

**Web-based Online Tools for A Shake Table
Laboratory**

by

Hoang-Phong Nguyen

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of
Master of Science in Civil and Environmental Engineering
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2002

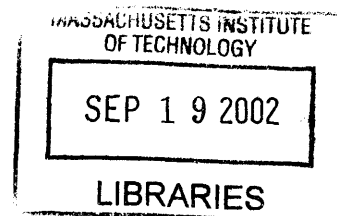
© Massachusetts Institute of Technology 2002. All rights reserved.

Author
Department of Civil and Environmental Engineering
August 16, 2002

Certified by.....
Kevin Amaratunga
Associate Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by ..
Oral Buyukozturk
Chairman, Departmental Committee on Graduate Studies

BARKER



Web-based Online Tools for A Shake Table Laboratory

by

Hoang-Phong Nguyen

Submitted to the Department of Civil and Environmental Engineering
on August 16, 2002, in partial fulfillment of the
requirements for the degree of
Master of Science in Civil and Environmental Engineering

Abstract

The Shake Table Lab at MIT is an educational project developed in the MIT I-Campus framework. This framework includes several virtual laboratories and its main goal is to provide remote access to the corresponding laboratory facilities. In particular, the Shake Table Lab involves a table station, which includes a hardware apparatus equipped with accelerometers to record responses to earthquakes and a proprietary software system to control the table. Thus, this project can illustrate the responses of a structure to different customizable inputs through the Web. The main objective of the Shake Table Lab is to provide a Web interface to remotely control the table and to run earthquake experiments. This thesis deals with the IT support of the project. First, we describe how we design a server-client architecture to provide users with recorded responses of the table to different earthquakes and with simulations of a model closed to the table. Then, we extend this architecture to enable users to run an earthquake with the table. The user interface is an Applet communicating, via RMI¹, with a Java application running under the MATLAB² interface controlling the proprietary software. We compare our software approach to different software approaches that can be applied to our particular case. Finally, we also provide a security and management framework to use the Shake Table and to extend its applications in the future.

Thesis Supervisor: Kevin Amaratunga

Title: Associate Professor of Civil and Environmental Engineering

¹Remote Method Invocation is a registered trademark of Sun Microsystem, Inc.

²MATLAB is a registered trademark of MathWorks, Inc.

Acknowledgments

In this section, I'd like to thank all the people who were involved in this project:

First, I would like to thank my advisor, Prof. Amaratunga for accepting to monitor this thesis and for all his guidance during this project.

I am much grateful to Prof. Kausel, the head of the Shake Table Project at MIT, who let me join this project.

I thank Irfan Baig, graduate student in charge of the dynamical part of the project, with whom I worked in a team during this project.

I would like to express some special thanks to Raghunathan Sudarshan for helping the team with his great technical and software knowledge.

I also want to thank Andreas Klimke for his software expertise and contribution in particular with Matlab and Java.

Finally, I would like to thank my family for encouraging me during my graduate study.

Contents

1	Introduction	13
1.1	The Description of the project	13
1.2	The Motivations and Objectives	14
1.3	Some Relations to other I-lab projects	15
1.4	The Roadmap of the thesis	16
2	Simulations	19
2.1	The Issues	19
2.2	The Recorded Data Response	20
2.2.1	The Presentation and Physical Approach	20
2.2.2	The Software Approach	21
2.3	The Simulation Software	24
2.3.1	The Presentation and Physical Approach	25
2.3.2	Software Approach	26
3	The Real Time Experiments	31
3.1	The Issues	31
3.2	The Hardware Architecture	31
3.2.1	The Table	31
3.2.2	The Accelerometers and the Active Mass Damper	32
3.2.3	The Power Module	33
3.2.4	The Data Acquisition Board	33
3.3	The Software Architecture	33

3.3.1	WinCon	33
3.3.2	MATLAB-SIMULINK	36
3.3.3	Integration of Java with SIMULINK models	37
3.4	The Real-Time Programs	39
3.4.1	Approach 1: MATLAB Invocation and Communication via the Java Runtime Class	39
3.4.2	Approach 2: Using a JNI Wrapper for MATLAB's C Engine	44
3.4.3	Approach 3: Using a Server-Client architecture and RMI	50
4	The Security and Management of the System	57
4.1	The Issues	57
4.2	The Security	57
4.2.1	The Software and the Servers	57
4.2.2	The Hardware	60
4.3	The Management of the System	63
5	The Conclusion and the Future Work	65
5.1	The Issues	65
5.2	The Software Improvements	65
5.2.1	Upgrading the framework	65
5.2.2	Other possible options	66
5.3	Hardware Improvements	67
	Bibliography	69

List of Figures

1-1	The shake table.	14
2-1	A movie of the shake Table in action.	20
2-2	The Global Client/Server architecture.	22
2-3	The UML architecture.	23
2-4	A simple XML structure.	24
2-5	The UI of the simulation.	29
3-1	The table moved by brushless servo motor.	32
3-2	An accelerometer mounted on the second floor.	32
3-3	The active mass damper.	33
3-4	The power module: the most important part for security.	34
3-5	The data acquisition board: the device to control the experiments. . .	35
3-6	A two PCs architecture showing the role of WinCon.	36
3-7	A UML diagram for the RMI architecture.	56
4-1	The different security steps to get Access to the Java UI.	59
4-2	The modular board used to switch on and off the power module. . . .	60

List of Tables

4.1 Pin values for the modular board	61
--	----

Chapter 1

Introduction

1.1 The Description of the project

This thesis is based on the I-Campus MIT Shake Table project. This Shake Table project is part of the University Consortium on Instruction Shake Tables (UCIST)[1]. UCIST is a university consortium formed in 1998, its goal is to enhance undergraduate education in earthquake engineering through the procurement of instructional shake tables.

Each university member of UCIST has a shake table station, which is equipment that can simulate mechanical wave propagations such as scaled earthquakes using a commercial software and a two-floor building apparatus. Using defined hardware-software package, UCIST members can study different topics such as non-structural seismic hazards in the home (University of Nevada), liquefaction analysis (Southern Illinois University), determining the natural frequencies and mode shapes of multi-degree freedom structures (Florida A&M), and experimental identification of dynamic properties of scaled frames (Penn State University).

Fig 1.1 shows an example of the shake table apparatus:



Figure 1-1: The shake table.

1.2 The Motivations and Objectives

As defined by Prof. Kausel, Head of the Shake Table Lab, the goal of the Shake Table project is to focus on developing a Web-based laboratory for studying earthquakes and vibration responses of buildings. The centerpiece is a sophisticated, commercially built laboratory apparatus, which can be used to excite, monitor and control model structures. The proposed effort is to extend the reach of this state-of-the-art research facility so that it is readily accessible to students in and beyond the classroom. The equipment is observable over the web. Students can then measure the response of the model and compare it with numerical predictions obtained using MOTION LAB, a

PC-based software system, some other Java simulations and the observed behavior of real life structures during actual earthquakes.

Basically, this project includes two parts:

- The structure part, which uses the shake table to study some particular physical problems. Among other subjects, this part involves earthquake study, mass damper influence analysis, system transfer function, inputs data/output data model analysis. More information about this part can be found in Irfan Baig's Master thesis[2].
- The IT part, whose goal is to support the multiple applications of the structure part through an extensible framework. It includes a software job and a hardware job. The software job is mostly high level and involves a web interface through server programming and through programs to link this interface to the hardware. The hardware part of the project is limited compared to the software part and future improvements on the hardware will be possible.

1.3 Some Relations to other I-lab projects

The Shake Table Lab (<http://flagpole.mit.edu/shaketable>) is one of several I-Labs under the I-Campus framework at MIT. Some of these laboratories include the Flagpole remote Web Lab (<http://flagpole.mit.edu/>), the Microelectronics Web Lab (<http://weblab.mit.edu>), the Photovoltaic Weather Station (<http://pvbase.mit.edu/-index.html>), and the Web Accessible Heat Exchanger Laboratory (<http://heatex.mit.edu>). The main goal of the Shake Table project is to provide students with an interactive tool that will enable them to learn from earthquake engineering at different levels through different applications. The challenges and difficulties can be found in two parts: software and hardware. In terms of software, the shake table can be controlled by a proprietary software system, which limits the applications and the degree of customization of the system because of limited available functions. The software challenge is to develop a customized system, which can be easily deployed on the

Web. The hardware challenge is to deal with a real time hardware system involving sensors with the same limitation about the degree of customization and some limited possibilities since the hardware was provided with the software as a package.

1.4 The Roadmap of the thesis

To simplify the reading, here is the road map of the thesis, which includes independent chapters:

- In Chapter 1, we describe the Shake Table project and we detail the objectives knowing the constraints coming from the existing software and hardware. To reach these goals, we have defined two degrees of interactivity described in Chapter 2 and in Chapter 3.
- In Chapter 2, we describe the first degree with a set of educational tools, which were developed as Java Applets. These tools work with archived data that are under text or XML format(without running the table) and have two functions: providing recorded data output of the Shake Table, and providing output results from a customized software model. In the first function, responses of the table to famous earthquakes are available to users on the Web. The second function allows users to change the structural parameters of a two-story building and to enter some input in order to know the corresponding output.
- The next level of interactivity is about the real-time experiments. Chapter 3 describes the software and hardware architecture of the Shake Table project and we discuss in detail the technology we use to enable users to run the Table via a Web interface and explain the choices we make to combine new programs with the existing third part software. We describe other software approaches that can be applied to the project and we compare them to the one we chose.
- All these simulations and experiments should be upgradeable and expandable and require a framework. Chapter 4 describes the need for security from a

software and a hardware points of view. In particular, we detail the security at the server side to limit the access to the table to authorized users and we explain how we programmed a switch to shut down the power in case of problems. Moreover, we gave the ideas for maintaining a reliable system that can work in the future with more users and more applications.

- Finally, chapter 5 presents a conclusion with the different improvements that can be done in the future. In particular, we define some improvements in terms of software and hardware to get more flexible and upgradeable tools.

Chapter 2

Simulations

2.1 The Issues

In this chapter, we present what we think is the first step to learn about earthquake Engineering: manipulating data earthquake such as displacements or accelerations, and analyzing responses of a structure to different inputs. For this purpose, we have defined two main functions: providing response to real earthquake data from the shake table, and building a simulation model that is fully customizable.

