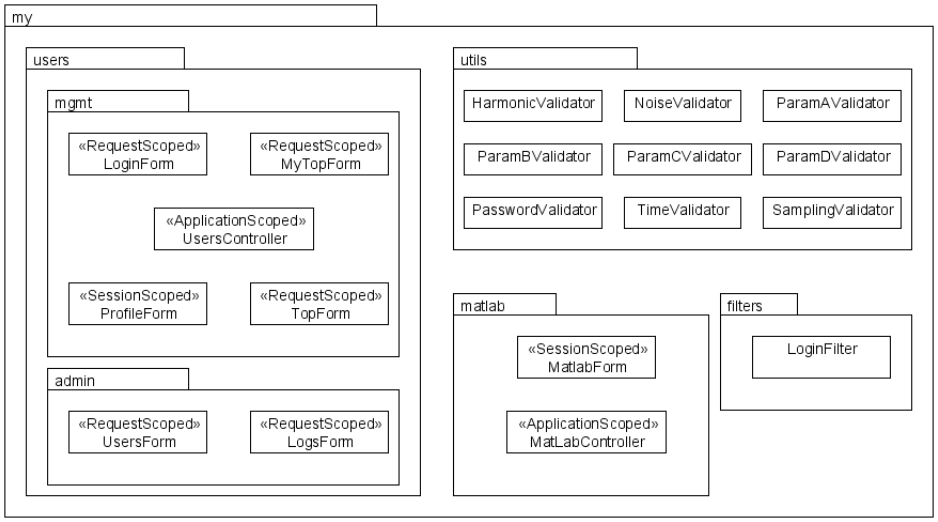

Figure 5.3: Controller Layer Package Diagram

5.2.3.1 Users Functions

The users functionalities in this web interface are distributed in seven managed beans as shown in Figure 5.4. The first managed bean that is used by the user is the "LoginForm" bean. It allows the user to enter his username and password which are mapped to the fields in this bean. The bean also contains an extra field that informs the user in the case of a wrong username or password. The HTML form is submitted to the "loginFunc" in the "UsersController" bean. This function simply collects the username and password from the "LoginForm" and asks the login service in the backend to validate them. According to the result, the user is either directed to the experiment page or receives an error message.

The "ProfileForm" managed bean is used in two different web pages, namely the registration page and the update profile page. This is because the similarity of the structure of the pages. It keeps track from the user profile data. In addition to that it has a boolean flag to disable the input element of the username in case of the update profile (it is not allowed to modify the username after registration). In a same manner the submit button label is different in the two pages, it is called "Save" in the update profile page and "Register" in the registration page. The HTML form is submitted to the "saveProfileFunc" in the "UsersController" bean. This function collects the profile data and pass ask the save profile service in the backend to register or update the profile. To have the update profile page preloaded with the

Figure 5.4: Users Management Functions

profile values, the function "editProfileFunc" is used.

Parts of the "ProfileForm" is also used in the change password page and the reset password page. The change password page uses the fields "oldPassword" and "password" to allow the user to modify his account password. This page submits to "changePasswordFunc" in the "UsersController" bean. On the other hand, the reset password page uses the "userName" and "email" fields to allow the user to reset his account password to a new randomly generated value. This page submits to "resetPasswordFunc" in the "UsersController" bean.

The "MyTopForm" managed bean is a simple bean used to retrieve the performance value of the logged in user to display it in the experiment page. The actual loading of the value is done at the "myInit" method which is annotated with "@PostConstruct" to declare that this method should be called directly after the constructor finishes. In the same manner, the "TopForm" managed bean is used to retrieve the list of top researchers according to their performance values.

For the administrator, the managed bean "LogsForm" is used to list all the logged events in the system. The administrator is able to select some events to be deleted. The selected events are then mapped to the list "selectedLogs". Moreover, the number of saved events is shown to the administrator (to keep track of system usage) and is mapped to the "size" field. Finally, whenever the administrator submits the form by clicking on the "Delete" button, the "delete" method is called and all the events that were in the "selectedLogs" are deleted. In a same manner, the "UsersForm" managed bean is used to list all the registered accounts in the database. Also, the administrator is able to select any user account to be deleted and also the method "delete" is used for deleting the "selectedUser".

5.2.3.2 MATLAB Functions

The experiment functionalities in this web interface are simply divided into "MatlabForm" and "MatLabController" managed beans as shown in Figure 5.5. The "MatlabForm" managed bean keeps track of every user input on the experiment page. The "distSelectedTab" is used to detect which tab is selected by the user in the disturbance signal part. Likely, the "algoSelectedTab" detects which tab is selected in the algorithm part. The fields "amplitudeHarmonic", "amplitudeWhite", "frequency" and "distUpFilename" are used in the input of the disturbance signal. However, the fields "paramA", "paramB", "paramC", "paramD", "paramTs" and "algoUpFilename" are used in the input of algorithm part. The field "time" is

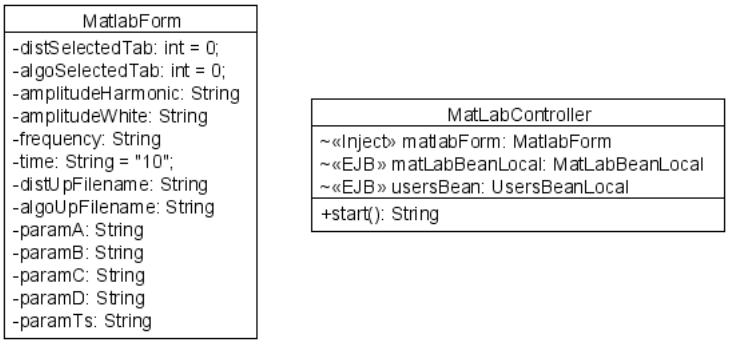

Figure 5.5: Experiment Functions

just a mapping for the simulation interval user input. It defaults to ten seconds. Finally, when the user submits the HTML form, it calls the "start" method in "MatLabController" managed bean. The main job of this method is to collect all the user input from the form, create a job array that holds all relevant information and sends it to the backend. If the request is successful, the backend submits the job to the queue and directly return to the "MatLabController" which informs the user to wait for the results via email.

Chapter 6

Additional Features

The development of this project was controlled by the continuous interaction between the company "Profactor" and the author of this thesis. That is, the started with an initial phase of requirements specifications. After the completion of this phase, the development lifecycle starts with selecting a subset of requirements to be implemented in the current development cycle. Next, another phase of refining the requirements specifications for the chosen requirements starts. This is usually achieved by a meeting with the company to establish a concrete understanding of the exact behavior for the chosen requirements. Finally, the development phase starts to implement those chosen requirements. This ends the current cycle and immediately starts a new cycle with a new subset of requirements. For the new cycle, the meeting held in the phase of refining the requirements includes also confirmation about the results for the implementation phase of the previous cycle.

It is usually the case that new requirements rise up during the development of a software project. These new requirements vary between new features (that would increase the usability of the system) to new critical requirements (that the system would not be able to operate correctly without them). The second type usually indicates that the initial requirements specification phase were not successful in figuring out the requirement of the whole system. It is also very difficult and costs a lot to recover from unsuccessful or incomplete requirements specification. On the other hand, it is possible to add new features to the system with minimum amount of cost. However, there are some cases where the architectural design of the system (that would satisfy the original requirements) may not be able to satisfy the new feature. In these cases, a compromise should be done to compare the gain of implementing this new feature against the cost of modifying the system design.

