A Web-Controllable Shaking Table for Remote Structural Testing Under Seismic Loading

by

Mazen Manasseh

Bachelor of Engineering, Civil Engineering, 2002
American University of Beirut

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

June 2004

Signature of Author ....
Department of Civil and Environmental Engineering
May 11, 2004

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

Accepted by ... .
Heidi Nepf
Chairman, Departmental Committee on Graduate Students

A Web-Controllable Shaking Table for Remote Structural Testing Under Seismic Loading

by

Mazen Manasseh

Submitted to the Department of Civil and Environmental Engineering on May 11, 2004 in partial fulfillment of the requirements for the degree of Master of Science in Civil and Environmental Engineering

ABSTRACT

The thesis presents a remotely accessible system for controlling a shaker table laboratory experiment. The Shake Table WebLab is implemented at MIT's Civil Engineering Department under the Microsoft-sponsored iLab initiative for the development of educationally-oriented virtual experiments. Facilitated accessibility, safe operation and expandability are essentials at the root of the design and implementation of the Shake Table WebLab.

The fully functional system allows students and researchers to excite a two-story structure, which is three feet high, by vibrating its base while receiving accelerometer readings from its three levels. Registered Internet users may upload their own input data, such as the seismic ground acceleration of a newly occurring earthquake, and therefore study the corresponding behavior of a real structure. The system is designed with an expandable architecture which enables future researchers to add functionalities that suit their fields of interest. Relevant fields of study include real-time signal processing and filtering techniques that would provide an understanding of how earthquakes affect a structure and therefore provide insight on means to minimize encountered damage in large-scale structures. An already developed tool utilizes frequency domain transfer functions to compare the measured structural response at the upper levels with a predictable result based on seismic vibrations applied at the structure's base.

Two main characteristics of the web-based application are interactivity, provided through synchronized control/response processes, and sensor-based monitoring of the experiment. The system is built on the Microsoft .Net Framework through server-hosted Active Server Pages and browser-embedded Windows Form Controls. Web Service methods are implemented for initiating remote processes. Throughout the thesis, I state the motivations for conducting this project, the different online activities and generic administrative features, and a description of the implemented technologies and system components.

Thesis Supervisor: Prof. Kevin Amaratunga
Title: Associate Professor of Civil and Environmental Engineering

ACKNOWLEDGEMENTS

This thesis is the result of my continuous research throughout the period I have been a Master's student at MIT. The outcome of my work, presented in the thesis, has only been possible due to the guidance and support from a number of MIT faculty, colleagues and business affiliations.

I would like to thank my advisor, Prof. Kevin Amaratunga who has never been short on providing me with inspirational guidance and unlimited support. His consultation has always been a source of enlightenment and motivation. Prof. Eduardo Kausel has also been dedicated to my project through his commitment to tackle the smallest details and provide valuable ideas. His wise approach helped me learn a lot throughout the course of this project. I would also like to thank Prof. Ruaidhri O'Connor for his availability during critical times by helping us resolve hardware difficulties. Thanks to all other faculty who have expressed there admiration of the project.

Microsoft Corporation has provided the necessary funds for the research through Project I-Campus as an alliance with MIT. Quanser Consulting Inc. has also been very cooperative in sorting out technical problems. Thanks to the I-Lab team for their continuous interest in and appreciation of the Shake Table WebLab.

I am thankful to all the assistance that I received from my fellow colleagues in Information Technology in Civil and Environmental Engineering. My final and greatest gratitude is towards my family – my father George, mother Rita, brother Christian and sister Grace – for being a boundless source of affection and strength.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 General Benefits of Remotely Accessible Labs

A recently evolving trend in engineering education is focused on replacing traditional means of experimentation with virtual laboratories which are remotely accessible through the web. Such a shift from the physical laboratory environment to an online experimentation space is sometimes accused of detaching students from an irreplaceable real-world experience gained through physical interaction with a laboratory setup (Khan, 2002). For instance, a laboratory presence is valuable for conducting experiments on reinforced concrete members or soil samples as understanding material behavior is crucial to the observer. For this reason, the primary intention of a virtual experiment shall be driven by a tendency to improve the quality of learning. In some cases, this aim may require encouraging students to familiarize themselves with the experiment through a virtual online version before a more fulfilling laboratory session (Powell et al, 2002). Based on the type of experiment and the equipment involved, a virtually-conducted experiment may, however, prove to be more rewarding from the perspectives of both student learning and the logistics of laboratory resources.

A remotely controllable laboratory as a replacement to a traditional one has valuable benefits in terms of advocating new trends in education and a more efficient management of laboratories. Online laboratories are key to distant learning programs (Waner & Tuttas, 2002). Moreover, online experimentation provides a means to tighten lab-to-lab collaboration among universities and research centers. As a result, research and student groups have access to a wider collection of experiments by accessing resources in geographically distant locations. Concerning the benefits of virtual labs in managing resources of a laboratory, a reduced cost is associated with sharing facilities among different educational departments. A single experimentation unit supported by a remotely accessible system, which caters for multiple user accessibility, is a more cost effective solution than having multiple experimentation units or several scheduled lab sessions conducted by an appointed assistant. Specialized laboratories are usually utilized 25% of the school week (Powell at al, 2002). Consequently, reducing the space occupied by laboratories and therefore the needed effort for maintaining equipment lead to a reduction in the overall costs as virtual experimentation becomes more frequently adopted.

1.2 Types of Web-Controllable Experiments

Virtual experiments, as described above, are associated with real laboratory equipment that is controlled remotely. While simulations are artificial, virtual experiments incorporate physical components that provide a sense of reality (Wagner & Tuttas, 2001). In proceeding efforts of the iLab[1] group to provide a shared system architecture that facilitates the implementation of common requirements for developing virtual laboratories, experiments have been categorized into three groups based on the specific nature of the experiment. This approach for categorization is reliant on the kind of interactivity involved between the user and the laboratory. The first type of virtual experiments is defined as 'batched'. Such an experiment expects the user to set values for input parameters at once before the experiment is started. After the experiment request has been processed, the user is provided with values for different output parameters. It is in this manner, through which input and output parameters are each collectively transferred between a client and the laboratory server, that the naming of the category is justified. The Microelectronics Weblab (http://weblab.mit.edu), for experimenting with semiconductor devices, is an example of a virtual laboratory based on a batched experiment. The second category is termed 'interactive' and involves more interaction between the client and the laboratory server as various input parameters can be changed while the experiment proceeds. The Heat Exchanger Project (http://heatex.mit.edu) for experimenting with concepts in thermodynamics is representative of an interactive

[1] The iLab is an interdepartmental research group at MIT aimed at developing virtual experiments.

experiment. The third category includes experiments that focus on streaming sensor data. This category requires intensive data transfer from a server to a client whereby the delivered data is collected in real-time by different sensors. The flagpole project (http://flagpole.mit.edu), which is considered a preparatory step towards the monitoring of physical infrastructure (Amaratunga & Sudarshan, 2002), serves as an instance of a sensor data streaming experiment.

1.3 Objective of the Shake Table WebLab

Providing accessibility to a shaker table apparatus through the Internet has been advocated by academic and research interests to facilitate efforts of understating vibratory effects on a structure. One major field of interest is the study of the vibration modes of structures under earthquakes.

Facilitated and secure accessibility are essentials at the root of the design and implementation of the Shake Table WebLab. The Internet serves as an easily accessible medium for students to perform their experiments. To illustrate, were students to conduct their experiments in the lab, they would need to devote precious time to get acquainted with the exacting procedures needed to operate the hardware components in a safe and error-free manner. By contrast, a web interface that is well-adapted to the specific needs of the users eliminates the need to spend course time instructing students on the proper operation of the test apparatus. Add to that the vulnerability of the shaker table itself when subject to loading above the safe operating limits or without proper calibration. Alternatively, through the Shake Table WebLab students are able to directly explore their ideas without being concerned about the technicalities of interfering directly with the laboratory setup. The web application is configured so as to eliminate possibilities of human error in activating the shaker table. Such errors may lead to operation under unsafe conditions and therefore jeopardize the lab setup and anyone near it. These necessary precautions are carried out programmatically through accessibility processes that automatically execute whenever a web user conducts an experiment. As to the shaker being reachable by researchers outside MIT, web accessibility enables, for example, a research group in Japan, interested in understanding the nature of a new earthquake, to upload the corresponding seismic content and compare the behavior of a mounted structure due to a new earthquake with that of another.

In addition, a remotely accessible shaker table has valuable outcome in the classroom as instructors convey concepts in structural dynamics. Accompanying class lessons with experimental demonstrations enhances traditional means of conveying course material, and it adds interest to a class in structural dynamics as students are then able to observe in real-time the illustration of theoretical concepts. An advantage to such an approach is being able to compare experimental results with those obtained from simulations. Eventually, in-class demonstration allows experimentation to proceed in parallel to lecturing without the need to arrange for dedicated lab sessions.

1.4 Thesis Outline

In this thesis, I present the Shake Table WebLab (http://flagpole.mit.edu:8000/shaketable), one of a group of iLab projects in progress at MIT under the sponsorship of Project iCampus: MIT-Microsoft Alliance. As a successor to the Flagpole Project (a sensor lab for the monitoring of wind loading on a 31m tall flagpole), the Shake Table WebLab presents an online laboratory that is both interactive and heavily reliant on accelerometer sensor readings. Through the Shake Table WebLab students can excite a two-story 3ft tall structure by vibrating its base while simultaneously monitoring its behavior as they receive accelerometer readings from three different levels. The Shake Table WebLab allows any web user to upload his/her own input data, such as the ground accelerations of a newly occurring earthquake, and therefore study the corresponding behavior of a real structure. For instance, an interest to structural engineering students is to observe the response to the applied loading as it is transferred to a higher level of the structure. Throughout this thesis, I state the motivations for conducting the project, the different online available activities and the various system components as well as special administrative features that have been implemented.

Chapter 2

HARDWARE COMPONENTS & INITIAL SYSTEM CONDITIONS

2.1 Hardware Components

The hardware setup used for the Shake Table WebLab is a product of Quanser Consulting Inc. The uniaxial servo-controlled electromechanical earthquake simulation system consists of the following components (see Figure 1): a shake table, a power module equipped with a microcontroller-based safety circuit to drive the table, a data acquisition board (MultiQ) to drive the power amplifier and collect sensor responses, and a two-floor test structure. The shake table consists of a 1 Hp brushless servo motor driving a ½" lead screw. An 18" x 18" linearly moving stage is coupled to a circulating ball nut driven by the lead screw. The table slides on low friction linear ball bearings on two shafts. Table 1 summarizes the characteristics of the shake table and the graph in Figure 2 plots the acceleration and amplitude limits against varying frequency. Proprietary is fin.

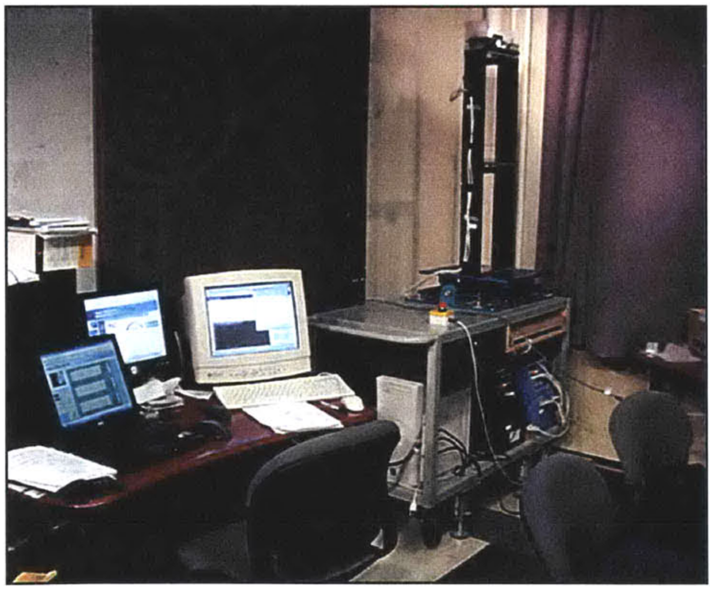


Figure 1. Shake table test structure and hardware assembly on the right. Serve computers on the left.

Table 1. Shake Table Parametric Characteristics

Parameter	Value
Table Dimensions	18 x 18 inches
Maximum Payload	33 Lbs
Operational Bandwidth	20 Hz
Peak Velocity	33 inches/sec.
Ball Screw Efficiency	90%
Maximum Force	700 N
Peak Acceleration	2.5 g
Stroke	+/- 3 inches
Weight	60 Lbs
Encoder/Lead Screw Resolution	0.000125 inches
Motor Maximum Torque	1.65 Nm
Linear Bearing Load Carrying Capacity	290 Lbs

Source: Quanser Consulting Inc. Shaker Table

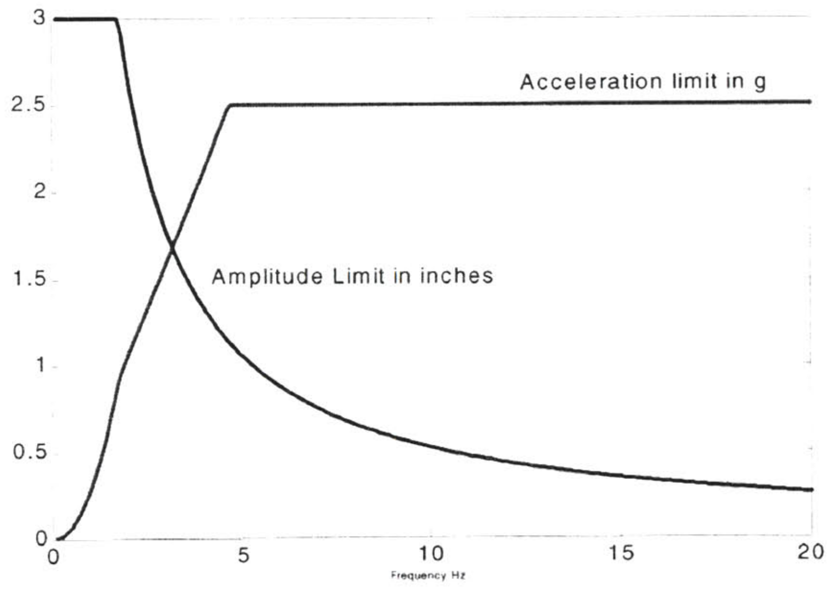


Figure 2. Acceleration and Amplitude Limits vs. Frequency

The assembled three-foot tall structure carries one accelerometer on each floor (Figure 3). A third accelerometer is attached to the moving base of the table. In addition, there are three position detection sensors embedded under the moving plate of the table. Located in the center, far left and far right positions (see Figure 4), those sensors serve to determine the location of the table relative to its allowable displacement limits.

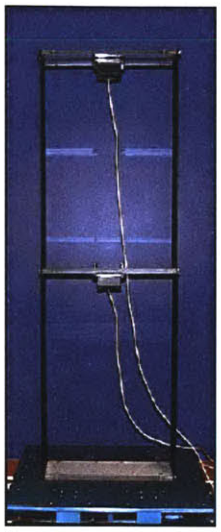


Figure 3. Two-level test structure

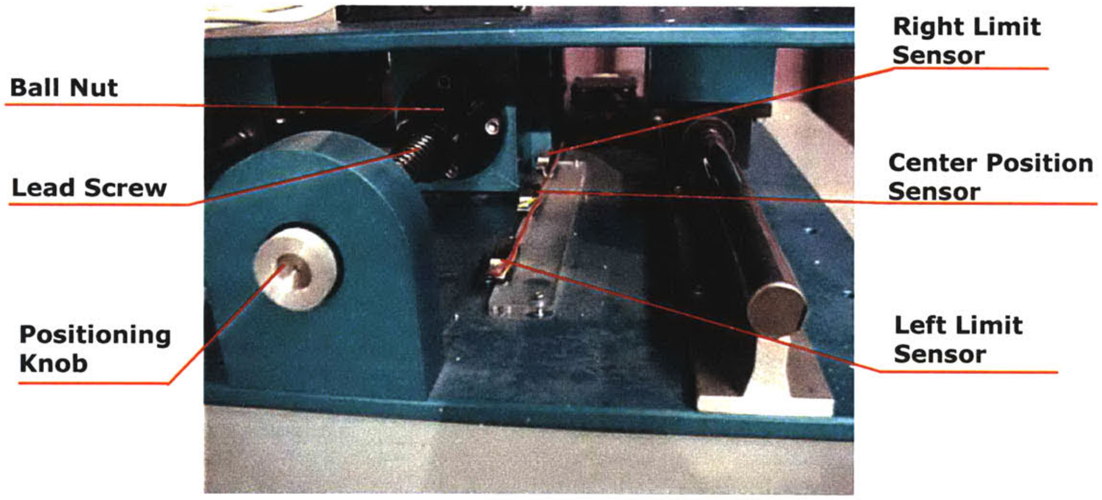


Figure 4. Close-up of the shaker table.

2.2 Initial Software Architecture

The rest of this chapter deals with the software application, namely WinCon 3.2, which is developed by Quanser Consulting and enables computerized control over the shaker table. Following is a description of WinCon's features and capabilities as initially intended to be used for a variety of implementations. However, since the aim of the project transcends WinCon's functionalities, an expanded architecture that describes the new system is presented in Chapter 3. Accordingly, the description of WinCon below only serves to provide an understanding of the software on which the newly implemented system design relies. Since the new system is built on top of WinCon, it is essential to present WinCon's intrinsic features first and then introduce the new architecture in the next chapter.

2.2.1 WinCon Software

WinCon is a Windows 95/98/NT application which consists of two components: WinCon Client and WinCon Server. WinCon Client is installed on a computer that is connected to the controller/data acquisition (MultiQ) board and therefore is responsible for driving the table and conveying sensor data. WinCon Server is another desktop application that entails user interface controls and display charts and would act to serve up the table with different experiments. WinCon Server communicates with a WinCon Client by establishing a TCP/IP socket connection through a well-defined port on the WinCon Client side. To illustrate, the server needs to be configured so as to connect to a TCP port exposed by the client. Various client-server scenarios are supported whereby a WinCon Server may connect to several clients and a WinCon Client may communicate with several servers. Hosting both the client and server applications on the same computer is also a possible implementation (Quanser Consulting Inc.).

WinCon Client

WinCon Client is a real-time component that executes special type files, namely WinCon Controller Library (.wcl) type file. Operation requires that WinCon Server transfer the controller file to the client where it is executed as the connection with one or more servers is maintained. WinCon Servers connected to the client receive real-time data as the experiment proceeds. Another means of running experiments locally on the client computer connected to the shaker table is possible by loading the controller file directly in WinCon Client. In the latter case, the user is not able to view any outputs from the system.

WinCon executable files are compiled from code generated out of Simulink models[2]. Simulink is a diagrammatic programming tool whereby a model is defined by a set of function-specific blocks and interconnecting links. Accordingly, the basis of any experiment executed in WinCon is a Simulink diagram. Refer to Appendix A for a sample Simulink diagram that loads its inputs (typically previously scaled seismic accelerations and system constants) from the Matlab workspace. Simulink blocks such as '*Shaker Table*', '*q_Clock*' and '*Stop Run*' are provided by Quanser for carrying out WinCon-specific operations. The '*Enable Shaker Amp*' block, for example, is required to activate the shaker table.

Being a real-time software that requires accurate time synchronization between the MultiQ board clock and the Windows operating system clock, WinCon Client imposes the need for VenturCom's Real Time Extension (RTX). The role of RTX is to enable windows to handle control-oriented and high-performance operations (VenturCom Inc., 2000).

WinCon Server

WinCon Server has several functionalities that enable users to change client parameters, upload experiments to be executed on the client, and change experiment input parameters in real-time. After downloading a WinCon controller library file to the client, WinCon Server is then capable of starting, manipulating input signals to and stopping the client. Moreover, WinCon server (Figure 5) is capable of plotting and saving data streams acquired from a specified WinCon Client. After customizing the

[2] Simulink is a Matlab-based package for modeling, simulating and analyzing dynamic systems the outputs of which change over time (The MathWorks, 2002).

input/output settings associated with a loaded wcl file, a user may choose to save his preferences as a WinCon Project File (.wcp) for future use.