For these functions, we tried to match objectives according to different constraints. For the first function, our goal is to provide a system where multiple users can observe an experiment being run, knowing that the table can only be accessed by one user at a time. Therefore, we have chosen to serve up some earthquake data or acceleration data representing the response of the Shake Table without actually running an experiment at the time. These data can be processed through our Java Applets (described in the following pages of the chapter) or through some MATLAB code we have written[2]. For the second function, we want to build a software simulation model that is fully customizable, and as interactive as possible involving inputting different types of data to a system. For each function, we detail the goals and the software approach we used to reach these goals.

2.2 The Recorded Data Response

The first level of interaction and the first step to familiarize users with earthquake engineering is to discover real earthquake data.

2.2.1 The Presentation and Physical Approach

From this software, users can choose to get the data of some particular famous earthquake such as the Elcentro earthquake (Imperial Valley, 1940), or the Hachinohe earthquake (Japan, 1994) or the Kobe earthquake (Japan, 1995).

After choosing the earthquake, users can visualize the recorded data as if it was a real time earthquake. The displacements of the table, the displacements of the first floor and the displacements of the second floor can be seen on graphics.

To get a better visual idea of earthquakes, a movie showing the table shaking is available with the displacements. As can be seen in figure 2.1:



Figure 2-1: A movie of the shake Table in action.

From this point, the data are ready for a new step: data analysis and signal processing. By integrating once and twice, we can get the accelerations of the different elements of the structure. Moreover, we can apply different signal processing tools to the recorded data such as Fourier Transform or Wavelet Transform in order to denoise the signal and to get the frequency analysis of the different signals.

The software will give some recorded data describing the response of the structure in order to compare the simulation and the real data.

2.2.2 The Software Approach

In order to study the response of the structure remotely over the web, we decide to use a system based on:

- Java Applets on the Client Side.
- Java Server on the Server Side.
- text files and XML files on the Server Side.

This system will work as a database system using data we got from experiments using the shake table or other data we can get from other experiments. However, we haven't implemented the system using an efficient database software, we only have been using some text files to archive the data and this point can be an extension to the project in order to manage more data efficiently.

Here are some descriptions of our classes:

- We have built a class Earthquake that contains all the attributes of an earthquake such as name of the earthquake, accelerations, velocities and displacements for all the floors.
- The Java Applet is the UI on the Client side and we instantiate a Earthquake object in it. To visualize the results, the Applet includes three instances of a ScopePanel object, that corresponds to a convenient plot similar to that of MATLAB for graphics. All these objects implement the interface Runnable so

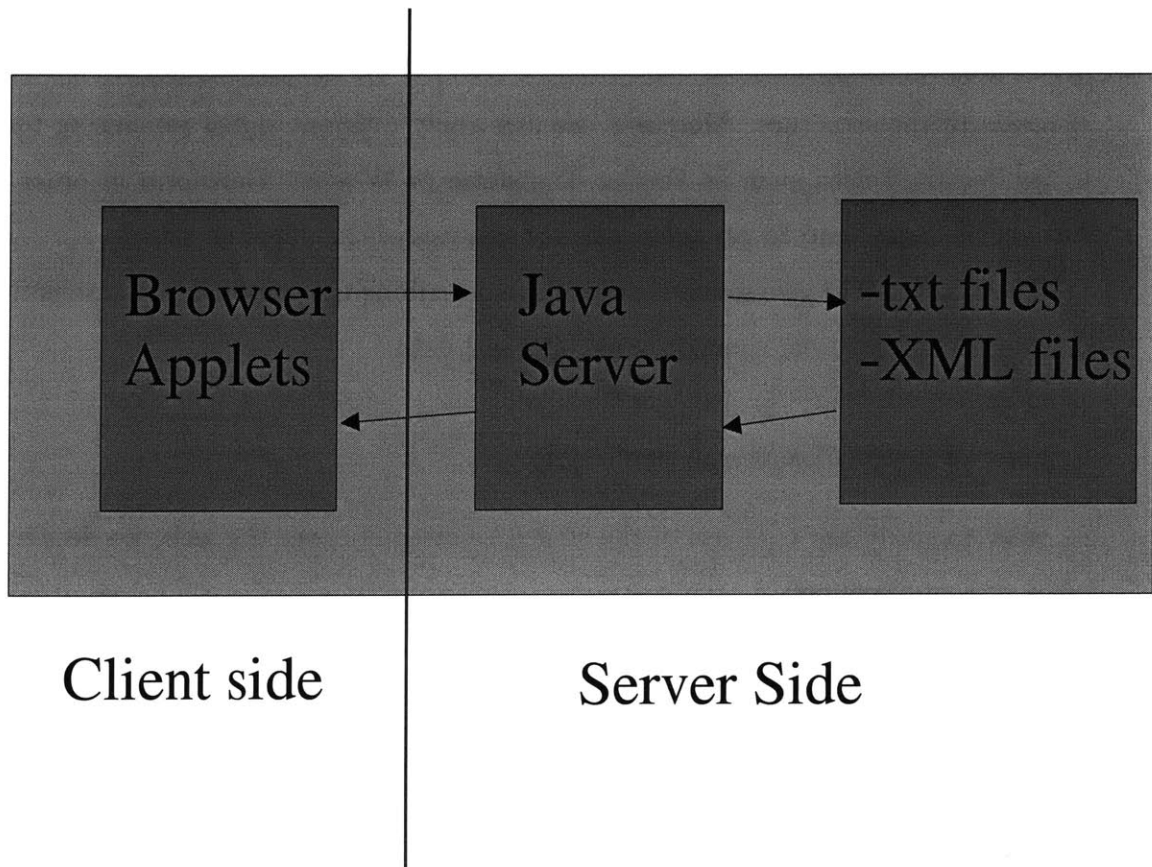


Figure 2-2: The Global Client/Server architecture.

that they can run together in a multi-threaded program. The goal of multi-threaded program is to get all the information on the screen as if they came in real-time. An other object of type Movie is instantiated in the Applet to get the recorded movie of the earthquake. This movie viewer is 100% Java and comes from the JMF¹ package available on the Sun Microsystem web site[3]. The link to the data is done via RMI², the Java client uses remote methods to access the data on the server side.

- The Server class is the class that implements the methods for the communication

¹Java™ Media Framework API (JMF) is registered trademark of Sun Microsystems, Inc.

²RMI stands for Remote Invocation Methods and is registered trademark of Sun Microsystems, Inc.

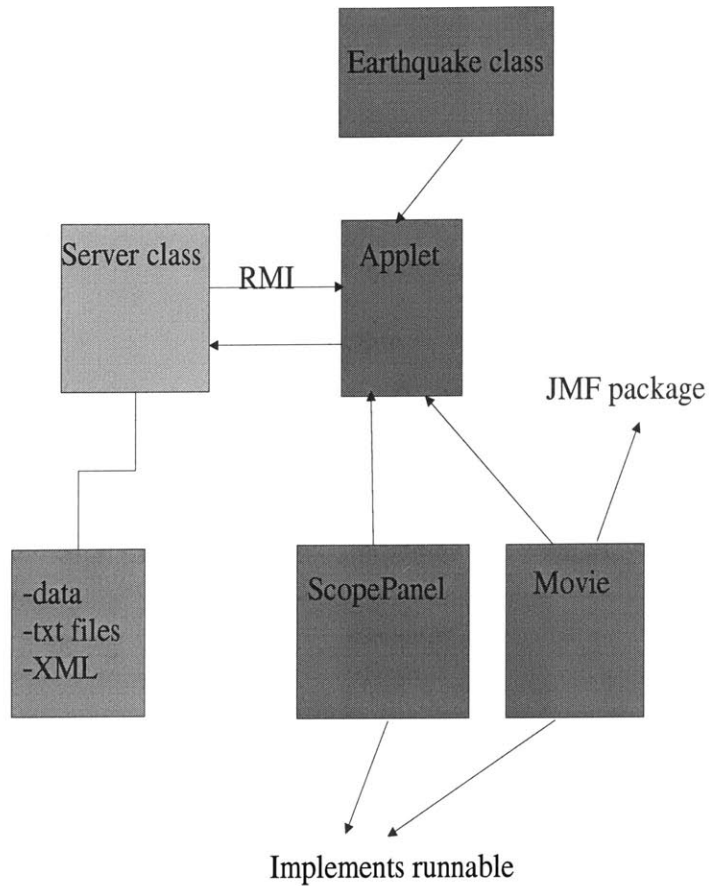


Figure 2-3: The UML architecture.

with the Java Applet. The methods enable the Applet to get the data which are on the Server. More software description are available in chapter 3 where the realtime communication is explained.

The earthquake data are stored on the server side as simple text files or XML³ files. First, we started with text files because simple text files are the most commonly used for storing earthquake data nowadays. We have moved to XML files in order to be compatible with other possible software or systems. The only point to upgrade our format was to define classes to generate and to read XML [4][5]formats.

³Extensible Markup Language

We have developed an XMLWriter class and an XMLViewer class. The XMLWriter class takes the standard data text files and build an XML file.

In figure 2.4, we present a simple summary of our XML model.

```
<?xml version = "1.0" encoding = "ASCII"?>
  <generalInfo>
    <name> </name>
    <date> </date>
    <location> </location>
    <magnitude> <magnitude>
  </generalInfo>

  <data>
    <timeIncrement> </timeIncrement>
    <displacements>
      <time0>
        <table> <table>
          <floor1> <floor1>
          <floor2> <floor2>
        </time0>
      ...
      <timen>
        <table> <table>
          <floor1> <floor1>
          <floor2> <floor2>
        </timen>
      </displacements>
    </data>
```

Figure 2-4: A simple XML structure.

2.3 The Simulation Software

Once we know what real earthquake data look like, it is interesting to compare the results with some models.

2.3.1 The Presentation and Physical Approach

The simulation is based on a simple model: a two-story model that is supposed to be equivalent to the shake Table. We have developed a MATLAB script and a Java version of a program for calculating response of a 2 DOF freedom model building which is modeled as discrete shear beam. For the computation, we use time step integration(Central Difference Method). Basically, here are the different steps: assembling the 2 x 2 mass matrix, assembling the 2 x 2 stiffness matrix, computing the Modal frequencies, assembling the damping Matrix, assembling the matrix containing the story load, calculating constants for time step integration, creating a 2 x 1 vector for initial conditions, performing the time step integration, finding the ground velocity and displacement, and getting the absolute responses.