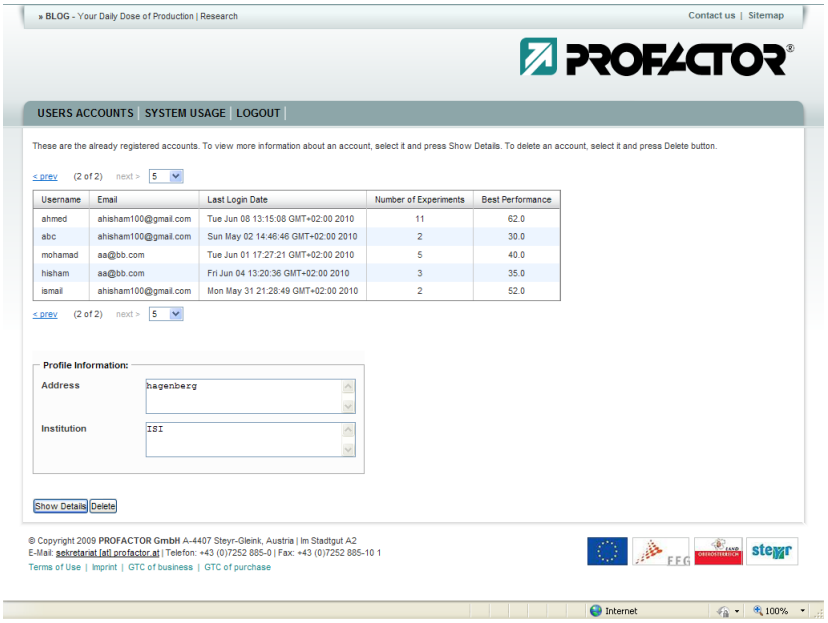

Figure 6.1: Administrator Interface - User Accounts

In this project, the new features were possible to be added to the system without modifying the system design a lot. Therefore, it was decided to implement these features as they effectively increase the usability. This chapter discusses the new features that were added to the system during the development phase.

6.1 Administrator Interface

After the development of the user account functionality (registration, login, logout, edit profile and reset forgotten password), the system administrator required that he is able to delete a user account at any time. In order to fulfil this new requirement, it was decided to have an administrator view in the web interface. This account is accessible from the normal "login" page. However, to restrict the usage of this account to only the system administrator, it has to be used from the server in the company itself. That is, it can not be used remotely over the Internet.

In this administrator view, a list of all registered accounts is displayed (Figure 6.1). This list is scrollable by a user defined amount of accounts per page (for example the figure shows the last five registered users accounts).

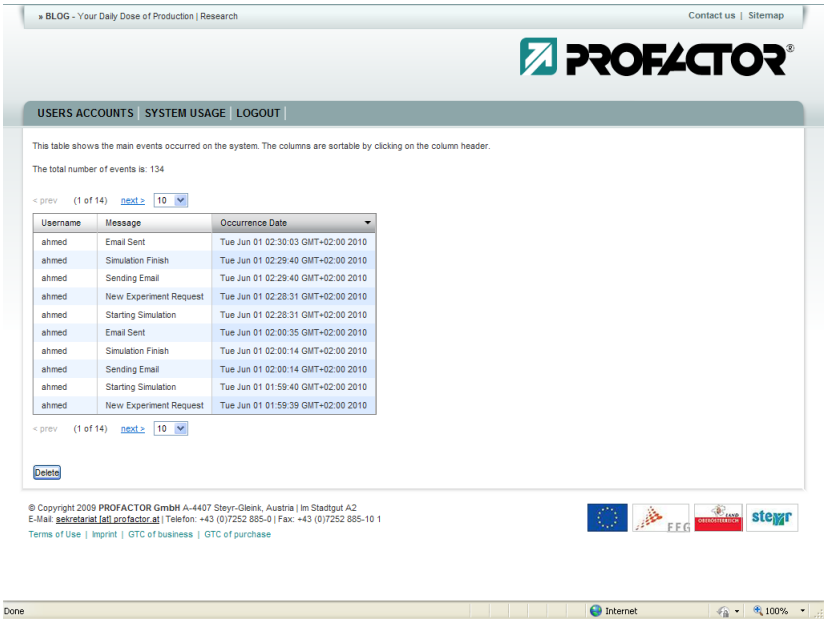

Figure 6.2: Administrator Interface - System Usage

For each account, the username, email address, last login date (if any), number of experiments performed and the best performance value achieved are displayed. These data reflect the usage of the system per user. Also, the administrator can select any row in the list (selecting user account) and click on the button "Show Details" at the bottom of the page. This retrieves further information about that user from the database (namely, his address and his institution). Finally, the administrator can select the user account and click on the button "Delete" to totally delete that account from the database. That user will not be able to use the system again unless he registers a new account.

Another page was created for the administrator view is the "System Usage" page (Figure 6.2). In this page, the administrator can keep track of the system usage for all users. As discussed in Section 4.2.2, important events are logged in the persistence layer. Each event is associated with the username responsible for it and its occurrence date and time. As in the "Users Accounts" page, this page contains a scrollable list that shows a user defined amount of events per page (for example the figure shows that there is 134 logged events and only the first ten are shown). The columns of this list can be sorted (ascendingly or descendingly) by the administrator. For example, the administrator can sort by username to get all the events associated with that user in one block, or sort by message itself to get all the events of same type in one block, finally he can sort by the occurrence date to get

the events in logical order of occurring (as shown in the figure). In addition to that, the administrator is able to select some events, he does not want to keep track of, and click on the button "Delete" to totally delete the selected events from the database.

6.2 Dynamic Configuration

During the development phase, it was clear that the "Simulink Scheme" (Section 4.2.1) may be modified from time to time according to needs. For example, addition of "Scope" units to monitor specific signals or modifications to properties of "rate transition" units to adjust the sampling rates for the model and the hardware. However, if the project is tightly coupled (heavily depends) to the initially provided scheme, it wouldn't be possible to do these modifications (actually not even simple renaming to already exist function or unit). Therefore it was decided to extract the dependency on MATLAB to an external configuration file that is parsed for each request on runtime. In the same way, if the hardware specifications or layout design was modified, the "Experiment Description" page will be out dated. The next sections discuss in details how these issues were resolved.

6.2.1 MATLAB Code

Extracting the dependency of MATLAB from the compiled Java classes, makes the project more tolerant to accept any modifications to the underlying system. Using the external configuration file, the administrator can write any MATLAB code that is then parsed by the application and send to MATLAB for execution. The application also buffers the output of MATLAB to add it to the email send to the user after execution. If for some command, the administrator does not want to send its output to the user, the command can be postfixed by a ";" that informs MATLAB that no output should be printed. This gives the administrator to control not only the execution code but also what to be sent to the user email (more details in section 6.2.1.2). In addition to that, the administrator can make use of MATLAB programming language and include extra validations to the user input and reject a simulation request if doesn't meet the new criteria. In this case, an email will be sent to the user containing the output of MATLAB code already executed till the point of rejecting the request. What can be achieved using this configuration file is unlimited and gives the administrator extreme flexibility to modify anything and the project will adapt to that modifications without any recompiling or redeploying the web interface. In

this project, a complete configuration file was written to be used with the current model. This file would only be used in case of modifications to that model.

The MATLAB code written in this configuration file should make use of the exact values entered by the user in the web interface. This is why the configuration file is parsed for each request. The configuration file defines a set of variables written in special format. During the parsing process, these variables are replaced by exact values from the user interface and the generated command (after replacing variables by values) is sent to MATLAB for execution.