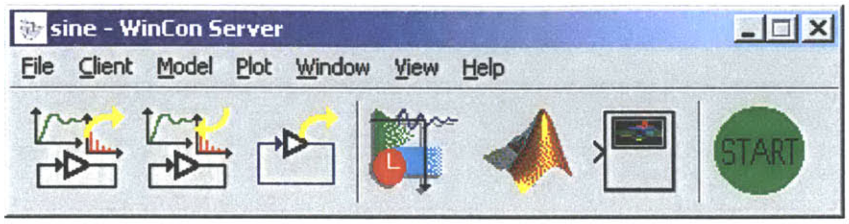
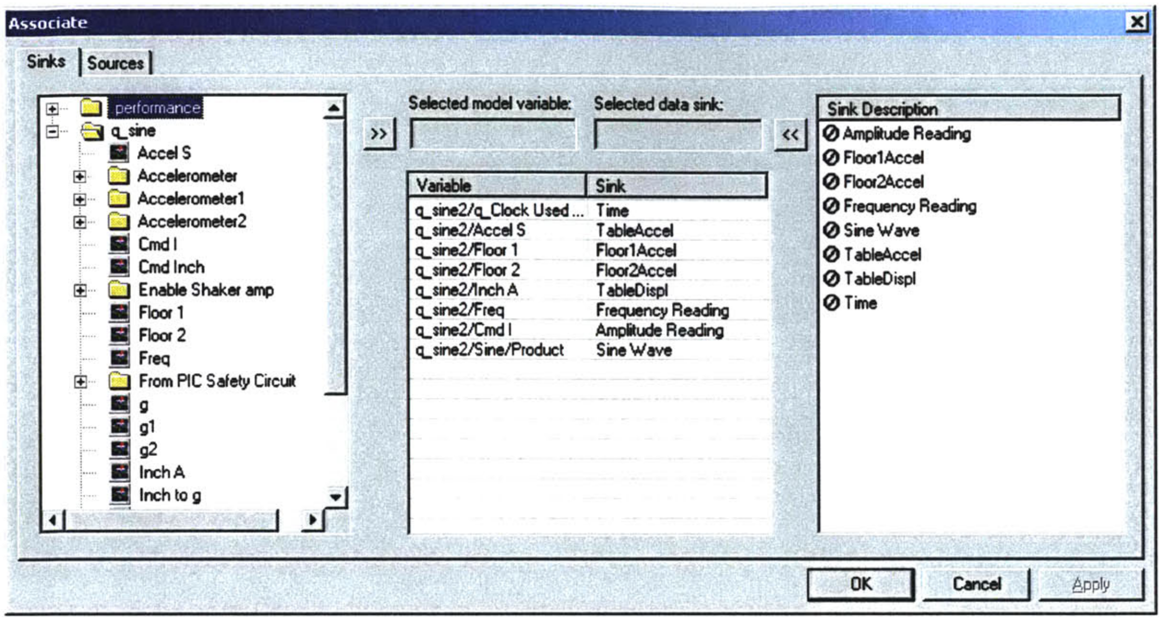


Figure 5. WinCon Server's control panel.

A particularly distinctive feature of WinCon Server is its External Interface Window (EIW) which supports integration with independent applications. Accordingly, special-purpose programs may be developed and used to alter input parameters and process output data streams. An external application communicates with the server's EIW through a dedicated TCP/IP socket connection identified by a designated port on the machine hosting WinCon Server. Figure 6 is a snapshot of the EIW's parameter association form. Left column fields are input/output scopes in the corresponding Simulink model of the WinCon project. The right column fields are the sink/source names sent from a web-client application when it first establishes a TCP connection with WinCon. Accordingly, this form serves to associated Simulink's sinks and sources with corresponding parameter names declared in an external application. More details as to the methodology of streaming data are provided in section 5.4.

2.2.2 Initial Principle of Operation and Limitations

Building the *Shake Table WebLab* with a flexible yet stable architecture has been one of the main concerns in designing the new system. A significant effort has been dedicated towards safe and facilitated operation. This section describes the processes that were required to be manually performed in order to conduct a sample experiment using the WinCon software. The following description provides a clarification of the initial processes that have been simplified later on with the new system (described in chapter 3) thus providing seamless user interaction with the server.



performance
q_sine
- Accel S
- Accelerometer
- Accelerometer1
- Accelerometer2
- Cmd I
- Cmd Inch
- Enable Shaker amp
- Floor 1
- Floor 2
- Freq
- From PIC Safety Circuit
- g
- g1
- g2
- Inch A
- Inch to g

Variable	Sink
q_sine2/q_Clock Used ...	Time
q_sine2/Accel S	TableAccel
q_sine2/Floor 1	Floor1Accel
q_sine2/Floor 2	Floor2Accel
q_sine2/Inch A	TableDispl
q_sine2/Freq	Frequency Reading
q_sine2/Cmd I	Amplitude Reading
q_sine2/Sine/Product	Sine Wave

Sink Description
- Amplitude Reading
- Floor1Accel
- Floor2Accel
- Frequency Reading
- Sine Wave
- TableAccel
- TableDispl
- Time

Figure 6. WinCon Server's sinks and sources association form.

Using the originally provided Simulink models to apply a new vibration – defined by file-stored accelerations - to the table is achievable through three steps: First, the Simulink model that reads input from the Matlab Workspace is opened with Simulink and the corresponding WinCon constants are loaded from a data file. Second, the new acceleration data is loaded and a scaling algorithm is applied to generated scaled-down ground displacements that are within the displacement limits of the shaker table. The input/output parameters for the scaling function are described as:

$$[Tc, Xc, Ac, Te] = q\_scale(t, a, x_{max}) \quad \ldots\ldots\ldots\ldots\ldots [5]$$

Input parameters:

t: array of time at equal sampling intervals

a: array of accelerations in g matching t

x_{max}: maximum displacement of the table from the center position

Output parameters:

Tc: command time array

Xc: position command array applied to the table in cm

Ac: acceleration array in g

Te: numeric value for the duration of the experiment in seconds

Applying the scaling function to seismic accelerations results in displacements (**Xc**) proportional to real ground displacements with a maximum equal to $\mathbf{x}_{\mathbf{max}}$ but with accelerations defined in **Ac** identical to the real accelerations in **a**. Third, when all required input parameters to the model are loaded into the workspace, the Simulink model is compiled. Matlab uses a C-compiler to generate intermediary code files and finally produce the WinCon Controller Library (.wcl) which is ready to execute in WinCon Client.

Though WinCon supports a client/server setting which allows carrying out experiments from outside the lab, relying on WinCon as the only means to establishing a remote connection to the laboratory setup does not exploit the full functionalities of the shake table as discussed in the next chapter. Furthermore, each WinCon experiment – in the form of a Simulink model – is static as to its input data. For instance, the Simulink model that runs an earthquake reads its ground accelerations from the Matlab workspace resulting in a compiled controller (.wcl) file that only serves to execute that particular earthquake. As a result, each stored earthquake has its own executable with the corresponding seismic data embedded within. Accordingly, the initial models necessitated as many executable (.wcl) files as there were earthquakes - though all the files would entail the same execution mechanism. The disadvantage of this principle is in executing new experiments for newly occurring earthquakes as that would require repeating the whole aforementioned compilation process. It is in this sense, that extending original operation to become more flexible (as new experiments can be seamlessly supported) and easily accessible (from a commonly used Internet browser) have been two major requirements in extending functionalities of the shake table lab.

Chapter 3

SYSTEM ARCHITECTURE

3.1 Problem Identification and Formulation of Adopted Solution

Before delving into the details of the implemented system architecture, I first reiterate the problem that the project aims at targeting. After providing a high-level description of the situation, various proposed alternatives are discussed as possible solutions and the adopted system is finally justified.

As presented earlier in Chapter 1, the project's aim is to provide web accessibility to a laboratory experiment in an attempt to allow interested groups to conduct experiments remotely. To achieve that goal, we need to understand the primary system conditions, chose a development platform that enables extending the original system, appropriately distribute software components among client and server computers and provide a safe and well-organized accessibility mechanism. For that purpose, I discussed earlier the initial state of the system as it operates prior to introducing any modification. In brief, the hardware components (driving motor and sensor devices) are accessible through Quanser's WinCon proprietary software which provides a means for remote accessibility, yet it does not address all the requirements of an easily accessible and extensible system. Accordingly, presented below are three alternatives which define distinct approaches towards an enhanced architecture:

1. *Relying on WinCon's client/server implementation to allow users to access the experiment remotely*: This approach entails implementing WinCon as the only software for conducting experiments. This is possible through WinCon's client/server connectivity by installing WinCon Client on the computer connected

to the shaker table while WinCon Server is installed on every user-computer to serve up experimental data to the laboratory computer (see Figure 7). While such an approach requires minimal effort in terms of software development, it, however, offers very little in terms of simplicity and manageability of user accessibility to the laboratory experiment. To illustrate, such an implementation is solely based on a TCP connection between user computers and the laboratory server. In addition to installing WinCon Server, users would need the whole Matlab/Simulink environment installed locally on their machines in order to compile new experiments as described in the 'Initial Principle of Operation' section of the previous chapter.

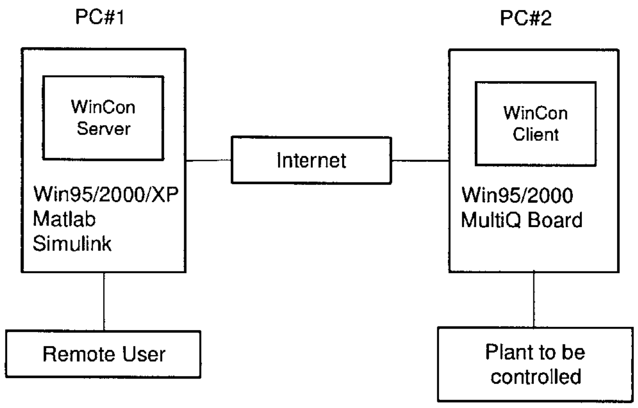


Figure 7. WinCon Client/Server implementation on separate computers.

2. *Replacing WinCon with a newly designed control program*: Bypassing WinCon as a means to establish direct access and control over the shaker table necessitates the development of an alternative program. Such a control program may be designed to intelligently handle multiple client requests and publish streaming sensor readings in real-time. The advantage of this option is that it provides total flexibility by incorporating all the favorable features early on in its design (such as communicating with a web-interface as a user-side application). However, totally disregarding WinCon's role as a user's mediator to the shaker table indicate additional challenges attributed to: 1) Understanding and reestablishing low-level programming needed to interface the new program with the microcontroller and data-acquisition devices, and 2) rebuilding several

components - such as booting-up the microcontroller and calibrating the table – which are already implemented by Quanser Consulting and are operational from within WinCon.

3. *Maintaining WinCon as a mediator between a newly developed web-based user application and the back-end laboratory setup*: The External Interface Window[3] (EIW) provided in WinCon Server may be used as a channel for communicating with additional applications that comply with its exposed protocol. Accordingly, while WinCon remains the only means of accessing the shaker table, a user web-based application may be supported. Though this approach would necessitate ensuring the reliability of WinCon and more precisely its EIW as a control program and data publishing server, the solution allows unrestricted extensibility of a client application that is easily accessible, user-friendly and upgradeable.

While the first option is least demanding in terms of development, it provides the least customization capabilities as adopting a previously developed application doesn't necessarily match newly defined requirements that have been imposed much later, for a different purpose and by a different group. On the other hand, the second option provides ultimate customization as all the system software is to be developed from scratch. However, designing a system from the ground up is time consuming especially that the details of the hardware interface are not well-specified. The third solution serves as a middle ground between the two extremes as it aims at maintaining the constructive aspects of WinCon, as an intermediary that activates the shaker and acquires sensor data streams, while the user interface is provided through a newly-built web application. Accordingly, the rest of this chapter reveals the system architecture and the technologies that have been implemented in delivering a solution based on the third concept.

3.2 Overview of the Software Architecture

The Shake Table WebLab is built on a multi-tier architecture that supports client-server interaction. A client is representative of an Internet user computer – for example a student

[3] For a description of the EIW, refer to the WinCon Server Section in Chapter 2.

accessing the shaker table website. The term server incorporates several computer systems located at the MIT Civil Engineering Labs that serve as intermediaries for establishing communication between a web client and the shake table controllable apparatus. Server-sided components are distributed among three machines with well-defined interconnections: a shake table control server, a web/database server, and a streaming video server. Distributing server tasks among the three machines is essential for maintaining acceptable system performance levels. Figure 8 is a depiction of the various system components, the software installed on each and the connecting channels that tie them together. For an overall understanding of the system architecture, I describe below the software technologies that have been used and elaborate on how they have been applied for the purpose of this system through an overview of the main roles of its components. The next chapters present an in-depth explanation of the functionalities provided by each component.

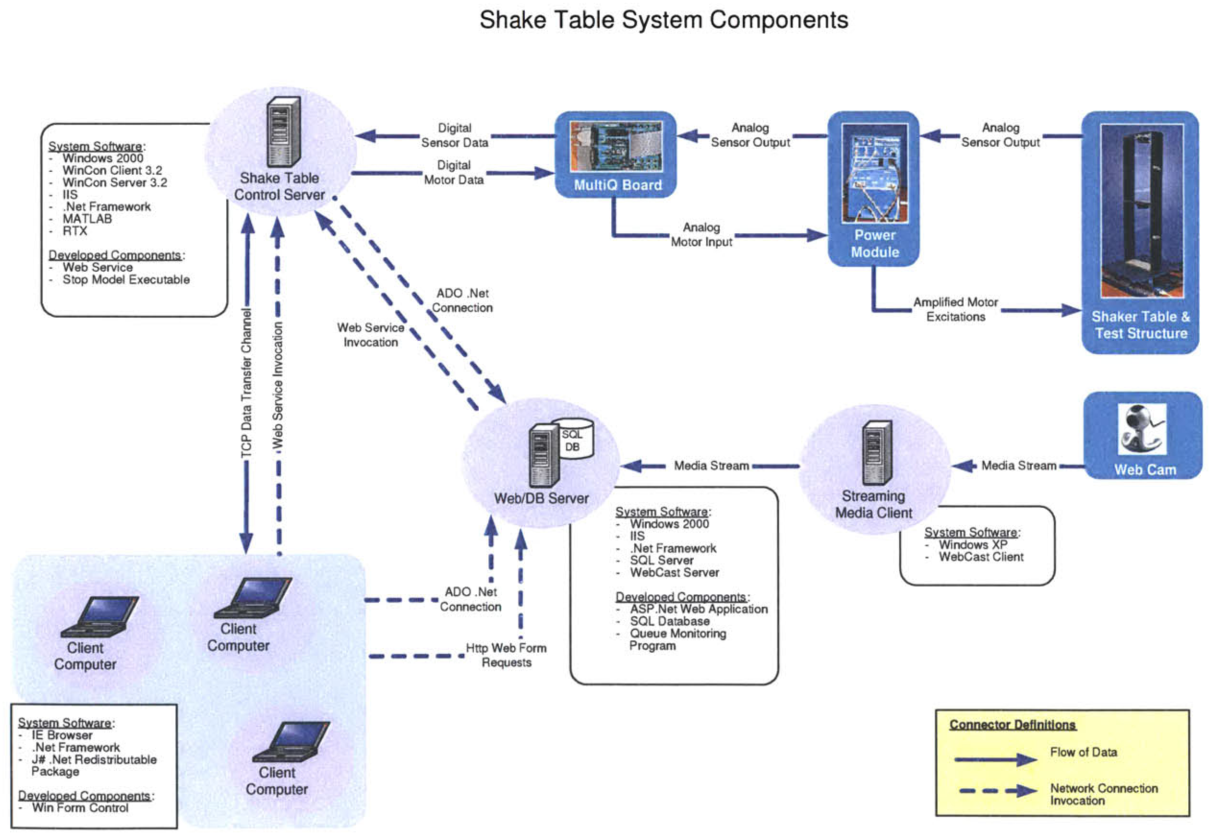


Figure 8. System architecture and the interrelationship among various hardware and software components.

3.3 Implemented Technologies

As illustrated in Figure 8, all system components include the Microsoft .Net Framework as a requirement for the developed software to execute. This section introduces the .Net Framework, its essentials and related applications. In addition, a justification for the selection of .Net as an adopted software platform for the Shake Table WebLab is discussed from the viewpoints of both its advantages and drawbacks.

3.3.1 The Microsoft .Net Framework and the CLR

The development and release of the .Net Framework by Microsoft Corporation was advocated by the need for a platform which primarily aims at software integration. To illustrate, software integration is meant both within the same operating system process and across the web. The Common Language Runtime (CLR) serves to provide integration at the same system level while XML-based Web Services serve to integrate software at an Internet scale. In a few words, "the CLR is a loader that brings components to life in an operating system process" (Box & Sells, 2003).

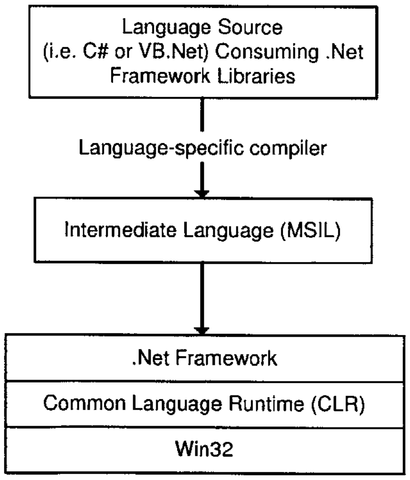

Language Source
(i.e. C# or VB.Net) Consuming .Net Framework Libraries
Language-specific compiler
Intermediate Language (MSIL)
.Net Framework
Common Language Runtime (CLR)
Win32

Figure 9. MS .Net Framework and the Common Intermediate Language.

As Windows NT 3.1 marked the end of the DOS era on July 9, 1993, the CLR release as part of the .Net Framework on February 13, 2002 marked the end of the Component

Object Model (COM) era. From a world were contracts among software components were merely defined by functional entry points, we later moved to COM, which defines contracts based on type definitions allowing the dynamic loading of code. Today the CLR serves to resolve problems in COM, mainly caused by the description of contracts between components. Unlike the COM object-model and Win32, the CLR functions on types that exist in a Common Intermediate Language (CIL) (see Figure 9). It is only until runtime that a type represented in the CIL is then converted to the native machine code by the Just-in-Time (JIT) compiler of the CLR. Consequently, CLR type definitions are logical – based on method names and signatures – rather than being physically represented by their memory addresses and offsets. Furthermore, as the CIL-to-native translation occurs on the deployment machine, the processor-specific layout rules will better match the processor architecture that the code executes on (Box & Sells, 2003).

Other characteristics of the CLR include: managed code execution, metadata specification in types, and Just-in-Time (JIT) type loading. The execution of managed code allows the CLR to be aware of various aspects of the running program such as the state of variables and the origins of stack frame code. An example of managed execution within the CLR is automatic garbage collection. It is true that with such features, programmers tend to be in less control of their program, yet their productivity increases when working at higher levels of abstraction as they become manipulators of types, objects and values instead of dealing with virtual memory and threads. As to type metadata, it serves to define the type contract which is essential for its translation from CIL to native code. More simply put, the metadata of a type exposes the names of the type methods with their associated signature (input/output parameters). The aim of the JIT compiler is to postpone type loading into memory until the program needs to access the code associated with those types. Consequently, not all parts of an assembly[4] are loaded into memory at the time of its execution as some parts are not loaded until being requested at runtime.

[4] An assembly is a collection of types.

3.3.2 Why use the .Net Framework?

Following the aforementioned briefing on the nature and features of the .Net Framework, two facts follow: First, since program components are compiled into the CLR-compliant Intermediate Language, this provides the advantage of writing programs in a particular language and reusing components written in other languages as the compilation of the written code results in assemblies in the common intermediate language. This is true of languages which have a CLR-compliant compiler such as the .Net built-in compilers for C#, VB.Net and C++. Second, since the compiled assemblies exist in a non-machine code format, the CLR engine is required on every system that is expected to execute programs compiled against the CLR.

Another framework to consider is based on the Java programming language and the Java Virtual Machine (JVM). Sun's Java 2 platform and the JVM have a lot in common with Microsoft's CLR. For example, both are based on types and execute managed code. However, there remains a lot to be said about the differences between the two platforms. Accordingly, determining which is better is only possible from the viewpoint of what is required of a new system implementation and which of these platforms better suit these requirements.

