Development of a Secure, Low-Cost, Web-Based Remote Technology Demonstrator© (WebRTD©)

Damian Barnes

Master of Science Thesis in Computer Science

July, 2004

Development of a Secure, Low-Cost, Web-Based Remote Technology Demonstrator© (WebRTD©)

Damian Barnes

Abstract

The Internet has acted as an effective catalyst for remote demonstration and experimentation in recent years through the medium of virtual laboratories. These laboratories can be especially useful in science and technology education, allowing topics to be presented to a geographically dispersed audience. There has also been a rapid proliferation in recent times of transactional systems for e-commerce applications. Similar to virtual laboratories, these online applications extend their content to a wide audience but normally achieve this using a different set of technologies. Commercial application servers are often used in the development of e-commerce solutions and provide features such as transaction management, extensibility and security. This thesis presents a new virtual laboratory which combines conventional virtual laboratory technologies with these useful application server features, producing a system which is known as Web-based Remote Technology Demonstrator©[1] (WebRTD©). The new system is potentially capable of managing a number of technology demonstrations using the one user interface.

A Stirling Engine demonstration was selected as the first application for the new system. This demonstration will provide the user with a basic understanding of the operation of the engine as well as communicating the thermodynamic principals associated with type of engine. The new system allows the user to start, stop and control the engine as well as download operational performance characteristics for learning.

[1] WebRTD© is copyright of the author

Acknowledgements

I wish to sincerely thank the following for this help and assistance throughout this project:

- My supervisor, Paul Dunne, for his encouragement and guidance.
- Enterprise Ireland, without whose financial support, this project would not have been possible.
- Steve Hollice of Omega Instruments, for his technical advice and insight into appropriate hardware solutions.
- Igor Pashuta, for his invaluable input on all matters involving C++.
- Oleksandr Oliynyk, for his suggestions on network design.
- Anssi Raittila, for sharing his broad ranging knowledge on CORBA in the C++ and Java domains.
- Brian Webster, for his expert advice on documentation issues.

Also, I am greatly indebted to Dr. John Lohan, without whose assistance, this body of research would never have been realised.

Table of Contents

List of Figures

Dedication:

To my parents, Brendan and Bernadette Barnes

Chapter 1

Introduction

1.1 Motivation for Developing a J2EE-Based Virtual Laboratory

The use of the Internet has been rising exponentially since the arrival of the World Wide Web (WWW) in the early nineties and is now a ubiquitous medium for commercial, personal and educational purposes. The educational and training sectors have benefited by introducing new methodologies for extending educational opportunities to a wider audience through various forms of distance learning initiatives.

Distance learning improves access to education and can advance the quality of education delivery. Martin and Haque [1] suggested that the traditional classroom could be replaced with a more intimate virtual environment. Indeed distance learning allows education to take place where traditional institutions do not currently exist, or in areas that have few educational opportunities. In the context of distance education for the engineering discipline, the learner-centred distance learning archetype could include dynamic demonstrations of theoretical engineering models. This will allow learners to affirm, translate and improve their understanding of theoretical knowledge using real-world applications. The distance education curriculum in engineering could evolve to include the creative use of virtual technologies.

Knight and DeWeerth [2] confirm that web technologies promote efficient learning and cater for diverse learning styles. While web-mediated access to static content offers a well structured document format as defined by the World Wide Web Consortium (W3C) [3], the maturing of technologies, such as Java, introduces a new level of interactivity which hitherto was not feasible. Crisp [4] states that Java applets embedded in training course web site material, can enhance educational material with applications that are responsive to learner choices and provide interactivity to engage learners in active learning. Those involved in designing online environments, therefore, must afford special attention to interactivity.

A considerable amount of work has already been carried out leveraging new web-based technologies to create new methods of enhancing communication between teacher and learner. Specifically, remote learning methods such as those offered by virtual laboratories, have the capability to reach more learners using specialised

instruction and self-paced learning. Virtual Laboratories provide an active learning environment where a number of teaching methods may be used. These can include interactive exercises, real-time feedback from the system and graphical modelling of data generated by the system. Reisman and Carr [5] indicated that students learn and retain more when they engage with instructional materials. In their research, they concluded that students learn 20% of the material taught by hearing, 40% by seeing and hearing, and 75% by seeing, hearing, and doing. These findings should form the basis of any design work being undertaken to develop new virtual laboratory systems.

The term "virtual laboratory" has been described in the previous paragraph as an effective medium for training and instruction. However, this term is ambiguous in this area of research so it is important at this point to stipulate what constitutes a virtual laboratory. Tutas and Wagner [6] define virtual laboratories as 'software simulations of physical devices (e.g. measurement instruments) or other real life systems (e.g. economic systems)'. Guggisberg *et al.* [7] realise that the term is used in literature for two distinct purposes. The first is that used to describe a simulation of laboratory infrastructure and the online virtual physics laboratory at [8] is an example of a collection of these simulations in the public domain. The second possible purpose of a virtual laboratory is that it be used for remote control and observation of real-world equipment, often referred to as remote laboratories, and are outlined by Hsu *et al.* [9]. The latter authors describe them as web-based education environments where instructors and learners participate in learning activities while geographically separated from each other. For the purposes of this report, a virtual laboratory is defined as an online laboratory offering remote access to laboratory equipment, workbenches and all types of experiments over the Internet. Interactions with such laboratories effect real laboratory equipment and not just software simulations of physical devices.

Ko *et al.* [10] outline how a virtual laboratory may be used for conducting experiments relating to frequency modulation over the Internet. This type of application is typical of a virtual laboratory and allows expensive equipment, such as oscilloscopes and signal generators, set up by trained instructors, to be used and experienced by a greater number of learners in a controlled environment. There may be other reasons, apart from cost and available expertise, why this type of technology is useful. Amaratunga and Sudarshan [11] reveal that there may be other situations where a traditional laboratory setting is not feasible and virtual laboratories may be

the only solution. They cite analysis of civil engineering and aerospace structures as examples of such situations. In these instances, operating environments would be hazardous for carrying out traditional experiments.

Virtual laboratories should provide control of a real test system. Röhrig and Jochheim [12] state that interactive experimentation on real-world plants improve the motivation of the students and also develop an engineering approach to solve realistic problems. Simulation is a good way to complement control education but in general, it cannot replace experimentation on real plants. Also students can observe dynamic phenomena that are often difficult to explain in writing. Macias *et al.* [13] indicate that their virtual laboratory reinforced the lecture material used in the teaching of electronic engineering topics. Baher [14] found that virtual laboratories used in thermodynamics actually promoted faster and easier understanding of concepts.

It is important to clarify what is meant by a "real-time system" in relation to this project. Due to the non-deterministic nature of the Internet and Intranets, it is actually impossible to offer a true real-time control and monitoring. The use of real-time operating systems and indeed real-time enabling hardware are beyond the scope of this project. The interface to the test equipment is a *soft* real-time system which offers timely and accurate feedback of data. Aktan *et al.* [15] highlight the importance of safety for these types of applications. To this end a local application, known as the test server, is charged with providing the first line of defence in realising a safe operating environment.

Virtual laboratories are in widespread use today and have been constantly evolving since the World Wide Web appeared in the early 1990s. Technologies such as Common Gateway Interface (CGI) provided the underlying functionality in early implementations and indeed Ko *et al.* [10] use this approach relatively recently in a remote electronics laboratory.

The emergence of Java 2 Enterprise Edition (J2EE) introduced a new standard for developing multi-tier distributed applications. This new standard has gained widespread acceptance in transaction dependant applications such as online banking and mobile phone subscriber portals. Recent trends in virtual laboratories have seen certain elements of the J2EE feature set used and are typified by Amaratunga and Sudarshan's [11] civil engineering based monitoring system. This project aims to illustrate how more of these features may be used to produce a more effective virtual laboratory.

A major reason that virtual laboratories are not implemented using J2EE, is the transactional nature of its architecture. Many researchers such as Röhrig and Jochheim [12] use Java applets for real-time control and monitoring. Allowing application servers[1] to be used requires a number of issues to be addressed including integration and the fact that commercial application servers have remained prohibitively expensive. However, open source versions have now become sufficiently stable and offer a cost effective, scalable and secure platform.

Free and open source software could also be utilised in other areas of the project, thereby lowering the overall cost base. Open source software gives the user the freedom to use, copy, distribute, examine, change and improve the software. These rights are stipulated in licenses such as the GNU[2] Public Licence (GPL) [16] copyrighted by the Free Software Foundation [17]. This license is intended to guarantee freedom-to-share and freedom-to-modify, thus ensuring that it is free for all its users. Open source software is usually made available free for download from either a specialised product site such as that used by the Apache web server [18] or from a group project site such as Sourceforge [19]. This is particularly useful in educational institutes where software costs can prohibit the introduction of new concepts.

Proprietary software is sold without any access to source code and it is therefore not possible to change, improve or further distribute the software. A license for proprietary software entitles a user to only use the software under certain conditions. Becoming dependent on a proprietary software vendor can result in adverse effects because new versions or releases of the product are not always initiated by the actual needs of the user, but rather by the product cycle of the vendor or supplier. A software upgrade can also require new hardware, resulting in even higher costs.

Mockus *et al.* [20] state that proponents of open source software declare that it has the capacity to compete successfully, and perhaps in many cases displace, traditional commercial software. They conclude that hybrid forms of development that borrow the most effective techniques from the Open Source Software (OSS) world may lead to high performance software process. They indicate that the quality of OSS

[1] An application server is a server program in a distributed network that provides the business logic for an application program.
[2] Recursive acronym for "GNU's Not Unix"

can be greater than commercial software as many more developers can be available to work on a specific problem and that they are motivated enough to work on it voluntarily.

1.2 Objectives of the Research Work

The principal objective of this research is to show how e-commerce principles can be applied to, and enhance virtual laboratories. Enterprise applications have benefited greatly from structured development techniques, using J2EE as the vehicle to deliver required features. This project aims to illustrate how the J2EE feature set may be used to produce a more effective virtual laboratory. These features include transaction management, extensibility and security.

Transaction management is a feature which ensures that transactional operations are carried out fully. For instance, if an online banking application was required to perform a credit transfer, it is important that the first account be successfully debited and the second account subsequently credited. Failure to carry out either of these transactions could lead to either the purchaser or vendor loosing out financially. While virtual laboratories may never deal with monetary data, it is important nonetheless, that information such as user settings and test result data are read, as well as written correctly to secondary storage.

Extensibility is a much coveted feature in transactional e-commerce applications and virtual laboratories alike. The modular component architecture of J2EE lends itself well to the scaling of applications and promotes code re-use. Functionality may be augmented without necessarily changing and re-compiling the project source code.

A comprehensive security solution is essential for every publicly available e-commerce application. There are a number of customised and standard solutions used to address common security issues. J2EE offers a standardised role-based component security framework, known as Java Authentication and Authorization Service (JAAS), which is enforced by the application server. This feature enables restrictions to be placed on individual operations of deployed components.

Single sign-on is a popular and useful security feature used by many e-commerce applications. In a client/server relationship, single sign-on is a session authentication process that permits a user to enter one name and password in order to access multiple applications. The single sign-on, which is requested at the

initialisation of the session, authenticates the user to access all applications to which they have been granted rights, and eliminates future authentication prompts when the user switches applications during that particular session. J2EE compliant application servers typically achieve this using XML deployment descriptors to configure the authentication type (basic or form-based), leaving the developer to use component code for handling business logic exclusively. Single sign-on could be used to give the user access to a number of technology demonstrations.

The result of employing the J2EE feature set in a virtual laboratory will produce a system known as the Web-based Remote Technology Demonstrator© (WebRTD©). This new system will offer the potential of managing a number of virtual laboratory demonstrations. In this instance, the WebRTD© has been developed to demonstrate one thermodynamics-based system, a Stirling Engine coupled with a control and monitoring system. It is anticipated that this application will be of considerable benefit to engineering students learning thermodynamics, as it facilitates the demonstration of theoretical knowledge in a practical and informative manner. The first trial is this application is scheduled for the first semester of the 2004/2005 academic year.

Cost control is a secondary objective of this work. This project will show how extensively open source software solutions can be used in order to keep the system cost to a minimum.

User interface design is also an important aspect of this work. The interface should be intuitive, user friendly and promote e-learning by effectively communicating to the learner the principles and any background information on the remote demonstration.

The adaptation of J2EE into the area of virtual laboratories has not previously been undertaken. As mentioned earlier, the cost of commercial application server software remains prohibitively high. Open source versions, such as JBoss, now offer the possibility of leveraging the J2EE feature set in a similar manner to expensive transaction-based e-commerce systems. Another reason why J2EE has not been applied to virtual laboratories is that integrating a soft real-time system into a transactional system presents difficulties with regard to how the two systems can interoperate successfully. It is useful, therefore, to examine relevant virtual laboratories and transactional systems to produce a hybrid solution for the remote technology demonstrator.

1.3 Thesis Outline

The number of services provided by the standard defined by J2EE continues to grow. This research endeavours to utilise the most effective services available in J2EE, from Enterprise Javabeans to the more recently added Web Services.

In Chapter 2, existing systems, in both the virtual laboratory and transaction-based realms, are analysed to determine the most suitable elements for inclusion in the new system.

Chapter 3 describes the most relevant features of J2EE used in this project. Other chapters contain references to these features, so this chapter may serve as a primer for the technologies involved.

Chapter 4 outlines the comprehensive system analysis and design undertaken in order to ascertain the correct approach to realise the system requirements. This work uses Sun Microsystems' blueprint design patterns [21] which provide a template for creating J2EE solutions using proven methodologies improving such facets as performance and scalability.

All content related to the implementation of the new system is presented in Chapter 5. The appropriate network protocols for control and monitoring of active test cycles are determined and applied to the soft real-time element of the application. Also the architecture of the user management system is defined using J2EE services. The operation of the new system, comprising of the J2EE-based and real-time elements, is presented.

The outcome of the new system is presented in Chapter 6 and the success of applying J2EE technology to a virtual laboratory is determined. Future directions for the system are also briefly explored in this chapter.

The appendices aim to provide a supplement regarding the technologies used in this thesis. Appendix A provides a glossary of terms. Appendix B and C present an overview of the fundamentals of Stirling Engines and the calculations involved when calculating the overall thermal efficiency respectively. Appendix D details the operation of the test server used to control and monitor the Stirling Engine used in the demonstration. Appendix E contains the contents of the principal XML deployment descriptor file required to deploy the Enterprise Javabeans.