To make the program customizable, we give two possible inputs from users:

- Loading earthquake data accelerations. These data can come from real earthquakes.
- Entering the time series of the equivalent forces applied to the structure by
 - Inputting the duration of simulation and the time increments.
 - Inputting the equivalent forces applied to the storeys.
 - Initializing responses (i.e, accelerations, velocities, and displacements).

The algorithm of the model was designed by Irfan Baig and detailed and explanations can be found in his master thesis [2].

For this simulation, most parameters should be customizable:

- the physical values for each floor: the masses, the stiffnesses, the initial conditions of accelerations, velocities, displacements.
- the displacements can come from some real earthquake data or the users should be able to use any functions to excite the structure.

An example of the User Interface with the displacements of the different floors of the model can be seen in figure 2.5 at the end of Chapter 2 on page 28.

2.3.2 Software Approach

In this part, we use almost the same software approach as that of the previous function. The main difference is that users can input data and get outputs from the system.

We know that in Java we have security restrictions for accessing files that are on local machines when using applets. There are different possibilities to overcome this security problem. Here are three ways to solve the problem using Java technologies

- Using a signed Applet.

When using a signed Applet, we can set up the client browser to grant some particular permissions to the Applet. For instance, applets can have permissions to read or write files on the client machine. To sign an Applet, we have to sign a jar file that includes the applet class. More information about signing a jar file can be found in [6], [7] and [8].

Advantages:

- There is only an Applet on the client side and few requirements in terms of software to install: a browser with a JVM.

Disadvantages:

- Signing an applet requires third party certificates such as Verisign certificates but they are not free. If we build our own certificate, it is likely that very few users will trust it and use our program.
 - Signing an Applet is cumbersome and sometimes difficult to debug.
- Using Java 1.4 and Java Web Start Technology.

Java 1.4⁴ is the latest version Java and it includes Java Web Start v1.0.1_02 [9] (bundled with the Java 2 SDK and Java 2 Runtime Environment (JRE)). Java Web Start is a new application-deployment technology that gives the power

⁴<http://java.sun.com/j2se/1.4/>

to launch full-featured applications with a single click from a Web browser. In particular, these applications can ask users if they accept to grant some particular rights, for instance to manipulate files on their own hard drive. Some example are available on the Sun Microsystem website.

Advantages:

- There is a full client side program on the client and after the first use, it is stored on the local hard disk ;therefore, there is no time delay for downloading the Applet packages.

Disadvantages:

- The package requires downloading a full package and software on the client side and not all users are willing to install a large amount of classes and programs for a single application.
 - The package is based on the Javax.swing and is not as fast as the Java.awt package. It requires a fast machine with a fast connection to get acceptable results.
- Using a Java server in particular servlets: our choice.

Users upload their files to our server. The server via a servlet reads the data and send them to the Applet or creates an XML file from this data using the same classes as described in the previous section about XML.

The advantages of our choice are the following:

- We use the same class architecture as that of the real earthquake data presentation.
- We can easily rely on the Java Server and upgrade the functions if required.
- We only use a Java Applet on the client side; hence, users only need the minimum requirements to use our programs.

At the same time, we have tried to tackle a problem of format for the earthquake data because there is no unique earthquake data format according to Prof. Kausel.

We have thought about different ways to input data to our program:

- a simple way to copy and paste data to a TextArea.
- uploaded files to our server. The data files can be read from a servlet and the servlet can send the data to an Applet.
- an XML format can be generated by our program and users can download the output at the end.

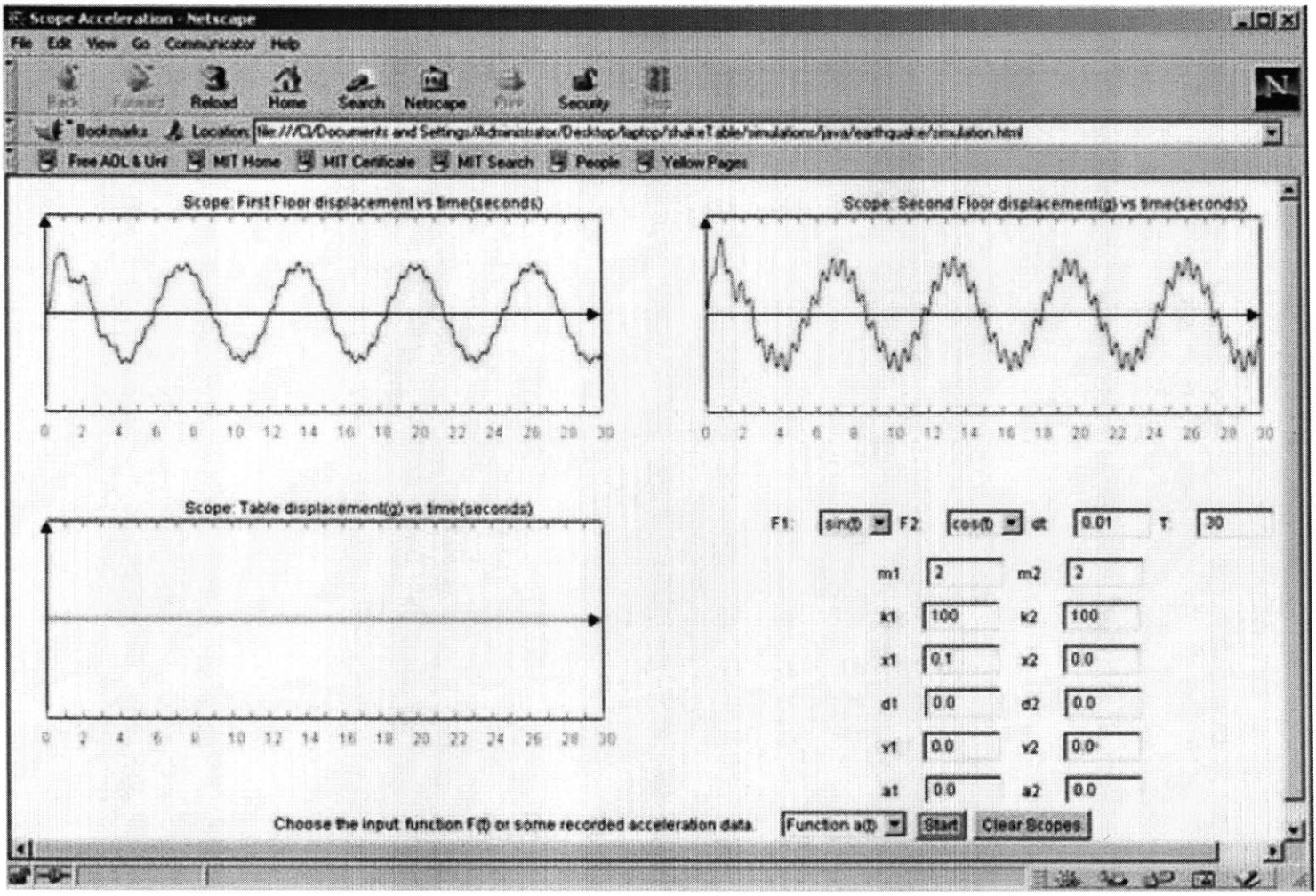


Figure 2-5: The UI of the simulation.

Chapter 3

The Real Time Experiments

3.1 The Issues

In this chapter, we present the real-time part of the Shake Table Lab. We describe in details the constraints we had in terms of hardware and software. Finally, we show the solutions we designed to fit the constraints, in particular, we give detailed examples for the code.

3.2 The Hardware Architecture

To give a better understanding of the way the Shake Table works, we present the most important pieces of hardware of the project:

3.2.1 The Table

The table includes 3 floors and is moved by a brushless servo motor driving a 1/2 lead screw. The table itself slides on low friction linear ball bearing on 2 ground hardened shafts.

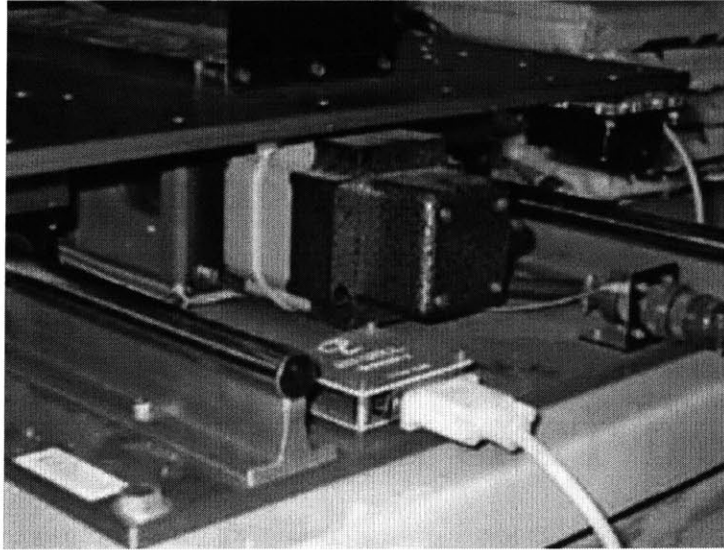


Figure 3-1: The table moved by brushless servo motor.

3.2.2 The Accelerometers and the Active Mass Damper

The accelerometers keep track of the displacements of the ground floor, of the first and second floor. They measure a range $\pm 5g$ within output of ± 5 Volts. They are wired to the data acquisition board. The active mass damper is also wired to the data acquisition board and it is mounted on the last floor of the table to damp the system during earthquakes. When using the active mass damper, we can get very different responses with the same inputs.

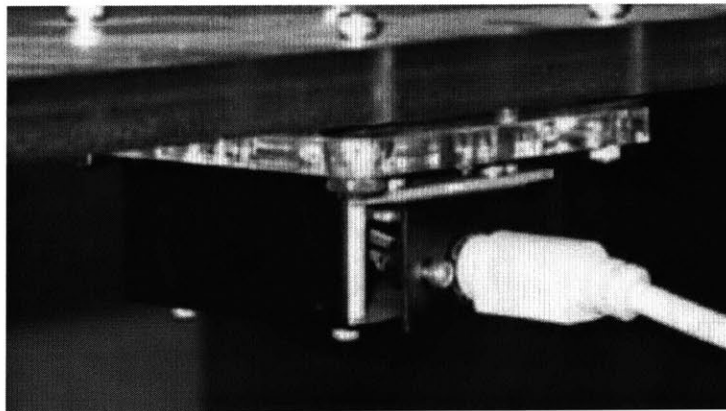


Figure 3-2: An accelerometer mounted on the second floor.