As mentioned earlier, two major objectives of the Shake Table WebLab are facilitated accessibility and feature extensibility. For this reason, I will focus on these two points as I present the advantages and disadvantages of both the Java and MS .Net platforms. Starting with the accessibility issue, the aim is to minimize effort on behalf of a user accessing the WebLab – this is more simply explained as users being asked to download and install minimal software. Though both platforms require a virtual engine to be installed – either the Java Runtime Environment or the .Net Framework Redistributable – yet we still need to investigate the types of supported operating systems. A current advantage of the Java alternative is that the JVM is platform independent as it supports Windows, Linux and Mac operating systems. On the other hand, the newly released CLR is only supported on a Windows-based system. Concerning feature extensibility, support for multiple languages is a considerate element. While the CLR is capable of supporting various languages (C#, VB, C++, J#) as previously explained, the JVM only supports the Java programming language.

As a result, while the JVM may be described as a platform independent environment, the .Net Framework is more of a language-independent engine. Another issue is the availability of valuable parts of code that have been previously written in Java. Consequently, it is essential to reuse those Java-based classes. Eventually, the .Net Framework was adopted as a common platform for the various system components. This decision allows future research groups, interested in applying their own programs (for instance adding advanced signal processing algorithms that run within the client application), to do so without worrying about the programming language they have to adhere to. Moreover, reusing previously-developed Java code has been possible through .Net's support for a J# language – a syntactically identical language to Java. As to supporting operating systems other than Windows, this is not currently possible, however, as the CLR becomes more mature, it is anticipated that initiatives will work towards extending the .Net platform to other operating systems. The ongoing Mono Project, for example, is one such initiative that aims at supporting the .Net environment on Unix-based operating systems.

3.3.3 ASP .Net

ASP.Net is a successor to the previous ASP Web programming model. While ASP provided a lot in server-sided processing of input HTML and dynamic generation of output HTML, ASP.Net offers much more in terms of performance levels, development efficiency, and code reusability. With ASP, server processes are based on script interpretation. On the contrary, ASP.Net handles HTML requests based on compiled code which avoids the need for recompilation on each request and therefore increases the server response time. In addition, generating Web forms in ASP.Net allows for an object-oriented approach to Web development. To illustrate, web forms consist of two essential files: an aspx file which contains the HTML-based design of the form and a code-behind file which may be written in any CLR-compliant language. Hence, the separation between the presentation (aspx) and server side processes (code-behind) of a web form is defined (Prosise, 2002).

3.3.4 Web Services

A Web service is another type of a web application. However, it is not intended to be used by end-users but rather to provide services to other applications, in this case an ASP.Net web application. Instead of having a user interface, a Web Service exposes its services through Web Methods defining an API (Application Program Interface) which other applications may consume. Unlike the Distributed Component Object Model (DCOM), Web services are based on XML SOAP messages as a means to transfer information across the web. For the purpose of the Shake Table WebLab, a Web service is hosted on the Control Server in order to trigger processes on that machine from a user's client application. The purpose is to gain control over WinCon which in turn activates the shaker and streams back sensor responses. Chapter 5 provides a description of the implemented Web Service and the use of its Web Methods.

3.3.5 Windows Form Controls

Despite the support for server-side code in ASP.Net, a real-time interactive experience could only be applied through the use of client-side program components. Accordingly, Windows Form Controls are implemented as client applications that load within a user's browser. Previously known as Windows ActiveX controls and similar to Java Applets, Win Form Controls are based on the .Net Framework and are identical to an ordinary Windows Form. However, the difference between the two is in the way that a form is launched. In the case of an ordinary Windows Form application, the program runs as a stand-alone executable. However, a Win Form Control is intended to be embedded within an HTML page and thus can only be accessed from within a web browser. Moreover, Win Form Controls run within the CLR and therefore require that the client machine: 1) have the .Net Framework installed, and 2) access the Shake Table WebLab from a CLR-enabled browser such as Internet Explorer.

As with the case of Java Script and ActiveX, Win Form Controls are associated with security concerns. This is true because a program that automatically loads itself and runs within a client computer makes its host environment susceptible to malicious execution if not loaded from a trusted source. Accordingly, these controls are operational only when granted permission by a particular user. The user would have to increase the security trust level associated with the shaker table website. Such security configurations are applied to both: the Internet browser and the .Net Framework.

Since a Wind Form Control executes within the CLR of the client's computer, it therefore functions independently of the ASP.Net application. However, for the purpose of the Shake Table WebLab, the control that loads within the browser has properties which are user-defined from within ASP pages. Accordingly, it is necessary to transform some predefined properties, such as the user's identity and the type of the selected experiment, from an asp page to its embedded form control. The following technique is used whereby customizable parameters of the control may be passed and defined when the control is first loaded:

- In the asp page the following object tag is used to locate the compiled library of the Win Form Control and pass parameters that are defined in the asp page to the control.

```
<OBJECT style='…' height='…' width='…' classid='server path to the control
      dll    file # class name' VIEWASTEXT>
<param name='first parameter name' value='value of first parameter'>
<param name='second parameter name' value='value of second parameter'>
<param name='third parameter name' value='value of third parameter'>
</OBJECT>
```

- In the control's main class, each parameter is declared with 'get' and 'set' properties in order to access the passed values as follows (C# syntax):

```
public string 'first parameter name'
{
      get
      {
            //Code to return a value
      }
      set
      {
            //Code to use the passed value
      }
}
```

3.4 Components of the Implemented System

3.4.1 Shake Table Control Server

Direct access to and control of the shaker table assembly is achieved through a dedicated server. The purpose of the control server is to translate client requests into server-side processes that gain control of the shaker. Accordingly, being connected to the MultiQ data acquisition board, this server is responsible for sending and acquiring data to and from the shaker table assembly. Digital data which the control server delivers to the MultiQ board consists of values of successive table displacements that correspond to a particular input signal. The MultiQ board would in turn provide the control server with data corresponding to:

1- *Sensor readings from three mounted accelerometers which reflect the behavior of the test structure.*
2- *Sensor readings from three position detectors located at the table's base.* (See Figure 4) These readings serve to inform the control server whether the table is properly positioned in order to avoid situations where excessive displacements cause the table to go beyond the boundary limits.

3.4.2 Web/Database Server

The second server computer has two main functionalities: a Web Server, namely Internet Information Services (IIS) hosting an ASP .Net application and a database server implemented in MS SQL. The web application serves as the access point for Internet users to control the shaker table. Managing accessibility, such as user authentication and queuing of multiple client requests, is a functionality of the Web Server. For this purpose, a database is implemented to store user accounts, previously loaded experiment records, queued clients, and other modifiable system parameters such as the frequency of the sensors' sampling rate and the status of the lab. In addition to users accessing the Web Server through their web browsers, the Control Server establishes a database connection – based on an ADO connection to SQL – mainly to access experiment-related information in the database.

3.4.3 Streaming Media Server

The web laboratory environment includes a live video display of the shaker table located in the Civil Engineering lab. Live video is broadcast through a streaming media server called WebCast. A web cam is connected to this server which in turn publishes the video stream to be viewable in the user's browser. The video display enables a user to identify the different modes of vibrations of the structure as it undergoes various dynamic excitations.

3.4.4 Clients

The online web application allows a real-time control of the shaker table while receiving streaming sensor data from the Control Server. Accordingly, communication starts through http messages between a client – through an IE browser – and both the Web Server and the Web Service hosted on the Control Server. Through an http transfer protocol the client application calls web methods that launch the necessary processes on the Control Server. Afterwards, the web application maintains an open channel for data transfer to and from the Control Server. For this reason, the web application incorporates a Win Form Control that establishes a TCP connection with WinCon's EIW. This control is embedded within an html page and therefore runs on the client's machine in Internet Explorer.

3.5 Data Model

A back-end database is implemented in SQL Server to manage the information associated with the following features: user accounts, uploaded experiments, queue status, activity logs, and other system variables. The database is not intended to store experiment information regarding individual experiments beyond the descriptive information that applies upon new experiment uploads. Accordingly, each experiment is represented once, in the database, regardless of the number of times that it is run. This approach is applied because the database does not handle experiment information such as input parameters and results. Such information which relates to user-specific instances of running an experiment are handled locally on the client's computer. Figure 10 presents the implemented data model. Following is a description of the major database entities:

Users: The users table stores user profiles. The "IsAdministrator" parameter defines whether the user is an administrator or not. The "UserAllowedDuration" parameter is the maximum time in minutes that a user is allowed for any experiment. A five minute value is set as default for accounts created by new users. However, this value can be changed by an administrator either upon creating a new user account from the Secure Area of the web application or by changing the profile of a previously registered user from the "User Account Management" page also listed under "Secure Area".

PreLoadedExp: This table holds records of experiments that acquire input from file. If an experiment is created by a regular user, "IsPublic" is set to false as the experiment is visible to that user only. Only administrator accounts may declare experiments as public. A one-to-many relationship exists between the "User" and "PreLoadedExp" entities as a user may have several experiments and each experiment shall be owned by a user designated by a unique "UserID".

Queue: The queue entity holds records of user requests starting at the time that a request to run an experiment is submitted until the experiment is stopped, either explicitly by the current user or an administrator, or automatically by the server when the allocated time limit is exceeded. Each request logged into "Queue" consists of the user's ID, a time limit, a most recent updated timestamp and a number which represents the user's turn in the queue. Records of the queue are managed dynamically by a monitoring server-hosted program as described in section 4.2.2. Accordingly, a queued item is either maintained or removed based on the status of the associated user. For instance, a user who is disconnected after submitting a request will be removed from the queue in order not to block accessibility of subsequent users.

The "LabStatus" field is configurable by an administrator and indicates whether the WebLab is available for regular user-access or not. The "Frequency" field is a system constant that specifies the rate at which sensor readings are taken. This rate depends on WinCon's configuration for acquiring sensor readings. Since the system is implemented with a time interval of 20 milliseconds for streaming data, "Frequency" is set to 50Hz. Accordingly, changing this predefined value requires reconfiguring WinCon project files to be compatible with the new interval.

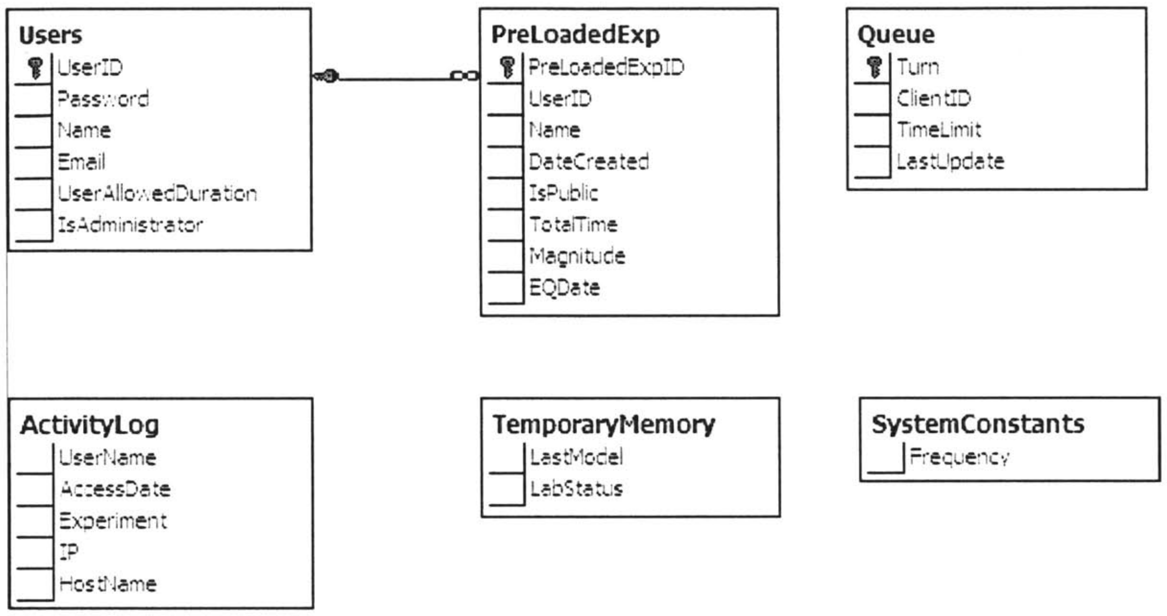


Figure 10. Data model diagram implemented in SQL Server.

Chapter 4

FUNCTIONALITIES OF THE WEB LAB

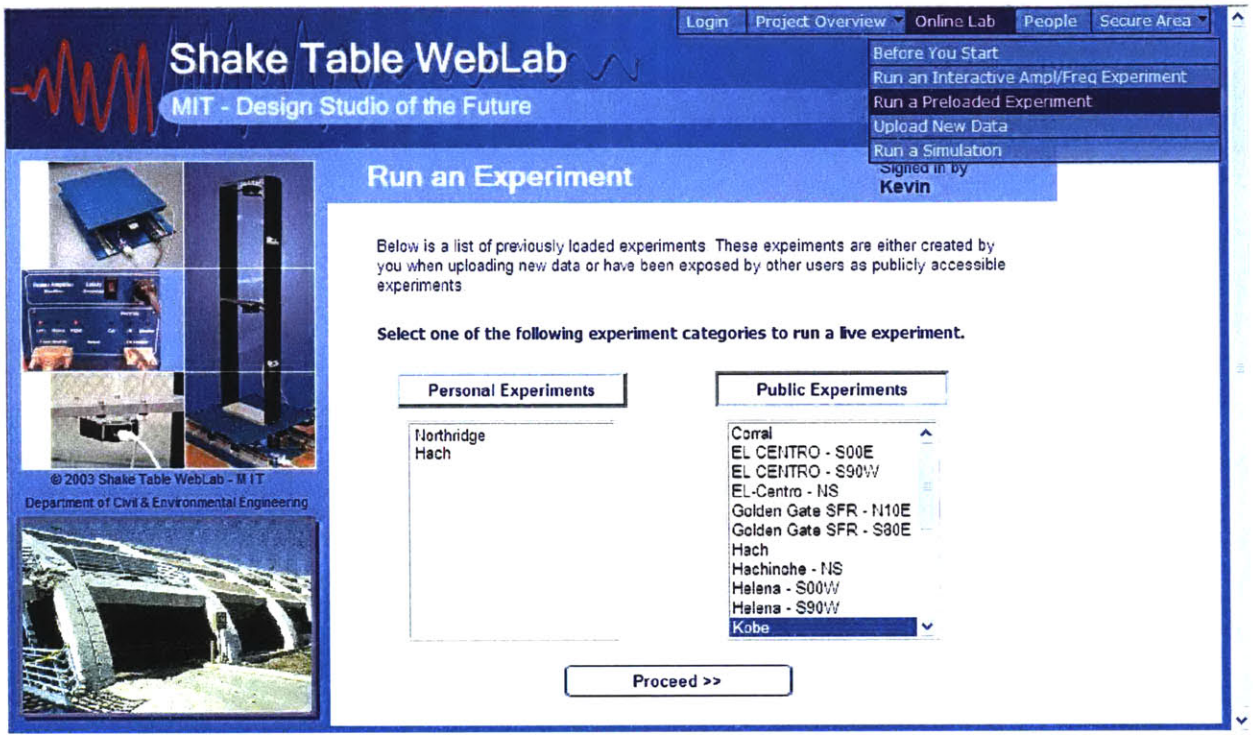
4.1 Online Experimentation Activities

Building the Shake Table WebLab with a flexible yet stable architecture has been one of the main concerns in design. This chapter presents the different online experimentation activities and user accessibility features of the WebLab. Three different types of activities are described below: previously loaded experiments, new experiments, and interactive sine-wave experiments.

4.1.1 Previously Loaded Experiments

Previously loaded experiments are available in two groups: publicly available to the whole user community and privately accessible by the experiment creator only (see Figure 11). Further illustration on user permissions and accessibility is provided in the 'User Account Management' section later in this chapter.

Figure 12 is the sequence diagram for processes that occur while running one of the stored experiments. The diagram serves to summarize the activities that execute on both the client and the server and clarifies which of these functions occur synchronously and which occur asynchronously on separate threads. While vertical lines represent the lifetime of a program, horizontal arrows represent method calls - as in the case of arrows directed towards the Web Service - or an action invoked by the application from which the arrow is initiated and affecting the application to which the arrow is directed. All arrows directed from the Web Service towards WinCon are good examples of the latter case. Ovals attached to a vertical application thread represent functions that synchronously execute within that thread as the program proceeds downwards. The shaded ovals, 'data transfer to/from WinCon' and 'process data', are further illustrated under 'Data Streaming Mechanism' in Chapter 5 and 'Coding Digital Filter Computations' in Chapter 6 respectively.



Below is a list of previously loaded experiments These expeiments are either created by you when uploading new data or have been exposed by other users as publicly accessible experiments

Select one of the following experiment categories to run a lve experiment.

Northridge
Hach

Corral
EL CENTRO - S00E
EL CENTRO - S90W
EL-Centro - NS
Golden Gate SFR - N10E
Golden Gate SFR - S80E
Hach
Hachinohe - NS
Helena - S00W
Helena - S90W
Kobe

Figure 11. Shake Table WebLab menu page showing lists of publicly and privately accessible earthquake experiments.

On the clients' computers the main program is the Win Form Control which loads within their browsers. (The means by which parameters are passed from the HTML page to the control is previously described in the 'Implemented Technologies' section of chapter 3.) Within this same system process, there are two other threads (namely Threads 1 and 2) which proceed asynchronously with the main thread. The purpose of '*Asynchronous Thread 1*' is to maintain the status of the current user in an SQL queue table. To illustrate, Thread 1 would update the timestamp of the user in the maintained queue at five second intervals to avoid being disconnected by a server-side queue monitoring program. The purpose of '*Asynchronous Thread 2*' is to renew socket connections with WinCon, on the Control Server, in order to avoid having the client disconnected while receiving data streams as the experiment proceeds. On the shake table control server, there are two main applications: a Web Service and WinCon[5]. A description of each of the web methods invoked on the Web Service is described in Chapter 5.

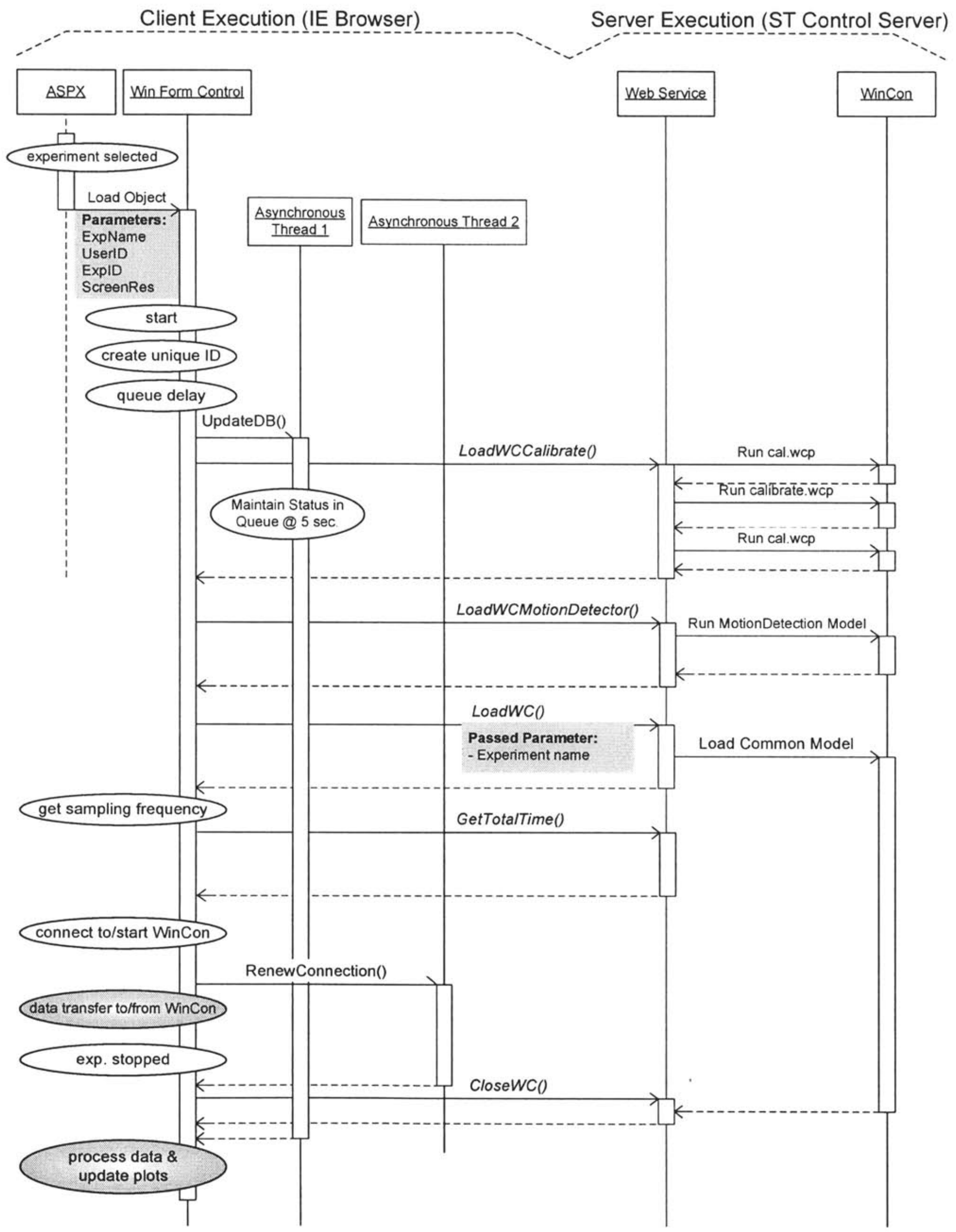


Figure 12. Sequence of processes for running a previously-loaded or an interactive experiment.