Chapter 2

Analysis of Relevant Web-Based Interactive Systems

2.1 Introduction

In order to design an effective virtual laboratory, it is important to examine existing web-based systems with a view to assessing the advantages and disadvantages of their respective technologies. This chapter focuses on two groups of interactive systems which deserve particular attention, these are;

Existing Virtual Laboratories

Current virtual laboratories will give a good indication of the most appropriate technologies for developing a remote technology demonstrator.

Transactional Systems

A significant element of the system will be based on a transaction-based model so therefore it is useful to examine previous online transactional systems.

2.2 Existing Virtual Laboratories

Chen *et al.* [22] present a remote Frequency Modulation laboratory built using the hardware elements shown in Figure 2.1.

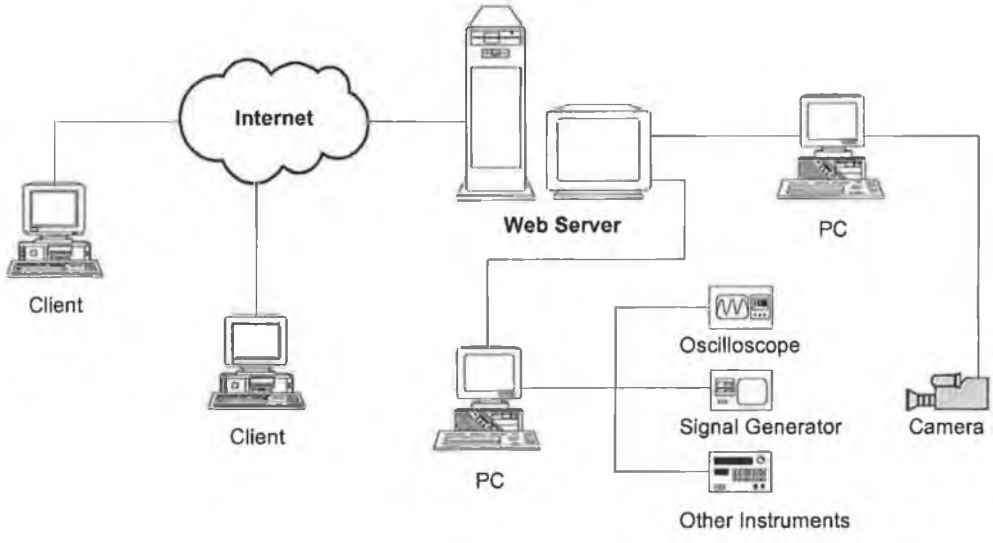

Figure 2.1: Remote Laboratory Hardware Elements (Source: Chen *et al.* [22])

This laboratory was designed using a 'double client-server architecture' (see Figure 2.2). This consists of a pair of interconnected client-server systems, the web server acting as both client and server. Ko *et al.* [10] also describe the design of a similar virtual laboratory.

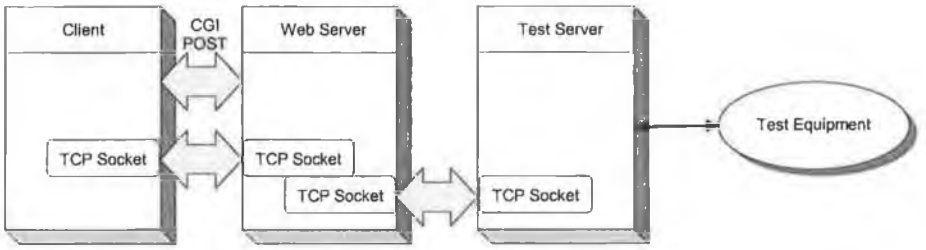

Figure 2.2: Double client-server architecture (Source: Ko *et al.* [10])

In their work Chen *et al.* [22] describe how the web browser communicates with the web server initially via Common Gateway Interface (CGI) and thereafter via Transmission Control Protocol (TCP) using a Java applet. The applet displays graphical representations of oscilloscopes and other scientific equipment and uses TCP to issue commands and read feedback data from the server. The web server uses yet another TCP connection to relay data to the test server. The latter connection is not seen by the client.

The initial user authentication is implemented using a Hypertext Transfer Protocol (HTTP) POST request. This is achieved using another TCP connection to write to the web server itself.

Control of the test equipment at any one time is restricted to one client. This is managed using a token system. This system is operated on a first-come first-served basis where the client must first obtain a valid session ID in order to issue commands to the test server. After having gained control of the test server, other users must wait until it becomes available again.

If this configuration were to be used in the proposed virtual laboratory, it would be preferable that others could at least observe the current process. Also, there is perhaps a potential security risk to the TCP servers in that there may be a threat from denial-of-service attacks, which may have unpredictable consequences for expensive test equipment.

Significant emphasis is placed on the facility having a real-time feedback display, realised by using a video camera relaying data through a video server. This is

an important feature of this particular application as the user's experience of controlling the equipment will be enhanced and indeed the user may be in a better position to react to the state of the equipment based on the visual feedback. While the proposed virtual laboratory may not implement the double client-server design as seen here, the concept of a de-coupled configuration does render the system more flexible and extensible. Specifically, it allows new demonstrations to be added just by changing the test server application and hardware.

One essential requirement of this type of test server, is that it be capable of using native libraries to control the test equipment using interfaces such as RS-232 and General Purpose Interface Bus (GPIB).

Aktan *et al.* [15] describe a virtual laboratory that enables a user to remotely control a robot arm. A client-server structure is used, but the client is required to properly install X-terminal software[1] first. Subsequent communication between the client and server is realised using User Datagram Protocol (UDP). According to Comer [23] UDP offers an unreliable delivery service. He adds that applications using UDP must take full responsibility for handling the problem of reliability, including message loss, duplication, delay, out-of-order delivery, and loss of connectivity. Comer also points out that developers often test network software using reliable, low-delay local area networks so testing may not expose potential failures. Thus, many application programs that rely on UDP work well on a local environment but fail in dramatic ways when used in a larger TCP Internet.

However, one potential benefit of using UDP is that, if the test equipment is already being controlled, then the possibility exists for others to observe the experiment. Safety concerns regarding the remote operation of a robot arm are highlighted and the authors clearly indicate that safety should be given the highest priority at the design stage.

UDP operates at a very low-level and is not suitable for issuing test server commands. TCP also provides a very primitive means of transferring data and particular care must be taken when passing parameters. Wang and Robinson [24] describe a high performance engineering computing portal based on Common Object Request Broker Architecture (CORBA). Their implementation uses a three-tier architecture, which includes Java applets, servlets and CORBA. The use of CORBA

[1] X-terminal software is used to emulate user interfaces from other operating systems.

enables language independent interoperability between multiple platforms. Orfali and Harkey [25] determine that the performance of CORBA implementations (or ORBs), combined with their greater usability make them a better alternative over socket programming. CORBA hides the low-level detail of making network connections and provides clearly defined error handling in the form of exceptions.

Ko *et al.* [26] recognise the limitations of the TCP protocol used and subsequently re-developed their electronics-based virtual laboratory. They improved on their Frequency Modulation remote experiment by using IP multicast. This protocol is a bandwidth-conserving technology where one copy of data is sent to a group or class D IP address. Each user may read from this address thereby eliminating the need for multiple connections to the server to be maintained. It is designed to be very scalable, as an increase in the number of remote users does not result in a corresponding increase in bandwidth. It is, however, built on top of UDP, which is an unreliable protocol. Data packets may arrive out of order or not arrive at all. These limitations should be considered at the design stage.

After analysing a number of virtual laboratory implementations, it is evident that there are a number of technologies that have already proved very effective in fulfilling such requirements as scalability and performance. However, with regard to the design of virtual laboratories, a structured approach to their design has not been formulated. This is due to a large degree to the wide range of criteria of these systems.

2.3 Transactional Systems

The next web-based systems that were studied, were broad-based transactional systems, in particular those built on the J2EE platform. Typical examples of these include online banking portals and e-commerce auction sites such as eBay [27].

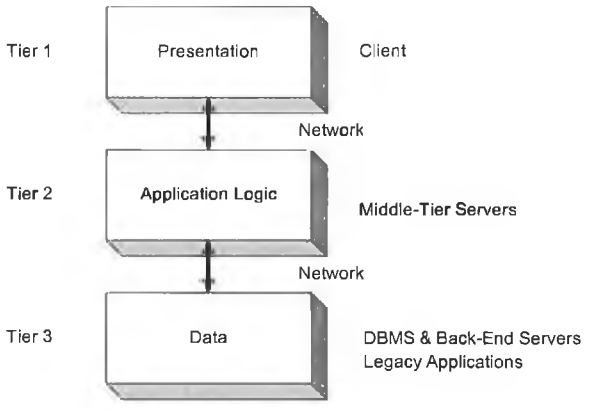

Figure 2.3: 3-Tier Client/Server Application Architecture (Source: Pour *et al.* [29])

Soldar *et al.* [28] argue that, if the number of e-commerce services are to continue to expand at their current rate, then the underlying technologies will need to change to provide greater flexibility, scalability and integration with enterprise systems. This would suggest that this approach should also be applied to the technology and educational sectors, in particular to web-based virtual laboratories. In the case of the proposed virtual laboratory, a generic framework should be devised to allow new test applications to integrated into the system while providing access to existing ones, thereby creating a technology portal. A shift from the traditional 3-tier architecture to one created using J2EE services is advocated. Pour *et al.* [29] outline the component groups in a 3-tier architecture, as shown in Figure 2.3. Presentation components operate in tier 1, application logic components in tier 2 (i.e. middle tier), and data components operate in tier 3.

The J2EE model presents a multi-tier architecture comprising of such services as servlets, JDBC and Enterprise Java Beans (see Figure 2.4).

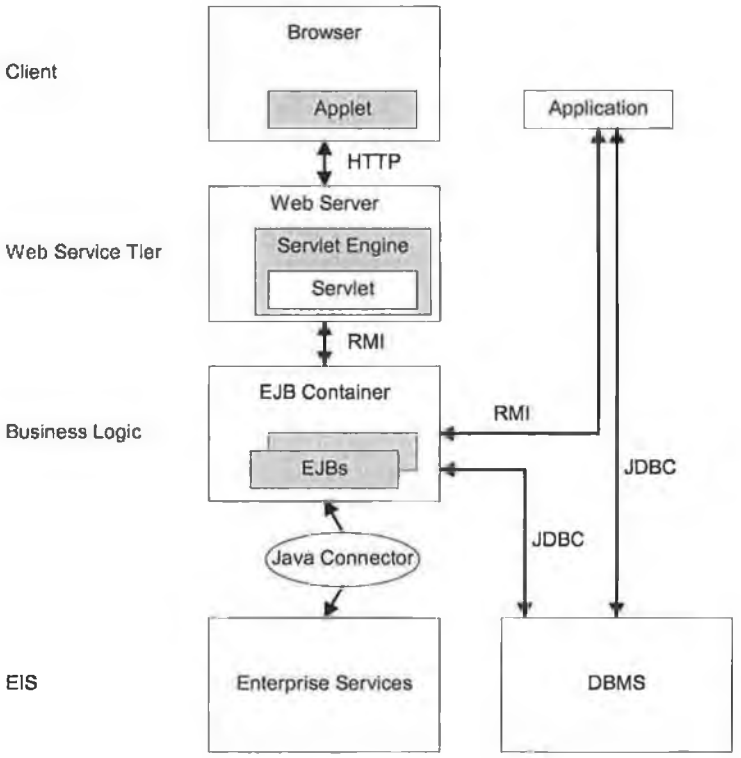

Figure 2.4: J2EE Multi-Tier architecture (Source: Soldar *et al.* [28])

Soldar *et al.* [28] take the position that the success of e-commerce is tightly related with the development of the Internet technologies, especially those based on platform independence. One of the key components for achieving this is the use of ubiquitous, low cost communication networks using Internet technologies and standards such as J2EE.

There are two traditional paradigms for software development. At one extreme, the project is developed entirely from scratch, with the help of only programming tools and libraries. At the other extreme, a commercial-off-the-shelf solution is purchased and configured to provide an approximate or "best-fit" solution. Pour [30] presents Component-Based Enterprise Software Engineering (CBESE) as a rapidly emerging trend in the software engineering area. This concept of component software represents the middle path, where an entire application is assembled from individual atomic components, developed by third parties.

The architecture of component-based systems is often significantly more demanding than that of traditional monolithic integrated solutions. To make development of efficient component-based applications a feasible task, a component model incorporates "best practice" designs by providing developers with design patterns, suggesting a standardised structure upon which distributed component-based systems should be based.

An important reference application is 'Java Pet Store' [31], a best-practices sample application from the "Java Enterprise Blueprints" program maintained by Sun Microsystems. It represents a typical e-commerce application. Customers can browse through a catalogue of products, select items of interest and place them in a shopping cart. Upon indicating readiness to buy what is in the shopping cart, the application displays a bill detailing prices and quantities. The customer can also create a permanent account with the on-line shop, which includes billing and shipping information.

Rice University Bidding System (RUBiS) [32] is an auction site prototype modelled after the popular e-commerce web-portal, eBay [27]. RUBiS implements the core functionality of an auction web site: selling, browsing and bidding on items. Visitors can search through a catalogue of items divided into several categories and belonging to different geographical regions. They can bid on items of interest, as well as put comments for other users. Users may also choose to sell an item, registering it and specifying several parameters, such as action duration, and initial, reserve and

buy-now prices. All non-browsing activities require creation of a permanent account on the web site before logging in. Candea *et al.* [33] use the popular open source application server, JBoss [34], for their RUBiS implementation.

Chambers *et al.* [35] design and develop a fully featured library management system using J2EE. The benefits of this approach are significant. Again using JBoss, it is possible to deliver a secure, cost-effective but yet easy to use application, which does not require specialised knowledge to maintain. The versatility of the system was proved by providing a web-based library catalogue search facility as well as an application for administration purposes. Figure 2.5 illustrates how the server program serves both of these clients.