Sample 8 shows how the format of the configuration file and how it can be used. The configuration file defines seven blocks of code. Each block is responsible for handling the user input for a specific part. For example, the interface defines three different methods to enter the disturbance signal (namely, "Harmonic", "White Noise" or "Custom"). Each method of these has its own set of variables (the Harmonic signal requires amplitude and frequency, the white noise requires only amplitude and the Custom signal requires path to user uploaded file). Thus, in the configuration file there are three blocks responsible for these three methods for the disturbance signal. At runtime, the user will choose only one method for the disturbance signal. Only the code block that corresponds to the chosen method is parsed and sent to MATLAB for execution. This abstracts the administrator from knowing which method was chosen by the user and lets him focus only on the MATLAB code for a block that is executed in case the user chose a certain block. In the same way the controller algorithm can be entered by two different methods (namely, "State-Space" or "Custom"). Accordingly, two more code blocks are responsible for these two methods. Another block that is always executed in the "Simulation" block, where all the commands responsible for compiling the model, writing it on the real-time processor and starting/stoping execution are defined. Finally, another block defines exactly configurations for the email attachments (more details in section 6.2.1.2).

The sample code shows the configuration block for the "Harmonic" method of entering the disturbance signal, the "Simulation" block and the email attachments configuration. In the first block "Harmonic", the administrator can use the specially formatted variables "$$HarmonicAmplitude$$" and "$$HarmonicFrequency$$" in the written code within this block. The first two lines of code shows the mapping between the these variables to variables "A" and "F" in the MATLAB workspace. So after executing these two lines, the MATLAB workspace will have definitions of the "A" and "F" variables that contains the values entered by the user for the parameters "Amplitude"

Sample 8 Sample Usage to Configuration File.

```
##############################################################
###
###
### Disturbance Signal Input - Harmonic
###
### Input: $$HarmonicAmplitude$$ - $$HarmonicFrequency$$
###
### Code:
A=$$HarmonicAmplitude$$
F=$$HarmonicFrequency$$
T=1/F;
open_system('demo.mdl')
##############################################################
###
###
### Experiment - Simulation
###
### Input: $$SimulationTime$$
###
### Code:
sim('demo.mdl',3)
close_system('demo.mdl',0)
##############################################################
###
###
### Email
###
### Code:
ScopeImage: Scope 200 300 scope.jpg
Attachment: scope.jpg
##############################################################
```

and "Frequency" respectively. Since, these two commands are not postfixed with ";", then MATLAB will echo the variable name and value back to the application which buffers all the output and sends to the user email. Thus, the user will receive the exact values he entered for his experiment. However, the third command is meant to be internal calculation that the user shouldn't know about (since it is postfixed with ";"). The last command in this block, open the Simulink Scheme named "demo.mdl". It is clear that the administrator can even replace the whole Simulink Scheme with a new version and just modify this command.

The next code block is the "Simulation" block. This block is always executed as it has no alternatives in the interface. It only defines one input "$$SimulationTime$$" which is the amount of time required for simulation. It is restricted by the interface to deny values greater than one minute. As previously explained, this variable is substituted by the real value entered by the user in the interface. However, the administrator may want to discard this value and have it as a constant (that is all experiments lasts for same amount of time). This is achieved by the first command in the block. The command explicitly define the duration to be three discarding the value entered by the user. The administrator may want to inform the user about that in a message that will be echoed back to the user email. The last command just forcedly closes the opened system.

6.2.1.1 Evaluation Function

After the simulation finishes, the system should evaluate the performance of the experiment. The performance here is defined as how much reduction in noise is achieved using this algorithm. The performance value is then calculated in the "Evaluation Function". As we discussed previously, if there is any modifications to the model, the current evaluation function would be outdated and may need refinement as well. Thus, it was decided to define this function externally as well (in MATLAB) and have the application just call it and get the performance value back. The application then compares the new performance value with the best performance value achieved by that user and all other users. This way, the evaluation function may be totally defined outside the application and can be modified and/or replaced at any time with no extra effort on the application side. Finally, the administrator can control whether to send the achieved performance value in the email sent to user or not.

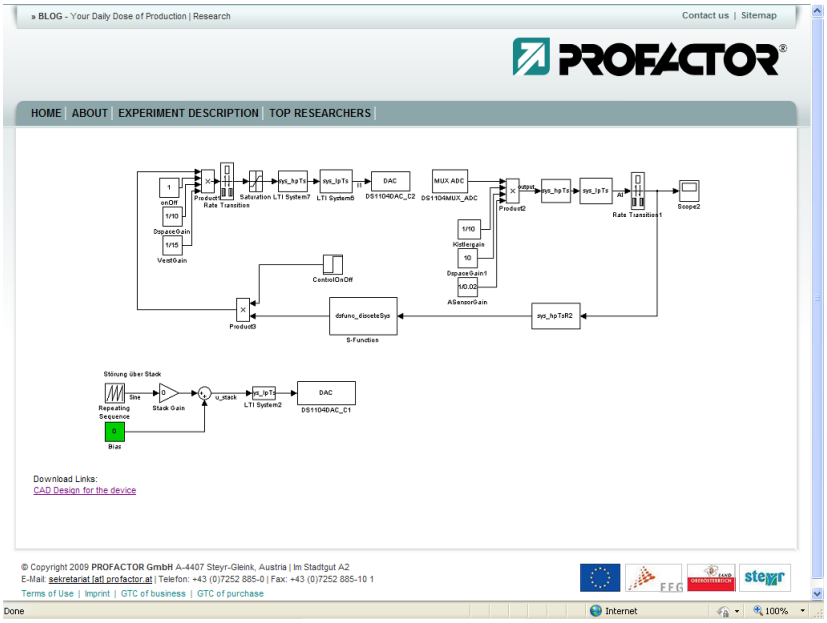

Figure 6.3: Experiment Description Page

6.2.1.2 Email Attachments

The last code block of the configuration file defines the email attachments. This is not similar to the previous code blocks. That is, it does not contain MATLAB code. However, it contains a more abstract commands used only to get images out of the "Scope" units in the Simulink model and attaches them to the user. The command syntax is very simply since it only starts with "ScopeImage:" to get an image or "Attachment:" to attach a file. The "ScopeImage:" command defines the name of the "Scope" unit to get the image from, the dimensions for the extracted image and the filename to be saved as. The application then parses this command and generates the exact commands to be executed by MATLAB. This abstracts the administrator from how to get images and writing repeated commands for each image and only focus on what images are required. The "Attachment:" command just attaches the specified file to the user email. It can be used for example to attach a file generated by MATLAB code written in one of the previous blocks or even attach a certain file to all users.

6.2.2 Experiment Description Page

The Experiment Description Page is intended to provide scientific description to the experiment. It should include the exact dimensions of the experiment device. Also, the positions of the disturbance and the actuator patch units and the vibration sensor. In addition to that, it may also include electrical specification for the used units. Moreover, a description of the simulating model can be also provided here.

As we discussed in the previous sections, if this page is statically programmed in the project, then it would be useless and/or misleading, if there were any modification to the device or the simulating model. Therefore, the administrator should be able to modify the contents of this page easily and safely with the minimum amount of effort. Figure 6.3 shows the content of this page. This page consists of three main parts, the first part is from the top of the page to the top of the model picture, the second part includes the model picture and download link and the third part is from the download link to the bottom of the page. The application defines an internal static page that contains the first and third parts. In-between these parts, it includes an "iframe" element. The "iframe" tag is a normal HTML tag that can be used to inline an page inside another. This is the main point here, since the "iframe" can embed a page in the already defined static page, we can have an external page (the second part) and link to it from the "iframe" element. In this way, the three parts are appended automatically and sent to user as if it is one page.