[5] WinCon is represented as a single application instead of showing WinCon Client and WinCon Server separately. This representation is sufficient for the purpose of the sequence diagram as both applications reside on the control server and starting WinCon Server would automatically launch WinCon Client on the same computer.

4.1.2 New Experiments

The second type of online activities allows users to create new experiments by adding items to their own list of preloaded experiments. This functionality is useful for experimenting with newly occurring earthquakes by uploading horizontal components of seismic accelerations and running the corresponding experiments. With the absence of a commonly adopted standard for seismic data representation, it is essential to interpret uploaded files based on a well-defined structure. The specified format, can however, accommodate various numeric layouts with a non-confining unit of measurement so as to avoid complications on behalf of a user in preparing a new experiment. Refer to Appendix B for the description of a newly uploaded data file format. In addition, uploading new earthquake data files requires server-sided checking and scaling down of displacements to a scale which is compatible with the capabilities of the shake table. Accordingly, the experiments are an exact representation of the real seismic accelerations, yet the ground displacement values are reduced systematically to a maximum of three inches for the displacement of the table's base. With the newly developed web application, all necessary system processes for uploading new data are carried out at the click of a button.

Sequence of Programmatic Processes

Figure 13 shows the sequence of processes entailed in creating a new experiment. Client input (such as the experiment name and local path of a data file) is defined through a Win Form Control while the server-side processes (on the Control Server) include a Web Service and Matlab functions. When a user submits a request – with a complete set of input parameters – to the Web Service to start processing the data, an asynchronous thread is initiated on the client to maintain its status in the queue. Even though the user doesn't activate the shaker in the process of creating an experiment, the request, which is received by the Control Server, is expected to reserve a considerable portion of the server's processor (mainly by executing Matlab commands) which would degrade the performance of simultaneously running experiments by other users. Therefore, a single queue is designed to handle all client requests to the Control Server – be they aimed at running experiments or creating new ones. The role of Matlab as a server-side data processing environment is further illustrated in the following section.

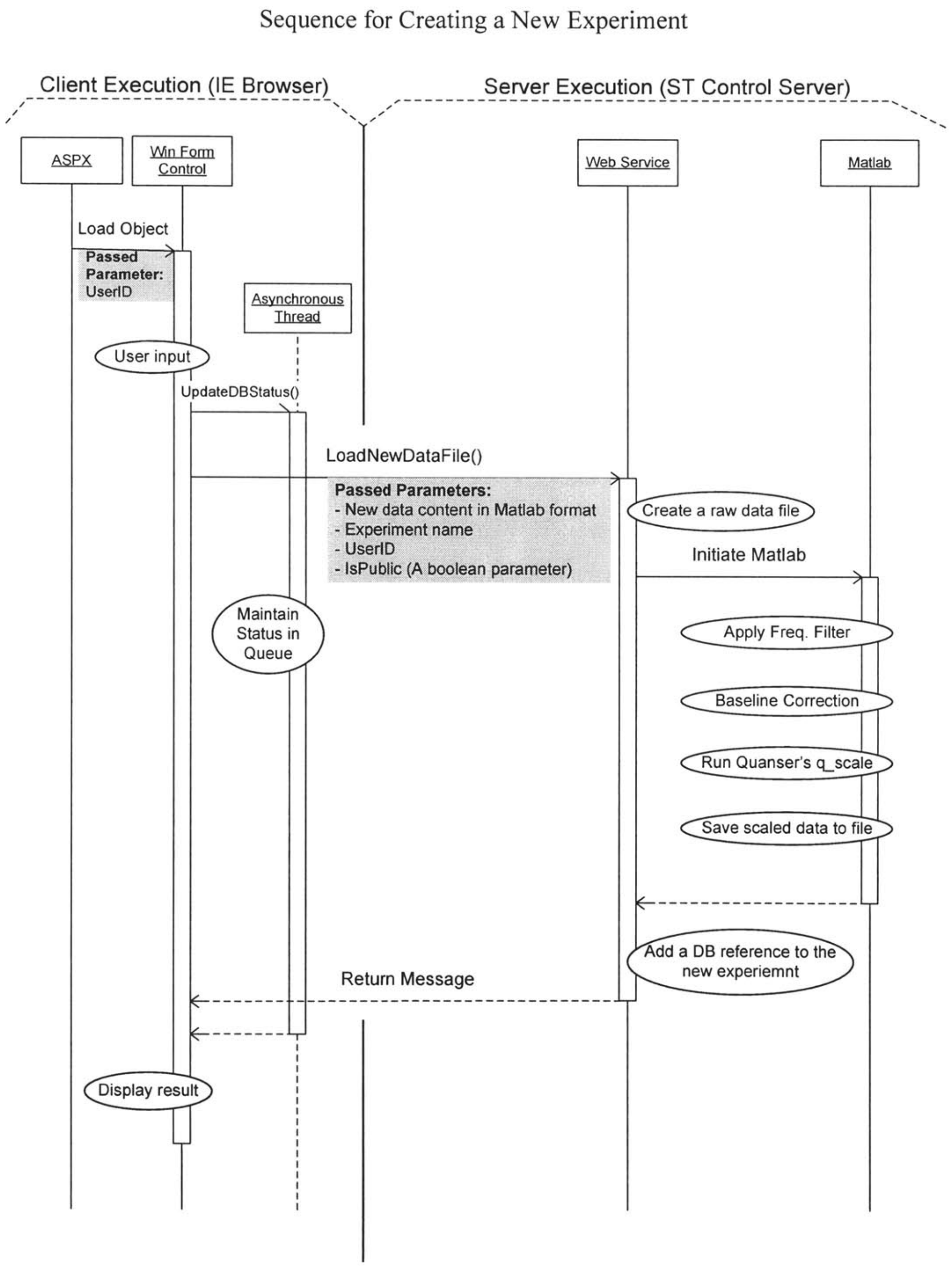


Figure 13. Sequence of processes for creating a new experiment.

Filtering Newly Uploaded Excitation Signals

As users may upload new input signals to the shaker, it is essential that the signals be filtered appropriately to avoid subjecting the shaker to high frequencies. The largest frequency that an input signal may entail is related to the sampling time interval which users state in their uploaded data files. A sampling time interval of 1ms, for example, would correspond to a maximum applicable frequency of 50 Hz, which is half the sampling frequency. One possible solution to this problem would entail restricting acceptable time intervals to values above a minimum threshold which would guarantee avoiding unfavorable experiments applied to the shaker. However, a user may use a low sampling rate for a more accurate signal representation without necessarily applying any high frequencies. For this reason, we resort to a more elaborate solution which makes use of a frequency filter. Accordingly, users may upload data with low sampling frequencies and the filter would screen the corresponding signal of any high-range frequencies.

The procedure used to process the newly uploaded data consists of two steps: first, a low-pass frequency filter is applied; second, a baseline correction of the filtered acceleration time series is performed. A low-pass filter with a maximum frequency of 20Hz is applied to the Fourier transform of the table input accelerations (see Figure 14). The filter is dynamically generated using Matlab's Finite Impulse Filter (FIR) based on a Hamming window. Refer to Appendix C for the Matlab filtering function code. Multiplying the filter by the Fourier transform would deprive the frequency spectrum of any frequencies above 20 Hz. The Inverse Fast Fourier Transform of the resulting spectrum would give back the acceleration time series excluding its high frequencies. As to the baseline correction, it is based on zero initial velocity and displacement conditions. A cubic curve as a baseline is finally subtracted from the acceleration values resulting from the filter to produce an acceleration series with a flattened baseline.

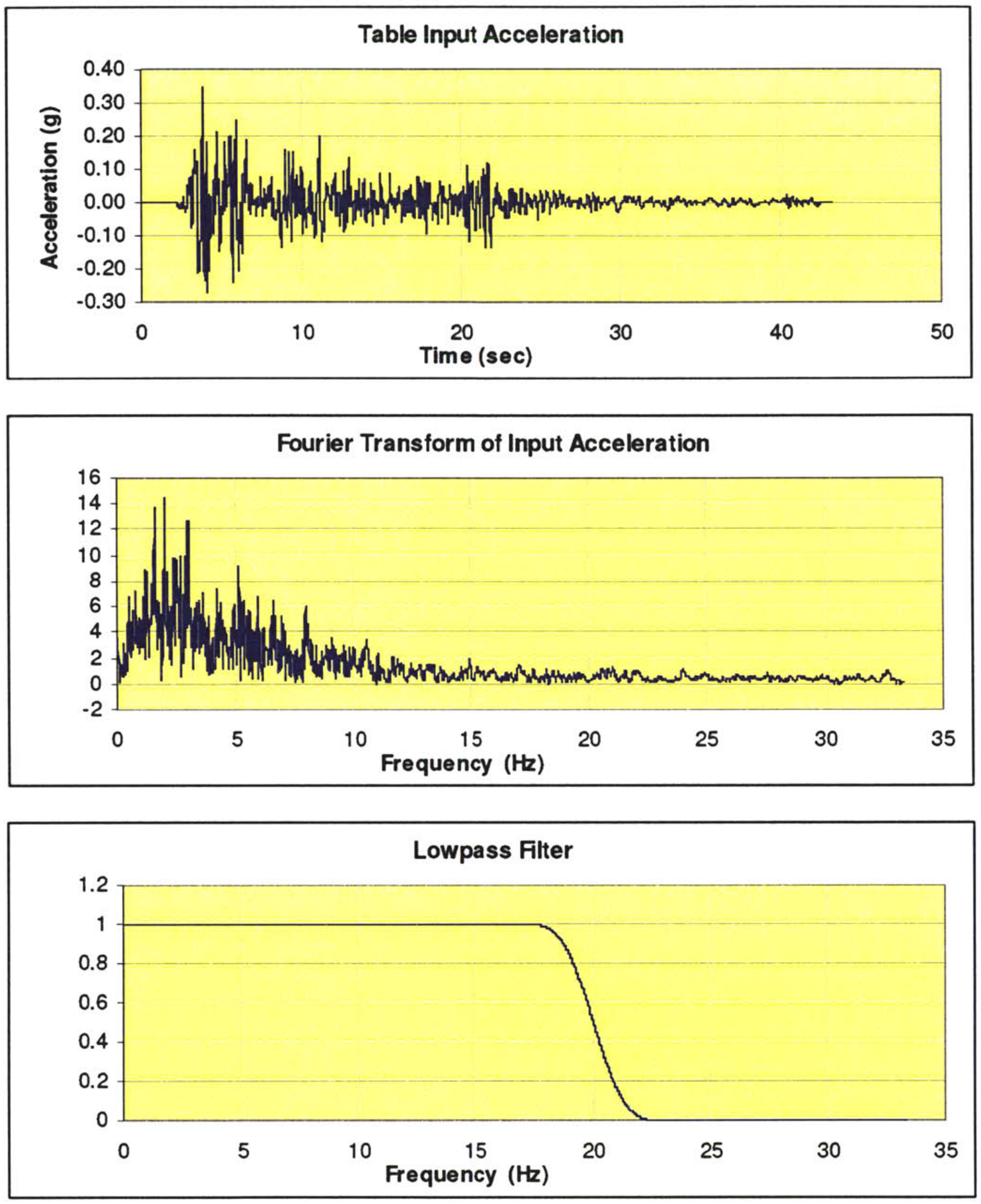


Figure 14. Applying a low-pass frequency filter to a newly added acceleration series.

4.1.3 Interactive Experiment

The first two types of experiments may be described as non-interactive in the sense that when the experiment is started, users have no further control of the shaker table besides observing the streamed-back sensor responses. In the previous cases, the streamed data represents four distinct parameters: three accelerometer readings and one displacement reading of the base. Two accelerometers are located on each of the two floors and one at the base. In contrary to the two previous categories, the third experimentation model is interactive whereby displacements of the shake table are dynamically set by the user instead of being read from file. To illustrate, the input displacements are defined by a sine wave signal, the frequency and amplitude of which are modified by the user as the experiment proceeds. In addition to the four sensor responses described above, the sine wave variation experiment provides three other signal readings that reflect the amplitude, frequency and input sine wave as applied to the shaker. Accordingly, users are able to simultaneously change the input displacement signal and monitor the behavior of the mounted structure as it responds to instant alterations conducted on the client side.

Figure 15 is a screenshot of the interactive experiment interface and the 'Sine Wave Variation' page which enables users to define an input wave. Two knob controls allow a user to incrementally change an input sine-wave while variations to the signal are instantaneously plotted. A user may shift back and forth between the 'Floor Accelerations' tab and the 'Sine Wave Variation' tab in order to observe the instantaneous effect of the input signal on the acceleration responses and their Fourier spectra. One useful experiment is to check out the modes of vibration of the test structure by increasing the frequency as the user observes structural deflections in the live video display.

Unlike previously loaded experiments whereby the input signal is filtered and scaled down to comply with the table's capabilities, the interactive experiment entails directly feeding user input to the table. For this reason, it is essential that a user be restricted to sine-wave variations that may be safely applied. Accordingly, knob controls are designed to validate user input based on criteria that ensures a safe operation of the shaker and avoids excessive deflections of the test structure. Table 2 is a list of maximum frequency values that may be set at various frequencies. In addition to maximum allowable frequencies, a forbidden frequency range that is centered on the dominant vibration mode of the test structure is applied[6]. Figure 16 is a plot of the amplitude-frequency boundary limit. The forbidden frequency range, designated by the lower and

[6] Refer to section 6.2 for more information on the modes of vibration of the test structure.

upper limit vertical series, bans frequencies at or near resonance conditions with the structure's first mode of vibration (1.4Hz).

Table 2. Amplitude/Frequency boundary limits for the interactive experiment.

Amplitude (inch)	Forbidden Frequency Range		Maximum Frequency (Hz)
	Lower Limit	Upper Limit	
2.5			1.1
2.4			1.1
2.3			1.1
2.2			1.1
2.1			1.1
2			1.1
1.9			1.1
1.8			1.1
1.7	1.1	1.9	2.2
1.6	1.1	1.9	2.2
1.5	1.1	1.9	2.4
1.4	1.1	1.9	2.4
1.3	1.1	1.9	2.6
1.2	1.1	1.9	2.6
1.1	1.2	1.8	2.6
1	1.2	1.8	2.6
0.9	1.3	1.7	3
0.8	1.3	1.7	3
0.7	1.3	1.6	3.5
0.6	1.3	1.6	3.5
0.5	1.3	1.6	3.5
0.4	1.3	1.6	4
0.3	1.3	1.6	4.5
0.2	1.4	1.6	5
0.1			5

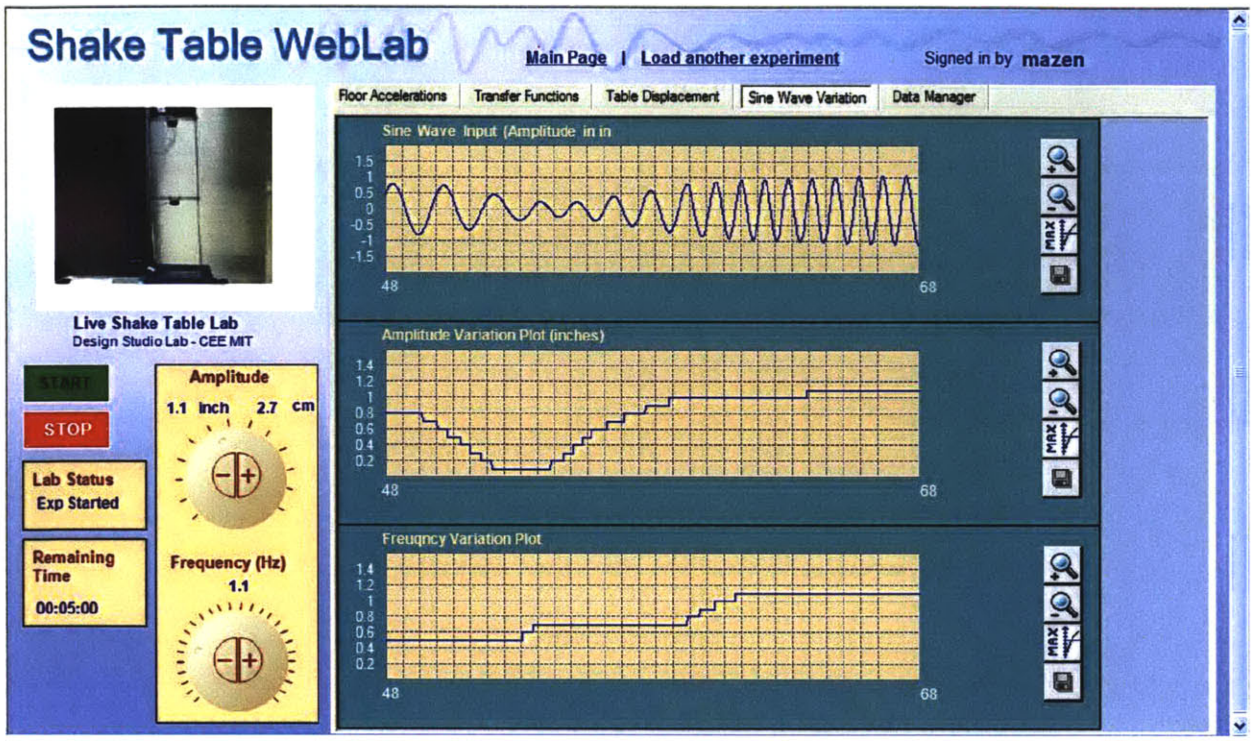


Figure 15. The WebLab's sine wave variation page implemented for the interactive experiment.