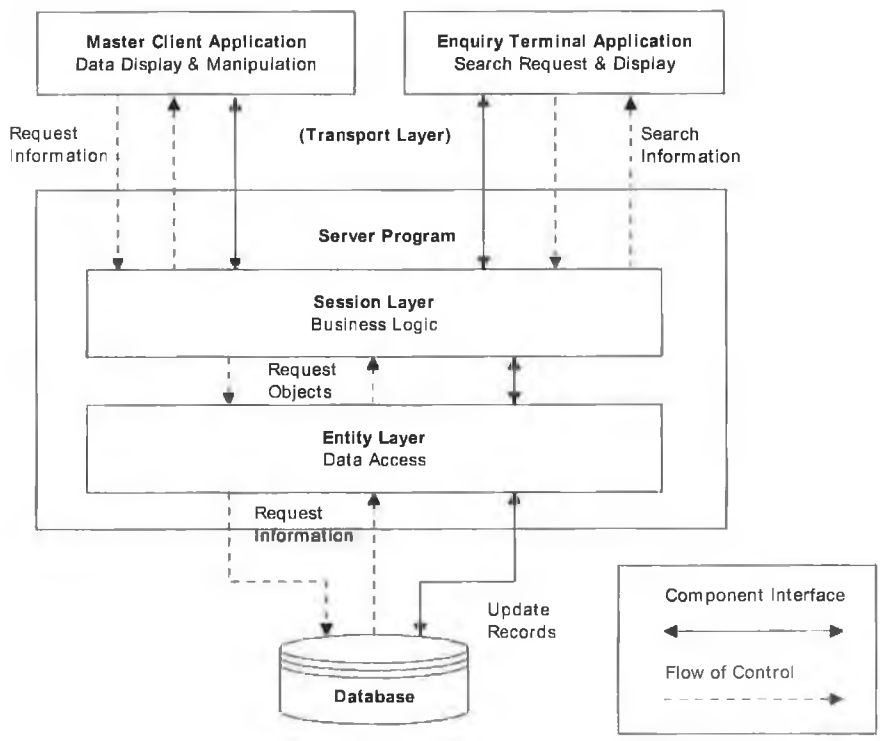

Figure 2.5: Multi-Tier Architecture Used by the Scots Library Management System (Source: Chambers *et al.* [35])

The enquiry terminal application is a Java servlet-based search facility, which obtains persistent entity bean data through the session layer. The session and entity layers are implemented using EJB[2]. The master client application is a Java Swing-based GUI application and communicates directly with the session layer using the application-based authentication.

[2] A description of these and other J2EE technologies is given in Chapter 3.

EJB is a good example of component-based enterprise software. Pour [29] gives ActiveX/Microsoft Transaction Server (MTS) as another example of a component-based solution. Indeed, Borges *et al.* [36] use ActiveX technology for their client user interface in a virtual laboratory and Pasquarette [37] describes a technique for creating virtual instruments using ActiveX technology. While this approach may be relatively straightforward, it relies on using proprietary software which is platform dependent. Java, on the other hand, while not open source, is available for download "royalty-free" under Sun Microsystems' licensing terms. This not only makes it more cost-effective, but also makes it possible for third party development teams in the open source community to provide useful additional products.

Another major difference between Java-based and ActiveX technologies lies in the approach that these two technologies have taken to address security of components. ActiveX relies only on the digital signature verification method while Java solutions provide higher level of security through a combination of security managers built into the language, digital signature verification, and the concept of trusted servers. In addition, ActiveX technology has been declining in popularity in recent times.

Allamaraju *et al.* [38] describe servlets as the basic building blocks of web applications. They provide a common programming model that is also the foundation for Java Server Pages (JSPs). Servlets, like CGI and sockets, provide a primitive form of middleware in client/server architecture. Like CGI scripts, they are executed on the server. However, instead of creating a new process for each request, all servlet requests are handled in the same process by separate threads. Therefore, servlet-based applications avoid overhead processing by eliminating the cost of starting a new process for each client connection. Additionally, servlets provide a Java-based solution that addresses the problems associated with server-side programming (i.e. non-extensible scripting solutions, platform-specific APIs, and incomplete interfaces). Servlets deliver the same functionality as CGI scripts but they are faster, cleaner, and easier to use. In addition, servlets provide lower deployment and maintenance costs.

Web Services are a relatively new addition to the J2EE standard and offer great potential in integrating dissimilar applications. Schattkowsky and Müller [39] describe how a novel system that uses Simple Object Access Protocol (SOAP), can greatly enhance Computer Supported Collaborative Work (CSCW). They provide an

example in the area of electronic design automation where existing legacy tools may be integrated into distributed workflows. They highlight not only the tool management and integration, but also the secure transport of design data as being of the utmost importance.

Supply Chain Management (SCM) plays a key role in the success of companies in the today's highly competitive global business environment. SCM is about managing the flow of goods, services, and information among suppliers, manufacturers, wholesalers, distributors, stores, consumers, and end users. Pour and Guo [40] present a new Web Service orientated architecture which supports the process integration and flexibility in SCM. The system encapsulates the details of application logic while providing consistent, secure and auditable access to a wide variety of software services needed in a supply chain. The system is independent of language, platform, and location and caters for the wide variety of software services needed. Figure 2.6 shows a related sequence diagram showing the interaction between customers and suppliers on the SCM system.

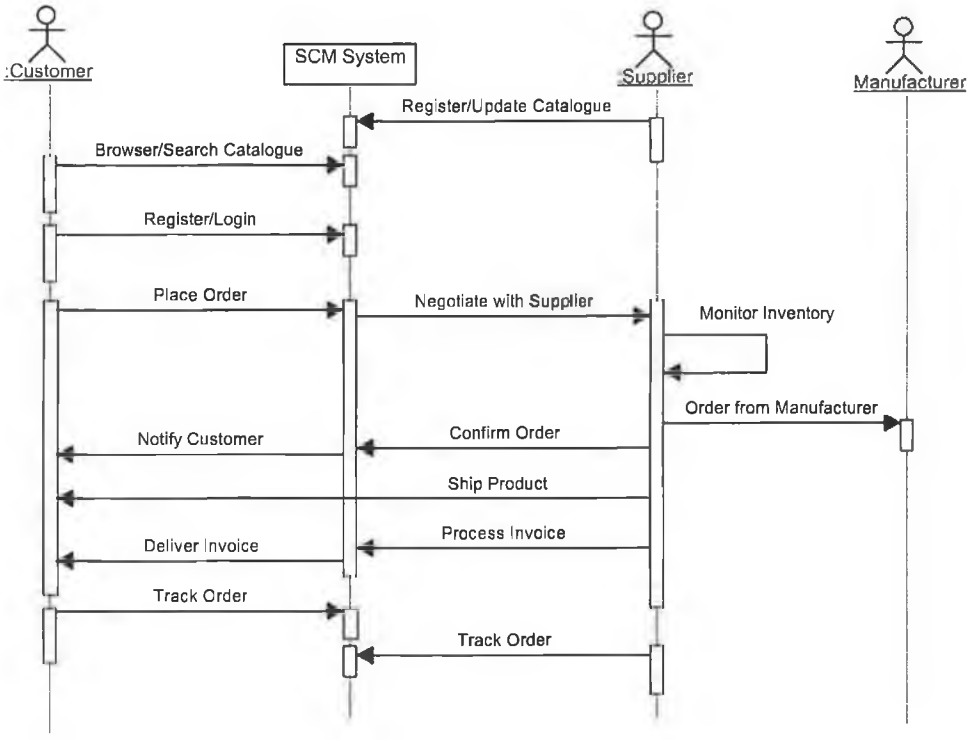

Figure 2.6: Related Sequence Diagram for Supply Chain Management System
(Source: Pour and Guo [40])

The architecture of the SCM system consists of three main tiers; presentation tier, middle tier, and data management tier. In the presentation tier, JSP and servlets are used for requests from client browsers to Web Services. The middle tier is implemented using JNDI, Java Message Service (JMS), JDBC, JavaMail, and Java API for XML Parsing (JAXP). A UDDI server was set up using a SOAP container to allow electronic publishing and querying information about business, services and technical information. The data management tier is occupied by a Microsoft SQL Server which is queried using JDBC.

Even after reviewing a relatively small number of transactional systems, it is obvious that structured approaches to their design have been well formulated. These approaches include the use of UML and best practices, such as design patterns.

2.4 Summary of Web-Based Interactive Systems

The analysis of existing virtual laboratories has shown that a whole range of technologies is being employed that could be applied to a remote technology demonstrator. Network protocols used include TCP, UDP and CORBA. It is necessary to first determine the reliability and performance requirements before selecting a protocol. While CGI is still in use for dynamic web content generation, other more performance-orientated methods, such as servlets, are being employed. It is evident that web pages with containing embedded Java applets are popular for providing client-side functionality for virtual laboratories. Safety issues should always receive particular consideration during development and operation. Security is also important. The token-based system used by Ko *et al.* [10] offers an effective means of securely operating a system where only one user may control the remote demonstration at any one time. Feedback mechanisms for the purposes of monitoring of the remote demonstrations can include test data and visual feedback.

There are numerous examples of successful transactional e-commerce web sites in existence such as eBay [27] and Amazon [41]. Requirements for such systems demand that they utilise technologies which provide substantial flexibility, scalability and the possibility to integrate with other enterprise systems. The multi-tier architecture of J2EE offers developers the potential to deliver such functionality taking full advantage of its platform independent operational model. Enterprise solutions using J2EE may be achieved using a component-oriented approach such as CBESE also employing proven best practices such as design patterns from Sun

Microsystems' "Java Enterprise Blueprints". Enterprise Javabeans and Java servlets are already used in enterprise research applications and should be used also for the proposed new remote technology demonstrator. Web Services also present a useful solution for a language and platform independent exchange of data between remote applications.

Virtual laboratories lack a structured approach to their design and development. In contrast, a number of mechanisms have been employed for developing transactional systems, including UML and design patterns.

J2EE does offer a low-cost, secure and scalable solution as an alternative to many existing virtual laboratory designs. However, there appears to be an apparent absence of remote laboratories based on the transactional systems. The reason for this may lie within the statement itself. Remote laboratories often comprise of real-time elements and transactional applications are not suited for this purpose due to the non-deterministic nature of many network protocols. Real-time control over the Web poses significant challenges. It is important to understand what services J2EE can provide for building the remote technology demonstrator. The next chapter describes J2EE technologies relevant to the development of a virtual laboratory.

Chapter 3

Java 2 Enterprise Edition

3.1 Introduction to J2EE

Sun Microsystems has, through the Java community, defined a collection of Java based technologies for server side application development and execution named the Java 2 Enterprise Edition (J2EE) [42]. The J2EE specification requires application servers to support a specific set of protocols and Java enterprise extensions such as Java Management Extensions (JMX) [43]. This ensures a consistent platform for deploying J2EE applications. This chapter describes the most relevant services of J2EE used in this project. A moderate level of detail is given to each service in order to provide sufficient detail that the reader may comprehend J2EE related arguments put forward in other chapters. Each service has itself, a standard specification. The full range of services is shown in Figure 3.1 below.

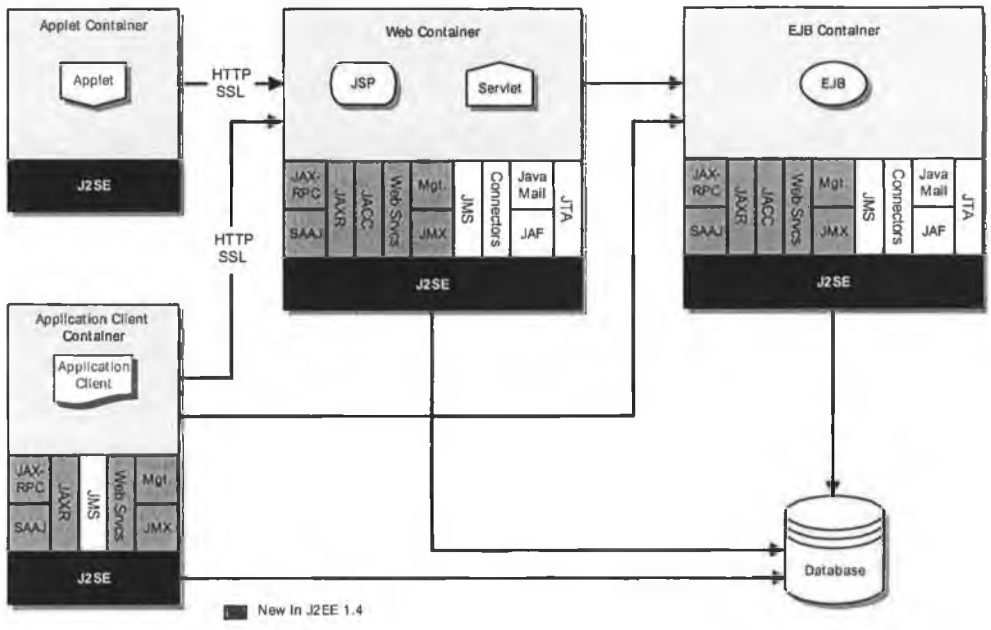

Figure 3.1: J2EE Architecture (Source: Sun Microsystems [42])

3.2 Enterprise JavaBeans (EJB)

The term, "Enterprise JavaBeans", describes both the name of the standard as well as the components it defines. It defines software architecture for server side components. The specification [44] defines architecture for building distributed object-oriented

applications in the Java programming language. EJB is a component technology that supports a distributed paradigm and may interact with other important technologies such as Web Services. It provides and hides many troublesome mechanisms for an application developer such as transaction control, persistency of data, thread control, and directory services. It supports a "Write Once, Run Anywhere" philosophy.

An EJB container is an abstract entity that represents the EJB server where an EJB lives. The EJB specification defines the rules governing the relationship of an EJB component and its container. An EJB container implements the management and control services for a class of EJBs that live in that container. More specifically, the EJB container provides implicit transaction control, transparent distribution services, security services, persistence management, and other services for life cycle management and instance pooling (including creation, activation, passivation, and destruction) on behalf of the EJBs that lives in that container.

3.3 Java Naming and Directory Interface (JNDI)

Prior to discussing the architecture of enterprise Java beans, it is useful to first outline the mechanism by which the components are accessed by clients. The Java Naming and Directory Interface (JNDI) provides an interface for accessing name and directory services such as Lightweight Directory Access Protocol (LDAP) directory services and Domain Name Service (DNS). JNDI enables Java programs to use name servers and directory servers to look up Java objects by name. This feature enables an application to locate distributed objects, which is essential in distributed programming. JNDI is a generic API that works with any name or directory server and as such it provides a common interface against existing directory and naming servers.

A directory service typically provides access to data structured in hierarchies, such as directories in a file system. It is also used to categorise data into hierarchies such as yellow pages. A naming service allows access to objects by name for instance, looking up an IP-address to a computer based on a name as in DNS.

JNDI is a core service in J2EE. It is used to locate enterprise beans and to access environment variables. Other resources such as databases and email facilities may also be accessed through JNDI.

3.4 EJB Architecture

An enterprise bean typically contains business logic that operates on the enterprise's data. Enterprise beans exist inside an EJB container, which is hosted by a J2EE Application Server. A J2EE application may also contain other containers such as a web container than handle requests from web clients. The EJB container manages the lifetime of various instances of enterprise bean classes. The container provides the set of interfaces defined in the J2EE specification which include transaction services, persistency of data, network transparency through use of RMI/IIOP or SOAP, thread handling, and JDBC connections to databases.