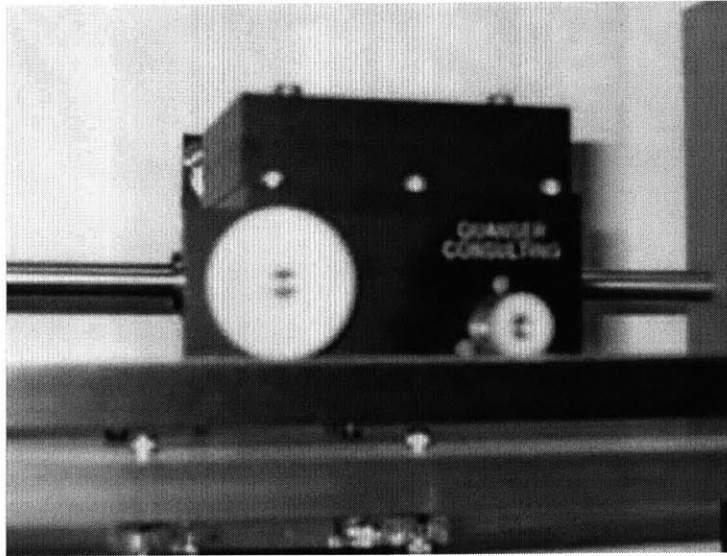


Figure 3-3: The active mass damper.

3.2.3 The Power Module

The power module powers the motor and also the accelerometers. This part is the main piece of hardware for security. The power should be shutdown in case of problems. In chapter 4, we present the security part for this device.

3.2.4 The Data Acquisition Board

The system is supplied with a MultiQ-2E data acquisition board. This piece is the main part in terms of controlling, monitoring the experiments using the table. The data acquisition board is wired to a computer from which we can access the table via a commercial software.

3.3 The Software Architecture

3.3.1 WinCon

WinCon is the proprietary software provided by Quanser, inc to simulate earthquakes from a computer. It runs SIMULINK generated code using the MATLAB Real Time



Figure 3-4: The power module: the most important part for security.

Workshop. WinCon consists of a Client and a Server. Here the vocabulary used by Quanser, Inc is different from the usual vocabulary. The client is the part of the program running on the machine wired to the Shake Table and the servers are the remote programs that can run the table remotely using the Web.

From the WinCon Server users can:

- Convert SIMULINK¹ diagrams to Wincon Controller Library files using Real-Time Workshop.
- Compile and link the code using Microsoft Visual C++.
- Download the Wincon Controller Library file to run on the WinCon Client.
- Start and stop the WinCon Client.
- Maintain TCP/IP communications with the WinCon Client.

¹SIMULINK, trademark of MathWorks, Inc, is an interactive tool for modeling, simulating, and analyzing dynamic, multidomain systems. It is included in MATLAB

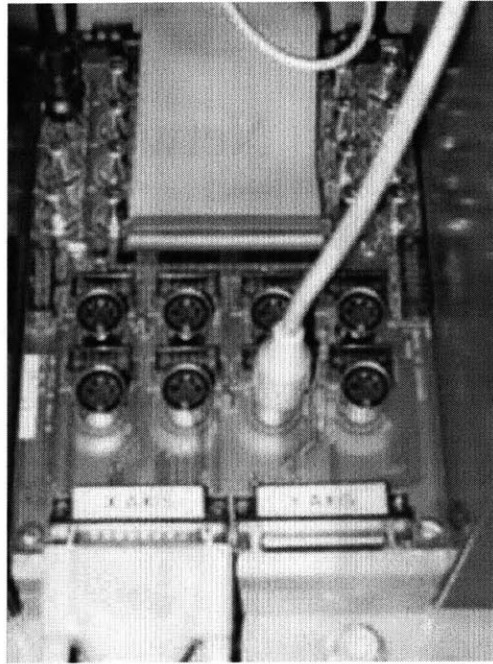


Figure 3-5: The data acquisition board: the device to control the experiments.

- Maintain communications with SIMULINK to perform real-time changes in the WinCon Client.
- Make changes to WinCon Client using Control Panels.
- Plot the data streamed from a desired WinCon Client in real-time.
- Save data to disks.

The WinCon client is the part on the machine that controls the table and can perform the following:

- receive the WinCon Controller from the Server.
- run the controller code.
- run a simulated earthquake and gets the data back from the data acquisition board.

A classical configuration between WinCon Server and WinCon Client is summarized in figure 3.6:

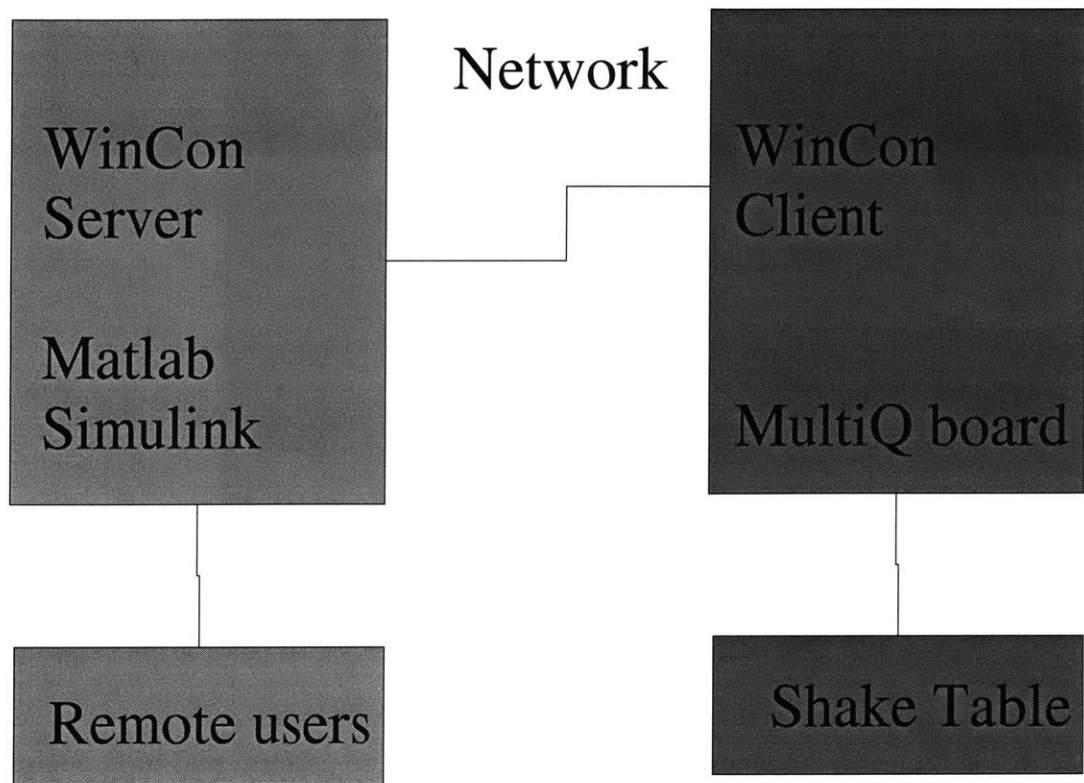


Figure 3-6: A two PCs architecture showing the role of WinCon.

3.3.2 MATLAB-SIMULINK

How can we design earthquakes and get the data we want? The MATLAB-SIMULINK can answer this question. We take data from a real earthquake and use MATLAB to rescale the earthquake to the size of the table in time and magnitudes. Using SIMULINK, we can design the diagrams that represent the earthquake simulation and that can define different outputs and controls such as the the displacements of the different floors or the activation of the Active Mass Damper. In addition to the data of the earthquake, these diagrams are then compiled using Microsoft Visual C++ to get a WinCon Controller Library file, which can be sent to WinCon to run the simulation we designed.

Moreover, Quanser Consulting provides MATLAB scripts which can be used to control WinCon from the MATLAB prompt. These are useful for applications which require automated execution such as gain schedule, data collection, adaptive learning control and parameter estimation. Among the main functions, we have used the following:

- WC_BUILD to build the code for a SIMULINK model.
- WC_START to start the code for a model.
- WC_STOP to stop the code.
- WC_NEWPLOT to open a new plot.
- WC_SAVEPLOT to save the data to a MAT file.

We have successfully used the WinCon architecture using a Client and a Server. However, we can see how much this configuration is restrictive since we need to have WinCon installed on every server on which we want to be able to run simulations. Moreover, the system is not very flexible at the moment and we cannot display the results the way we want or we cannot have a direct access to the earthquake data to do some signal processing analysis for instance. One important problem concerns all the requirements of this configuration: to have MATLAB, WinCon installed on each Server.

3.3.3 Integration of Java with SIMULINK models

In order to give access to a larger number of students or visitors who don't necessarily have access to this commercial software, we have developed a Java Web based interface to give control to the shake Table. This Java interface can solve the problem of customization of the results. We can have the data formatted as we want and we took the same format as described in our simulation part in chapter 2. These data can be analysed using different signal processing tool available in Java.

The most difficult problem is to link our Java interface to the existing software and hardware we have described in the beginning of this chapter. To solve the problem, we have three main ideas:

- Control the hardware directly at the accelerometers and data acquisition board level. This solution is very clear in the way that we don't have to access the Quanser Consulting program. Yet, this solution requires some solid low level programming and above all programming an equivalent of what Quanser Consulting have done. This solution requires some time and should be considered in the long run. However, some part of this solution is still implementable, programs to get access to the accelerometers can be used to get access to the data without using the whole Quanser process.
- Use the C code generated by Visual C++ to design our own C code programs to run simulations and control the table parameters. One difficulty of this solution is that the software is provided as is without any API to develop the program or to control it. This option can be implemented but some copyrights issues may occur about using some commercial software code.
- We can use the MATLAB script functions provided by Quanser Consulting to run the basic steps of the programs. We can then run simulations on the table without WinCon Server using a browser. The main idea is to link some Java programs to run some Matlab scripts. Hence, requirements for users are decreased to a simple browser including a Java Virtual Machine. This solution is fast to implement and can be the beginning of a more complete framework for this kind of experiments.