The administrator required that the content of this page would be images, text and downloadable links. Thus, a skeleton page was developed that has the same look and feel of the whole web interface and also contain these three elements. The administrator can modify this page (for example adding new picture by copy/paste the HTML element from the skeleton page) dynamically at runtime without any need to recompile or redeploy the whole web interface. Nevertheless, the external page is normal HTML web page, that is it is not restricted to those three elements (required by the administrator) and can contain any other HTML elements provided that their style (look and feel) matches the web interface (or explicitly coded in the page). A final warning here is that since the included web page is external to the web interface, therefore it is persisted with the application. That means that, if the web interface would be hosted on another server in the future, this page has to be transferred manually as well. To protect that page from not being accidentally deleted, it was decided to put it on the server directory (and not on the root directory for example "C:\").

Chapter 7

Typical Usage Scenario

In this chapter, we discuss a sample usage scenario for the system. For each step in this scenario, the corresponding screenshot of the web interface is shown. Basically, the scenario starts with a new user that browses through the "Public Web Pages" and ends with the user gets into the page "Top Researchers". The sections in this chapter differentiate between "Public Web Pages" and "Access Restricted Web Pages".

7.1 Public Web Pages

These are the pages that are open to everyone to see (i.e. they are not restricted to registered users). From the technical point of view, these pages do not expect the visitor to be directed to them from within the web interface. That means, the user can safely bookmark these pages in his own web browser to easily access them directly afterwards.

The starting page in this category is the "login" page (Figure 7.1). This page allows the visitor to enter his account username and password to be an authenticated user (and thus use the system). Before registering a new user account, the visitor may visit the also public web page "about" (Figure 7.2) to get a general idea about what it is about. This page mainly informs the visitor that the system respects his privacy and does not persist any data related to his experiments. Also, that page provides general background information for the problem on research and the role of the device used in the experiment. Finally, it shows a simple "How-To" description of how to experiment with the system over the web interface.

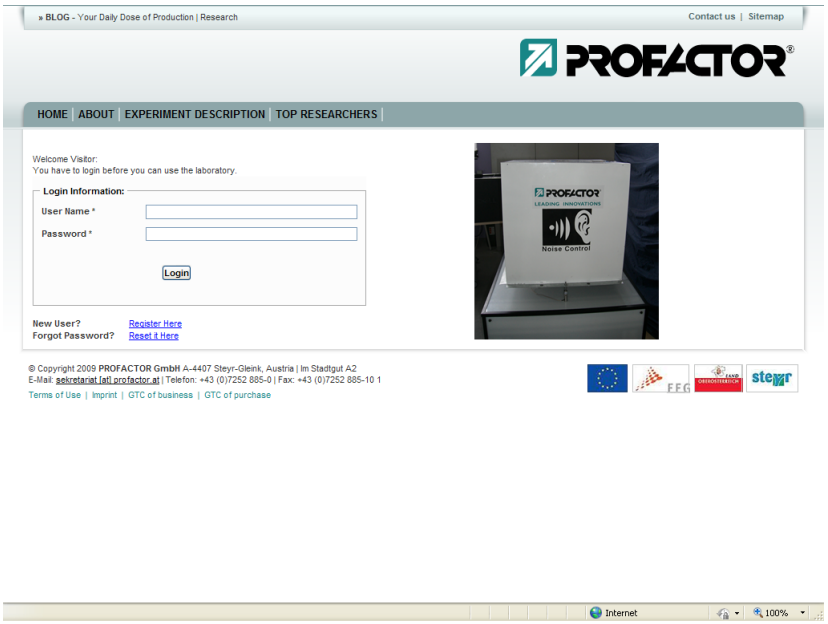

Figure 7.1: Login Page

Figure 7.2: About Page

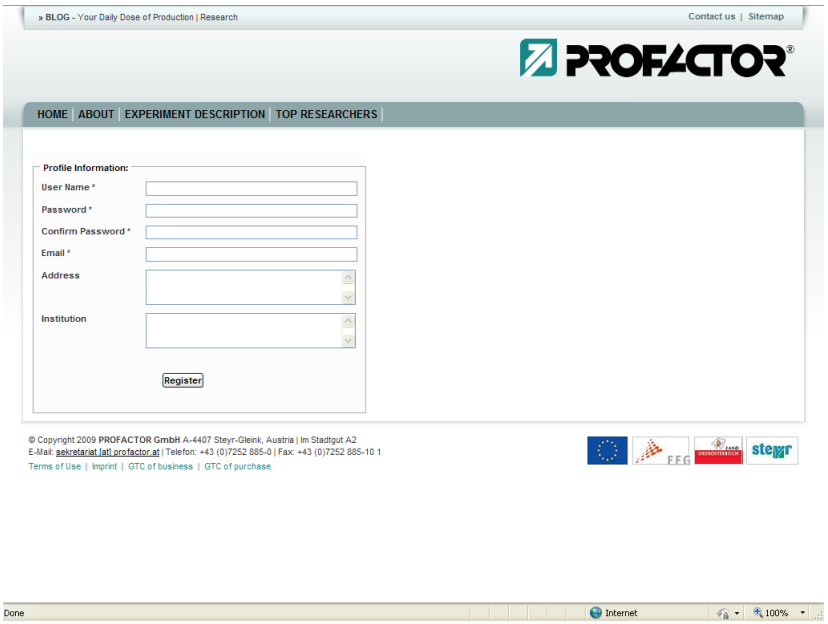

Figure 7.3: Registration Page

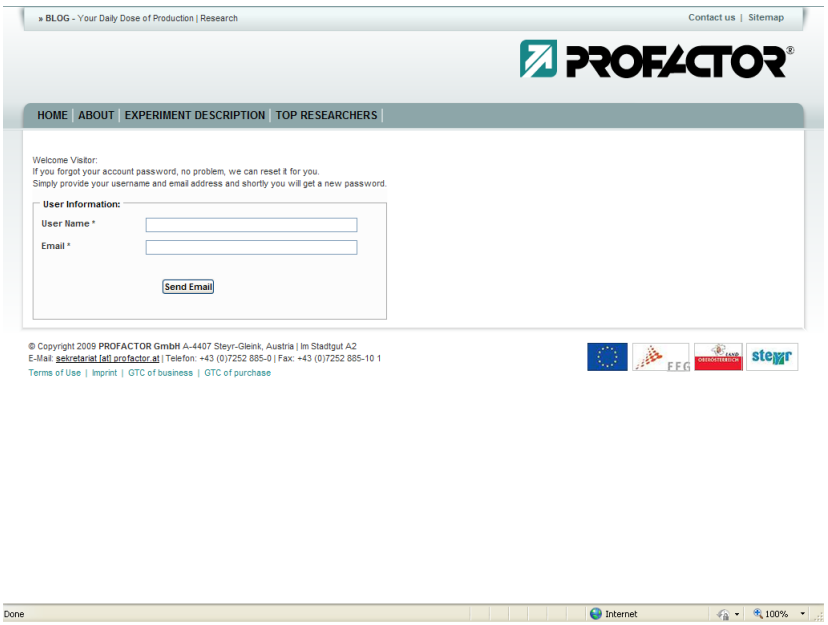

Figure 7.4: Reset Password Page

If the visitor likes the system and wants to register for a free-of-charge account, he is able to do that through the link "Register Here" below the "Login Information" box. This link redirects the visitor to the "register" page (Figure 7.3). The registration page allows the visitor to enter his username and password along with his email and some further optional data (namely, address and institution). Upon completing this form and clicking the "Register" button, the visitor has successfully registered for a new user account. In case the visitor chooses a username that is already registered, he gets an error message. If however, that visitor remembers his username but forgot his password, he can reset his password through the link "Reset it Here" below the "Login Information" box in the "login" page. This link redirects the visitor to the "register" page (Figure 7.4).