An enterprise bean is specified through a set of interfaces and bean class. The exception to the rule is the message-driven bean, which does not have any interfaces. Every EJB has a home object which acts as a factory (i.e. based the factory design pattern as outlined by Gamma *et al.* [45]). The home interface specifies the methods for creating or finding beans. The EJB object itself is a proxy object which results in corresponding business methods being made on the bean instance on behalf of the client.

A client does not interact directly with the enterprise bean; it must do so through the interfaces of the bean, which may be local or remote. A local interface may be used, if the bean to be accessed resides in the same Java Virtual Machine (JVM). Remote interfaces are used when accessing beans outside the client JVM. The local and remote interfaces contain business methods that are specific for the enterprise bean. The enterprise bean instance class implements the business methods defined in local or remote interface.

It is preferable to use local interfaces when looking up enterprise objects as a direct reference is returned. Clients are required to use RMI/IIOP to look up the requested object using remote interfaces.

The container is responsible for making the home interfaces of its deployed enterprise beans available for clients. Thus, the client can look up the home interface for a specific enterprise bean using JNDI.

When a client accesses a specific home object, it will create an EJB object instance and result in the creation of an enterprise bean. The creation mechanism itself is vendor specific. A unique reference to the EJB object is returned to the client. The client can then access the business methods through the EJB object, which is responsible for forwarding the method call to the allocated enterprise bean.

The enterprise bean instance class contains the implementation of the methods that are specified in interfaces. These instances are created and managed at runtime by the EJB Container as well as mediating overall client access. An enterprise bean can be customised at deployment time by editing its environment entries.

3.5 Enterprise Beans

The enterprise bean class contains the implementation of the methods that are specified in interfaces. There is a specific name convention that has to be followed. There are three types of Enterprise Javabeans: session, entity and message driven beans.

3.5.1 Session Beans

A session bean represents a single client inside the J2EE Application server. It is created by the client by invoking a create method in the home interface and it exists only for the duration of a single session. The session bean normally represents some business logic. Although session beans can be transactional, the container does not guarantee recovery after a system crash. Session beans must therefore manage their own persistency of data.

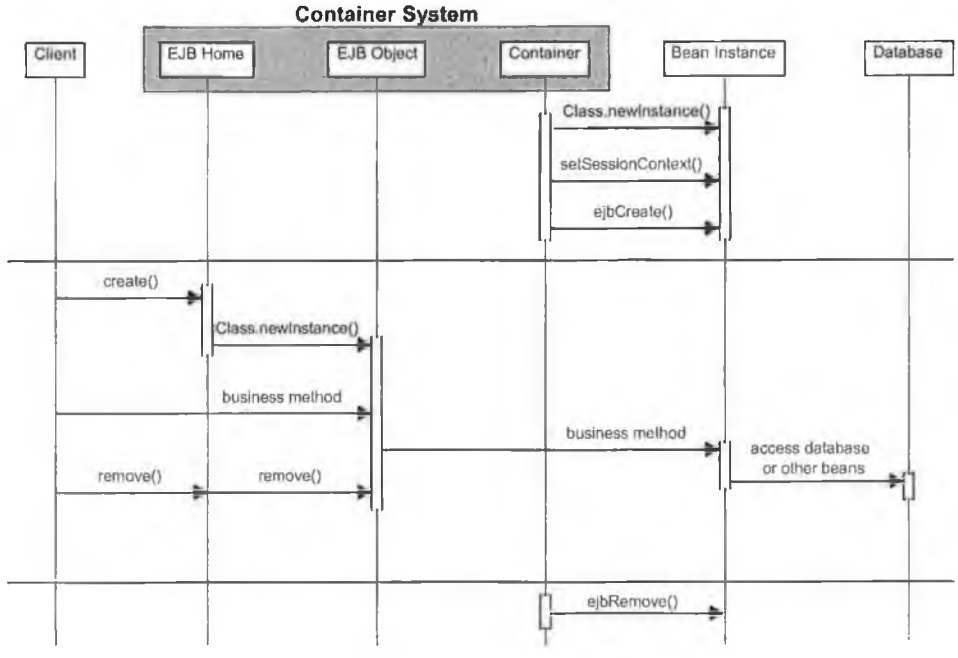

Figure 3.2: Creation and removal of stateless session beans (Source: Monson-Haefel [46])

There are two types of session beans. The first type of session bean is known as a stateless bean. This EJB does not maintain the conversational state between invocation of the bean's business methods. Figure 3.2 shows the creation and removal of a stateless session bean as described by Monson-Haefel [46]. Notice that the container creates the bean instance prior to any calls being made by the client. Stateless session beans are normally pooled so that they may be used by multiple clients. A stateless session bean can also provide a Web Service endpoint, which can be used by Web Service clients.

Stateful session beans are the other types of session beans available but, in contrast, maintain conversational state between method invocations. Once a stateful session bean is instantiated and assigned to an EJB object, it is dedicated to that EJB object for its entire life cycle. Figure 3.3 shows how the client causes the bean instance to be created after calling the home interface create method.

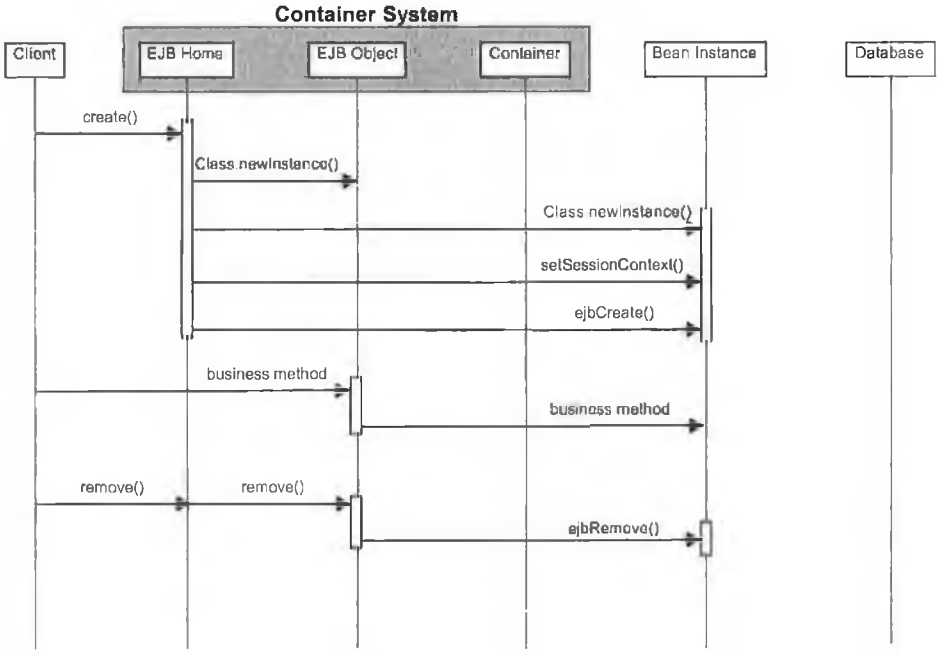

Figure 3.3: Activation process of stateful session beans (Source: Monson-Haefel [46])

Stateless session beans offer a more efficient means of representing business logic and should be used if possible. However, it may be required to maintain information relating to a previous invocation in a member variable (e.g. user details) so a stateful session bean may be required.

3.5.2 Entity Beans

Entity beans represent persistent data of an application. The entity bean interface includes methods for accessing the data. An entity bean is identified with a unique primary key and the home interface must have a method that is used to find an entity bean based on the primary key. The JDBC API is used to store data in the underlying database. Entity beans are often associated with database transactions and may handle concurrent access from multiple clients.

There are two different types of entity beans. The first type is Container-Managed Persistence (CMP) based entity beans where the container manages the storing of application data. The second type is Bean-Managed Persistence (BMP) based entity beans where the bean itself is responsible for storing and restoring the data. Data may be read and saved to the database using an entity bean's get and set methods respectively. The container automatically generates methods for finding entity beans based on the method signature in the home interface.

Until recently BMP was used to bridge the gap between the application requirements are the limitations found in the CMP approach. Much has been done to resolve this issue with the introduction of Container-Managed Relationships (CMR) and EJB Query Language (EJB-QL). CMR allows entity relationships to be defined using the application descriptors at deploy time. This is in effect configurable normalisation of the application data objects. EJB-QL is a recent addition to CMP which data related objects to be located in a platform independent way using a limited subset of SQL.

3.5.3 Message-Driven Beans

The final category of enterprise beans are message-driven beans (MDBs). MDBs are not used in this project but a description on their use is included here for completeness. Monson-Haefel [46] describes MDBs are stateless, server-side, transaction-aware components for processing asynchronous Java Message Service (JMS) messages. JMS itself is a standard vendor-neutral API that is part of the J2EE platform and can be used to access enterprise messaging systems. Enterprise messaging systems (a.k.a. message-oriented middleware) facilitate the exchange of messages among software applications over a network. Many commercial enterprise messaging products currently support JMS, including IBM's MQSeries and BEA's WebLogic JMS service.

Message-driven beans process messages delivered via JMS. Message-driven beans can receive JMS messages and process them. While a message-driven bean is responsible for processing messages, its container takes care of automatically managing the component's entire environment, including transactions, security, resources, concurrency, and message acknowledgement.

One of the most important aspects of message-driven beans is that they can consume and process messages concurrently. This capability provides a significant advantage over traditional JMS clients, which must be custom-built to manage resources, transactions, and security in a multithreaded environment. The message-driven bean containers provided by EJB manage concurrency automatically, so the bean developer can focus on the business logic of processing the messages. The MDB can receive hundreds of JMS messages from various applications and process them all at the same time, because numerous instances of the MDB can execute concurrently in the container.

A message-driven bean is a complete enterprise bean, just like a session or entity bean, but there are some important differences. While a message-driven bean has a bean class and Extensible Mark-up Language (XML) [47] deployment descriptor, it does not have component interfaces (e.g. remote & home interfaces). The component interfaces are absent because the message-driven bean is not accessible via the Java RMI API; it responds only to asynchronous messages.

3.6 Enterprise Bean Environment

In order to facilitate component reuse, an EJB component needs to be customised to fit into different environments that exist for the application. The EJB specification specifies an environment mechanism to allow customisation of the enterprise bean's business logic, use of external resources or references to other components without accessing the enterprise bean's source code. All environment variables are defined in XML deployment descriptors. The container is responsible at deployment to read the deployment descriptors and put the environment variables into JNDI. The environment variables are available to the bean at runtime through the JNDI interfaces.

Several types of environment variables may be defined and targeted for different purposes. The most interesting ones in this study are described in the following sections.

3.6.1 Environment Entries

The enterprise bean's environment entries allow the enterprise bean to be customised without the need to access or change the enterprise bean's source code. Each enterprise bean defines its own set of environment entries. All instances of an enterprise bean share the same environment entries. The environment entries are not accessible to other enterprise beans. Enterprise beans are not allowed to modify the bean's environment at runtime.

At deployment time all environment entries that are accessed in the beans source code have to be declared. An environment entry is declared using `env-entry` elements in the deployment descriptor for the bean. An `env-entry` element consists of an optional description of the environment entry, the environment entry name, the expected Java type of the environment entry value, and an optional environment entry value. The environment entry value must be of type String, Character, Integer, Boolean, Double, Byte, Short, Long and Float.

3.6.2 Deployment Descriptors

Al environment variables are declared in XML deployment descriptor files which are read at deployment time by the container, enabling the information to be accessed using JNDI. The container uses part of the environment information in the deployment descriptors to configure different properties of the enterprise beans.

3.7 CORBA as a J2EE Service

JavaIDL is Sun Microsystems' CORBA implementation which allows J2EE application components to invoke external CORBA objects using the Internet Inter-ORB Protocol (IIOP) protocol. CORBA itself is a specification for a distributed application framework developed by the Object Management Group (OMG) [48]. CORBA objects may be written in any language and normally use OMG's Interface Definition Language (IDL) to define operations being offered to clients. Clients and objects are not directly aware of each other's locations within the network, allowing relocation of one (or both) with no impact on the other. Clients and objects running within different vendors' ORBs can communicate just as if they were located within the same ORB.

Due to the platform-independence of the CORBA architecture [49] and support from many different vendors, CORBA-compliant products and frameworks

exist for most operating systems and hardware platforms, allowing CORBA applications to span many different types of systems, from small, handheld devices to mainframes.

The J2EE and CORBA architectures provide similar benefits to the programmer developing a distributed application. Each provides a framework that supports location transparency of distributed objects. Each offers multi-platform support and as a result permits applications to run in heterogeneous environments. The underlying implementation language, however, is where these two architectures diverge.

J2EE makes development of distributed applications easier by allowing programmers to work entirely in Java. As J2EE is based completely on the Java language, this allows the application components to benefit from Java's 'Write Once, Run Anywhere' model. However, integration with existing or third party non-Java applications can be particularly difficult as a platform and language specific solution may be required. Making distributed components available to non-Java clients can also present difficulties as many application servers as yet have incomplete support for callbacks using IIOP.

In contrast to J2EE, CORBA permits development in numerous supported programming languages, addressing the use of APIs and legacy code written in different languages. In addition, performance-critical components can be isolated and developed in lower level, more performance orientated languages such as C++. On the downside, there is the learning curve of using IDL and understanding the IDL-to-language mappings. In addition, no programming languages ports as easily from platform to platform as Java, so moving non-Java CORBA components from one platform to another can present significant expense in time and effort, as well as increasing the risk of new bugs occurring. In summary, CORBA allows more flexibility, and potentially performance, at the expense of portability.

By integrating the CORBA and J2EE architectures, distributed applications can leverage the aforementioned benefits while many of the associated limitations can be reduced or even eliminated.

3.8 Servlets

Servlets are used to dynamically generate Hypertext Markup Language (HTML), Extended HTML (XHTML), and XML output using Java technology. They are

programs that run on a web server, acting as a middle layer between a request coming from a web browser or other Hypertext Transfer Protocol (HTTP) client and databases or applications on the HTTP server. Their purpose is to:

1. Read any data sent by the user. This data is usually entered in a form on a web page, but could also come from a Java applet or a custom HTTP client program.
2. Look up any other information about the request that is embedded in the HTTP request. This may include information about browser capabilities, cookies, and client host address for example.
3. Generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a legacy application, or computing the response directly.
4. Format the results inside a document. In most cases, this involves embedding the information inside an HTML page.
5. Set the appropriate HTTP response parameters. This means telling the browser what type of document is being returned (e.g. HTML), setting cookies and caching parameters, and other such tasks.
6. Send the document back to the client. This document may be sent in text format (HTML), binary format (e.g. JPEG images), or even in a compressed format like gzip that is layered on top of some other underlying format.