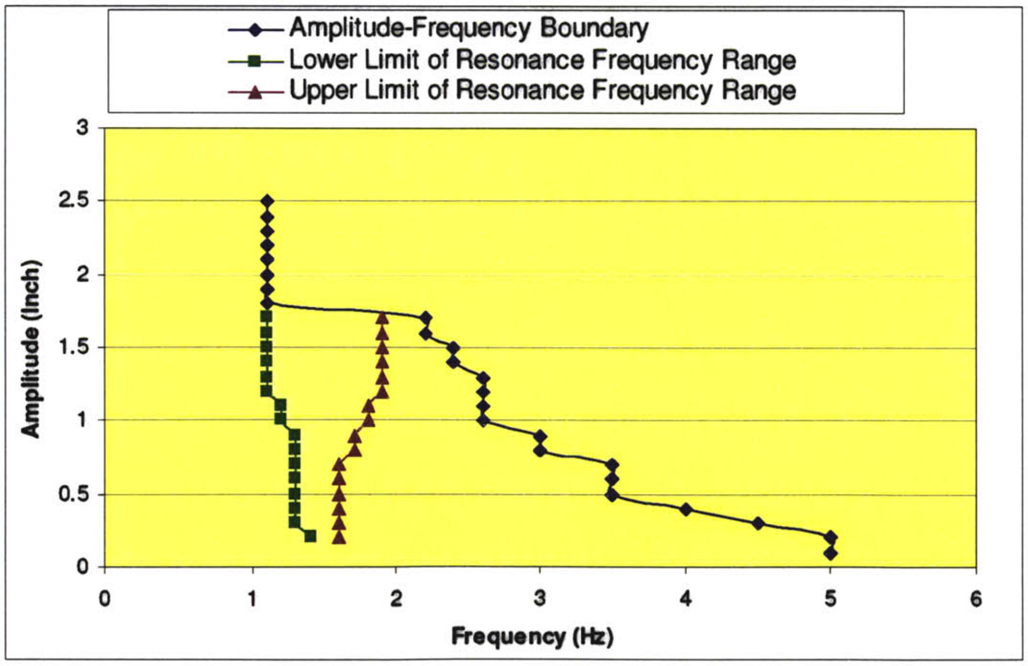


Figure 16. Allowable amplitude/frequency range for the sine-wave interactive experiment.

4.2 User Account Management and Accessibility Features

An essential feature of an online experimentation environment is the manageability of user accounts in terms of authenticating accessibility and authorizing permission to restricted activities. Moreover, a web channel to control laboratory equipment necessitates a request queuing mechanism that avoids the possibility of simultaneous multiple client-control scenarios. I explain next why these features are vital requirements for the shake table WebLab and how they have been implemented.

4.2.1 User Account Management

Access to the Shake Table WebLab requires registration which can be carried out online. Accessing any of the online lab activities entails a user login with a username and password. ‘Administrator’ and ‘Developer’ are two special types of user accounts that are given added accessibility privileges. A user with an administrative account can view, edit and remove other registered users. Another functionality restricted to an administrator is the ability to make uploaded earthquake experiments publicly accessible. Newly created experiments by ordinary users are only visible to the experiment owners who have uploaded the new data files. Administrators may, however, choose to configure new experiments as public. Consequently, the added experiment will be displayed under the public experiments list and may therefore be activated by any registered user. An administrator account is of value to a course instructor or an assistant who needs to manage student profiles and add to the public experiment archive. Creating an administrative account is only possible through another logged-in administrator.

As to the ‘Developer’ account, it is intended for a technical developer who is to maintain and upgrade the system. To prevent access to the shaker table at times when the system is being upgraded, a protection key is implemented. Accordingly, an administrator can lock the online accessible channel to the shaker whereby none of the registered users except the ‘Developer’ can run experiments. This approach serves as a precaution against unfavorable outcomes when the table undergoes testing.

4.2.2 User Request Queuing Mechanism

In order to organize online requests to activate the table, a client monitoring solution is implemented with the following functionalities: queue user requests in order to prevent more than one client from activating an experiment at the same time, remove any disconnected client from the queue, and stop the shaker table in case the active user was disconnected. The queue includes user requests that might be cancelled as when a user closes his/her browser. Accordingly, a dynamic queuing mechanism has been implemented to continuously check client status and grant lab access on a first-come-first-serve basis.

The aforementioned features are implemented with a queue-monitoring program hosted on the Web Server. When a logged-in user attempts to run an experiment, the client application (as a Windows Form Control) creates a unique ID which is logged into an SQL Server queue table with the corresponding time of request. During the time before a user is granted control permission, two different programmatic functions proceed: one running on each queued client computer, and one continuously monitoring program on the Web Server. The aim of the processes that run on the client side is to update the client status on the server at five second intervals by renewing the client's time stamp in the queue. Moreover, this same process keeps users informed of their rank in the queue and the estimated remaining time for their turn. The client-side processes are illustrated in Figure 17 which presents the implemented model for handling user requests. Meantime the monitoring program on the server checks for any updates in the database queue table at five second intervals. As a result, the queue monitoring program removes any queued item older than ten seconds; if the status of the user who is in charge of the shaker table hasn't been updated for more than ten seconds, the program disposes the corresponding user from the queue after stopping the table in case it is in operation mode. Refer to Figure 18 for the process diagram of the continuously running queue-monitoring program.

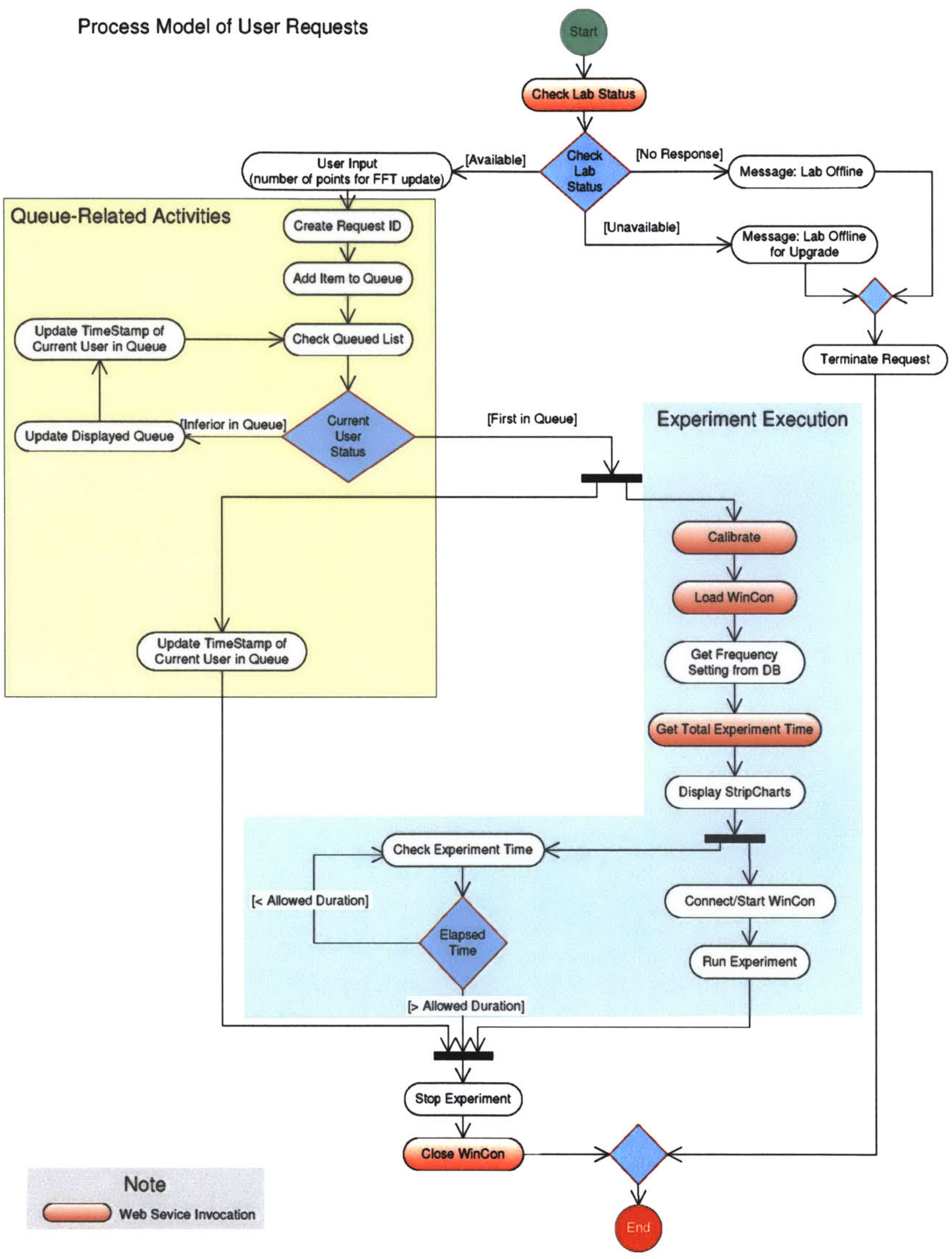


Figure 17. Process model of web-user requests to operate the shaker table.

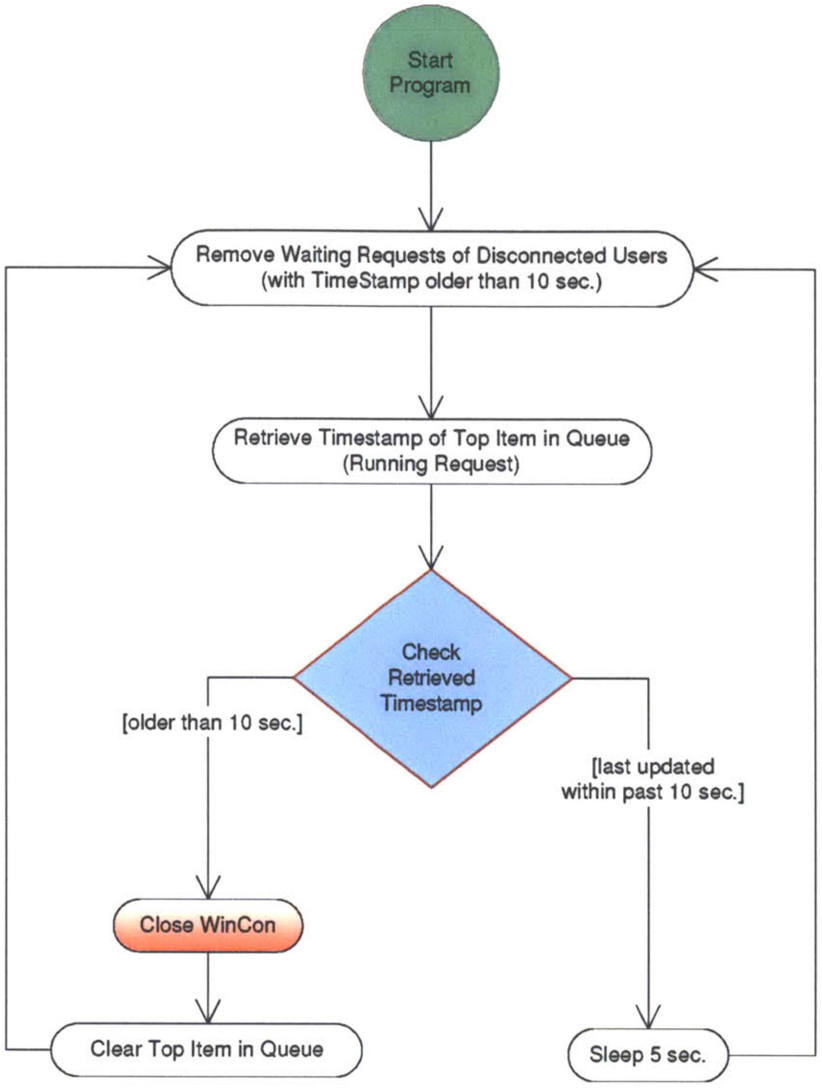

Start Program
Remove Waiting Requests of Disconnected Users (with TimeStamp older than 10 sec.)
Retrieve Timestamp of Top Item in Queue (Running Request)
Check Retrieved Timestamp
[older than 10 sec.]
[last updated within past 10 sec.]
Close WinCon
Clear Top Item in Queue
Sleep 5 sec.

Figure 18. Process model of the queue-monitoring program.

As illustrated in the process diagrams, queuing user requests entails excessive database querying. To illustrate, the server-hosted program updates the queue table every five seconds. In addition, every client placing a request to conduct an activity is repeatedly renewing its presence in the queue table at the same time interval. To improve performance, SQL stored procedures are implemented to handle database operations. Figure 19 and Figure 20 are two sample stored procedures used in maintaining the queue table.

```
CREATE PROCEDURE dbo.Proc_QueueCheck
        (
                @ClientID varchar(50)
        )
AS
        UPDATE Queue SET LastUpdate=GetDate() WHERE ClientID=@ClientID

        SELECT * FROM Queue

RETURN
GO
```

Figure 19. SQL Stored procedure for updating user status in queue.

```
CREATE PROCEDURE dbo.Proc_RemoveDeadQueueItems
(
        @CloseWCOut int OUTPUT,
        @ClientIDOut varchar(50) OUTPUT
)
AS
        DECLARE @TopLastUpdate datetime
        SET @CloseWCOut=0
        SELECT TOP 1 @TopLastUpdate = LastUpdate, @ClientIDOut = ClientID  FROM Queue

        IF DATEADD(second,-10,GetDate()) > @TopLastUpdate
                BEGIN
                        SET @CloseWCOut = 1
                END

        DELETE FROM Queue
                WHERE (DATEADD(second,-10,GetDate())> LastUpdate)
                AND ClientID <> @ClientIDOut
RETURN
GO
```

Figure 20. SQL stored procedure for removing client requests older than 10 seconds.

Chapter 5

SHAKE TABLE CONTROL SERVER

5.1 Overview of the Control Server

The shaker table Control Server has direct access to the table apparatus through a data acquisition/controller board which is in turn connected to a power module. This same server runs Quanser Consulting's WinCon proprietary software package for driving the shake table. WinCon was originally targeted for client/server operation where WinCon server and WinCon client are installed on separate computers with a TCP data transfer channel between the two. However, for the purpose of the project, both of those programs are installed on the Shake Table Control Server. As described in Chapter 2, the WinCon Server application is accompanied by an external Interface that provides an additional TCP channel for other applications to communicate with the shake table. Since the aim of the project is to provide lab accessibility through an Internet browser, a user friendly client-side web application is needed. This application relies on WinCon as an intermediary for controlling/monitoring the shake table equipment.

In order to establish a connection between the web application and WinCon, an essential mechanism is required to automatically carry out processes on the shake table control server. Accordingly, a Web Service is hosted in Internet Information Services (IIS) to translate web client requests, carried out through the web application, into corresponding activities on the control server. For instance, a user request to start an experiment triggers a web service method to check the status of the control server and accordingly calibrate the shake table before initiating the experiment. Another web method is then triggered to launch WinCon and load the corresponding experiment files.

Other major applications hosted on the Control Server include Matlab as a means to scale down original earthquake data – when a user chooses to experiment with new earthquakes - into displacement ranges that comply with the Shake Table capabilities. Functions within Matlab are triggered through a web method. This method uploads the new data file from the user's computer, launches Matlab as a background process running on the control server and sends the necessary commands to the Matlab environment[7]. The data file that results from the uploading and scaling procedures is stored under an identifiable directory on the Control Server in order to be read by WinCon when the corresponding experiment is requested.

5.2 WinCon Configuration

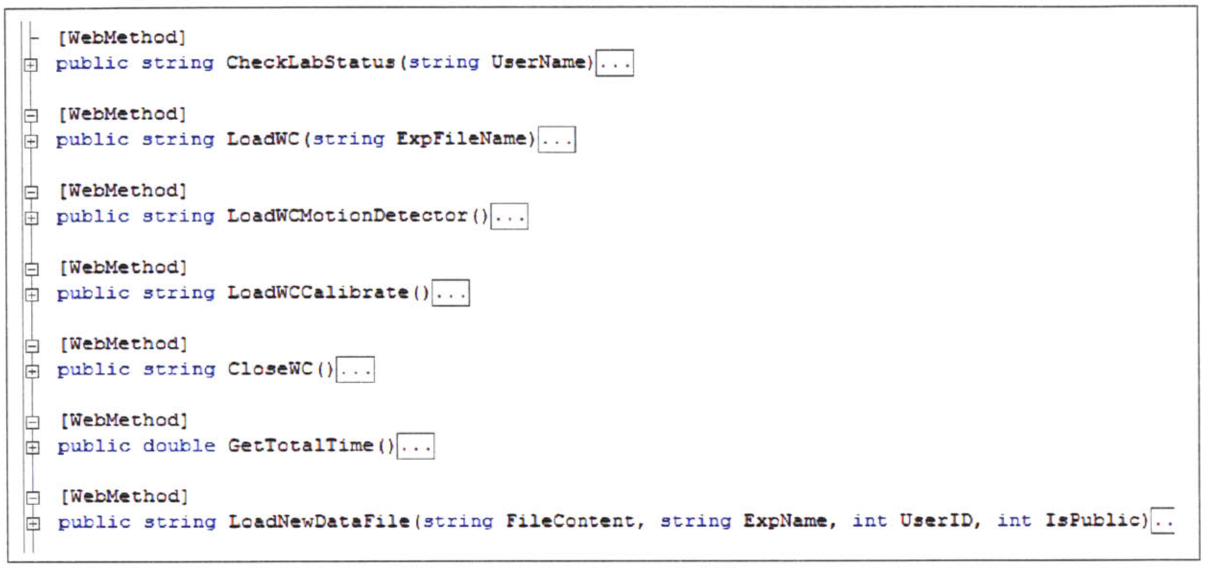
Since an outside web-based application handles remote accessibility to the laboratory, both WinCon Client and Server are hosted on the control server. As described in the 'Initial Principle of Operation' section of Chapter 2, WinCon is based on Simulink models that are then transformed into WinCon executable files by a C-compiler. The limitation of the originally supplied models is in the fact that they acquire their input from the Matlab workspace (sample provided in Appendix A); thus, their compilation would result in the input data being embedded within the executable. Alternatively, the models are changed so as to read their input from a separate data file. Accordingly, the new earthquake model permits the same WinCon executable to run an indefinite number of excitations. As a result, instead of generating a distinct WinCon file for each experiment, we end up with one executable only and as many data files as there are experiments.

5.3 Web Service Implementation

Web Methods are implemented on the Control Server mainly to handle remote access to both WinCon and Matlab. Refer to Figure 21 for a listing of the Web Methods as written in C#, with their corresponding input/output parameters. Following is a brief description

[7] Interfacing the MS .Net-based web service (Prosise, 2002) with Matlab is achieved through a dedicated CLR (Common Language Runtime) compliant type developed in VB .Net for interconnecting .Net applications with Matlab.

of the functionality provided by each method. Refer to the sequence diagrams in Figure 12 and Figure 13 for an overview of how these methods are invoked.



```
[WebMethod]
public string CheckLabStatus(string UserName)...

[WebMethod]
public string LoadWC(string ExpFileName)...

[WebMethod]
public string LoadWCMotionDetector()...

[WebMethod]
public string LoadWCCalibrate()...

[WebMethod]
public string CloseWC()...

[WebMethod]
public double GetTotalTime()...

[WebMethod]
public string LoadNewDataFile(string FileContent, string ExpName, int UserID, int IsPublic)..
```

Figure 21. Signatures of Web Methods implemented in the Control Server's Web Service.

- *CheckLabStatus* is called by a client to determine the status of the laboratory just as the user request to run an experiment is sent. The laboratory status is maintained in the SQL database and stored as 'LabStatus' (refer to the 'Data Model Section' in Chapter 3 for more information about the database). Accordingly, the method is expected to return the state of the lab as being either available or not. The returned value is also dependant on the type of user issuing the request. For this reason, even though the lab status, stored in the database, may be unavailable, a 'developer' user is still granted access to the lab. In addition to checking the lab status being preset as available or not, client calls made to this method serve to detect if the control server is on and ready to receive experiment requests. While a 'not available' return value would be translated to a user as 'the WebLab being disconnected for maintenance', not receiving any response after calling this method would indicate to a client that the control server is in an offline state and, in both cases, the experiment request would be discontinued.

- *LoadWCCalibrate* positions the table in a centered state so as to allow for maximum displacements without exceeding the boundary limits of the shaker.

- *LoadWCMotionDetector* checks whether the test structure is still moving due to oscillations from a prior experiment.

- *GetTotalTime* retrieves the duration of a previously loaded experiment which a user has chosen to run. The client application uses the experiment duration to fit the recorded response data points with the maximum plotting width of the graph panels. This serves to provide the clearest possible real-time plots depending on the size of the acquired data. The duration of an experiment is stored in a data file which is located on the control server and contains all information pertaining to that particular experiment.