In the next section, we explain how we design a Java interface to get access to some MATLAB program and to communicate between MATLAB and Java. We detailed different ways to do it and compare them.

3.4 The Real-Time Programs

MATLAB is today's most popular scientific computing environment. This is not by coincidence: With MATLAB, carefully designed and implemented numerical algorithms and data structure packages are accessible through an intuitive as well as powerful user interface. Many companies and universities use MATLAB to develop new algorithms, or perform graphical evaluation of data sets, to name just a few of its many applications. Furthermore, MATLAB is an open system, i.e. users can add functionality by providing their own routines.

MATLAB can interface with Fortran or C programs through a so-called C or Fortran Engine. This allows compiled programs to open and communicate with a MATLAB session and thereby make use of many of MATLAB's features. Unfortunately, a Java engine with similar capabilities is not yet available (with MATLAB version 6.1). MATLAB can access Java programs. However, as of now, there are no Java packages included in MATLAB that provide the link in the other direction.

Why are we interested in interfacing Java with MATLAB in the first place? First of all, there is no need to re-implement many of MATLAB's functions in Java if we can just access these routines. More importantly, we would like to design Web applications that perform calculations using MATLAB.

In the forthcoming sections, we present three ways of accessing MATLAB from a Java program. This could be a regular Java application, but also a Servlet or JSP ² page.

3.4.1 Approach 1: MATLAB Invocation and Communication via the Java Runtime Class

Description

This approach uses the Java Runtime class to start a new MATLAB process. Communication is achieved through acquiring MATLAB's standard input and output

²JSP and JavaServer Pages are registered trademarks of Sun Microsystems, Inc.

streams.

Implementation

Let us assume we would like to write Java code to access MATLAB which looks simply like this:

```
import MATLAB.*;
import java.io.*;

public class Main {
    public static void main(String[] args) {
        Engine engine = new MATLAB.Engine();
        try {
            engine.open("matlab -nosplash -nojvm");
            System.out.println(engine.getOutputString(400));
            engine.evalString("A = 5;");
            engine.evalString("B = 15;");
            engine.evalString("A*B");
            System.out.println(engine.getOutputString(400));
        }
        catch (IOException e) {
        }
    }
}
```

We would like to import some sort of package that contains all our MATLAB Interaction classes (in this example called 'MATLAB'). Then, we just want to create a new instance of our MATLAB Engine, start a new MATLAB process, and then send commands to this process, and receive results back.

The Java code below shows how to open a MATLAB session. This is the first method of our class 'Engine' that will provide the functions to access MATLAB.

First of all, we need to declare a couple of private variables. `p` is the handle to the MATLAB process. `in`, `out`, and `err` will be used to get hold of MATLAB's standard input/output streams.

The public method `open` may be called from a main program. The argument `startcmd` that is passed should contain a valid start string, such as `'matlab'`. Other possibilities for a start string would be `'matlab -nojvm -nosplash'` to suppress the MATLAB splash screen and startup of MATLAB's Java Virtual Machine ³, or `'ssh hostname /bin/csh -c 'setenv DISPLAY <hostname>:0; matlab'` to run the MATLAB process on another machine through a secure shell. The method itself does nothing but invoking the MATLAB process by calling the `'exec'` method, and obtaining the input/output streams.

```
package MATLAB;

public class Engine {

    private Process p;

    private BufferedInputStream in;
    private BufferedOutputStream out;
    private BufferedInputStream err;

    public void open(String startcmd)
        throws IOException {
        try {
            System.err.println("Opening MATLAB...");

            p = Runtime.getRuntime().exec(startcmd);

            out = new BufferedOutputStream(p.getOutputStream());
```

³MATLAB 6 starts its own Java processes by default. The GUI, for example, is Java based.

```

        in = new BufferedInputStream(p.getInputStream());
        err = new BufferedInputStream(p.getErrorStream());
    }
    catch(IOException e) {
        System.err.println("MATLAB could not be opened.");
        e.printStackTrace();
        throw(e);
    }
}
}

```

The next methods we need to provide are to send and receive information. To do this, we just feed data to the input stream, and retrieve data from the output stream of the MATLAB process. We make these methods synchronized to avoid any interference between multiple writes and reads to the MATLAB streams.

The method `evalString` sends a string to the MATLAB process for evaluation. Since the input/output pipes are binary, we need to convert the `String` to an array of bytes with the `getBytes` method. `enc` denotes the character encoding that is used to perform this conversion. A platform's default character encoding can be determined with the following code sequence:

```

InputStreamReader reader =
    new InputStreamReader(new ByteArrayInputStream(new byte[1]));
enc = reader.getEncoding();

```

Finally, we write the byte array to the stream and flush it.

```

public synchronized void evalString(String str) throws IOException {
    try {
        System.err.println("Evaluation string: "+str);
        byte[] buffer = str.getBytes(enc);
        out.write(buffer, 0, buffer.length);
    }
}

```

```

        out.flush();
    }
    catch(IOException e) {
        System.err.println("IO Exception occurred while sending data to MATLAB");
        e.printStackTrace();
        throw(e);
    }
}

```

The last method we would like to present in this section is the `getOutputString` method. We read a specified number of bytes from the MATLAB output stream that is piped and accessible through the buffered input stream in:

```

public synchronized String getOutputString (int numberOfChars)
    throws IOException {
    byte[] buffer = new byte[numberOfChars];
    try {
        System.err.println("Opening input buffer");
        in.read(buffer, 0, numberOfChars);
    }
    catch(IOException e) {
        System.err.println("An IO Exception occurred.");
        e.printStackTrace();
    }
    return new String(buffer);
}

```

With this method, our core communication class is complete. This class should suffice for simple applications. We can run any set of MATLAB commands and retrieve the results. A big drawback, however, is the fact that we will eventually have to parse the output stream to interpret the results instead of just displaying them. This would become necessary if we would like to further process the results in Java,

for example.

Advantages and Disadvantages

Let's look at the advantages and disadvantages of this approach:

Advantages:

- MATLAB can be started directly from a Java class, either on the same machine, or, under unix-based systems, even on another machine.
- platform-independent.
- Very little Java code required.
- Will most likely work with future versions of MATLAB.
- No additional installation procedures or system setup changes required.

Disadvantages:

- Transferring of data is I/O, character- and stream-based, therefore inefficient especially for larger amounts of data (high latency, low transfer rates compared to direct memory access).
- Parsing of the MATLAB output stream has to be done manually.

3.4.2 Approach 2: Using a JNI Wrapper for MATLAB's C Engine

Description

With Java's Native Interface (JNI) [16][17][18], we can write a wrapper class for The Mathworks' C Engine library [20]. While this is somewhat cumbersome to do, we can take advantage of all the functionality that this library provides for use with C.

Building a Java application that contains native libraries is more difficult than compiling a pure Java application. Therefore, we describe this process in more detail. We have also included an example make file in the appendix (for Linux systems).

Implementation

We would like to keep the main program the same- the complexity of the native interface does not need to bother the application programmer that uses the MATLAB engine. Let us just change the class name to account for the use of native methods with the implementation of approach 2:

```
NativeEngine engine = new MATLAB.NativeEngine();
```

Let us now take a look at the implementation of the `NativeEngine` class. It basically contains only the function headers of the native wrapper library (which is written in C).

```
package MATLAB;

import java.io.*;

public class NativeEngine {
    public native void open(String startcmd) throws IOException;
    public native void evalString(String str);
    public native String getOutputString(int numberOfChars);
    static {
        System.loadLibrary("engineJavaMatlab");
    }
}
```

As with approach 1, we provide the same three functions `open`, `evalString`, and `getOutputString`. The function header of the `open` method indicates that we have provided error handling for this function. This would apply for the other functions,

too. However, we did not include this since it is repetitive and can be easily added by the reader. A good resource regarding error handling with JNI can be found in [17].

The `System.loadLibrary` statement is an essential part of each class containing native functions— Here, we can tell the JRE⁴ which library has to be loaded at runtime that contains the declared methods. You might wonder how the JRE knows about the path where this library can be found— we will discuss this later.

Let us now look into the wrapper class written in C, `NativeEngine.c`.

```
#include <jni.h>
#include "Matlab_NativeEngine.h"
#include <stdio.h>
#include "engine.h"

#define DEFAULT_BUFFER_SIZE 1024

Engine* ep;

char outputBuffer[DEFAULT_BUFFER_SIZE];
```

The first include statement `#include <jni.h>` provides the header file for the Java Native Interface (this header file is part of any standard Java distribution). Furthermore, we include `MATLAB_NativeEngine.h`. This header file can be generated automatically from the `NativeEngine.class` file. We will discuss this later. Finally, we need `engine.h` (provided by the Mathworks, this file is included with a standard MATLAB distribution) that declares the MATLAB C engine functions. The remaining lines are some global variable definitions, notably a handle to the MATLAB connection `ep`.

The method `Java_Matlab_NativeEngine_open` takes care of opening a MATLAB session:

⁴Java Runtime Environment

```

JNIEXPORT void JNICALL
Java_Matlab_NativeEngine_open(JNIEnv *env, jobject obj,
    jstring startcmd) {

    const char *c_string;
    c_string = (*env)->GetStringUTFChars(env, startcmd, 0);
    if (!(ep = engOpen(c_string))) {
        jclass exception;
        (*env)->ReleaseStringUTFChars(env, startcmd, c_string);
        exception = (*env)->FindClass(env, "java/io/IOException");
        if (exception == 0) return;
        (*env)->ThrowNew(env, exception, "Opening MATLAB failed.");
        return;
    }
    (*env)->ReleaseStringUTFChars(env, startcmd, c_string);
    engOutputBuffer(ep, outputBuffer, DEFAULT_BUFFERSIZE);
}

```

The long function name corresponds with the declaration in the automatically generated header file `MATLAB_NativeEngine.h`. It includes the package name and class name to avoid naming conflicts with other methods. The interface pointer `env` and the pointer to this object `obj` are passed with any JNI function (they are required to make JNI Interface functions available from the native subroutines [19]). The `startcmd` argument is a Java String object, as indicated by the variable type `jstring`.