Many client requests can be satisfied by returning pre-built documents, and these requests would be handled by the server without invoking servlets. In many cases, however, a static result is not sufficient, and a page needs to be generated for each request.

3.9 Java Server Pages

JSPs are also used to dynamically generate web content such as HTML, XHTML, and XML in web applications. JSP technology enables easy authoring of web pages, which allows web designers rather than programmers focus on the presentation. Embedded code known as "scriptlets" may be added to the page to be dynamically translated into HTML content on activation of the page.

3.10 HTTP and HTTPS

Web components in an application server service HTTP and HTTP Secure (HTTPS) requests. The servlet specification itself only requires support for HTTP but the application server must be capable of operating HTTP and HTTPS (HTTP 1.0 over Secure Socket Layer (SSL) 3.0). HTTPS is often used to ensure data confidentiality particularly when transmitting sensitive information such as passwords or credit card information.

3.11 Java Transaction API

J2EE requires the provision for Atomicity, Consistency, Isolation, Durability (ACID) transactions which is implemented by the Java Transaction API (JTA). The application developer should never have to deal directly with the JTA, especially when using CMP for data persistence where the EJB container handles the transaction management. The JTA provides mechanism for handling commit and the rollback of transactions as well as ensuring ACID properties of a transaction.

It is important to outline the properties of an ACID transaction at this stage. To be atomic, a transaction must execute completely or not at all. This means that every task within a unit-of-work must execute without error. If any of the tasks fails, the entire unit-of-work or transaction is aborted, meaning that any changes to the data are undone. If all the tasks execute successfully, the transaction is committed, which means that the changes to the data are made permanent or durable. Consistency is a transactional characteristic that must be enforced by both the transactional system and the application developer. Consistency refers to the integrity of the underlying data store. The transactional system fulfils its obligation for consistency by ensuring that a transaction is atomic, isolated, and durable. The application developer must ensure that the database has appropriate constraints (primary keys, referential integrity, and so forth) and that the unit-of-work, the business logic, does not result in inconsistent data (i.e., data that is not in harmony with the real world it represents). In an account transfer, for example, a debit to one account must equal the credit to another account. For isolation, a transaction must be allowed to execute without interference from other processes or transactions. In other words, the data that a transaction accesses cannot be affected by any other part of the system until the transaction or unit-of-work is completed. Durability means that all the data changes made during the course of a transaction must be written to some type of physical storage before the transaction is

successfully completed. This ensures that the changes are not lost if the system crashes.

3.12 JavaMail

Many Internet applications require the ability to send email notifications, so the J2EE platform includes the JavaMail to process Internet mail. The JavaMail specification provides a collection of abstract classes that define the common objects and their interfaces for any general mail system. By virtue of having interfaces to mail sessions, messages, transports, and stores, programmers have an easy means of sending and receiving emails in custom designed applications. Sun Microsystems define a transport as a service that has the capability to send messages to their destination. The most commonly used transport type is the ubiquitous Simple Mail Transfer Protocol (SMTP) transport. A store is defined as a retrieval service for messages.

Sun Microsystems' reference JavaMail implementation includes providers for some of the most essential Internet protocols and specifications. These include SMTP, Post Office Protocol 3 (POP3), Internet Message Access Protocol (IMAP), and Multipurpose Internet Mail Extensions (MIME).

3.13 Web Services

Web Services are a very recent addition to the J2EE API framework and complement the existing features in the standard. Web Services are applications that are defined, published and accessed across the Web. It consists of a set of technologies enabling loosely coupled applications to expose the service interface using Web Service Description Language (WSDL), a protocol that enables communication between applications, Simple Object Access Protocol (SOAP) and a registry, Universal Description Discovery and Integration (UDDI) for publishing the services. Together these technologies provide a Service Oriented Architecture (SOA). Web Services are an answer to the need to integrate business applications, enabling exchange of business information in real time. To that purpose, an open and implementation independent standard is needed, which emphasises services rather than interconnection of computers. Although Java RMI and CORBA are middleware for connecting distributed applications together, they are limited in integrating a wide variety of systems that exist over a rather unstable and firewall protected Internet. The SOAP protocol can run on different communication protocols such as HTTP, and it

supports asynchronous communication of messages. Current Web Service standards support only synchronised communication between service endpoints but it is intended to provide asynchronous communication between service endpoints in future. This will enable networks to route service requests as SOAP messages to the correct service endpoint like computers are addressed on the Internet today.

Web Services are based on the three main technologies SOAP, WSDL and UDDI. In addition there is XML [47], which is a text-based language used to describe data, a core technology for describing service interfaces and the content of SOAP messages. SOAP supports the encoding of arbitrary data, usually described in XML, and the transfer of data over a channel between communication endpoints. SOAP is wire protocol neutral and therefore can be used over different protocols like for instance HTTP, FTP and SMTP. However due to the ubiquity of the HTTP protocol, most implementations typically use HTTP. SOAP is a lightweight mechanism for exchange of data between heterogeneous applications that is independent of operating systems, programming languages and network protocols.

SOAP supports a language independent Remote Procedure Call (RPC) protocol mechanism. Specification of different encoding rules enables exchange of application specific data types. SOAP supports also definition of schemas for data types. This makes SOAP a good protocol for integrating of heterogeneous applications. WSDL is an XML based language that describes the functionality of Web Services. It has similarities with the Interface Description Language (IDL) used to define CORBA interfaces. WSDL describes the operations a Web Service offers including the parameters of each operation and the return value. But WDSL does not describe the semantics of the operations. WSDL enables method calls to a Web Service regardless of the selected RPC mechanism.

UDDI provides global directory services for Web Services. It supports also a protocol for publishing and discovering of services. The Web Services are described with WSDL and are stored in the UDDI as WSDL files. The WSDL files are retrieved from the UDDI by an application developer making it possible to make calls to Web Services.

Web Services have been integrated into recent application servers, offering EJB access to applications independent of language, platform and location. The foundation for this integration was the inclusion of the XML based technologies in J2EE. An EJB application typically exposes a Web Service interface through a

stateless session bean. The Web Service client view may be initially defined by a WSDL document and then mapped to a Web Service endpoint.

J2EE supports the Java API for XML Messaging (JAXM) protocol, which enables Java applications to send and receive document oriented XML messages. A message bean may receive SOAP messages instead of JMS messages. The message bean must then implement a specific interface for receiving a SOAP message with attachment.

An enterprise bean may also access a Web Service as an ordinary Web Service client, but invoking Web Services will introduce an unpredictable delay since the calls are synchronous. A Web Service endpoint is accessible for a bean through the JNDI API.

3.14 Security

The intended security goals are clearly defined in the J2EE specification. Transparency is important and J2EE component developers should not be required to know about security. Isolating the provision of security services ensures greater portability and offers possible responsibility of this function to a System Administrator. Security mechanisms and declarations used by applications should not impose a particular security policy, but facilitate the implementation of security policies specific to the particular J2EE installation or application. XML deployment descriptors provide a level of abstraction by describing environment-specific security roles, users, and policies. J2EE requires security behaviours and deployment contracts to be implementable using a variety of popular security technologies, but does not dictate which technologies should be used. Application components executing in a J2EE product must be able to invoke services provided in a J2EE product from a different vendor, whether with the same or a different security policy. These services may be provided by web components or enterprise beans.

Java security technology originally focused on creating a safe environment in which to run potentially untrusted code downloaded from a public network. The approach in relation to security was somewhat unusual in that it first concentrated where the code came from, rather than who was running it. A new security standard, Java Authentication and Authorisation Service (JAAS), was provided to address this situation. Lai *et al.* [50] describe JAAS as a means of enforcing access controls based on the identity of the user who runs the code as well as supplementing the security

based on the code's origin. JAAS is based on a standard known as the Pluggable Authentication Module (PAM) framework, which allows different modules to be added for authentication purposes. A client using JAAS can be configured to use different login techniques, such as a simple user-name/password dialog or a smart card reader connected to the computer's USB port. A server using JAAS can be configured to authenticate the user's identity and credentials using different back-end security services. Policies may be used to impose fine grained access control a particular user or on a group of users depending on their defined security role.

3.15 Summary of J2EE

Enterprise Javabeans constitutes a significant proportion of the J2EE standard. It defines software architecture for server side components and currently has three types of components namely; session, entity and message-driven beans. Session beans, which manage business method calls, may be either stateful or stateless depending on whether or not conversational state is required to be maintained between method invocations. Entity beans represent persistent data of an application and may support either container or bean managed persistence. Message-driven beans are stateless transaction-aware components for processing asynchronous JMS messages. XML deployment descriptor files play an important role in providing modularity in the J2EE architecture and allowing applications to be extensible without necessarily requiring a component code change. Environmental entries such as host addresses of third party servers may be specified in these deployment descriptors.

There are a number of other services in J2EE apart from EJB. JavaIDL is a CORBA implementation which allows J2EE application components to invoke external CORBA objects using the IIOP protocol. Servlets and JSPs provide dynamic web content and are also used process user requests. The Java Transaction API enables the EJB container to perform ACID transactions. JavaMail is another useful service that may be used for programmatically sending and receiving email messages. Web Services consist of a number of services such as WSDL, SOAP and UDDI which allow applications to asynchronously send messages independent of platform, language and location. The security goals of J2EE include transparency for the application developer. JAAS is an important J2EE security mechanism which may be used to enforce access controls based on the identity of the user.

Chapter 4

System Analysis and Design

4.1 Introduction

System analysis and design is an essential stage of the software development process. It helps with the project's architecture definition and implementation methodology. During this stage, the system under consideration will be defined and the conditions in which it will operate outlined so that broad guidelines of design may be determined. This chapter will describe the requirements of the remote technology demonstrator as well as outlining the approach that is to be taken to provide a solution for these requirements.

4.2 Unified Modelling Language

This thesis has already used the Unified Modelling Language (UML) [51] to describe existing software implementations. According to Fowler and Scott [52], the fundamental reason for using UML is communication. Natural language is too imprecise and intended meanings are lost when it comes to more complex concepts. Code is precise but too detailed. Louis *et al.* [53] found use case and sequence diagrams to be two of the most important UML diagrams, allowing them to communicate clearly with end users and domain experts and confirm the understanding of the system's requirements. UML will allow better design and reveal the best possible direction for the project to take. Relationships between the components of a system are grasped more easily when the design is represented graphically using a modelling language. UML provides a rich set of modelling tools to describe dynamic and static aspects of the system being designed. Based on this argument, extensive use of UML will be made to convey design concepts and issues.

4.3 Defining System Requirements

Gathering and agreeing on requirements is fundamental to a successful project. This does not necessarily imply that all requirements need to be fixed before any architecture, design, and coding is done, but it is important for the developer to understand what needs to be built.

An initial requirement description may be described as follows:

The operation and control of a Stirling Engine, similar to that depicted in Figure 4.1, will be the main focus of this demonstration. An overview of Stirling Engines is given in Appendix B. Other test equipment required includes a Programmable Power Supply (PPS) for supplying the electrical/thermal power input and the Data Acquisition (DAQ) unit for sensor measurement. The primary objectives are to provide the user with a basic understanding of the operation of the engine and to observe the thermodynamic cycle involved that facilitates the transfer of thermal power to mechanical power. More specific secondary objectives will include determining the overall thermal efficiency of the engine over time. This may be done by measuring the electrical heat energy input versus the mechanical work output. This efficiency may be varied by adjusting the user-configured operating parameters of the test cycle such as thermal power input. Also of interest may be the corresponding changes of the hot and cold plate temperature difference and rotational speed of the flywheel as a function of the input power. The user may see how a Stirling Engine operates and may observe the thermodynamic cycle involved.

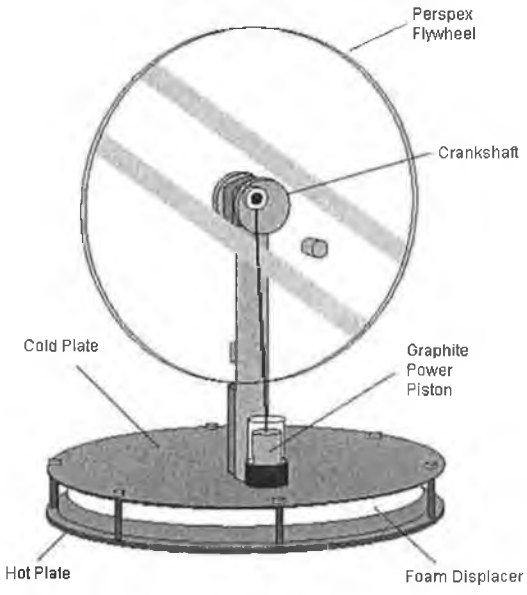

Figure 4.1: Stirling Engine Being Used in the Demonstration

A web-based interface should enable users to log onto the system and access features relating to management of their account and experimental procedure. The system caters for two types of users: administrators and observers. Administrators are relatively few in number and have unrestricted access to the available experimental test resources. Observers may be numerous and view results from available

demonstrations in real-time, but may not control the demonstration. A typical operational scenario involving administrators and observers might consist of a tutor conducting a demonstration for students who may be geographically dispersed.

Users may wish to change their account password and modify their account details such as the name of the person who has access to the account. In addition users will be able to save a setting to indicate whether or not they wish to receive an email summary of test cycle results on completion of the experiment. Users may view their history records showing previous actions carried out. These can also be used as an audit trail for administrator users when looking for details about particular account management activities or test equipment access. All users can view a currently active demonstration, that is, one which is being controlled by an administrator. There should be introductory help section available here which could be enhanced by also adding a related simulation of the test apparatus.

Administrators are permitted to carry out additional operations not allowed to observers. These include the creation and removal of users through the web interface. Also only administrators may request control of a test server controlling the test equipment. This will operate on a first-come first-served basis. If another administrator is already controlling the test server, then only observation functionality for the experiment will be available. This functionality will consist of a timely updates over an Intranet of active test data as well as visual feedback.

The Stirling Engine used in the experiment will be controlled and managed by a test server written in a compiled C++. All the programming interfaces to the experiment hardware (PPS and DAQ unit) are readily available in native language interfaces such as C++ and Pascal. A special Java Native Interface (JNI) bridge would have to be developed to communicate with the test hardware if a Java solution was used. Greater performance is an added benefit of using C++ as compiled code is faster than interpreted code. By developing in C++ it is also possible to explore the language independent options for network communication in the J2EE standard such as JavaIDL and Web Services.

4.4 Real-Time System Design

An interactive web-based interface is required for controlling the Remote Technology Demonstrator (RTD). It is important to replicate the experience of actually being present at the site of the experiment itself. This may be realised with visual feedback

of the apparatus working, complimented with the live feedback of data displayed in a meaningful graphical format such as a strip chart.