- *LoadWC* loads the selected experiment into WinCon. There are two directories on the control server which contain experimental data. The first one, named 'Scaled Data Files', contains a data file for each experiment excitation that has been uploaded and scaled appropriately. The second directory, 'Active Directory', contains all WinCon project files (such as the WinCon project files used for table calibration and the sine-sweep interactive experiment). One of the WinCon files under the 'Active Directory' is a common file that reads input accelerations from a data file located under the same directory. Accordingly, in order to run a particular experiment, the corresponding data file is copied from 'Scaled Data Files' into 'Active Directory' with a name recognizable by the common WinCon project. The experiment name, therefore, becomes meaningless to the WinCon project file as long as the correct data is loaded into 'Active Directory'.

- *CloseWC* would stop a running experiment and close the WinCon software. Since stopping a running WinCon experiment is only possible by establishing a TCP socket connection with WinCon Server's EIW, an instance of the *ExtComm* class needs to be initiated. (Refer to section 5.4 for information on *Extcomm.*)

Accordingly, the Web Method *CloseWC* creates an *ExtComm* object and invokes its *stopmodel()* function which sends the appropriate command to discontinue WinCon's operating model.

- *LoadNewDataFile* creates a new experiment as it takes four parameters from the client: the experimental data in a Matlab format, the name of the new experiment as declared by the user, the user's ID, and a Boolean key which specifies whether the experiment to be created is publicly accessible by the whole user community or privately owned by its creator. As described earlier in Figure 13, this method would initiate a set of Matlab functions that would handle filtering and scaling-down the originally submitted excitation data. Ultimately, a scaled data file that represents the new experiment is stored under the 'Scaled Data Files' directory with a name consisting of a combination of the supplied experiment name and the creator's UserID. Hence, a user may use an experiment name only once in his list of new experiments while the same name can be given to several experiments belonging to different users. Finally, this Web Method adds a new database reference to the created experiment and associates it with the creators UserID.

5.4 Data Streaming Mechanism

As the aforementioned Web Methods serve to launch WinCon on the Control Server and configure it with the appropriate experimental data, there is no further client-server interaction via the Web Service until the client triggers a request to stop the experiment. Instead, a TCP socket connection takes care of transferring data, back and forth, between a client (Win Form Control) and the Control Server.

The protocol for socket communication of an 'Outside Application'[8] with WinCon is established by Quanser Consulting Inc. Refer to Appendix D for an illustration of the commands which may be issued from or received by an 'Outside

[8] An 'Outside Application' (OA) is the naming given by Quanser to a program written to communicate with WinCon Server by using a socket connection and complying with a predefined protocol.

Application' after launching WinCon Server and its External Interface Window. Since a client/server socket connection establishes an open communication channel through which data is being transmitted, a client side application was designed so as to provide real-time information retrieval from the server and proper event handling techniques. Accordingly, a client socket is launched in its own thread to listen to data published by the server socket on a predefined port. As new data is made available on the server, the client application retrieves the posted information and carries out appropriate event handling procedures. Refer to Appendix E for a class model of the J# program that handles socket connections to WinCon. Since the source code for the represented classes was implemented in Java, a .Net compatible version of the libraries was created with the J# CLR-compliant compiler. Accordingly, once compiled, the libraries are accessible by the client application written in C#. The class model includes a listing of the fields and methods of each class. Designed by Quanser Consulting, the object model entails three main classes: *AsyncCommThread* which inherits the system *Thread*, the abstract class *Comm*, and the *ExtComm* class which inherits *Comm*. Client communication with WinCon's EIW is established by initiating an *ExtComm* object and calling *connect()* and *startModel()* successively. The *commThread*[9] object of type *AsyncCommThread* is responsible for listening to the server socket for any new bytes ready to be read.

5.4.1 Socket Connection Renewal

One problem concerning socket connections to WinCon is related to a connection timeout occurring at a one minute period after initiating the connection. Since the WinCon application is responsible for initiating the server-side socket, it is, therefore, not possible to control the timing out of the connection. This is attributed to the fact that WinCon is a proprietary software, the source code of which is not publicly available. In order to avoid having a client application disconnected, a data renewal mechanism is implemented so that communications between a client and the server are uninterrupted. Figure 22 depicts the data renewal mechanism whereby the maximum lifespan of a socket connection is reduced to 50 seconds and a new socket is initiated instead. The

[9] *commThread* is a property of *Comm* and is of type *AsyncCommThread.*

main running thread in this diagram, '*Connection Renewal Thread*', corresponds to '*Asynchronous Thread 2*' of the generalized sequence diagram in Figure 12. A five-second overlap is used after a new socket is connected and before disconnecting the previous socket. This approach serves to avoid losing segments of data streams at the interchange between two consecutive sockets.

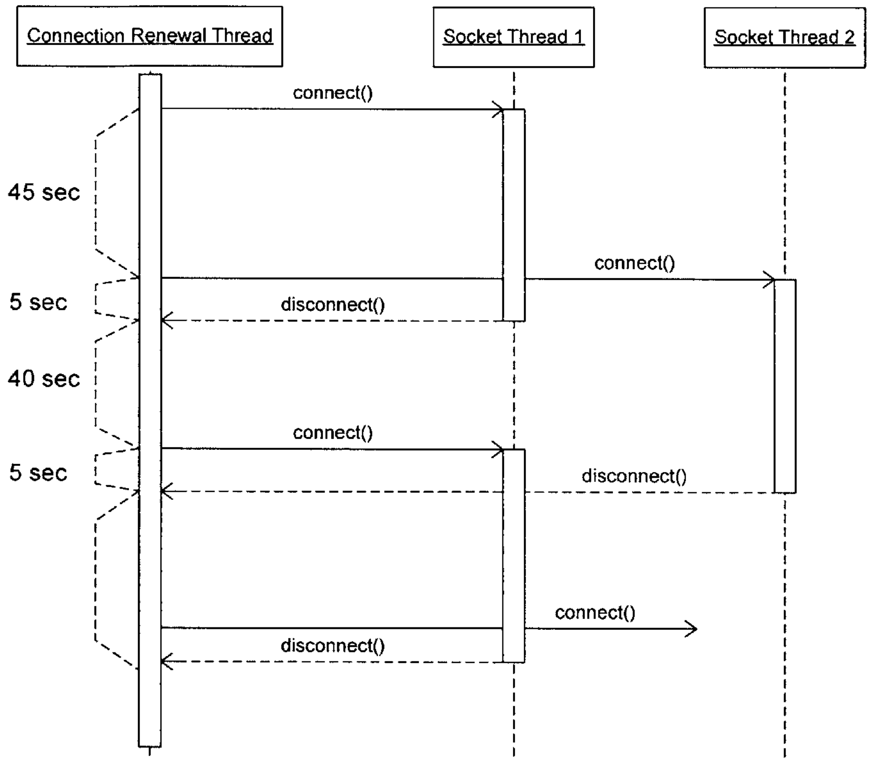
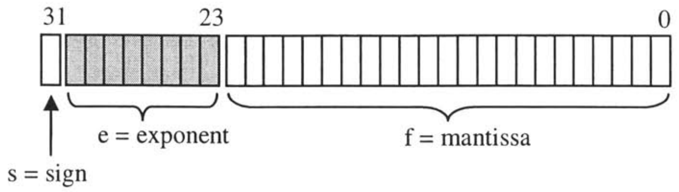


Figure 22. Renewing socket connections from a client application to the Control Server.

5.4.2 Data Transfer Based on the IEEE Standard for Single-Precision Floating Points

Data transfer across the socket connection is based on 8-bit bytes whereby every four bytes form a meaningful float. The float number is in a Single Precision Floating Point format consisting of a one sign bit (0 for positive and 1 for negative), followed by an eight bit exponent, and finally a 23 bit mantissa.



The single float value represented by the 32 bits can be obtained as follows:

$$float = (-1)^s \times 2^{e-127} \times (1.f)_2$$

Before reading data from the server, the client socket waits until four bytes are available, whereby the four bytes are treated as a single float of 32 bits (see the process diagram in Figure 23). Float numbers read by the client may be either:

- one of the communication command numbers listed in Appendix D
- a data value (such as an acceleration reading)

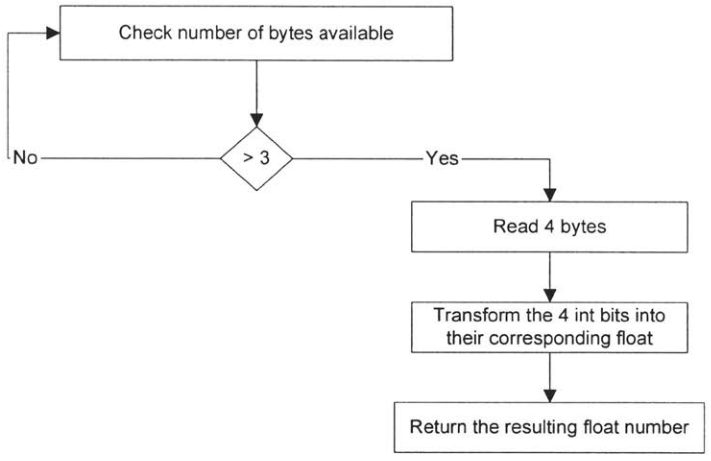

Check number of bytes available
> 3
No
Yes
Read 4 bytes
Transform the 4 int bits into their corresponding float
Return the resulting float number

Figure 23. Process diagram for reading one 32-bit float number.

The diagram in Figure 24[10] illustrates the process through which the client application listens to incoming messages from WinCon and translates received command numbers into corresponding procedures. This diagram illustrates messages that are received by an 'Outside Application' and thus only shows the commands that are sent from the EIW to an OA.

[10] The function of the shaded box, 'Read one float number', is illustrated in Figure 23.

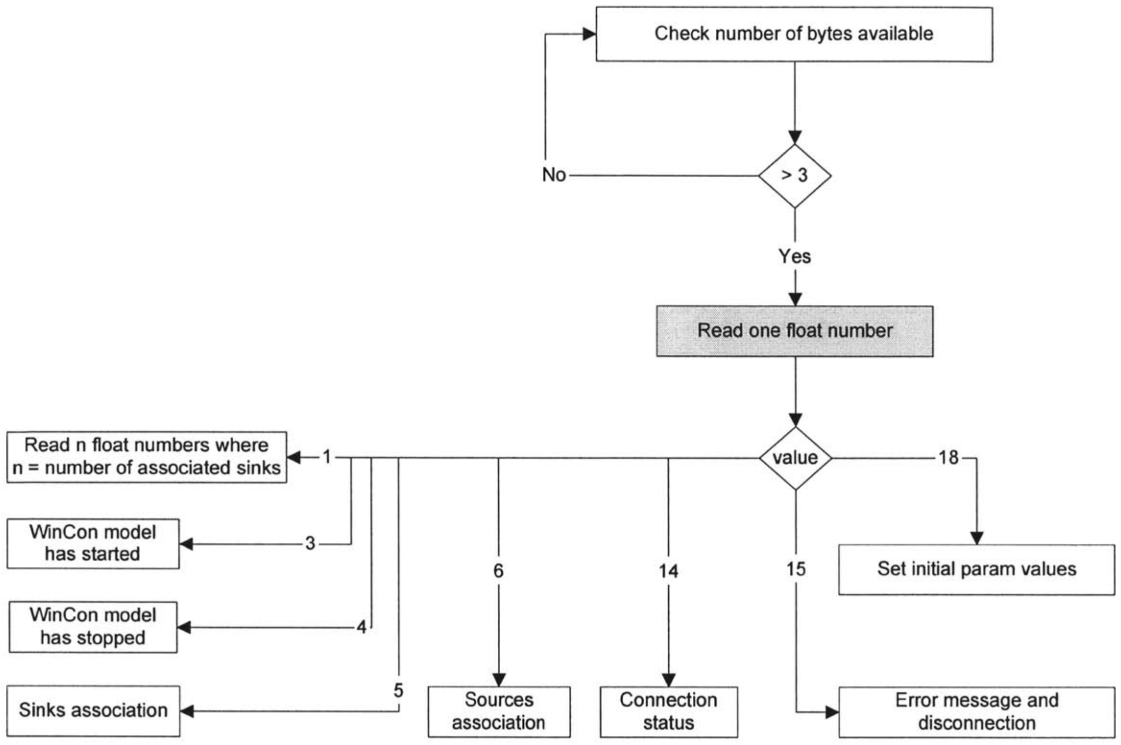


Figure 24. Process diagram for interpreting commands sent from the Control Server to the client web application.

Chapter 6

FEATURES OF THE WEB APPLICATION

6.1 Browser-Embedded Tools

Users access the lab through an Internet browser. An embedded Windows Form Control (which is similar to a Java applet) provides a rich control and plotting interface. The client application is expandable in terms of the data processing tools that might be added depending on the purpose of the user. The common experiment interface shown in Figure 25 provides the following features:

- Real-time plotting of measured acceleration responses
- Fast Fourier Spectra dynamically updated as the experiment proceeds
- Transfer function spectra for each floor and the extraction of dominant frequencies
- Capability to save plots to data files
- Simultaneous plotting of actual and predicted responses in variant line colors
- Capability to export/import transfer functions to and from files: This is implemented in such a way that, after a user runs an experiment, he/she may upload one or more transfer functions which are named by the user. Dropdown list boxes on the plotting interface are dynamically populated with the new uploads. Users may then choose a transfer function in order to predict a floor response and compare it with the actual measurement.
- A live video display of the shaker table

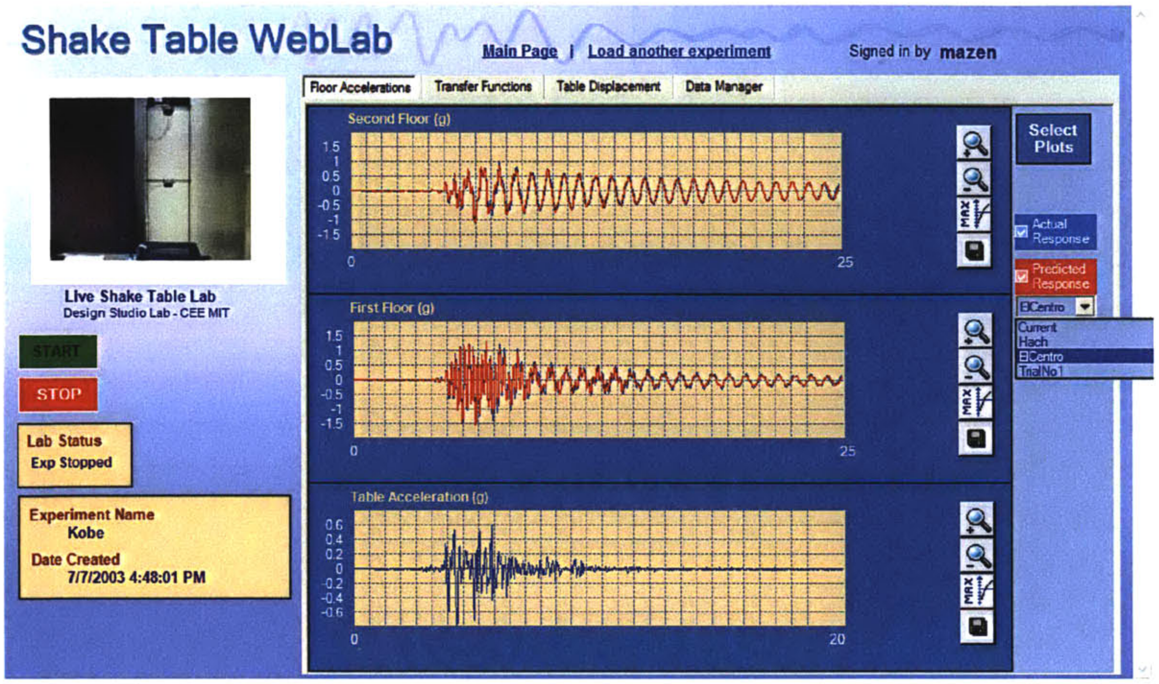


Figure 25. Snapshot of the actual and predicted floor responses for the Kobe earthquake.

6.2 Computation of Fast Fourier Transforms (FFT)

A frequency-domain representation of the structural response instead of a time domain one provides further insight into the nature of the structure and its state of vibration as it responds to excitations applied to its base. Accordingly, a Fast Fourier Transform (FFT) is generated for each of the three time histories associated with the accelerometers on the three levels. Following is the definition of the Fourier Transform which transforms time-based data (*x(t)*: represents time accelerations) into a frequency-based spectrum X(f).

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi f t} dt$$

Knowing the maximum frequency of the Fourier spectrum is possible by looking at the data sampling frequency. Acceleration readings are acquired at a time interval of 20 milliseconds. This value is configured within all the WinCon experiment settings so as to publish data to the windows form application at 20ms intervals. Since at least two data points are required to define a period on the plotted time histories, the minimum period would therefore be twice the period of data collection. Accordingly, the maximum detected frequency of vibration would be half the frequency at which data is collected.

T = 0.02 sec. (Time interval for data collection)

Freq = 1/0.02=50 Hz (Frequency of data collection)

Hence, the maximum response frequency, called the Nyquist frequency, is calculated as:

F_{max}= Freq/2 = 25 Hz (Nyquist Frequency)

The usefulness of FFT computation in the case of a shake table is to pinpoint the dominant modes of vibration of the test structure and therefore know its natural frequencies. To illustrate, the poles of a plotted FFT represent the dominating frequencies. The significance of a structure's natural frequencies is in identifying which types of ground accelerations would subject the structure to excessive deformation. A situation where the ground excitation has a frequency close to the structure's natural frequency would result in harmony between the ground movement and that of the structure, a case referred to as resonance. As the displacement of the ground and the structure are more in sync, the structural deformation is additive to that of the ground and therefore result in great displacement.

The Shake Table WebLab offers an interactive medium to observe and discover the aforementioned properties of the test structure by running the sine wave input experiment introduced in Chapter 4. In addition to a complete FFT being plotted when the user stops the experiment, FFT spectra are also dynamically generated as the experiment proceeds. The implemented algorithm for FFT computation requires that the time-histories consist of a number of points (N) which is a power of 2 (for example N=256, 512, 1024 points). Accordingly, carrying out the FFT on a set of points is accomplished by padding zeros to the end of the data set up to the closest power of 2. While calculating FFTs based on smaller ranges of data points result in a more real-time experience for the user, a greater range results in a delayed yet more accurate presentation of a frequency plot. For this reason, the user is prompted to specify one of three rates (256, 512 or 1024 sample points per plot) at which FFT spectra shall be displayed.

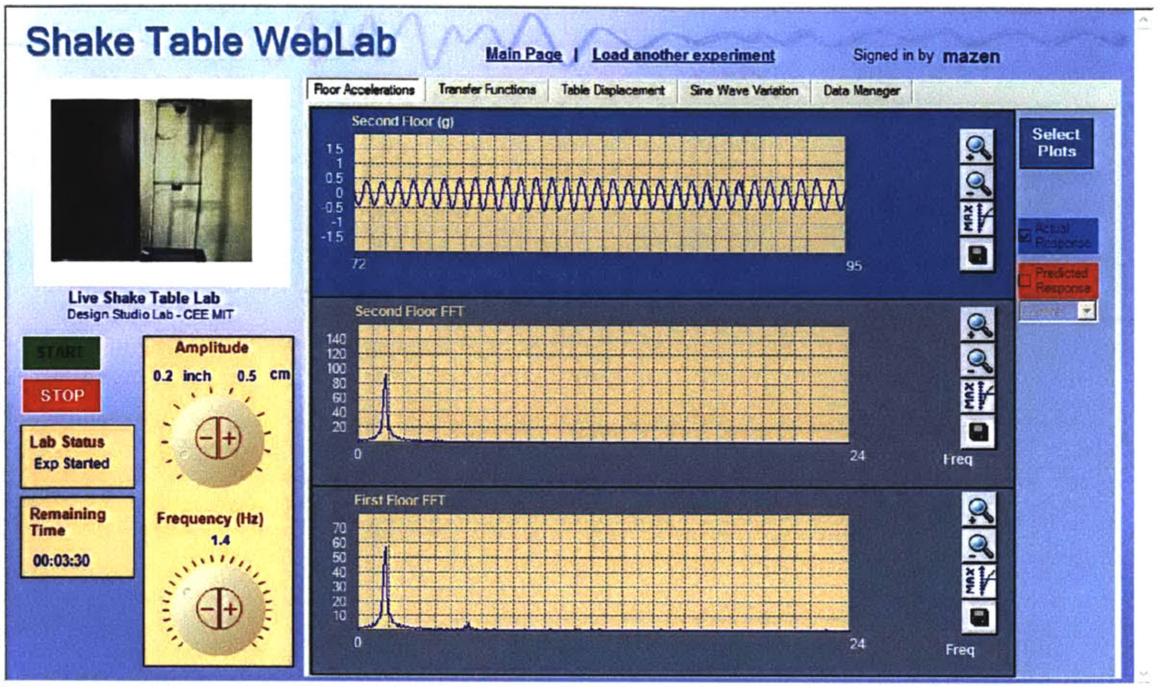
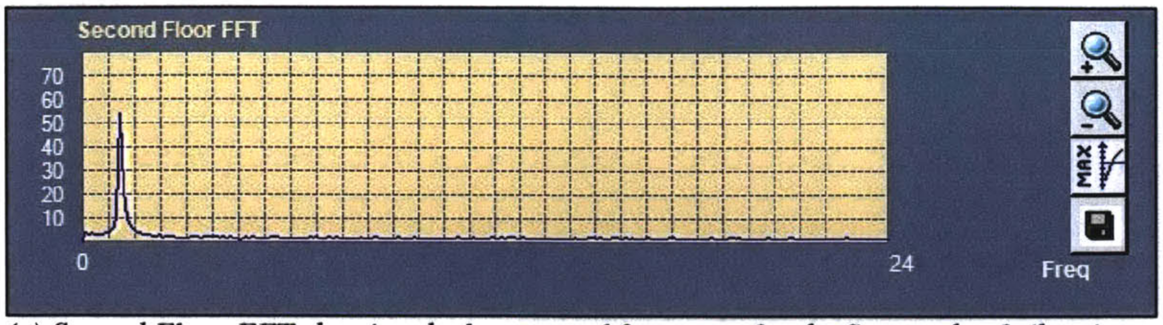


Figure 26. Sine wave experiment showing plots of the second-floor acceleration and FFT spectra.