The reset password page only contains two fields to be entered by an (already registered) user, the username and his email address. If these two fields matches a user account already registered in the database, the application will generate a random password and assign it to that user account. Also the randomly generated password is sent to that user email. The user may use his username along with the new generated password to authenticate himself to the system. It is recommended that the user changes this password manually to a more memorizable password as the randomly generated password is alphanumeric string (i.e. it is difficult to memorize).

7.2 Access Restricted Web Pages

Upon providing valid account information, the visitor is authorized and is allowed to navigate to the second type of pages "Access Restricted Web Pages". The first page in this type is the "experiment" page, which is the most important page in this project. That is because it is the page where the user can input his experiment parameters and submit the simulation request to the server.

The experiment page is shown in Figures 7.5 and 7.6. This page provides the user with different options to input his experiment configuration. Basically, the user needs a disturbance signal, a controller algorithm and a simulation time. For the disturbance signal part, he can either choose a pre-defined signal and configure its parameters (namely, the frequency and amplitude in case of harmonic signal and amplitude in case of white noise signal) or define his custom signal and upload it to the server. Figure 7.5 shows the user using the harmonic disturbance signal while Figure 7.6 shows the usage of a custom user-defined signal.

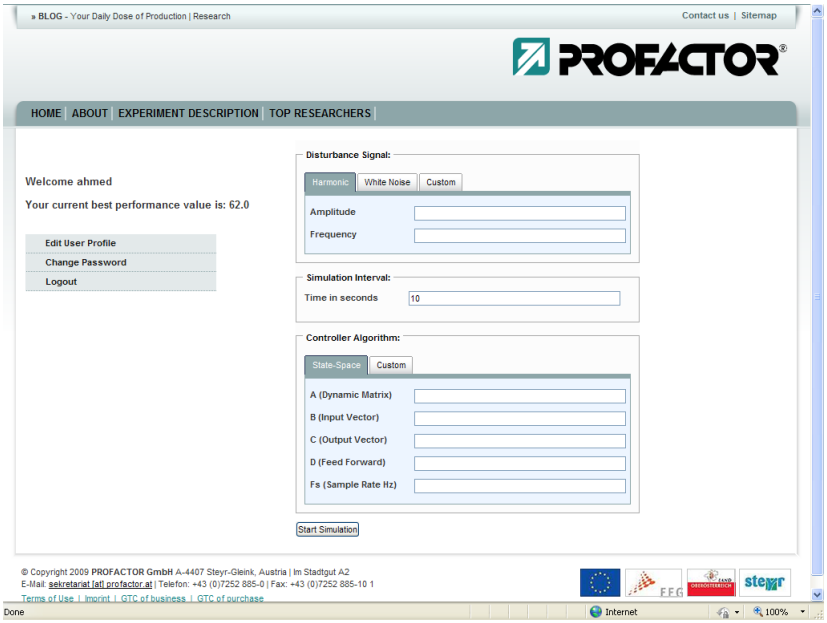

Figure 7.5: Experiment Page

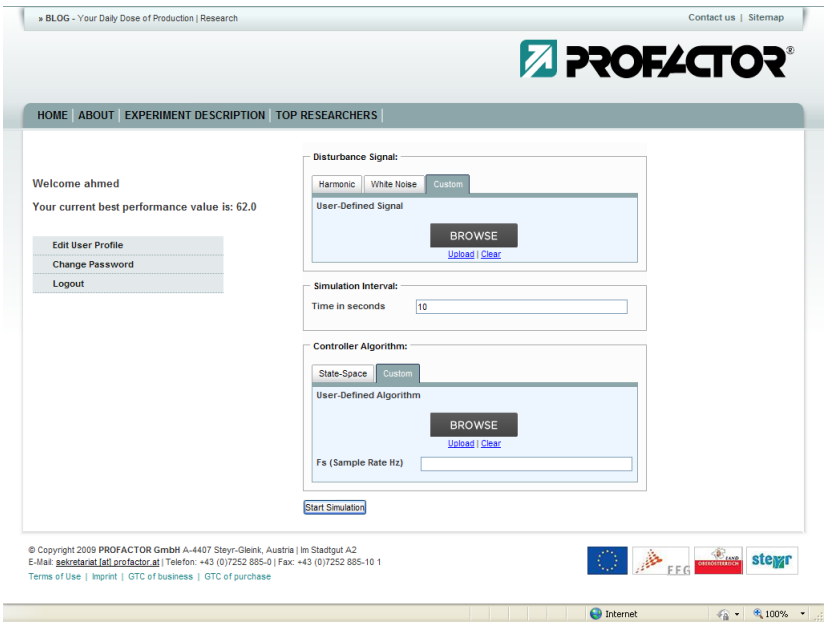

Figure 7.6: Experiment Page

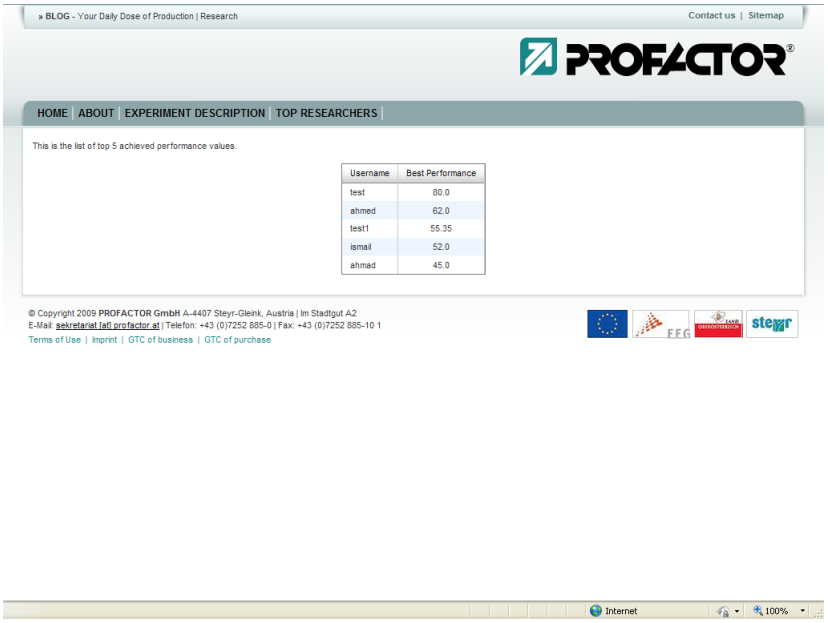

Figure 7.7: Top Researchers Page

The second box allows the user to enter the amount of time required to run the simulation. This by default has the value of ten seconds. However, the user can change it to any number between one second and one minute.