While the live feedback of test data is useful to indicate the current state of the experimental procedure, it would not be feasible for clients to save this data remotely while it is being generated. The most reliable method of recording the large volumes of data generated by the experiment is to save it locally first, on the test server machine, and then process it after when the test cycle has completed. The saved data can be sent to the J2EE-based system for long-term storage, permitting users to plots results independently of the test server.

From the analysis of existing virtual laboratories in Chapter 2, it was decided to use a Swing-based Java applet to implement the real-time functionality. This technology is robust and can be delivered to the clients through a web browser. However, in this instance, it must be properly integrated with the J2EE-based system taking operational and security considerations into account.

4.5 J2EE-Based System Overview

Enterprise Javabeans technology addresses a number of issues of generic distributed systems by simplifying the development of enterprise applications. Mos and Murphy [54] state that this inherent benefit alone, does not guarantee that they will perform as expected under heavy loads. There is still a need for good design and coding practices in developing these applications and this should be applied equally to all non-EJB components in the system.

Figure 4.2 shows a use case diagram for the interaction between the remote Java applet and the J2EE-based system. Note that while an administrator may perform any of the tasks of an observer, only critical functions are highlighted here. Administration of accounts includes tasks such as the creation and removal of users from the system. It is clearly intended that only administrator users be allowed control test equipment and this should result in the storing of test data to a database. A means of real-time feedback is necessary, as this will influence decisions made when controlling test equipment.

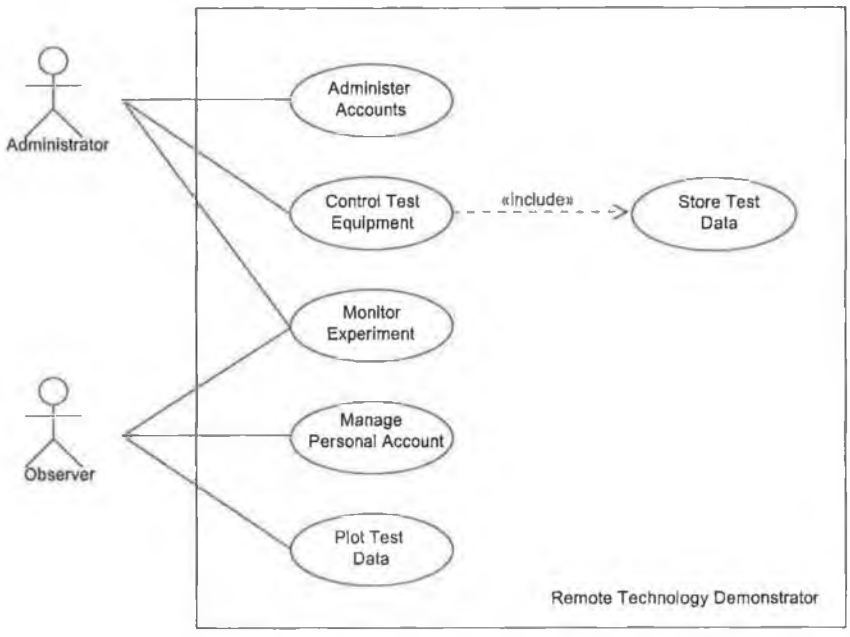

Figure 4.2: Use Case Diagram for Remote Technology Demonstrator

Observers should also be allowed to obtain real-time feedback of the test equipment but this is intended as a means of illustrating the behaviour of various elements in the experiment. All users will be capable of managing such settings as password and personal account name (e.g. John Smith).

Post-experiment processing, in the case of the thermal efficiency test procedure, should involve the calculation of the average efficiency, operating temperature and total sampling duration. These result details should be emailed to all users who wish to be informed of the test results immediately on conclusion of the experiment. This will allow the user to store the test data at their convenience. The application server should handle this email forwarding. This approach focuses the entire computational overhead on the test server. The test result plot mechanism will allow observers to independently view graphical test results of completed experiments.

Expanding the administrator use case in Figure 4.3 shows the proposed sequence of operations of how new user account provisioning might be achieved. Ideally an authentication mechanism should be integrated with a single sign-on facility, preserving the user's security credentials for every operation. Events details of the creation of the new user should be written to secondary storage using a logging mechanism.

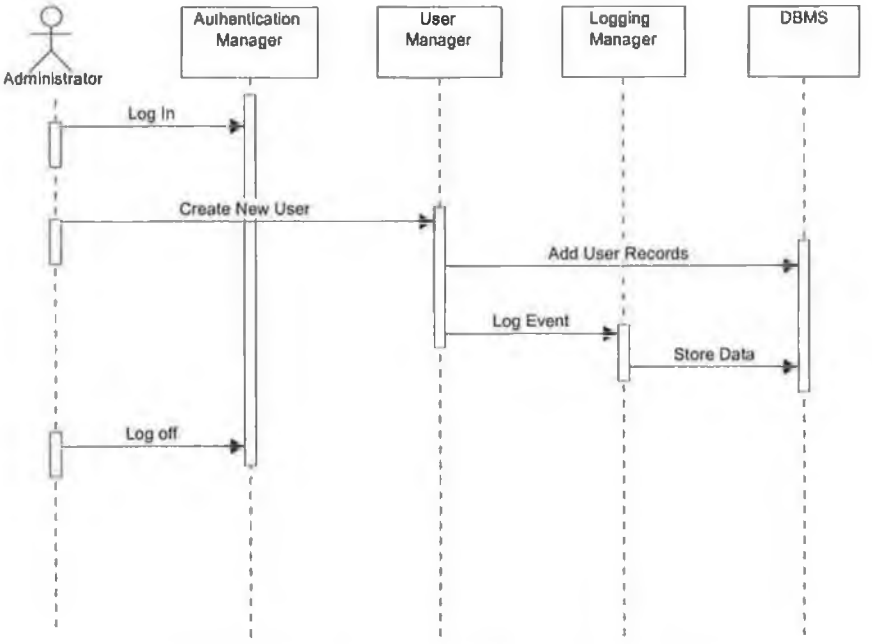

Figure 4.3: Sequence Diagram for Creating a New User

The sequence for removing a user should be similar but should allow for the administrator to view the user's details before removing the user from the system (Figure 4.4).

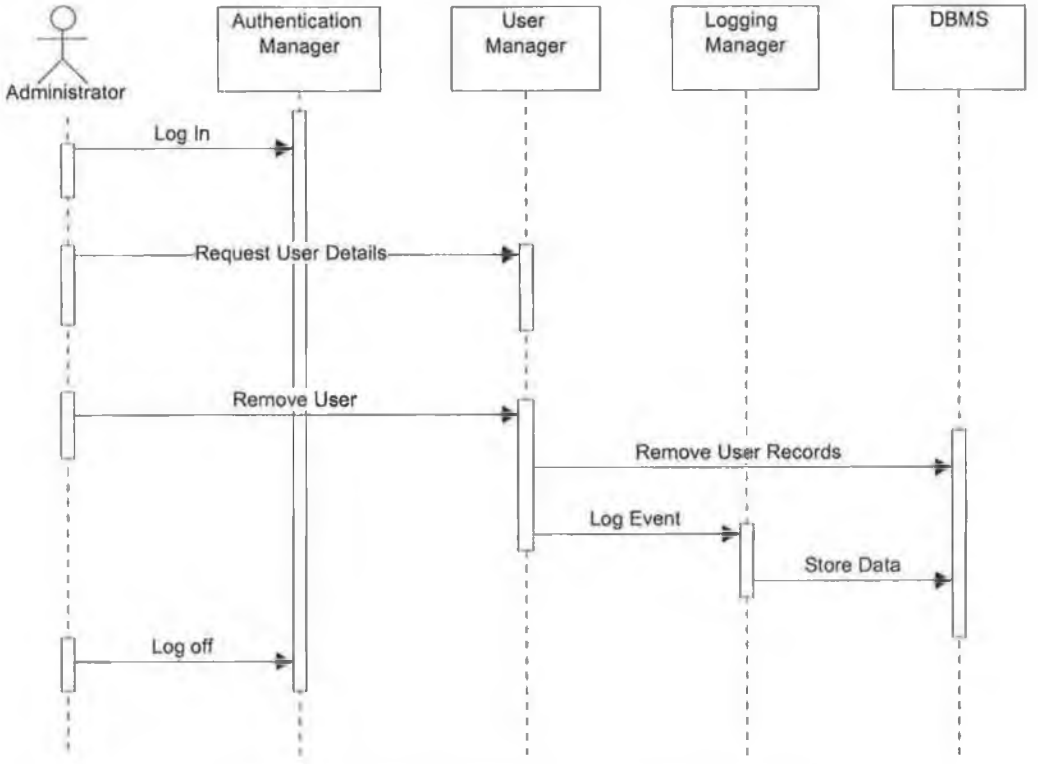

Figure 4.4: Sequence Diagram for Removing an Existing User

It is correct to say that most operations will take this format. Subsequent to authentication, the user management component will handle the business logic and event logging will be performed for critical operations.

4.6 Pattern-Based Development

Gamma *et al.* [45] define the quintessential set of core design patterns, proven design techniques, which provide an effective means of solving common programming problems. A new range of design patterns has emerged in the form of Sun Microsystems' blueprint patterns [21] specifically for J2EE-based platforms. These patterns not only ease the development process of J2EE applications, but also improve the quality of the produced software. Hammouda and Koskimies [55] utilise some of the most useful business-tier and presentation-tier design patterns in their application.

The communication between the presentation layer and business layer in distributed business applications often leads to tight coupling between clients and the business tier. The interaction could get so complex that maintaining the system becomes difficult and network performance can be adversely affected. The solution to this problem is to provide a simpler interface that reduces the number of business objects exposed to the client over the network and encapsulates the complexity of this interaction. At run-time, the client calls a method on a session façade, which in turn calls several methods on individual business objects. The Session Façade pattern [56] is based on the Façade design pattern outlined by Gamma *et al.* [45]. Figure 4.5 shows an example of a UML class diagram representing the pattern. Components behind the session façade interface may vary greatly from those shown here.

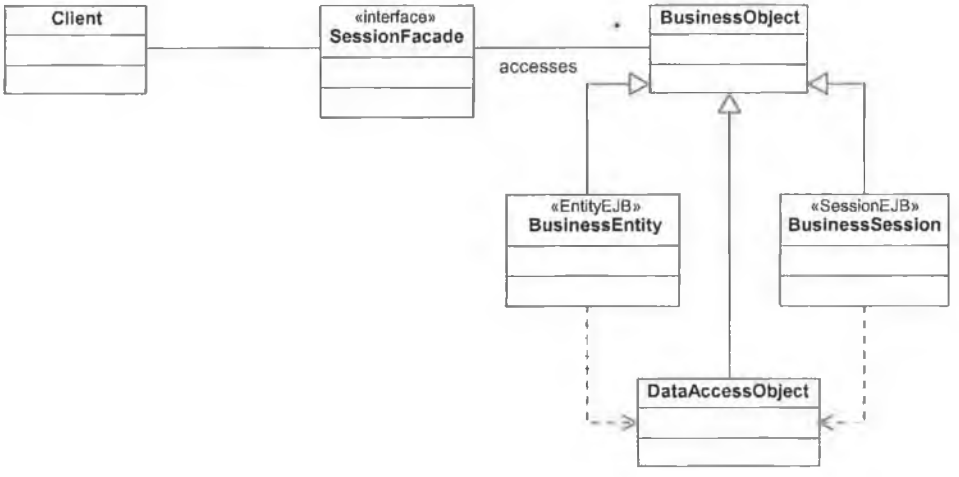

Figure 4.5: Session Façade Class Diagram (Source: Sun Microsystems [56])

The primary benefit of Session Façade pattern is to provide a centralised control over the business tier and ease of understanding and the maintainability of the system. In addition, the façade represents an access control layer to manage the relationships between user requests and business methods, and a transactional control layer where a transaction starts by calling a number of methods on the individual entities and commits by returning to the client.

The issue of enterprise service lookup is an important one. Enterprise applications require a way to look up the service objects that provide access to distributed components. This is normally done using JNDI to look up enterprise bean home interfaces, JMS components, data sources, connections, and connection factories. Repetitious lookup code makes code difficult to read and maintain. Furthermore, unnecessary JNDI initial context creation and service object lookups can cause performance problems. Hammouda and Koskimies [55] utilise Sun Microsystems' Service Locator blueprint pattern [57] in their general architectural tool. The Service Locator pattern centralises distributed service object lookups using JNDI, provides a centralised point of control, and may act as a cache that eliminates redundant lookups. It also encapsulates any vendor-specific features of the lookup process. Figure 4.6 represents the strategy of using the service locator pattern.

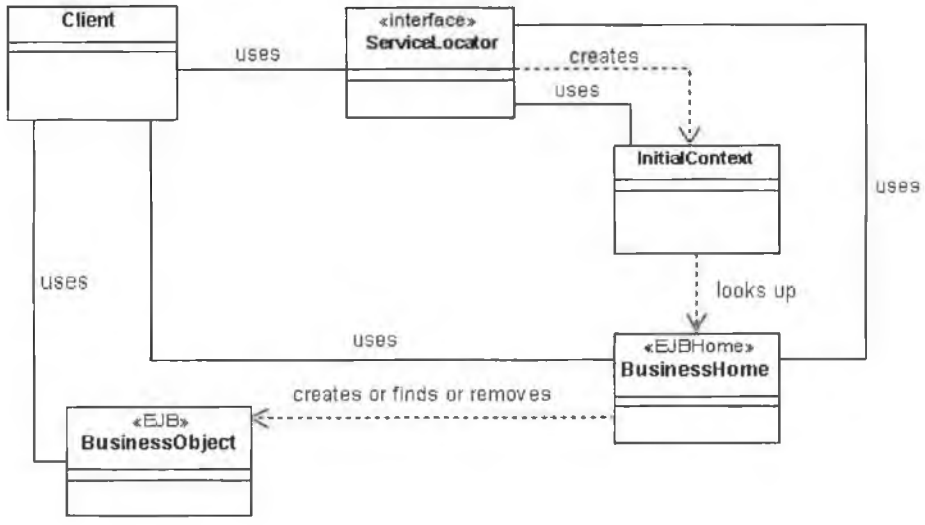

Figure 4.6: Service Locator Pattern Class Diagram (Source: Sun Microsystems [57])

There are a number of benefits of using this pattern. Firstly, the complexity of lookup and creation are completely encapsulated by the service location process. It provides a uniform service access point for clients and also facilitates easier development. It improves network performance as lookup calls are aggregated on the

server. Finally it reduces redundant lookups and object creation through caching and thereby improves client performance.