Let us go through the function body. The C Engine function `engOpen` [20] that we would like to call to open a MATLAB session takes a C-style string as input argument (i.e. an array of characters terminated by zero). We can use the JNI function `GetStringUTFChars` to perform the necessary conversion of our Java String object. We have to remember to free the allocated memory after use with the `ReleaseStringUTFChars` method to avoid memory leaks (C does not provide garbage

collection). Next, we try to open a MATLAB session with the converted start command `c_string`. If successful, `engOpen` returns a handle to this MATLAB session. We store this pointer to `ep` for later use. Otherwise, we throw an `IOException` that can be processed by the calling routine. Finally, we call the MATLAB C Engine method `engOutputBuffer` to store the MATLAB output stream data to our own buffer (by default, the MATLAB C Engine discards all generated output, so we need to do this to receive and process results).

Let us proceed with the native wrapper methods for sending commands to MATLAB and receiving results back:

```
JNIEXPORT void JNICALL
Java_Matlab_NativeEngine_evalString(JNIEnv *env, jobject obj,
    jstring str) {
    const char *c_string;
    c_string = (*env)->GetStringUTFChars(env, str, 0);
    engEvalString(ep, c_string);
    (*env)->ReleaseStringUTFChars(env, str, c_string);
}
```

```
JNIEXPORT jstring JNICALL
Java_Matlab_NativeEngine_getOutputString(JNIEnv *env, jobject obj,
    jint numberOfChars) {
    char c_string[numberOfChars];
    memcpy(c_string, outputBuffer, numberOfChars);
    return (*env)->NewStringUTF(env, c_string);
}
```

These methods are a bit shorter since we have omitted the error handling, which could be similar to the one of the `engOpen` method from above. As we did before, when transferring Strings from Java to MATLAB and vice versa, we have to take care of the proper String conversion to C format.

Sending Strings to MATLAB is simple via the MATLAB C Engine library. The method we need to access in the library is called `engEvalString`. This method takes care of the I/O operation, i.e. transfers the data to the MATLAB session. To do this, the method needs a handle to the session that we stored in `ep` when opening MATLAB with `engOpen`.

Returning results to MATLAB easy, too. We do not even need to call a function of the MATLAB C Engine to do this, since we have instructed the Engine to store results in our buffer `outputBuffer` when opening the MATLAB session. So we just need to copy the specified number of characters from our allocated output buffer to a String that we pass back to the calling Java routine.

Compilation

As mentioned earlier, compiling and linking a Java application containing native libraries requires more work than translating a pure Java application to byte code. A detailed explanation on how to do this can be found in [17], for instance. Here's a brief summary:

1. Translate the Java source files to byte code (class files) with the `javac` compiler
2. Generate the native library header file (here: `NativeEngine.h`).
3. Compile and link the native library with a C compiler (e.g. the GNU Compiler Collection `gcc` under Linux) as shared library.

The appendix contains an example make file for Linux for your reference. Before starting the application, make sure you provide a valid `LD_LIBRARY_PATH` variable that points to the directories containing the shared libraries.

Advantages and Disadvantages

Let's look at the advantages and disadvantages of the second approach:

Advantages:

- MATLAB can be started directly from a Java class, either on the same machine, or, under unix-based systems, even on another machine.
- The MATLAB C Engine library functions are accessible. Special functions to retrieve arrays or send arrays to MATLAB are already implemented and can be easily used with little additional programming effort.

Disadvantages:

- Transferring of data is I/O, character- and stream-based, therefore inefficient especially for larger amounts of data (high latency, low transfer rates compared to direct memory access).
- Platform-dependent. Different compilation, installation and setup routines apply to different operating systems.
- Will most likely not work with future versions of MATLAB without small adjustments and re-compilation.
- Cumbersome implementation. Harder to debug because of the native code parts.

3.4.3 Approach 3: Using a Server-Client architecture and RMI

Description

The main idea of this approach is to use a server-client architecture with multiple clients. The first client (say client1) can be an Java applet and the second client(client2) can be a Java application running under the Java virtual machine (included in MATLAB since version 6).

These clients can communicate via the server and Remote Method Invocation (RMI)[22][23][24]. Therefore, we can send inputs from client1 to client2, which will do some computations and send the results back to client1. The application that runs under MATLAB will be started by some MATLAB script[25].

Implementation

In this section, we are going to see a simple implementation of this approach.

First of all, the way to start MATLAB from client1 is similar to that of approach 1. The only difference is that it is included in a remote method that can be used by client1. For instance, this method can run a particular m file that will create the Java MATLAB client2. In the following lines, we can see the interface that contains the method to start a MATLAB script.

First, we define the method in the interface:

```
package RMIPack;

public interface ChatServer extends java.rmi.Remote {
    /**
     * Chatter calls this to start a MATLAB script.
     */
    public void startMatlab(String message)
        throws java.rmi.RemoteException,
            java.io.IOException;
}
```

Then, the method is implemented in the remote object class:

```
/**
 * Called by client1 to start a matlab file
 */
public void startMatlab(String message) throws
    java.rmi.RemoteException,
    java.io.IOException
{
```

```
    Process p = Runtime.getRuntime().exec(message);  
}
```

When executing this method, a MATLAB script is run in a MATLAB session. For the string message, we can choose: message = "myPath/matlab.exe Multiply(A,B).m". For instance, the method can call the following simple script:

```
%Multiply.m  
%test to send data to an applet  
product = function(A,B)  
  
%import the package required for server/client operations  
import RMIPack.*  
  
%create a matlab client object  
client2 = matlabClient  
  
%MATLAB does the computation  
product = A*B  
  
%connect this client2 to the server and  
%send the results back from client2 to client1  
client2.myTest(product);
```

The syntax for Java under MATLAB is different from the usual Java syntax and information about it can be found in [25] or on line.

Now, let's see in details some elements of the matlabClient class:

```
public class matlabClient extends ChatClient{  
  
    public static void myTest(int[] args) {
```

```

matlabClient client = new matlabClient();

//properly disconnect
try {
client.disconnect();
} catch (Exception ex) {ex.printStackTrace(); return;}

//create a chatter
client.start();

//logon to the server
try {
client.connect();
System.out.println("Connected");
} catch (Exception ex) {ex.printStackTrace(); return;}

//send a message to client1.
client.sendMessage(args);

//properly disconnect
try {
client.disconnect();
} catch (Exception ex) {ex.printStackTrace(); return;}

    }//end myTest

} //end ChatClient class

```

The matlabClient class inherits from the ChatClient class that contains the sendMessage() method to send a message to another client. This method is the following:

```

public void sendMessage(double myDouble){

server.chat(my_name, MyDouble);

    } catch (java.rmi.RemoteException ex) {ex.printStackTrace();}
    catch (java.io.IOException ex){ex.printStackTrace();}

} //end sendMessage

```

The class ChatClient uses a remote method that is in the remote object:

```

/**
 * Called by clients to broadcast chat messages
 */
public void chat(String name, double myDouble) throws java.rmi.RemoteException
{
    Enumeration enume = chatters.elements();

    while (enume.hasMoreElements()) {
        UserInfo u = (UserInfo) enume.nextElement();
        u.chatter.receiveChat(name, myDouble);
    }
}

```

All the connected clients will receive the outputs. As we can see, the clients can use the same methods to communicate to each other. They use methods that use RMI methods from the server. This system is fairly high level and enables the client to communicate with the default socket system without taking care of it.

A UML diagram to give the big picture of the architecture of the system can be found in Figure 3.7 on page 56.

Advantages and Disadvantages

Let's look at the advantages and disadvantages of the third approach:

Advantages:

- MATLAB can be started directly from a Java class, either on the same machine, or, under unix-based systems, even on another machine.
- Communication 100% Java between Java clients. By programming the methods, the system is fully customizable, in particular with the proper interface.
- This approach is almost platform-independent since it only uses Java but some server programming depends on the platform, in particular for securities.

Disadvantages:

- Requires a Java Server (even if Tomcat is free) and some Server programming.
- no simple implementation even for simple applications and requires some reliable RMI programming.
- Requires some MATLAB script programming and mixing Java and MATLAB may not be always very reliable.

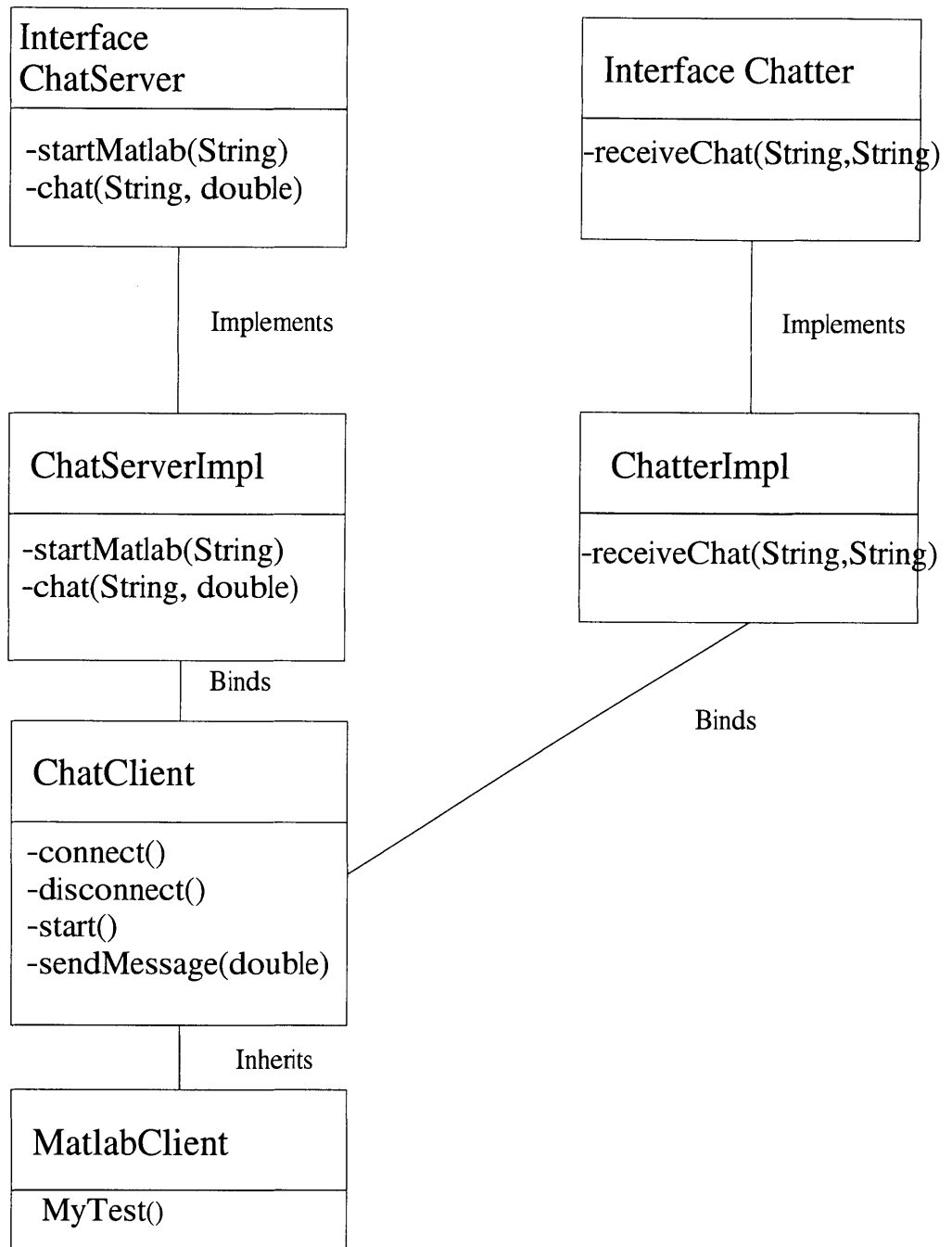


Figure 3-7: A UML diagram for the RMI architecture.