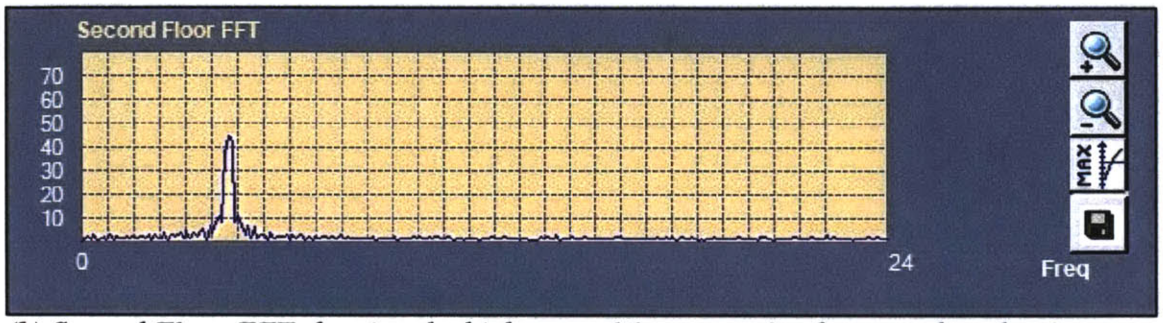
As shown in Figure 26, by changing the frequency knob, a user is provided with updated frequency spectra as the shaker vibrates. The peaks of the FFT reflect the instantaneous dominant vibrating frequencies of each floor based on the immediate modification in the input sine wave signal that the user carries out. By setting a low refreshing rate based on 256 data points, a user is provided with more frequent FFT plots to identify the prevailing mode of vibration. Figure 27 displays snapshots of the FFT plot for the second floor readings at three different times of the same experiment. Refer to Table 3 for information relating to each snapshot. Figure 27(c) corresponds to the final FFT which is based on the whole experiment data. The two peaks refer to the two natural frequencies of the structure namely, 1.5Hz and 5.3Hz. Figure 27 (a) and (b) correspond to intermediate FFTs taken when the input frequency is set to 1.4 and 5 Hz respectively.

Table 3. Summary relating FFT spectra to modes of vibration.

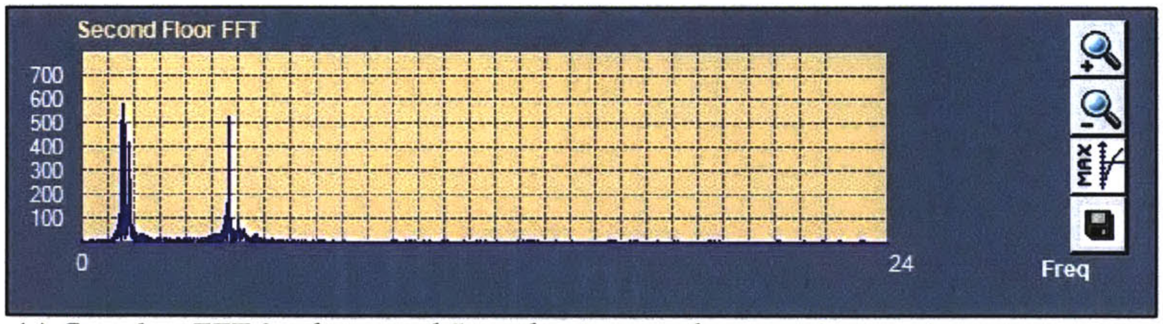
Vibration Mode	Figure Label	Input Sine Wave Frequency (Hz)	Mode Shape Figure	Number of Nodes
First Mode	Figure 27(a)	1.4	Figure 28(b)	0
Second Mode	Figure 27(b)	5	Figure 28(c)	1



***(a)** Second Floor FFT showing the low natural frequency for the first mode of vibration.*



***(b)** Second Floor FFT showing the high natural frequency for the second mode of vibration.*



***(c)** Complete FFT for the second floor showing two dominant frequencies.*

Figure 27. Snapshots of FFT spectra taken at different times during an interactive experiment.

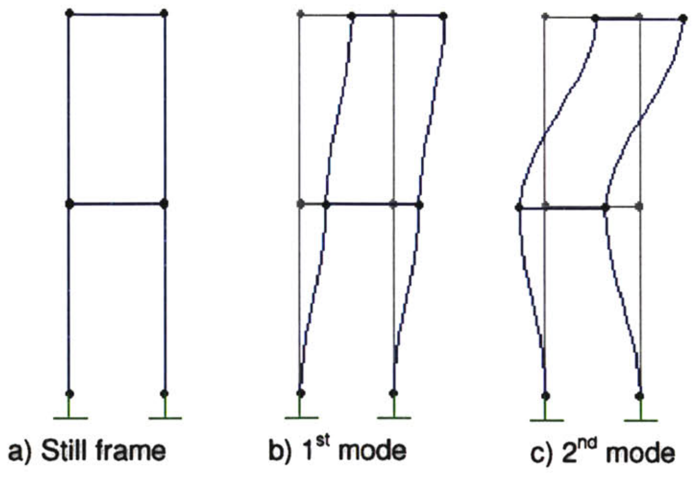
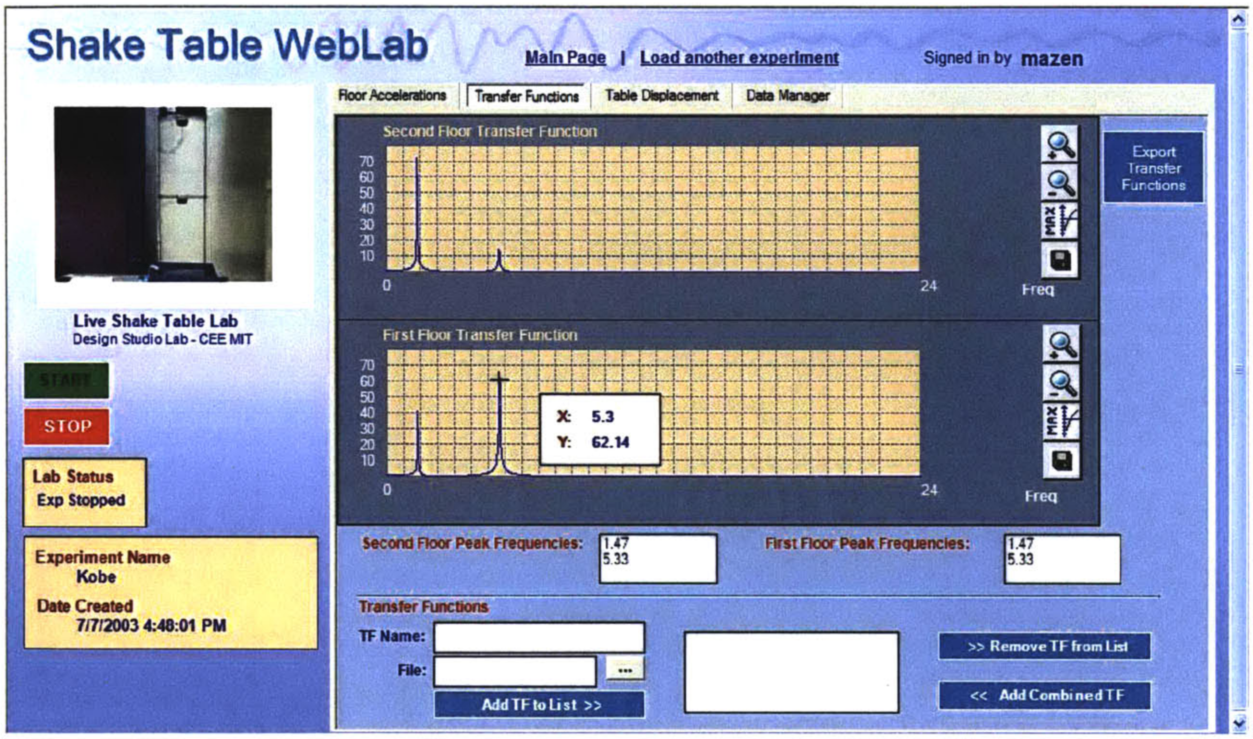


Figure 28. Structural behavior in different vibration modes.

6.3 Computation of Transfer Functions

Another developed functionality is accessible through the '*Transfer Functions*' tab shown in Figure 29. The aim of this tool is to determine the transfer functions from the ground accelerations to each of the upper floor accelerations. Accordingly, what is required is a digital filter which transforms input acceleration signals at the base into a response signal at an upper floor.



Signed in by mazen

X: 5.3
Y: 62.14

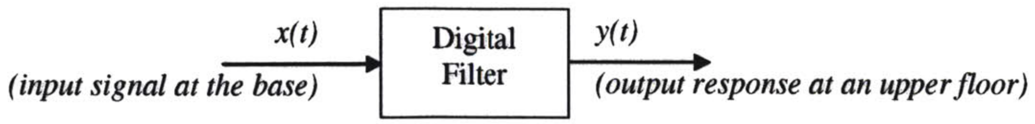
Lab Status
Exp Stopped

Experiment Name
Kobe
Date Created
7/7/2003 4:48:01 PM

Second Floor Peak Frequencies: 1.47 5.33

First Floor Peak Frequencies: 1.47 5.33

Figure 29. Transfer function tab view displays two plots showing the frequency peaks of each floor.


x(t)
(input signal at the base)
Digital Filter
y(t)
(output response at an upper floor)

In identifying a system for the black box, we look for a causal filter whereby values of y are not dependant on future values of x. Moreover, since we are dealing with infinite time histories of acceleration, the filter which is to be implemented needs to have an infinite sum. Accordingly, an Auto-Regressive Moving Average (ARMA) model is used to represent an Infinite Impulse Response (IIR) filter as such:

$$\sum_{k=0}^{N} a_k y_{n-k} = \sum_{k=0}^{M} b_k x_{n-k} \qquad \cdots \; (1)$$

$$y_n = \sum_{k=0}^{M} b_k x_{n-k} - \sum_{k=1}^{N} a_k y_{n-k} \qquad \cdots \; (2)$$

where a and b are sequences with lengths M and N respectively, and the coefficient a_0 is normalized as 1.

In expanding the last equations, values for x and y with subscripts less than zero are considered as zero:

$$\begin{aligned} y_0 &= b_0 x_0 \\ y_1 &= b_0 x_1 + b_1 x_0 - a_1 y_0 \\ y_2 &= b_0 x_2 + b_1 x_1 + b_2 x_0 - a_1 y_1 - a_2 \\ &\vdots \end{aligned}$$

In matrix form, the last equation is written as:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_P \end{bmatrix} = \begin{bmatrix} x_0 & 0 & 0 & \cdots & 0 \\ x_1 & x_0 & 0 & \ddots & 0 \\ x_2 & x_1 & x_0 & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{M-1} & x_{M-2} & x_{M-3} & \ddots & 0 \\ \hline x_M & x_{M-1} & x_{M-2} & \cdots & x_0 \\ x_{M+1} & x_M & x_{M-1} & \cdots & x_1 \\ \vdots & \vdots & \vdots & & \vdots \\ x_P & x_{P-1} & x_{P-2} & \cdots & x_{P-M} \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ y_0 & 0 & 0 & \ddots & 0 \\ y_2 & y_0 & 0 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ y_{N-2} & y_{N-3} & & \ddots & 0 \\ \hline y_{N-1} & y_{N-2} & y_{N-3} & \cdots & y_0 \\ y_N & y_{N-1} & y_{N-2} & \cdots & y_1 \\ \vdots & \vdots & \vdots & & \vdots \\ y_{P-1} & y_{P-1} & y_{P-2} & \cdots & y_{P-N} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix}$$

more compactly:

$$y = Xb - Ya \qquad \cdots \; (3)$$

or

$$y = Wc \qquad \cdots \; (4)$$

where:

$$W = [X \quad -Y] \quad and \quad c = [b^T \quad a^T]^T$$

This system of equations is over-determinate because it has more rows (P) than unknown variables (M+N+1). Therefore, the least-squares method is adopted as a solution by multiplying each side of the equation by the transposed matrix:

$$W^T y = W^T Wc \qquad \cdots \ (4)$$

$$c = \left[W^T W\right]^{-1} W^T y \qquad \cdots \ (5)$$

The digital filter to be found can be represented by the coefficients of vector *c* which is, in turn, reliant on matrices X and Y. However, the matrix expansion of Eq. 2 contains inaccuracies which result from the fact that the initial boundary values entail missing numbers which have been replaced by zero. A more accurate calculation of the vector *c* would therefore be based on the modified matrices, X' and Y', instead of X and Y. The modification entails disregarding initial rows in the matrices which contain missing values. The disregarded rows are enclosed in a dashed boundary in the above matrix representation and the resulting matrices are modified as follows:

$$X' = \begin{bmatrix} x_M & x_{M-1} & x_{M-2} & \cdots & x_0 \\ x_{M+1} & x_M & x_{M-1} & \cdots & x_1 \\ \vdots & \vdots & \vdots & & \vdots \\ x_P & x_{P-1} & x_{P-2} & \cdots & x_{P-M} \end{bmatrix} \qquad Y' = \begin{bmatrix} y_{N-1} & y_{N-2} & y_{N-3} & \cdots & y_0 \\ y_N & y_{N-1} & y_{N-2} & \cdots & y_1 \\ \vdots & \vdots & \vdots & & \vdots \\ y_{P-1} & y_{P-1} & y_{P-2} & \cdots & y_{P-N} \end{bmatrix}$$

and $W = \left[X' \quad -Y'\right]$ is used in equation 5.

Accordingly, by finding *c*, vectors a and b which characterize the digital filter are determined. Having determined the system which relates the output values y to input values *x* by using known values for *x* and *y*, we can therefore apply the resulting filter to predict the response to any other input signal *x*. The flow chart in Figure 30 summarizes the process of applying a digital filter to predict floor responses.

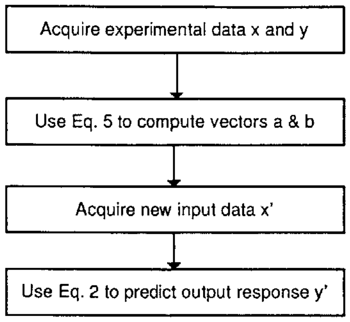

Acquire experimental data x and y
Use Eq. 5 to compute vectors a & b
Acquire new input data x'
Use Eq. 2 to predict output response y'

Figure 30. Flow chart of transfer function application

The peak frequencies of the system can be extracted by obtaining the transfer function. This is done by writing the recursion equation in the frequency domain (see Figure 31). The poles of the transfer function are obtained by finding the roots of polynomial A in the denominator of equation 7.

$$A(\omega).Y(\omega) = B(\omega).X(\omega) \qquad \cdots \quad (6)$$

$$Y(\omega) = \frac{B(\omega)}{A(\omega)} X(\omega) = H(\omega).X(\omega) \qquad \cdots \quad (7)$$

where X, Y, A and B are the Fourier transform complex vectors of x(t), y(t), *a* and *b* respectively.

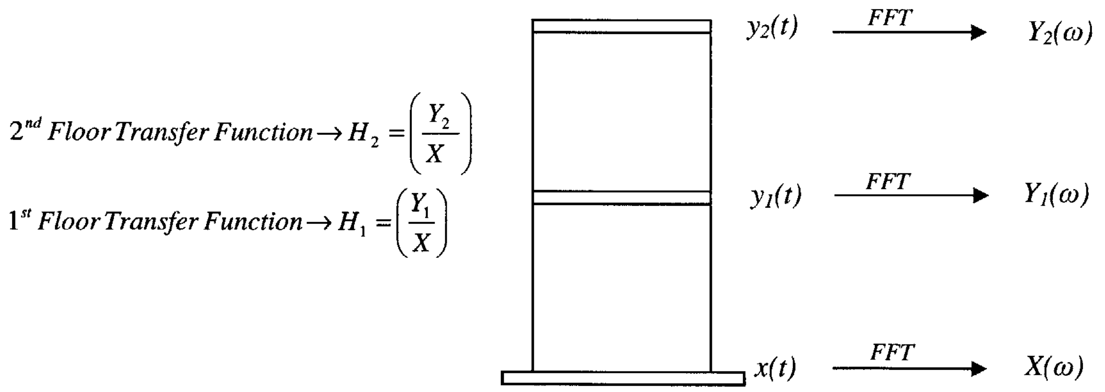

2^{nd} Floor Transfer Function $\rightarrow H_2 = \left(\frac{Y_2}{X}\right)$
1^{st} Floor Transfer Function $\rightarrow H_1 = \left(\frac{Y_1}{X}\right)$
$y_2(t) \xrightarrow{FFT} Y_2(\omega)$
$y_1(t) \xrightarrow{FFT} Y_1(\omega)$
$x(t) \xrightarrow{FFT} X(\omega)$

Figure 31. Frequency-domain representation of transfer functions.

6.4 Coding Digital Filter Computations

This section entails the signatures of functions implemented in the C# language to carry out transfer function computations. Accordingly, when an experiment is stopped and the client is disconnected from the server, the following functions are executed synchronously after a client has acquired a complete set of acceleration readings:

1. *Compute Complete Fourier Transforms*:

   After accumulating full acceleration streams, a complete FFT spectrum is calculated and plotted for each of the three levels of the test structure responses. Since the time-frame of an experiment is not predefined, as it may vary from a few seconds up to several minutes, the size of the FFT spectra is not predetermined. Since the FFT algorithms[11] used are based on 2^n data points, it is necessary to pad the gathered acceleration time histories with zeros up to the nearest 2^n number before calling the function that computes the Fast Fourier Transforms.

2. *Compute Transfer Functions*:

   The method `PlotTransferFunctions()` is implemented to compute the transfer functions from the ground-level input accelerations to each of the two upper-floor levels. Accordingly, `PlotTransferFunctions()` would, in turn, call `ComputeTransferFunction` twice, each time supplying the input parameters corresponding to a different floor. The signature of `ComputeTransferFunction` is as follows:

```
TransferFunction ComputeTransferFunction(double[] Input,
                        double[] Output, out ArrayList PeakFreq){…}
```

   `ComputeTransferFunction` takes as input two double arrays: `Input` is the array containing table accelerations while Output contains one of the floor accelerations. The third parameter `PeakFreq` serves as an output container of the peak frequencies in the transfer function. The final result of this method would be stored into a user-

[11] The FFT algorithm implemented in C# is depicted from a C-based algorithm by Press et al, Numerical Recipes, Chp. 12 Fast Fourier Transforms.

defined struct named *TransferFunction* and would entail all the characteristics of the digital filter. The *TransferFunction* struct contains the following parameters:

- two double arrays, namely, *a* and *b*, which contain the transfer function coefficients
- an array of the Complex class, *TFComplex*, to store the complex form of the computed transfer function

A struct is implemented for storing the transfer function information instead of a class. This is mainly due to the fact that this struct is not expected to inherit other classes and thus it would be more efficient storing the transfer function data as a value type.