The third box defines the controller algorithm for the experiment. Also, the user can choose here between a pre-defined algorithm and only configure its parameters or upload his own algorithm. The pre-defined algorithm here is the "State-Space" equations. It is a well-known algorithm in the control theory. It has four parameters to be configured (namely, "A" Dynamic Matrix, "B" Input Vector, "C" Output Vector and "D" Feed Forward). The values provided by the user are constrained by these rules, "A" has to be a square matrix, "B" has to be a column vector and "C" has to be a row vector. In addition to these variables, the user has to enter a "Sample Rate" value for his simulation request. This value has to be provided whatever the method chosen by the user to enter his controller algorithm. That is, if the user chooses to upload his own algorithm, he has to provide the "Sample Rate" as well. Figure 7.5 shows the user using the "State-Space" equations while Figure 7.6 shows the usage of custom user-defined algorithm.

Upon finishing the input of all parameters and passing the values validation, the user can click the "" button and have his request sent to server to be processed. On the server side, the requests are enqueued one after another and instantly a reply message is sent to user browser. The system

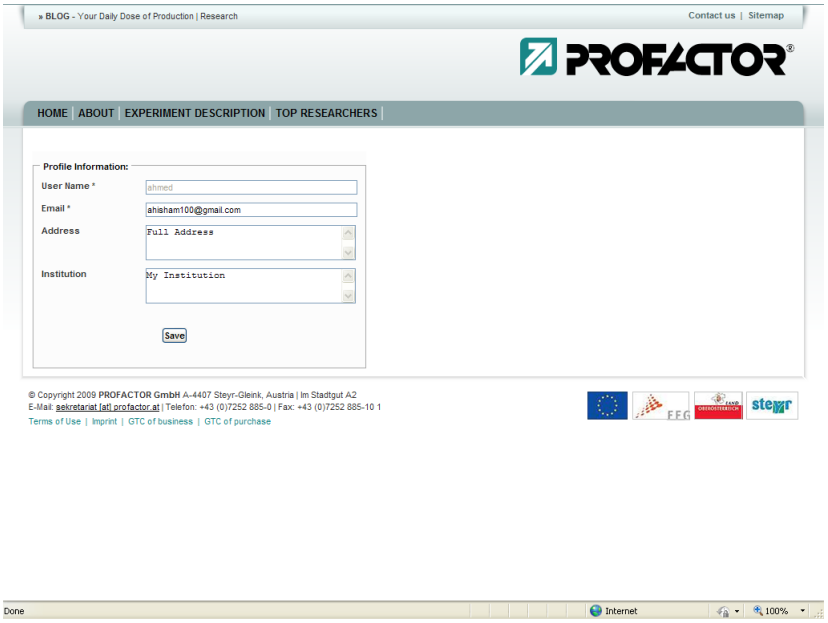

Figure 7.8: Profile Page

then process each message in queue in order of receiving them. After the simulation has finished, a report is generated and sent to the user email. Also, a performance value is calculated for each experiment. This is done for each user so that they know the best achieved value so far. The page "Top Researchers" (Figure 7.7) shows the best five researchers according to their best achieved performance values.

Finally, the user can show his profile data in the "edit profile" page (Figure 7.8). This page shows the profile of the already logged in user. The profile includes the username (which can be updated after registration), the email address, the user address and the institution. These fields can be updated for newer information and be saved in the database through this page. Also the user can modify his account password using the "change password" page (Figure 7.9). This page asks the user to enter his current password (to ensure that he is indeed the user and that his session was not hijacked) along with the new password (which has to be confirmed by writing it twice). By clicking the "Change Password" button, the new password will be encrypted and replaces the old password in the database.

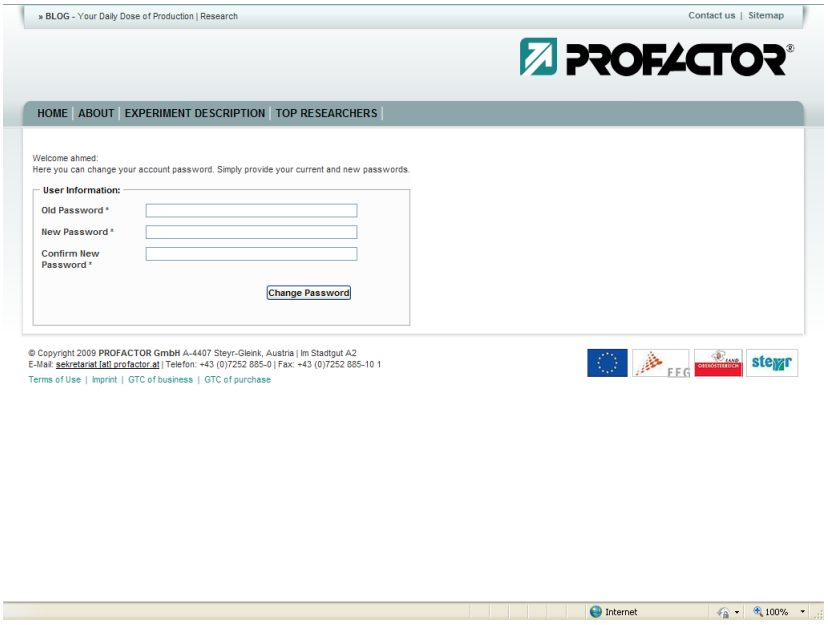

Figure 7.9: Change Password Page

Chapter 8

Conclusion

In this thesis, a virtual laboratory has been designed and implemented that enables world wide researchers to use a noise control experimental device. The project started from the MATLAB/Simulink controller layer and builds up till the user interface layer. The most vital components of this project are the connection between the web server and the MATLAB/Simulink controller and the synchronization mechanism used to queue the requests and perform them one-by-one in the order of arrival. The design principles of these components were discussed in the previous chapters (especially in Chapter 4).

The project was capable of achieving the primal objective set to it, namely, the exposure of the real device capabilities to world wide researchers. Also, the requirement of users privacy was achieved (by the different methods discussed in previous chapters, especially Chapters 4 and 5). The basic security level was applied to the web interface to disable the common software vulnerabilities. However, it was planned to have the feature of encrypted communication between the researcher and the server so that even if a hacker gets access to the communication, he receives it encrypted and of no valuable use. We decided to postpone this feature until a SSL signed security certificate is acquired. However, the web server installed is capable of providing the encrypted connection, once the certificate is given. Thus the enabling of this feature is just achieved by a modification of the configuration file.

The nature of the real device denies it from fulfilling multiple simultaneous requests. That is the reason of implementing a synchronization mechanism. While this is clearly a bottleneck in the whole system, it does not represent a problem now, since no heavy traffic is expected. Despite that, the architecture allows extending the underlying layer of real devices by incorporating replicates of the real device. With a simple modification in the

software code, it would be possible to make use of the newly added devices and thus fulfilling multiple simultaneous requests. This also increases the availability of the whole system (if one system stops working, others can still fulfill requests till it gets working again). It is recommended as future work for this project to extend the capabilities of the underlying real devices.

As it was shown in Chapter 6, the project enables the system administrator to totally modify its behavior (in particular, the way it deals with user input and commands sent to the real device) at runtime without recompiling or redeploying the whole web interface. A very interesting point of enhancement here is to also be able to modify the user interface. Modifying the user interface at runtime would be difficult (though possible at first glance), but it might be enough to have a tool that automatically generates the web interface from an easy to write configuration file. By achieving this, one would have a dynamically customizable virtual laboratory that is not specific to any real device and thus portable to any real laboratory.

Bibliography

[1] Adobe. open source framework — Adobe Flex. http://www.adobe.com/products/flex/. [Online; accessed May-2010].