Chapter 4

The Security and Management of the System

4.1 The Issues

In terms of security, in addition to the required security of a classical server, we present two aspects we have found fundamental for our project: the first one is the access to the control of the table, which has been decided to be restricted to some users, the second aspect is about the security of the hardware, in particular the option to shut down the power if there is an accident.

4.2 The Security

4.2.1 The Software and the Servers

For the security part of our Java server, since it was an experimental project, we have only used the security options provided by the Tomcat Server¹ [10][11].

Basically, Tomcat provides some security realms (mechanisms) that give abilities to administrators to set up a degree of constraints for each web service. When accessing a Servlet, users will be challenged and will have to enter valid username and

¹Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies.

password. The server will check these information in an XML file on the server, the default location of the MemoryRealms XML file is the

```
<tomcat_home>/conf/tomcat-users.xml.
```

It looks like the following:

```
<tomcat-users>
<user name="tomcat" password="tomcat" roles="tomcat" />
<user name="role1" password="tomcat" roles="role1" />
<user name="both" password="tomcat" roles="tomcat,role1" />
</tomcat-users>
```

In terms of code, Tomcat has this type of realm functionality built in. The component that provides this functionality is the `org.apache.catalina.Realm` interface.

```
<tomcat_home>/src/catalina/src/share/org/apache/catalina/Realm.java
```

The class that defines the memory realm is `org.apache.catalina.realm.MemoryRealm`.

In addition to this level of security from the server, we have developed our own security system. The incentive to restrict the access to the shake table are only based on the fact that unknowns exist since we haven't programmed all the software that is involved in the control of the table and since we prefer to give access to the table in priority to faculty who use it for educational or research purposes.

We have used a servlet that checks the IP addresses of the users. If the IP address is part of an authorized list, the servlet can generate an Applet that is the User Interface to control the shake table using RMI and the architecture we have described in chapter 3. To get the IP address of a client that uses our Servlet, we used the `javax.servlet.http.HttpServletRequest.getRemoteAddr()` method. From this point, we can define a XML file that contains all the authorized IP addresses.

We are going to see how we designed this servlet that generates our applet, this architecture of it relies on John B. Smith's online course on Advanced Java [12].

The main idea is to design a multi-threaded Servlet: The first thread returns the generated HTML and starts the server listen thread, if it is not already running.

The second thread listens for a client connection and, when it gets one, starts a thread to handle that connection. The third thread provides services to a single client connection. There may be multiple instances of this thread class for multiple connections

The different steps for security are summarized in Figure 4.1:

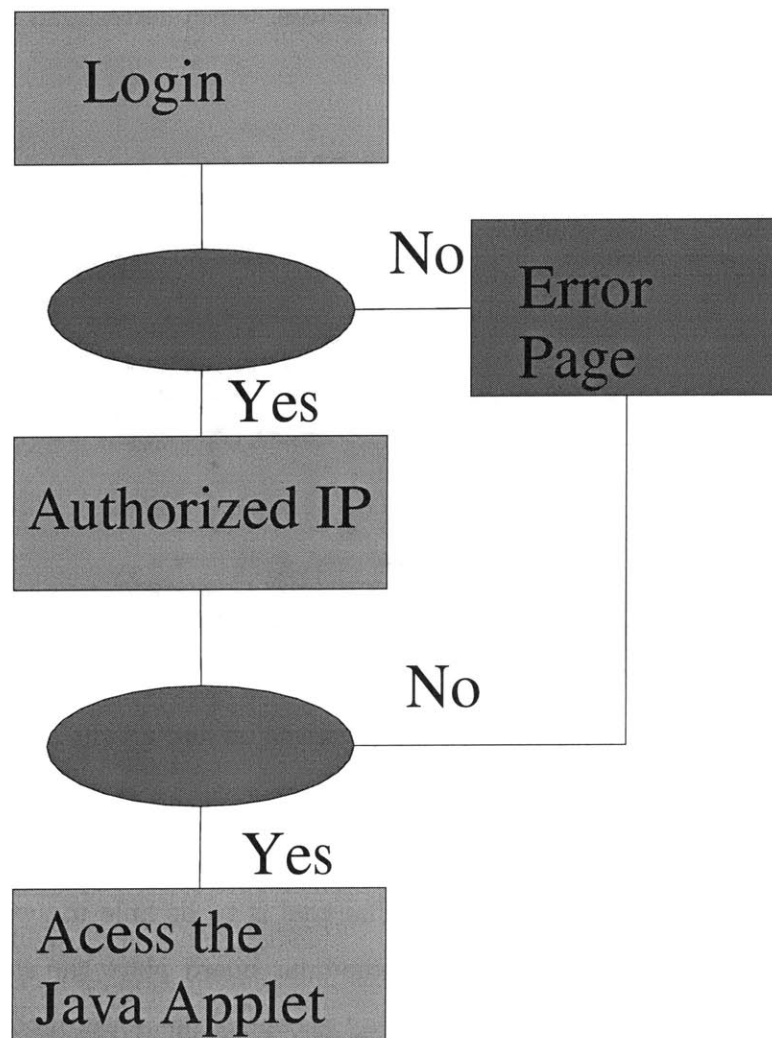


Figure 4-1: The different security steps to get Access to the Java UI.

4.2.2 The Hardware

Since, accidents damaging the shake table can occur during experiments, we have developed a Web Switch function based on a switch board to shut down the power of the table. We are going to describe the different steps and levels of this web Switch function.

First of all, let's describe the hardware: we used the SDDRB4 modular board by B&B Electronics. This modular board has four relays. The I/O connections to the I/O lines are made through DB25 connectors, which correspond to parallel port plugs.

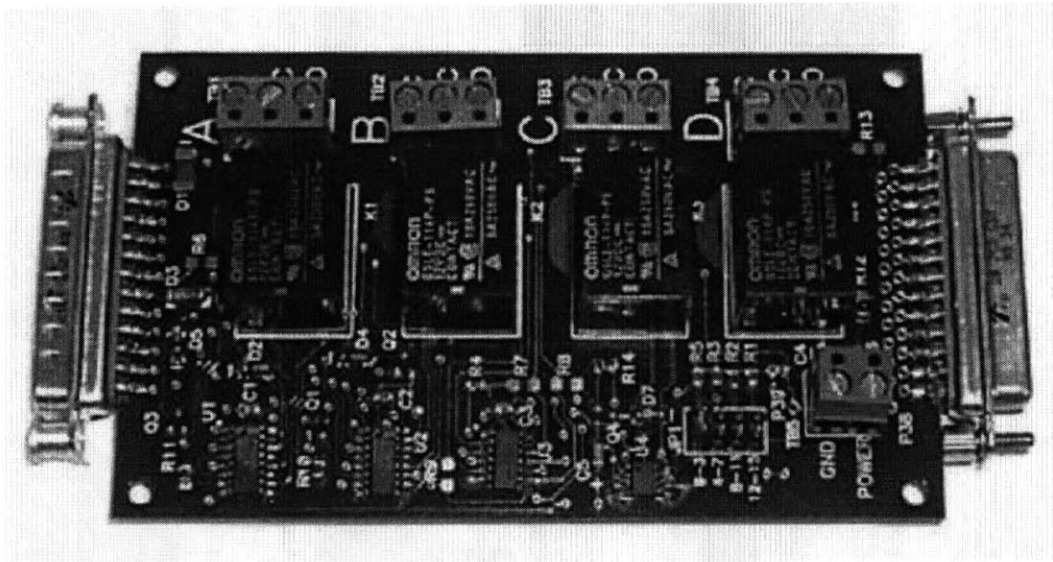


Figure 4-2: The modular board used to switch on and off the power module.

This modular board is connected to the pc and to the power module respectively via a parallel cable and a power plug. The goal is to be able to switch on and off the power module from the pc and the modular board plays the role of a remote interrupter. To start, we used only one relay to control the power module but we will be able to connect other device to this board with the other relays. To control the relay, we have basically to send some voltage through the parallel port of the pc and when choosing particular pins, we can activate the board.

The following table shows the channel each relay (A, B, C, or D) connects to

Table 4.1: Pin values for the modular board

JP1	A		B		C		D	
Position	Channel	Pin	Channel	Pin	Channel	Pin	Channel	Pin
0-3	0	9	1	10	2	11	3	12
4-7	4	13	5	25	6	24	7	23
8-11	8	22	9	21	10	19	11	18
12-15	12	17	13	16	14	15	15	14

depending on the position of JP1.

More information on parallel port programming can be found on Tomi Engdahl's[13] tutorial and a lot of details are available on Jan Axelson's[14] web page.

In our case, we only used the relay A and we had only to send data to PIN#9:

- to set PIN#9 to logic 1 and send the value: 128 to the parallel port: 2^9 .
- to set all the PINs to logic 0 and send 0 the parallel port of the PC.

As an Operating System, we have been using Microsoft Windows 2000 Server, which imposes strict security on hardware communication from users programs. In other words, direct communications are not possible as they are under Windows 98 or Windows 95. One solution we found is to use a driver, simple kernel mode driver for Windows NT/2000/XP that gives programs access to I/O ports. We used UserPort from Tomas Franzon, updated 5/31/01 and which is inspired by the article "Direct Port I/O and Windows NT" by Dale Roberts, which describes a way to defeat NT's security for port I/O.