In J2EE applications, the client needs to exchange data with the business tier. For instance, the business components, implemented by session beans and entity beans, often need to return data to the client by invoking multiple get methods. Every method invocation is a remote call and is associated with network overhead. So the increase of these methods can significantly degrade application performance. The solution to this problem is to use a "Value Object" to encapsulate the business data transferred between the client and the business components. Instead of invoking multiple getters and setters for every field, a single method call is used to send and retrieve the needed data.

Presentation-tier patterns are equally as important as business-tier patterns, the Front Controller being one of the most useful. View navigation is a key issue in web-based enterprise applications. Due to the fact that views usually share common logic, a centralised access point for view navigation can be introduced in order to remove code duplication and improve view manageability. The Front Controller pattern controls and co-ordinates the processing code across multiple requests. It centralises the decision with respect how to retrieve and process the requests. A common strategy to implement this pattern is to use command pattern as described by Gamma *et al.* [45]. Figure 4.7 shows the sequence diagram representing the Front Controller pattern. It depicts how a controller handles a request.

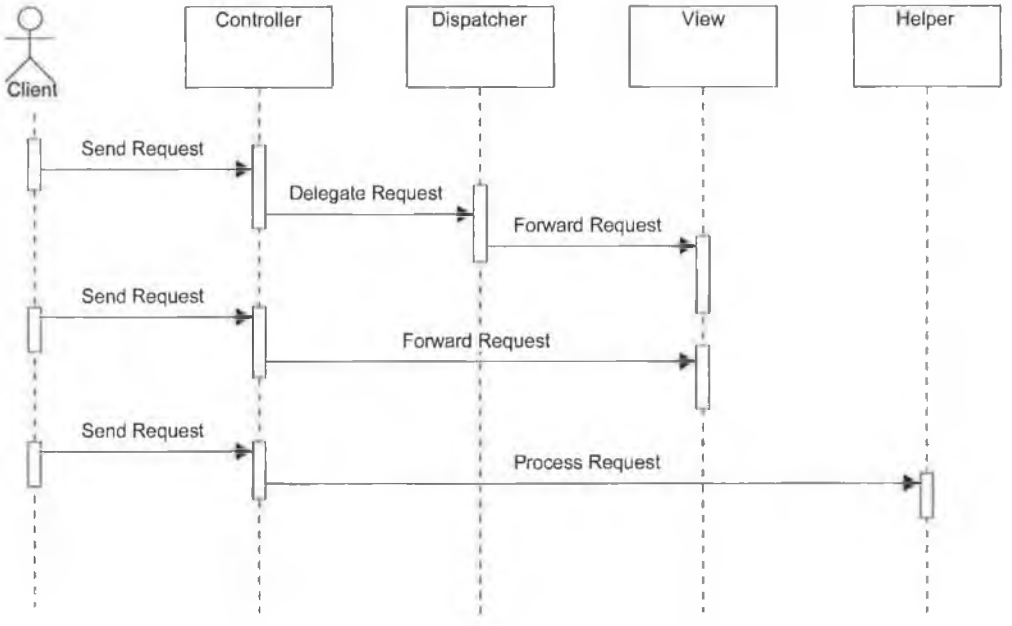

Figure 4.7: Front Controller Sequence Diagram (Source: Sun Microsystems [58])

The controller is the initial contact point for handling all requests in the system. The controller may delegate to a helper to complete authentication and authorisation of a user or to initiate contact retrieval. A dispatcher is responsible for view management and navigation, managing the choice of the next view to present to the user, and providing the mechanism for directing control to this resource. A helper is responsible for helping a view or controller complete its processing. Thus, helpers have numerous responsibilities, including gathering data required by the view and storing this intermediate model, in which case the helper is sometimes referred to as a value-bean.

The system needs to control the flow of execution and the navigation between views. In particular, the system needs to know to which view to dispatch next based on the request. It is vital to separate the logic on deciding which view comes next from the view components themselves. The dispatch mechanism is known as the Dispatcher View pattern. This pattern is an ideal mechanism for encapsulating much of the logic behind managing the navigation sequence through the web interface. The pattern does not perform heavy processing on the request but can be seen as a simple forwarding facility.

The Front Controller and the Dispatcher View patterns combine to produce an easily managed framework for web-based portals. One of the major problems they overcome relates to the way in which servlets present web content. In the early days of the servlet specification, content was embedded into the source code of the servlet in order to produce dynamic content. This resulted in code that was very difficult to manage and maintain. A number of third party web template solutions such as Webmacro [60] were produced in the interim, but it was not until Java Server Pages (JSPs) were introduced, that this issue could be satisfactorily addressed. By using the Front Controller and the Dispatcher View patterns from Sun Microsystems' design blueprints [21], developers could finally produce truly management solutions using what is known as the servlet Front Strategy. In this configuration, the Front Controller uses a servlet as a focal point for all requests made to a web application and the Dispatcher View component, in turn, directs the output towards a specific JSP.

While the Front Controller and the Dispatcher View patterns will manage client requests and redirect output there is still a requirement to adapt the web content further, based on the client user type. Soldar *et al.* [28] highlight servlet filters as an effective mechanism for dynamically transforming of header and payload information

in both servlet requests and responses. Typical uses of filter components include logging, image conversion, data compression, encryption, access events, data caching and authentication. The final item listed is of particular interest for this application. Users will be offered different functionality via the web interface depending on their user type, i.e. administrator or observer. Administrators can expect to have greater access to resources than observers.

Sun Microsystems cite the Intercepting Filter pattern [61] as a presentation-tier request handling mechanism receiving many different types of requests, which require varied types of processing. Some requests are simply forwarded to the appropriate handler component, while other requests must be modified, audited, or uncompressed before being further processed. While the Front Controller pattern solves some similar problems, it is better suited to handling core processing. Also, using the Intercepting Filter pattern abstracts the behaviour of handling authentication by using the web application's XML deployment descriptor. This allows for the filter component to be substituted by another if necessary without any code change. Another benefit of using servlet filters is that they may be chained together again without any code change. Figure 4.8 shows the class diagram for this type of configuration.

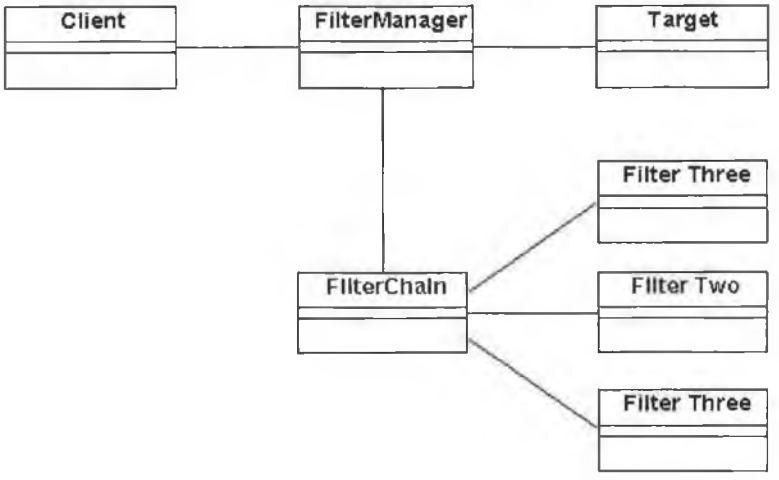

Figure 4.8: Intercepting Filter pattern class diagram (Source: Sun Microsystems [61])

4.7 Integrating the Real-Time and J2EE-Based Systems

The real-time and J2EE-Based Systems should be seamlessly integrated and maintain a good level of security in the combined overall system. A proxy test manager should act as a gateway to process access requests on behalf of the client. This event should

be logged and an access identifier should be passed back to the client. The administrator client should subsequently be able to issue control commands to the test server controlling the equipment. A broadcast system such as the one used by Ko *et al.* [26] is preferred but other possible solutions should also be analysed prior to proceeding with this option. As indicated previously, the test server should be responsible for ensuring that the processed test data is stored in the database. Figure 4.9 depicts a sequence diagram for the operation of the integrated system.

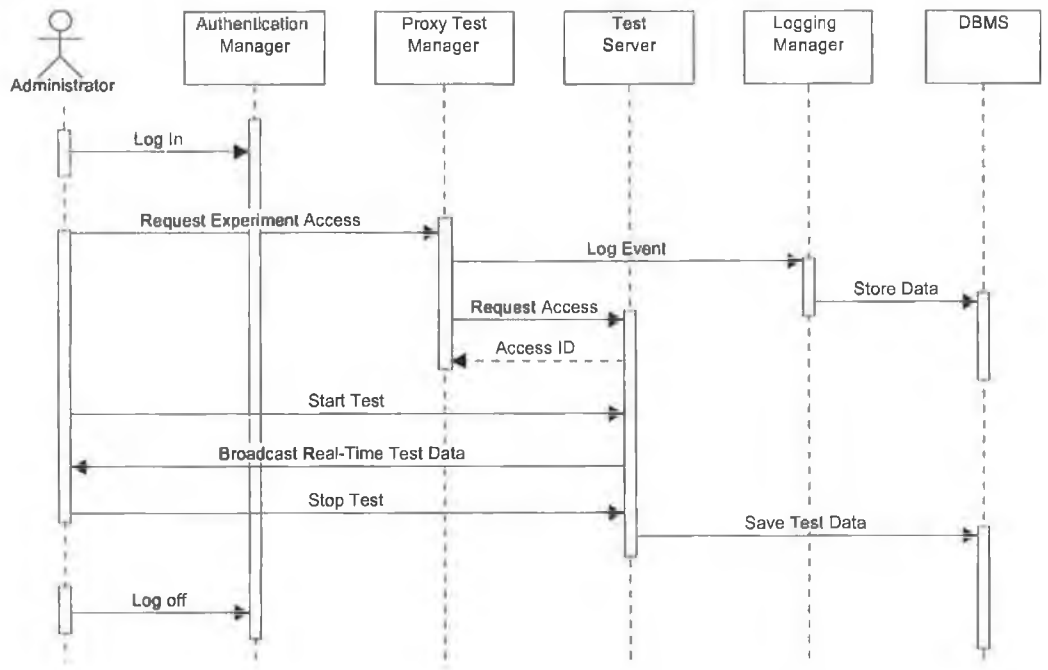

Figure 4.9: Real-Time Test System Integrated with J2EE-Based System

Observer clients will only require the use of the broadcast feedback mechanism in the test framework shown. As shown in Figure 4.10, everything else will remain in place.

Although the Stirling Engine demonstration equipment poses a low potential risk of injury, all possible safety precautions should nonetheless be taken to protect both the users as well as the equipment involved.

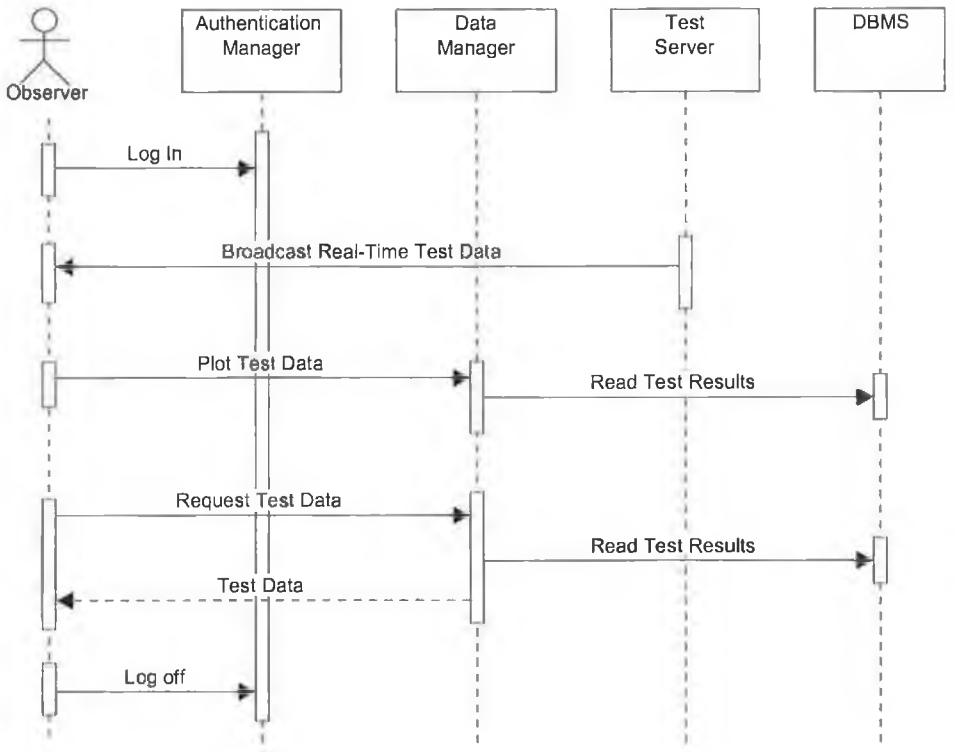

Figure 4.10: Sequence of Observer Involvement in Test Procedure

Administrators will typically only be concerned with controlling and monitoring the experiment while observers will be interested in dynamically plotting and saving the test results.

4.8 Summary of the System Analysis and Design

UML is used to communicate the system requirements and model the proposed sequence of operations for the proposed remote technology demonstrator. The first demonstration will be based on a Stirling Engine and will aim to show how this type of engine operates as well as being able to determine its overall thermal efficiency.

The system will support administrator and observer user types. It is intended that administrator users will act as the tutors or instructors of the system, while observer user accounts will be used by those learning the underlying concepts. Administrators will be capable of using all features accessible to observers including with additional privileged features. These will include user creation and removal as well as control of available demonstration resources.

The system is responsible for transferring resultant test data in a language independent manner between the C++ test server and the Java-based remote

technology demonstrator. Learners will be able to graphically plot test results for analysis after the test cycle has completed. A single sign-on feature will preserve the user's security credentials to allow access to multiple available applications.

A number of J2EE Blueprint design patterns will be used for developing the J2EE-based system. The Session Façade pattern will enhance performance by re-routing entity bean requests through session beans. The Service Locator pattern will be used for distributed service object lookups. The Front Controller and Dispatcher View patterns will be used for view management and navigation in the web or presentation tier. The Intercepting Filter pattern will also be used on the presentation-tier, as a request handling mechanism, carrying out various processing on the requests themselves.

The real-time and J2EE-based systems should be seamlessly integrated and maintain a good level of safety and security in the combined overall system.

Chapter 5

Remote Technology Demonstrator Implementation

5.1 Introduction

This chapter will reveal the full complement of open source software chosen to implement the system. The hardware architecture is also described before attempting to determine the most appropriate soft real-time system for controlling and monitoring the Stirling Engine demonstration. The previous chapter outlined the intended approach for the development of the remote technology demonstrator using blueprint design patterns. The core J2EE-based system utilises the proposed design patterns, implementing the required user account functionality. Security issues are also addressed using Java technology.