[2] Apache. The Apache HTTP Server Project. http://httpd.apache.org/. [Online; accessed May-2010].

[3] Antonio Bicchi, Alessandro Coppelli, Francesco Quarto, Luigi Rizzo, and Aldo Balestrino. Breaking the Lab's Walls: Tele-Laboratories at the University of Pisa. In *ICRA - IEEE International Conference on Robotics and Automation*, pages 1903–1908, Seoul, Korea, May 2001.

[4] Claudiu Chiculita and Laurentiu Frangu. A Web Based Remote Control Laboratory. In *6th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida, July 14-18 2002. IIIS.

[5] Django. The Web framework for perfectionists with deadlines. http://www.djangoproject.com/. [Online; accessed May-2010].

[6] dSPACE. dSPACE - Connector and LED Panels. http://www.dspaceinc.com/ww/en/inc/home/products/hw/singbord/conledpanels.cfm. [Online; accessed April-2010].

[7] Network Working Group. Internet Message Access Protocol - Version 4rev1. http://www.ietf.org/rfc/rfc3501.txt. [Online; accessed May-2010].

[8] Network Working Group. Post Office Protocol - Version 3. http://www.ietf.org/rfc/rfc1939.txt. [Online; accessed May-2010].

[9] Network Working Group. Simple Mail Transfer Protocol. http://www.ietf.org/rfc/rfc2821.txt. [Online; accessed May-2010].

[10] Numerical Algorithms Group. Fortran 2003. ftp://ftp.nag.co.uk/sc22wg5/N1601-N1650/N1601.pdf.gz. [Online; accessed May-2010].

[11] NATIONAL INSTRUMENTS. NI LabVIEW - The Software That Powers Virtual Instrumentation - National Instruments. http://www.ni.com/labview/. [Online; accessed March-2010].

[12] MathWorks. MATLAB - The Language Of Technical Computing. http://www.mathworks.com/products/matlab/. [Online; accessed March-2010].

[13] MathWorks. Simulink - Simulation and Model-Based Design. http://www.mathworks.com/products/simulink/. [Online; accessed March-2010].

[14] MathWorks. *MATLAB 7 External Interfaces*, chapter 6. September 2009. http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/apiext.pdf.

[15] Microsoft. Common Language Runtime. http://msdn.microsoft.com/en-us/library/ddk909ch.aspx. [Online; accessed May-2010].

[16] Microsoft. Microsoft Corporation. http://www.microsoft.com/. [Online; accessed April-2010].

[17] Microsoft. Microsoft NetMeeting. http://www.microsoft.com/downloads/details.aspx?FamilyID=26c9da7c-f778-4422-a6f4-efb8abba021e&displaylang=en. [Online; accessed May-2010].

[18] Stefan Mller. JMatLink. http://jmatlink.sourceforge.net/index.php. [Online; accessed May-2010].

[19] National Institute of Standards and Technology. SECURE HASH STANDARD. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[20] ORACLE. Applets. http://java.sun.com/applets/. [Online; accessed May-2010].

[21] ORACLE. Bean Validation. http://java.sun.com/javaee/6/docs/tutorial/doc/gircz.html. [Online; accessed May-2010].

[22] ORACLE. GlassFish, Open Source Application Server. https://glassfish.dev.java.net/. [Online; accessed May-2010].

[23] ORACLE. Java BluePrints - J2EE Patterns. http://java.sun.com/blueprints/patterns/MVC-detailed.html. [Online; accessed May-2010].

[24] ORACLE. Java EE at a Glance. http://java.sun.com/javaee/. [Online; accessed May-2010].

[25] ORACLE. Java Message Service (JMS). http://java.sun.com/products/jms/. [Online; accessed May-2010].

[26] ORACLE. Java Native Interface. http://java.sun.com/javase/6/docs/technotes/guides/jni/index.html. [Online; accessed May-2010].

[27] ORACLE. Java Persistence API. http://java.sun.com/javaee/technologies/persistence.jsp. [Online; accessed May-2010].

[28] ORACLE. Java SE Overview. http://java.sun.com/javase/. [Online; accessed May-2010].

[29] ORACLE. Java SE Technologies - Database. http://java.sun.com/javase/technologies/database/. [Online; accessed March-2010].

[30] ORACLE. JavaMail API. http://java.sun.com/products/javamail/. [Online; accessed May-2010].

[31] ORACLE. JavaServer Facelets. https://facelets.dev.java.net/. [Online; accessed May-2010].

[32] ORACLE. JavaServer Faces Technology. http://java.sun.com/javaee/javaserverfaces/. [Online; accessed May-2010].

[33] ORACLE. Managing Entities. http://java.sun.com/javaee/6/docs/tutorial/doc/bnbqw.html. [Online; accessed May-2010].

[34] ORACLE. MySQL :: The world's most popular open source database. http://www.mysql.com/. [Online; accessed May-2010].

[35] ORACLE. The Java Community Process(SM) Program. http://www.jcp.org. [Online; accessed April-2010].

[36] ORACLE. The Java Media Framework API (JMF). http://java.sun.com/javase/technologies/desktop/media/jmf/. [Online; accessed May-2010].

[37] ORACLE. The Java Persistence Query Language. http://java.sun.com/javaee/6/docs/tutorial/doc/bnbtg.html. [Online; accessed May-2010].

[38] ORACLE. Unified Expression Language. http://java.sun.com/products/jsp/reference/techart/unifiedEL.html. [Online; accessed May-2010].

[39] ORACLE. What Is a Message-Driven Bean? http://java.sun.com/javaee/6/docs/tutorial/doc/gipko.html. [Online; accessed May-2010].

[40] ORACLE. What Is a Session Bean? http://java.sun.com/javaee/6/docs/tutorial/doc/gipjg.html. [Online; accessed May-2010].

[41] PHP. PHP: Hypertext Preprocessor. http://php.net/. [Online; accessed May-2010].

[42] PROFACTOR. PROFACTOR - Leading Innovations - Home. http://www.profactor.at. [Online; accessed March-2010].

[43] Rafael Puerto, Luis M. Jimenez, Oscar Reinoso, Cesar Fernandez, and Ramon Neco. Remote Control Laboratory using MATLAB and SIMULINK: Application to a DC Motor Model. *Computer Applications in Engineering Education*, 2009.

[44] Python. Python Programming Language – Official Website. http://www.python.org/. [Online; accessed May-2010].

[45] QUANSER. Quanser. http://www.quanser.com/. [Online; accessed March-2010].

[46] Christof Rohrig and Andreas Jochheim. Java-based Framework for Remote Access to Laboratory Experiments. In *IFAC/IEEE Symposium on Advances in Control Education*, Gold Coast, Australia, December 17-19 2000. Elsevier Science Ltd.

[47] RoR. Ruby on Rails. http://rubyonrails.org/. [Online; accessed May-2010].

[48] The Internet Society. Hypertext Transfer Protocol – HTTP/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616.html. [Online; accessed May-2010].

[49] The Internet Society. The Common Gateway Interface (CGI) Version 1.1. http://www.ietf.org/rfc/rfc3875.txt. [Online; accessed May-2010].

[50] TCPIPGuide. The TCP/IP Guide. http://www.tcpipguide.com/free/. [Online; accessed May-2010].

[51] HUGHES TECHNOLOGIES. Hughes Technologies : The home of Mini SQL (mSQL). http://www.hughes.com.au/. [Online; accessed March-2010].

[52] Prime Technology. PrimeFaces. http://www.primefaces.org/. [Online; accessed May-2010].