The main idea of drivers is the following: Under Windows NT, there are only two I/O privilege levels used, level 0 and level 3. Usermode programs will run in privilege level 3, while device drivers and the kernel will run in privilege level 0, commonly referred to as ring 0. This allows the trusted operating system and drivers running in kernel mode to access the ports, while preventing less trusted usermode processes from touching the I/O ports and causing conflicts. All usermode programs should talk to a device driver which arbitrates access.

Once the driver is installed on the machine, we can program some C code to communicate with the parallel port, we provide here a simple example:

```
//this example sends a value to the parallel port
//and reads the value of the parallel port.
```

```
#include <windows.h>
static BOOL bPrivException = FALSE;

void outport(UINT portid, UINT value)
{
    __asm mov edx,portid;
    __asm mov eax,value;
    __asm out dx,ax;
}

void outportb(UINT portid, BYTE value)
{
    __asm mov edx,portid
    __asm mov al,value
    __asm out dx,al
}

BYTE inportb(UINT portid)
{
    unsigned char value;

    __asm mov edx,portid
    __asm in al,dx
    __asm mov value,al
    return value;
}
```

```

UINT inport(UINT portid)
{
    int value=0;
    __asm mov edx,portid
    __asm in ax,dx
    __asm mov value,eax
    return value;
}

void main(void)
{
    BYTE value;

    outport(0x378, 3);
    value = inport(0x378);
    printf("\nvalue is %d",value);

} //end main()

```

This C program sends the value 3 to the parallel port whose value is 0x378. Then, it checks what value was sent to the parallel port. Using our RMI architecture, we can link this C code to our existing Java programs.

4.3 The Management of the System

All the management of our system relies on some Java server programming, which is robust and reliable. If upgrades are necessary, we just have to modify the server programming by adapting our servlets or our RMI code by using some more functions.

In particular to deal with some important piece of hardware such as the table, we need to give access to the table to only one user at a time, if the users are among the users that successfully passed the first security steps we have described in the beginning of this chapter.

To overcome this problem, we have used some server programming. The main idea is like building a chat server: many chatters can come to the chat room and we have to keep track of them and give them a queuing number to get access to the shake table once the previous user is done with the table.

The RMI server contains a static Heap, which is an essentially complete binary tree. More information about Heaps can be found in Brassard and Bratley textbook [15]. A Heap is a structure that can easily be implemented through arrays and that can store numbers and sort them in quite a fast way: $O(\text{Lg}(n))$ (although, we know that we are not going to have that many users...)

When a user connects to the server, the server gives the new user a queuing number and store it in the Heap, keeping track of the user's IP address at the same time. When the current user of the table logs off, the Heap is updated by percolating. If a user decides not to queue, the Heap is updated by sifting. Thus, users are always tracked because of the Heap and because of the two RMI methods that tell the server that a new user enters the queue and that a particular user quits the queue.

Chapter 5

The Conclusion and the Future Work

5.1 The Issues

During the term I worked on the project, the team always tried to find the fastest ways to reach goals. Therefore, we did not always explore all possible solutions to reach our goals. In this chapter, we present the first conclusions and the future improvements or options in terms of software and hardware.

5.2 The Software Improvements

5.2.1 Upgrading the framework

We can consider two categories of functions to improve the existing framework:

- The functions that can be improved or added. Some more functions such as signal processing data could be added to our project. They could either be implemented in MATLAB and using our RMI framework, the server would run some MATLAB scripts in order to analyse the data and send the outputs to the user via a MATLAB client as described in Chapter 3. An other option is to program some more functions on the Client side in particular to add some

signal processing functions to our applet. The advantage of the second choice is that the computations would be done on the Client side and would not use the memory of the server. Moreover, each user would perform a customized analysis.

- The functions that do not exist: a database function. In order to give more data to users, we could use a database to store all the earthquake data. This database will be able to provide much more earthquake data than the current project does. Moreover, users should be able to add more data to increase the global database.

In terms of software, a database server could be added to the project. For instance, we could use MS-SQL Server 2000 or an Oracle database. Basically, through JDBC-ODBC bridges, one server could get database connectivity. Then, we could upgrade our Java Remote Invocation framework to make interaction with the database transparent to the end user so that each client does not need to add the data source and connect directly to the database. This system would avoid more security problems.

- Integrating the software simulations with the Shake Table experiments. An interesting point for educational purposes is to combine the simulations with the program that give output response from the Shake Table. Thus, users could be able to compare the validity of the model used in the simulations. We could also check the way we get the values from the table, namely to know if the filtering used by the data acquisition programs are correct.

5.2.2 Other possible options

In terms of Server, we have used Jakarta Tomcat server, which is one of the most reliable Java server. We could have used a Microsoft configuration:

- MS Server: IIS 5.0.
- C# as business logic language.

- ASP.NET for the Web Services.
- MS-SQL server 2000 as the database.

In terms of software, we could implement in the long run some other options developed in Chapter 3, in particular, using the C code generated by the compilation of the SIMULINK diagrams. This way, we would avoid using the third party software developed by Quanser and we would rely more on a complete program, which enables more functions and more flexibility.

5.3 Hardware Improvements

In terms of hardware, there are two important points that can be easily upgraded:

- The hardware architecture of our server. At the moment, we have been using one machine that does all the work. The configuration of this server should be upgraded in particular with a better processor and some more RAM. Moreover, the work should be divided into different machines with different tasks. One machine could be used to store the data and another machine could be used to access more efficiently the table. Using RMI would be helpful to divide the works between different programs.
- The hardware of the table. We could improve some device used on the table. For instance, we could use more accurate accelerometers and program them so that we could have an independent system to get the data from the table. It would be a good way to compare the results with the ones we have been getting so far.

Other device could be added to the table as suggested by Prof. Kausel in order to test more physical behavior such as wave propagation through liquids during an earthquakes.

Bibliography

- [1] S.J Dyke; S.M Johnson; R.T Ranf; and J.M Caicedo and M. Soto-Fournier. *Advancing Earthquake Engineering Education Through A Cooperative Effort Based On Instructional Shake Tables*. 7th US National Conference on Earthquake Engineering, EERI Boston, July 21-25,2002.
- [2] Irfan Baig. *Instructional Shake Table for Demonstration in Structural Dynamics*. Master Thesis, CEE, MIT, 2002.
- [3] *Documentation on the Java™ Media Framework APIs* Sun Microsystems, Inc. 11 Apr. 2002. <http://java.sun.com/products/java-media/jmf/>
- [4] *Documentation on the Extensible Markup Language (XML)W3C*, 2000. <http://www.w3.org/XML/>
- [5] Brett McLaughlin. *Java & XML*. O'Reilly, 2000.
- [6] Sanket Bakshi, *White paper about Accessing Native code from Java applets*, 2000. <http://www.ebsolutech.com/ebsolute/technologies/whitepapers/app2jni/index.html>
- [7] *Ten steps to sign an Applet*, Sun Microsystem Forum. <http://forum.java.sun.com/thread.jsp?forum=&63thread=132769>
- [8] *Sun Microsystem tutorial about applets*, Sun Microsystem, Inc. <http://java.sun.com/docs/books/tutorial/security1.2/tour1/index.html>
- [9] *Java Web Start documentation*, Sun Microsystem, Inc. <http://java.sun.com/j2se/1.4/docs/guide/jws/index.html>

- [10] *Online Documentation on the Jakarta Tomcat Server*, Apache Software Foundation . <http://jakarta.apache.org/tomcat/>
- [11] James Goodwill *Using Tomcat 4 Security Realms*, O'Reilly on Java.com journal. <http://www.onjava.com/pub/a/onjava/2001/07/24/tomcat.html>
- [12] John B. Smith *Threaded Self Loading Server Servlet*, Advanced WWW Programming, Department of Computer Science University of North Carolina. http://www.cs.unc.edu/Courses/comp118/docs/lessons-java/java_servlets/threadedserver.html
- [13] Tomi Engdahl *Tomi Engdahl's online tutorial about parallel programming*. http://www.hut.fi/Misc/Electronics/circuits/parallel_output.html
- [14] Jan Axelson *Jan Axelson's webpage*. <http://www.lvr.com/parport.htm>
- [15] Gilles Brassard, Paul Bratley. *Fundamentals on Algorithmics*. Prentice Hall, 1996.
- [16] Austin, Calvin; Pawlan, Monica. *Advanced Programming for the Java 2 Platform: Chapter 5: Java Native Interface (JNI) Technology*. Addison Wesley Professional, 2000.
- [17] Stearns, Beth. *Trail: Java Native Interface* Sun Microsystems, Inc. 11 Apr. 2002. <http://java.sun.com/docs/books/tutorial/native1.1/stepbystep/intex.html>
- [18] Epp, Diana M. *Interfacing C with Java in Linux – The JNI Solution*. C-Scene #4, 1997. 11 Apr. 2002. <http://cscene.org/CS4/CS4-04.html>
- [19] *Java Native Interface Specification*. Sun Microsystems, Inc. 16 May 1997. 23 Apr. 2002. <http://java.sun.com/products/jdk/1.2/docs/guide/jni/spec/jniTOC.doc.html>
- [20] *Matlab Manual: External Interfaces: Chapter 4: Calling Matlab from C and Fortran Programs*. Version 6. The Mathworks, Inc. 11 Apr. 2002.

<http://www.mathworks.com/access/helpdesk/help/techdoc/matlab-external/ch06engi.shtml>

- [21] Mueller, Stefan. *JMatLink Library*. 16 Aug. 2001. 29 Apr. 2002.
<http://www.held-mueller.de/JMatLink/>
- [22] *Java Remote Method Invocation*. Sun Microsystems, Inc. 21 Jun 2002.
<http://java.sun.com/j2se/1.4/docs/guide/rmi/>
- [23] Tom Valesky's RMI tutorial.
<http://patriot.net/~tvalesky/easyrmi.html>
- [24] Kenneth Baclawski's RMI tutorial.
http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html
- [25] Hanselman, Littlefield. *Mastering MATLAB 6, A comprehensive Tutorial and Reference: Chapter 35: Extending Matlab with Java*. Prentice Hall, 2001.