Two additional functionalities of `ComputeTransferFunction` are:

a- Generating the frequency spectrum of the transfer function through `ComputeTFSpectrum` as:

```
void ComputeTFSpectrum(ref TransferFunction TF, int nn){…}
```

The TF object in the above method call should already contain its coefficient vectors *a* and *b*. The integer *nn* is the number of points in the time domain which is to be used for generating a spectrum with a matching size.

b- Finding the peak frequencies of the spectrum through the `roots` method as:

```
string roots(Complex[] a, int m, ref Complex[] roots, int polish)
```

This method finds the roots of the polynomial:

$$p = a[0] + x.a[1] + x^2.a[2] + \ \dots \ + x^m.a[m]$$

where the array *a* contains the coefficients of vector *a* of the transfer function described in the previous section.

3. *Compute Predicted Accelerations*:

Transfer functions calculated in the previous step are used to generate predicted floor accelerations through the following method:

```
double[] ComputePredictedAccel (TransferFunction TF,
                                                 double[] InputAccel)
```

`TF` shall contain its coefficient vectors *a* and *b*, and `InputAccel` is an array of the table's accelerations as an excitation signal. The returned array contains the values of the predicted floor response corresponding to transfer function `TF`.

Chapter 7

SUMMARY AND FUTURE WORKS

7.1 Summary

This thesis presented various aspects of the design and implementation of a shake table remote laboratory. The Shake Table WebLab provides an online space for the real-time interactive control of a shake table. I discussed the effectiveness of online experimentation in general and focused on the deliverables of the project. The goal of the project is defined from the perspective of transforming the initial system into desired final operation conditions. Accordingly, various alternatives are evaluated before detailing the implemented solution. Since the implemented system is based on the MS .Net Framework, a concise explanation of various aspects of this framework is presented. The overall system architecture, its various components and detailed implementation are then elaborated together with generic features that are of valuable use in other similar remote laboratories.

7.2 Future Works

While the outcome of this project work is a fully functional remotely-controllable shaker table, there are several aspects of this web lab, more specifically as a shaker table experiment, that may be given further consideration. Based on my involvement throughout the course of this project, I present below areas which are prospects for further development and research. Future work aspects are separated into two categories: technical (through software and hardware improvements) and collaborative (with other research groups).

7.2.1 Improving Software & Hardware Components

Means to enhance the current architecture entail adding software components as well as experimenting with innovative sensor devices. Upgrades to the Shake Table WebLab are possible as additional course-specific applications can be easily added to the already implemented web application. Since the foundation of a remotely accessible system is developed, future works are aimed at processing output data as the structure responds to different vibrations and therefore allow students to compare theoretical means of understanding structural dynamics with real-case applications. Moreover, an easily accessible shaker table with a sensor-intensive application provides a suitable space for experimenting with real-time signal processing algorithms. A constructive improvement of the current software entails providing an API for others to use from within their own programs in order to perform specific data processing tasks such as the estimation of damping ratios. As to the hardware setup itself, future efforts may entail experimenting with wireless sensors – including applications of smart materials like shape memory alloys and piezoelectric wafers - as well as various model structures. For that purpose, the current framework would have to be scaled-up to support more sensors depending on the complexity of newly developed model structures.

7.2.2 Collaborating with Ongoing Research Initiatives

Extending the functionality of the already established web lab is more rewarding when conducted in collaboration with ongoing research efforts in the areas of both web-accessible laboratories and earthquake engineering. Concerning ongoing projects focusing on web laboratories, the MIT iLab project is an ongoing initiative that aims at a shared architecture providing a backbone for web-based laboratories. Accordingly, adapting the current Shake Table WebLab with the iLab shared architecture requires rethinking administrative and accessibility features in the light of what the iLab team provides in terms of lab-specific versus brokerage services.

As to exchanging research experience with other earthquake engineering groups, the Network for Earthquake Engineering Simulation (NEESgrid) is one consortium that aims at advancing the study of earthquake engineering by enhancing collaboration among various institutions. The goal is to provide an infrastructure so that various earthquake engineering research groups, experimenting with either physical or numerical simulation, can take advantage of the results of already accomplished experiments. In addition, a shake table has several experimental benefits in understanding civil engineering phenomena under seismic conditions. Besides structural testing, examining geotechnical soil behavior due to seismic vibrations is another application that requires the implementation of different types of sensor devices. Liquefaction, for instance, is a suitable phenomenon to be further examined with the use of a shaker table.

7.3 Lessons Learned

Determining what is required of the new Shake Table WebLab has been conducted in close collaboration with Eduardo Kausel, professor of structural engineering, and Kevin Amaratunga, professor of information technologies. However, setting a predefined set of requirements prior to development had necessitated some flexibility in the system architecture that accommodates future extensions. Flexibility allows implementing ideas that were not included in the primary deliverables of the web lab. Following are some important aspects that have been stressed throughout the course of the implementation:

- Using a variety of software suites has proven efficient in delivering a fully-functional system in a short period of time. This was possible by being able to make use of more than one programming language and different programming environments. To illustrate, though most of the newly developed components are .Net-based and were developed in C#, reusing previously written Java classes, without their conversion to C#, was very productive in the development phase. In addition, being unrestrictive as to a programming platform allows for more efficient handling of programming tasks. For example, software components that handle the graphical user interface were developed in Visual Studio .Net which provides sufficient efficiency for programming graphical objects. Conversely, programming algorithmic components that require invoking a considerable number of mathematical functions is much easily implemented in Matlab which is more suitable for mathematically-intensive programs.

- Of course, using several programming platforms, such as MS .Net and Matlab, requires proper distribution of programmed tasks between client-side and server-side execution. The aim is to avoid burdening users with demanding requirements that are essential for accessing the system. Accordingly, in order to avoid having users install Matlab on their computers, all Matlab functions are executed on the server. A client-side program is capable of invoking those functions via a Web Service hosted on the same server. In this case, instead of requiring a Matlab license for each user, only one Matlab license is needed for the server installation.

- The Shake Table Weblab hasn't emerged in its final condition from a ground up approach. The fact that there was considerable software that we reused in implementing the new architecture had both its advantages and drawbacks. As more elaborately explained in Chapter 3, the new architecture is based on software components originally accompanying the hardware setup. Yet incorporating those components without further modification would have resulted in a much tedious and inefficient means of operating the hardware. Accordingly, it was necessary to reengineer the essential processes of originally-supplied components and transform them into ones which are adequate for incorporation in a more robust and larger-scoped system.

REFERENCES

1. Amaratunga, K., and Sudarshan, R., "A Virtual Laboratory for Real-Time Monitoring of Civil Engineering Infrastructure", *International Conference on Engineering Education*, August 18-21, 2002.

2. Box, D., and Sells, C., "Essentials .Net", Volume 1, The Common Language Runtime, Addison Wesley, Boston, 2003.

3. Caicedo, J., Betancourt, S., and Dyke, S., "Introduction to Dynamics of Structures", A Project Developed for the University Consortium on Instructional Shake Tables, Washington University in Saint Louis.

4. Khan, R., "Software Architecture for Web-Accessible Heat Exchanger Experiment", *SM Thesis, Department of Civil and Environmental Engineering*, MIT, 2002.

5. Powell, R. et Al, "Using Web-Based Technology in Laboratory Instruction to Reduce Costs", Computer Applications in Engineering Education, Wiley Periodicals Inc., Volume 10, Issue 4, 2002.

6. Press, W., Teukolsy, S., Veterling, W., and Flannery, B. (2002), "Numerical Recipes in C", Second Edition, *Cambridge University Press.*

7. Prosise, J., "Programming Microsoft .NET", p.521-560, 417-170, *Microsoft Press*, Washington, 2002.

8. Quanser Consulting Inc., "UCIST Shake Table".

9. Quanser Consulting Inc., "WinCon 3.2".

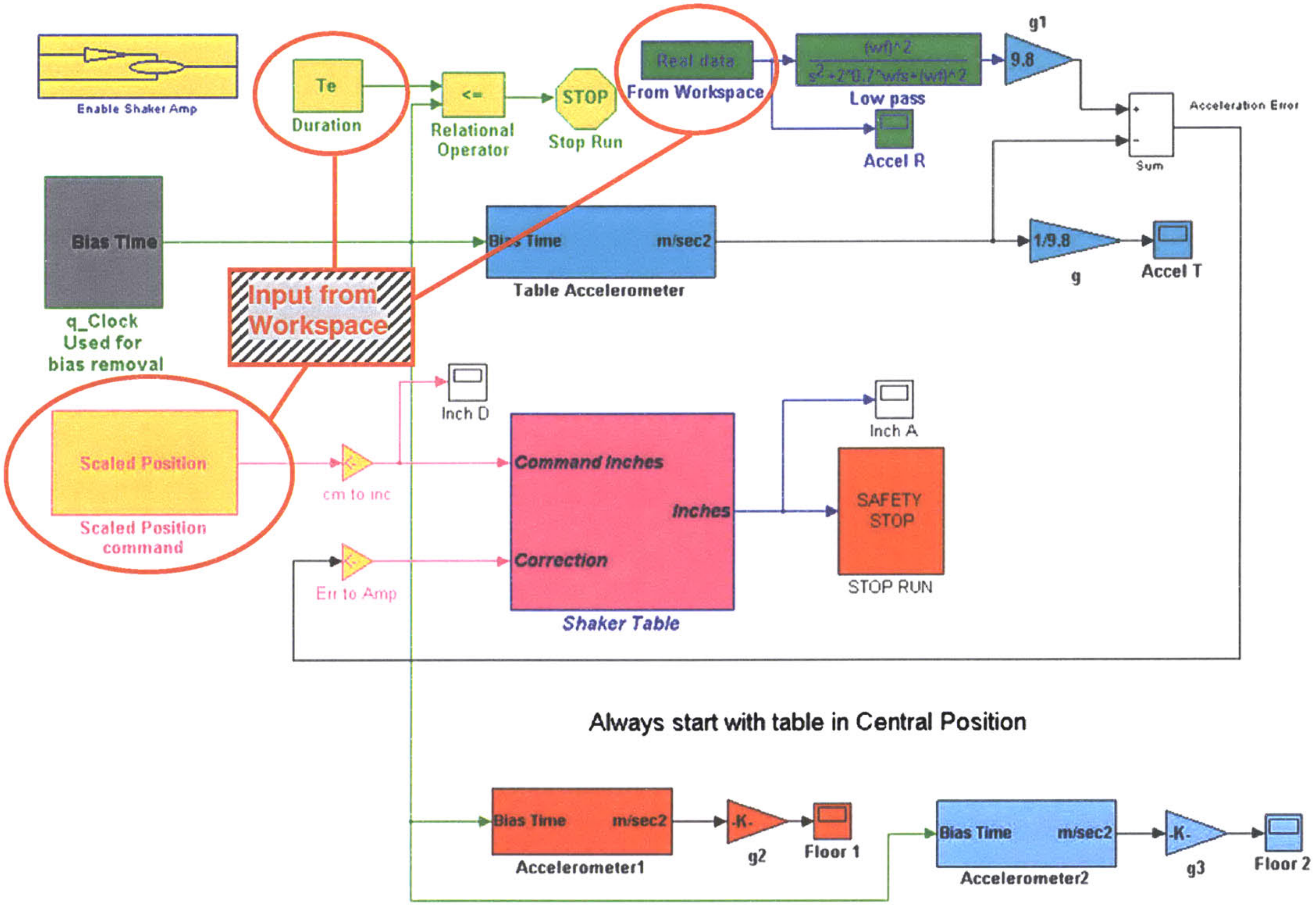
10. The MathWorks, "Simulink Model-Based and System-Based Design", Version 5, Natick, MA, 2002.
11. The MathWorks, "Simulink Model-Based and System-Based Design", Version 5, Natick, MA, 2002.

12. VentureCom Inc., "RTX 5.0 User's Guide", Cambridge, MA, 2000.

13. Wagner, B., and Tuttas, J., "Distributed Online Laboratories", *Engineering Education and Research, iNEER*, 2001.

APPENDIX

APPENDIX A – Original Sample of a WinCon Compatible Simulink Model



An experiment requires as input time related accelerations. The time interval for accelerations should be fixed. Accordingly, your data file must state the time interval (in seconds) of successive accelerations on the first line. The acceleration data should be provided on the second line onwards. One or more accelerations can be displayed on the same line. In case there is more than one acceleration value on the same line, these values must be separated by at least one space. There is no restriction as to the number of accelerations on a single line. No other information may be available in the file. The text should consist of numeric values only and saved as a .txt or .dat file.

File Format:

```
[Time Interval dt in Seconds ]
[Acc#1 @t=0] [Accel#2 @t=dt] [Accel#3 @ t=2dt]
[Acc#4 @t=3dt] [Acc#5 @t=4dt] [Accel#6 @t=5dt]
...
...
...
...
```

Sample 1:

```
0.02
134.0650    0.1000000E-02    127.8370    83.27400    -117.4720
 -242.1620    -332.3140    -407.2950    -463.2890    -487.0100
.....
.....
.....
.....
```

Sample 2:

```
0.01
7.957000
8.075000
7.882000
10.36500
.
.
.
.
```

APPENDIX C – Matlab Function for a Low-Pass Frequency Filter

```
function result = FrequencyFilter(data, dt, high)

nn=size(data,1);
if rem(nn,2)~=0
    data=data(1:size(data,1)-1);
end
nn=size(data,1);
ft=fft(data);

n=nn/2;
nf1=n+1;
MaxFreq=1/2/dt;

trim2=high*nf1/MaxFreq; %high = 20Hz

if trim2==nf1
    trim2=nf1-1;
end

%Creates a lowpass filter
b=fir1(48, trim2/nf1, 'low');
[h]=freqz(b,1,nf1);

ft=ft(1:nf1);
ft=ft.*h;

ft=[ft;conj(ft(nf1:-1:2))];
result=real(ifft(ft));
```

Parameter Definition:

Input:

- *data* is the array of numeric values to be filtered.
- *dt* is the time interval between two consecutive values in *data.*
- *high* is the upper limit frequency value in the applied filter.

Output:

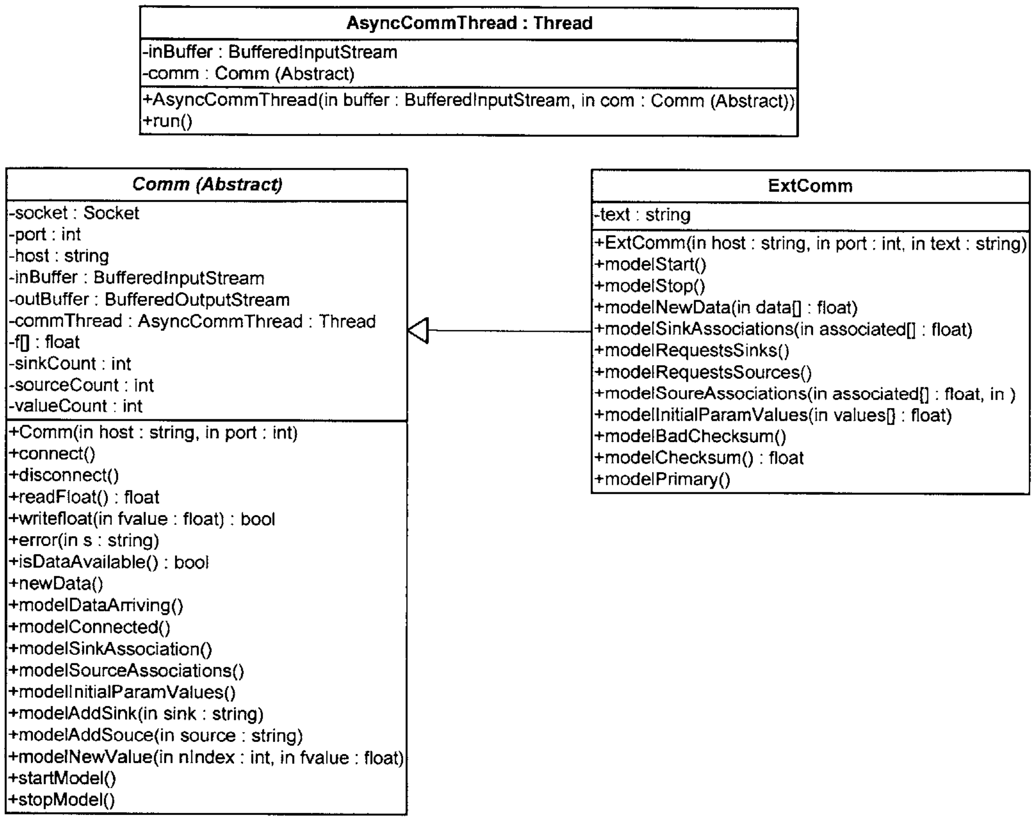
- *result* is the array of filtered numeric values.

Table 4. Command-based protocol for WinCon communication with an Outside Application.

Command	Name	Direction	Data
1	NEW_DATA	EIW → OA	Vector of values
2	NEW_VALUE	OA → EIW	Index into associated sources followed by new value
3	MODEL_STARTED	EIW → OA	None
4	MODEL_STOPED	EIW → OA	None
5	SINK_ASSOCIATIONS	EIW → OA	Number of associated sinks followed by vector indices
6	SOURCE_ASSOCIATIONS	EIW → OA	Number of associated sources followed by vector of indices
7	EXTERAL_CHECKSUM	OA → EIW	Checksum value
8	BEGIN_SINK_LIST	OA → EIW	None
9	NEW_SINK	OA → EIW	Size of sink name followed by character string
10	END_SINK_LIST	OA → EIW	None
11	BEGIN_SOURCE_LIST	OA → EIW	None
12	NEW_SOURCE	OA → EIW	Size of source name followed by character string
13	END_SOURCE_LIST	OA → EIW	None
14	CONNECTION_STATUS	EIW → OA	Connection status value
15	BAD_CHECKSUM	EIW → OA	None
16	START_MODEL	OA → EIW	None
17	STOP_MODEL	OA → EIW	None
18	INITIAL_PARAM_VALUES	EIW → OA	Number of parameters followed by vector of values
19	EXT_CMD_INVALID	Internal	None

Source: WinCon 3.2, Quanser Consulting Inc.

APPENDIX E – Object model for establishing communication between an OA and WinCon


AsyncCommThread : Thread
-inBuffer : BufferedInputStream
-comm : Comm (Abstract)
+AsyncCommThread(in buffer : BufferedInputStream, in com : Comm (Abstract))
+run()
Comm (Abstract)
-socket : Socket
-port : int
-host : string
-inBuffer : BufferedInputStream
-outBuffer : BufferedOutputStream
-commThread : AsyncCommThread : Thread
-f[] : float
-sinkCount : int
-sourceCount : int
-valueCount : int
+Comm(in host : string, in port : int)
+connect()
+disconnect()
+readFloat() : float
+writefloat(in fvalue : float) : bool
+error(in s : string)
+isDataAvailable() : bool
+newData()
+modelDataArriving()
+modelConnected()
+modelSinkAssociation()
+modelSourceAssociations()
+modelInitialParamValues()
+modelAddSink(in sink : string)
+modelAddSouce(in source : string)
+modelNewValue(in nIndex : int, in fvalue : float)
+startModel()
+stopModel()
ExtComm
-text : string
+ExtComm(in host : string, in port : int, in text : string)
+modelStart()
+modelStop()
+modelNewData(in data[] : float)
+modelSinkAssociations(in associated[] : float)
+modelRequestsSinks()
+modelRequestsSources()
+modelSoureAssociations(in associated[] : float, in)
+modelInitialParamValues(in values[] : float)
+modelBadChecksum()
+modelChecksum() : float
+modelPrimary()

(Source code by Quanser Consulting Inc.)