[53] W3C. SOAP. http://www.w3.org/TR/soap12-part1/. [Online; accessed May-2010].

[54] W3C. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). http://www.w3.org/TR/xhtml1/. [Online; accessed May-2010].

[55] W3C. XMLHttpRequest. http://www.w3.org/TR/XMLHttpRequest/. [Online; accessed May-2010].

Ahmad Hisham Ismail

(Last Updated 20/6/2010)

Personal Data

Place and Date of Birth:	Cairo, Egypt — 14 June 1986
Address:	18 Abou Egila st, 1st-district Nasr City, Cairo, 11371
Phone:	012 7364577
Email:	ahmadmhisham@gmail.com

Education

2008-2010

Masters Education

I prepared for the Master of Science from the last year of University Education. I applied for the **International Studies in Informatics** program at the **Johannes Kepler Universitat Linz** in Austria.

Masters Project

My Master thesis is titled **Remotely Controlling a Mechanical Laboratory via the Internet**. It is about designing and implementing a web interface for a mechanical laboratory that enables remote users from controlling it. The technology used is **JavaEE 6**.

2004-2009

University Education

I achieved my **Bachelor of Science** in major **Computer Science and Engineering** at the **German University in Cairo**. The final cumulative GPA is **1.34**.

Graduation Project

My Bachelor Thesis is titled **Design and Implementation of Computer Controlled 6-DOF Articulated Robotic Arm**. The technology used is **Java**. I got the excellent grade **A+**. It was published as the system of the month in the "University of Waterloo" robotics newsletter (`http://horizon.uwaterloo.ca/ras/system_of_month.htm`).

1991-2004

School Education

I completed my school education in the **Talaa' El-Kamal Islamic School**. I got the **"Thanaweya Amma"** certificate with the grade of **91.7%**.

Experience

2009

2 Months Freelancing for Raya

I was enrolled as a freelancer in an **E-Banking** system. My role was mainly enhancing the user interface and general testing for the whole system.

RoboCup - Festo Hockey Challenge Cup

This is a demonstration of capabilities of Festo robot called "Robotino". In this cup, all teams are playing Hockey with these robots. The competence here is in the software level (programming the robots to play as a team and score goals). I was assisting the GUC team in the implementation of computer vision algorithms and system (Linux) level tasks. The GUC team got the 2nd place in this cup. Note, I have no official certificate!

2006-2009
University

3 Months Junior Teaching Assistant in GUC

I was teaching **Computer Programming Lab** course. This course is about teaching the Object-Oriented features. This is done through the development of a game application (Engine, Interface and AI). My responsibility was to teach the concepts in class and evaluate the games developed by students.

3 Months Raya Internship

I had an internship at Raya Software Department. I was enrolled in an **E-Government** project. The project was in its final phase "Fixing Bugs". The project was based on the "Struts" framework.

3 Months Junior Teaching Assistant in GUC

I was teaching **Introduction to Computer Programming** course. This course is about teaching the basics of computer programming and software writing skills in Java Programming Language. My responsibility was to teach Java syntax (through algorithms) in the class and to correct the quizzes and exams.

1 Month Raya Internship

This internship was about general **computer maintenance**. I was in the customer care center that receives the malfunction computers and repairs them.

Skills

Software Developing

Programming Languages

Experienced in **Java** Programming Language (more in **Standard Edition** and **Enterprise Edition** and less in **Micro Edition**). Strong knowledge about **C/C++** Languages. Basic Knowledge about **C#** and **Visual Basic** Languages. For other paradigms, familiar with **Prolog** (including **Constraint Programming**) and **Haskell**. For system level, familiar with **Assembly** (especially x86 and MIPS architectures).

Scripting Languages

Basic Knowledge about **Python** and **R** (Machine Learning).

Architecture

Familiar with standard notations of **UML** design diagrams. Capable of applying different **design patterns**.

Web Technologies

Experienced in different Java-based Web application frameworks (**Struts** and **JSF**). Strong knowledge about basic **JSP/Servlets**. Capable of dealing with different application servers (**Glassfish, JBoss and Tomcat**). Strong knowledge in different internet protocols. Beside Java, basic knowledge about **PHP** and **.NET** frameworks. For client side, familiar with **Javascript** language, **CSS** styles, browsers compatibility (**xhtml**) and **AJAX** based requests.

Databases

Experienced with Relational Databases like **MySQL** and basic knowledge about **Oracle**. Familiar with querying languages like **SQL** and **JPQL**. Strong knowledge about **ORM** concepts (**Hibernate**). For small scaled applications, familiar with **XML** format as data storage (using **DTD and XML Schema**) and processing (using **XPath and XSLT**).

Graphics

Experienced in **OpenGL** and **OpenCV**. Familiar with computer vision problems and algorithms.

Platforms

Operating Systems

Experienced in working with **Microsoft Windows XP**. Strong knowledge with using **Linux** distributions (**Ubuntu and Suse**) and less knowledge with (**Gentoo and ArchLinux**). Basic knowledge about operating systems concepts (Booting, Schedular and Device Drivers).

Applications

Experienced in working with **Microsoft Office suite** and LaTeX. For software development, experienced in **Eclipse**, **Netbeans** and **Visual Studio**. For web development, capable of using **XAMPP** (or install individual servers) along with **phpmyadmin**. Basic knowledge with **Matlab/Simulink**.

Hardware	Digital Systems Strong Knowledge in designing **embedded systems** and **controlling devices** (especially using **PIC** micro-controller). Basic Knowledge about **computer architecture** and designing digital logic systems (simple **processors/ALUs**).

University Course's Projects

Ordered descendingly according to project scale.

Movies DB	Java Web Interface. Designed according to MVC approach. The technologies used are Servlets and JSPs. The Movies data itself was stored in XML files (because of a requirement only!). Users can add, edit or search movies or actors profiles.
E-Learning	Java graphical application. Management tool for ELearning Software. Users are assigned roles with certain privileges. After the login, every user is presented by his role panel and can perform certain tasks through the application. The database used is MySQL.
Billiards	C++ OpenGL 3D-game. With the wide set of the used keyboard keys, the user is allowed to move freely in the room and aims at the white ball. It is 2-player game. The top-view is always shown aside.
Triominoes	Java graphical game. It is similar to common dominoes game but with triangular tiles instead of rectangular. This game allow 6-participants (at most) to play together, each one can be human or robot. AI for robots is also implemented.
Voice Calculator	Java Sphinx4 application. This is a simple calculator that can perform basic operations of 2 numbers. The main challenge is to allow the user to say the numbers and the operators while the calculator will recognize this expression, evaluates it and return the result to the user.
Sudoku Solver	Java Prolog (Constraint Programming) graphical application. It displays an empty board (4x4 or 9x9). Then, the user can enter some values and finally submit the board. The core solver will results in all Sudoku's boards that satisfy the submitted values.
Football Teams	Prolog console application. Given a database of players, it can arrange their seating order and assign them lockers according to set of requirements.
Right Edge Follower	Java Fuzzy console application. It is a simulation for a robot that has to follow an edge on his right. The challenge was to implement a Fuzzy controller to accomplish this task given the readings of sensors.
Chatting	Java graphical application. It is a chatting room that establishes 2 concurrent threads (at each user) to send and receive data messages simultaneously. There is also a list of the currently online users.

Languages

Arabic:	Mothertongue
English:	Fluent
German:	Basic Knowledge

References furnished upon request.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, das ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Hagenberg, am 5. Juli 2010

Ahmad Ismail