5.2 Software Selection

One of the underlying objectives of this project was to address the overall cost factor by using open source software where possible. To this end several open source products were assessed and subsequently used in this work.

The open source application server, JBoss [34], was used along with the Apache's open source web server Tomcat [62]. SOAP services were realised using Axis [63], another open source product from the Apache software group. An open source Database Management System (DBMS), MySQL [64], was used at the back-end for data persistence. JFreeChart [65], an open source, graphing and charting library was used to plot test result data. Sun's reference JavaMail implementation was used for the application's email requirements.

While the test server was implemented using proprietary technologies, some open source elements were employed. OmniORB [66] was the CORBA implementation used for test server control and EasySOAP++ [67] was used as a SOAP client. Both of these products utilise other open source products, namely, the Python language tools [68] and the Expat XML parser [69], respectively.

5.3 Hardware Architecture

Figure 5.1 shows a simplified line diagram of the hardware layout of the Stirling Engine demonstration. The starter motor, omitted from this diagram, is used to overcome the initial inertia in the engine flywheel at start-up.

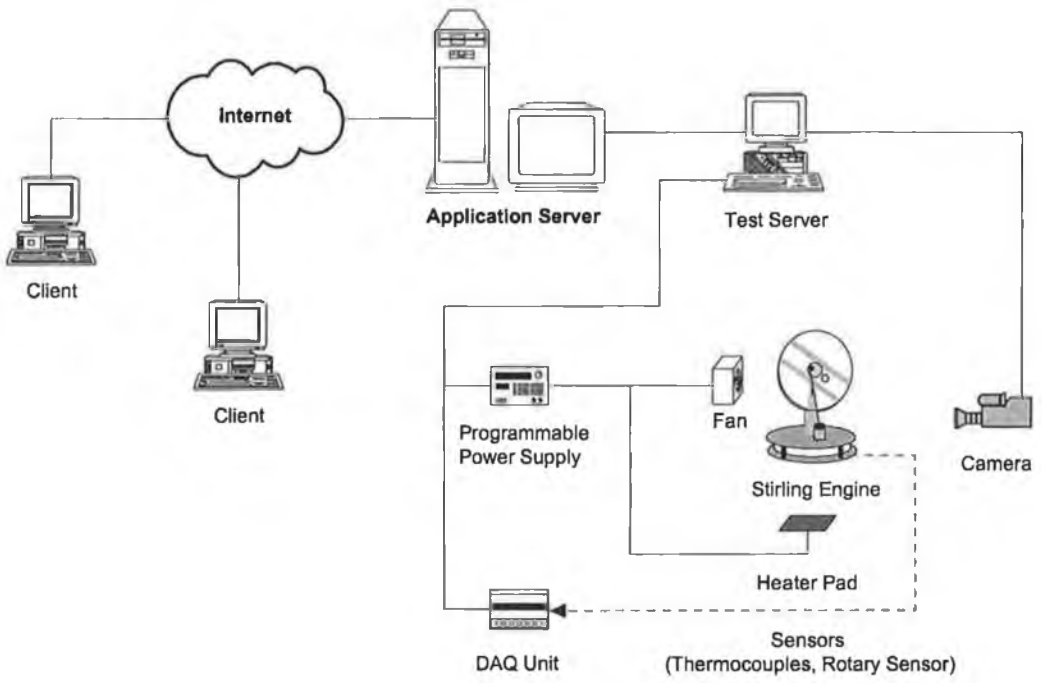

Figure 5.1: Hardware Architecture of the Stirling Engine Demonstration

Two thermocouples were used to read the hot and cold plate temperatures of the Stirling Engine. A Commercial Off-The-Shelf (COTS) DAQ system was used to record performance data from all sensors. This consisted of a multi-function DAQ unit which contained a number of power outputs and analogue input and output channels. This unit was used to read the thermocouple temperatures. An analogue output from this unit was used to drive the starter motor. A transistor proximity sensor was used for determining the rotary speed of the flywheel. This was powered by a 5 volt DC power output from the DAQ unit and the output was read back through the DAQ unit through an analogue input channel. This circuit was activated by a piece of aluminium mounted on the edge of the Perspex Stirling Engine flywheel. The proximity sensor detected the presence of the metal which caused an output signal to be generated and subsequently detected by the test server.

The heater pad and cooling fan were used to heat the hot plate and cool the cold plate of the Stirling Engine respectively. The heater pad and cooling fan required additional power and were powered by a Programmable Power Supply (PPS) controlled though an RS-232 interface. This allowed the user to experiment with different operational temperatures in the test procedure. A web-cam is used to provide visual feedback. The full complement of test equipment used in the Stirling Engine demonstrator is shown in Figure 5.2.

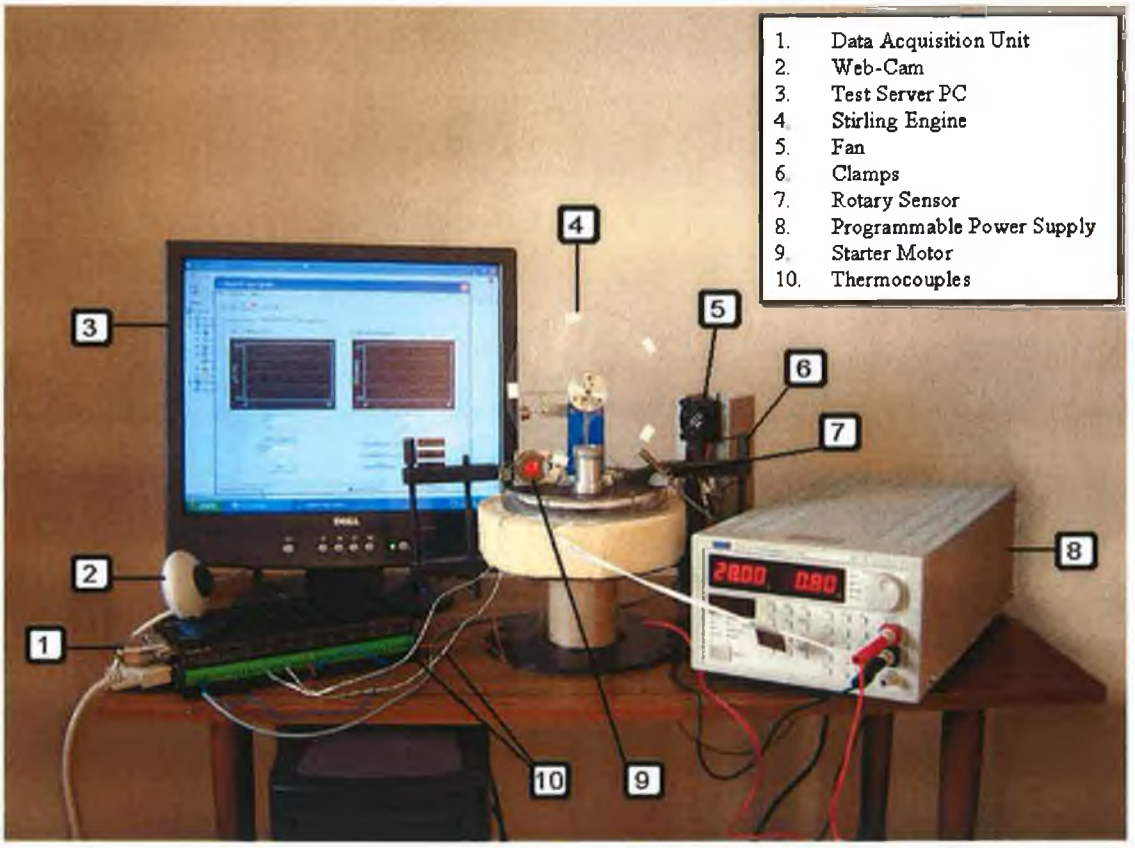

Figure 5.2: Test Equipment Used for Stirling Engine Demonstrator

5.4 Real-Time System

The real-time system of the Remote Technology Demonstrator (RTD) is based heavily on existing systems such that developed by Ko *et al.* [10]. This combines the best features of those virtual laboratories. Additionally, the real-time element has been designed to interoperate with the J2EE system, allowing all the additional requirements of their work to be realised. This includes transaction management, extensibility and security.

5.4.1 Real-Time Control

In Chapter 2, it was noted that Ko *et al.* [26] re-evaluate the protocol used by their earlier virtual laboratory [10], opting to use IP multicast instead of TCP for test data monitoring purposes. Obviously, protocol selection is critical when developing a soft real-time distributed system. This is particularly true if it is intended that a large number of users are to receive data from the application over the network. Despite changing the protocol for data monitoring, Ko *et al.* [26] retain TCP as the protocol for controlling the remote experiment. The requirements are similar to those inherent in the remote technology demonstrator application and their approach may be

leveraged to derive the control functionality of the real-time element of the virtual laboratory.

With regard to the control element of the new system, network traffic is low as only one person may control the test equipment at any one time. Reliability is the most important factor as the administrator may wish to react to a system event to modify the operational parameters or abort the procedure. For instance, the administrator may wish to increase the power output from the PPS powering the heater pad if the Stirling Engine is not being heated quickly enough.

Ko *et al.* [26] use a reliable TCP based system for their control requirements. However, the downside of this approach is that the developer must address issues such as connection management, error code translation and socket data-to-control parameter mapping. For this work, CORBA was selected over TCP as it avoided the aforementioned problems and offered an equally reliable protocol as confirmed by Orfali and Harkey [25].

An administrator user may request access to the test equipment from the test server. The test server[1] itself only accepts requests from a configurable list of host addresses. If this condition is met and if the test server is not already being controlled by another user, then the administrator may proceed to use the server. If authorised, the administrator may invoke operations on the Stirling Engine control interface, which includes operations for starting and stopping the engine. The interface itself is implemented using CORBA through its Interface Definition Language (IDL). The IDL defines all available control operations for the test server and is made remotely available though the IIOP protocol.

5.4.2 Soft Real-Time System Safety

As mentioned earlier, the real-time system used in this research is a soft real-time system delivering timely and accurate feedback of data. Srinivasan *et al.* [70] declare that in soft real-time applications, tasks are allowed to miss their deadlines. When using video conferencing software, for instance, it would not be critical if a video frame update was not delivered to the client provided that the bulk of the video frames were received successfully. Puschner *et al.* [71] acknowledge that standard Java is

[1] Appendix D describes the operation of the test server application.

however not suited for programming hard real-time systems. The situation for soft real-time use of Java is not quite straightforward either.

On the web interface, no software-related to the primary safety features of the system are provided as network latency increases the potential delay in responding to a critical situation. As a result, all such functionality resides within the test server.

Soft real-time feedback of test data is required to give the user an indication of the progress of the active demonstration. As all critical safety requirements are removed, it is acceptable if data monitoring feedback tasks miss their deadlines.

5.4.3 Client Monitoring Protocol Evaluation

A performance evaluation was undertaken to ascertain the most suitable protocol for real-time test data monitoring. A key requirement was that a potentially large number of network users, both administrators and observers, could simultaneously observe data from experiments that were being conducted remotely. Consequently, a large volume of data was required to be delivered to these users.

A standard test was devised to compare four possible protocols, TCP, IIOP, UDP and IP multicast. A server was developed for each protocol which generated sine wave data and relayed this information to a test client. The test client applet could then graphically plot the data and the expected sine wave could be observed. The test was run on a 500 MHz PC and an average of 480 data points per second were generated. Each test was run for thirty seconds for each protocol. The network used was 10/100 Ethernet Local Area Network (LAN) with tree topology. There were approximately 75 users actively using the system at the time of testing and the network traffic was mainly TCP based.

This test applet would be a prototype of a more advanced client applet, which could receive real-time data from the Stirling Engine test server. It is important to note that there will be a delay when using Java applets, as a security manager layer is involved when processing data from remote hosts. Figure 5.3 shows the composite test client where the TCP protocol is selected. Sun's appletviewer tool was used and its security policy was adjusted to allow the transfer rate and time to be logged to file.

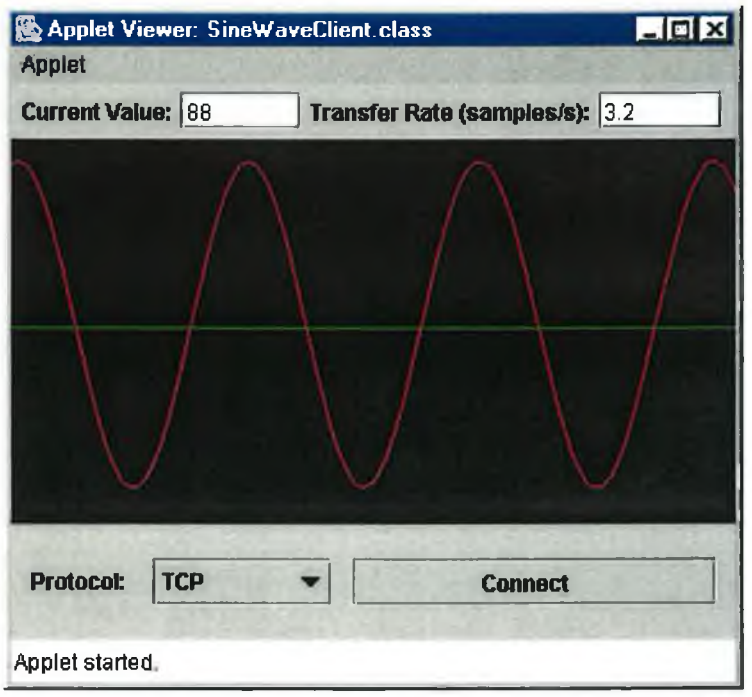

Figure 5.3: Composite Test Client Applet after TCP Test Cycle

As expected TCP provided a very reliable sine wave display. However, after a few seconds into the test, the transfer rate began to drop off dramatically and the plot rate slowed noticeably. This was due in principal to the client being overwhelmed with data from the network. TCP's reliability comes at the price of increased network traffic in the form of acknowledgement packets for data received. This problem is further exasperated in this system when more users request to observe the test.

JavaIDL was used in the client's IIOP mode for sine wave data testing. The findings showed that this CORBA implementation is not suitable for real-time monitoring of test data. Although the sine wave showed no signs of distortion as expected, having similar reliability to the TCP test. However, the transfer rate was quite slow and was in fact the slowest of the four protocols tested.

UDP provided a faster transfer rate but, as mentioned earlier in Chapter 2, it is unreliable. Initially the sine wave is well formed, but it becomes apparent that data packets are being lost. Figure 5.4 shows an active client where this effect is shown.

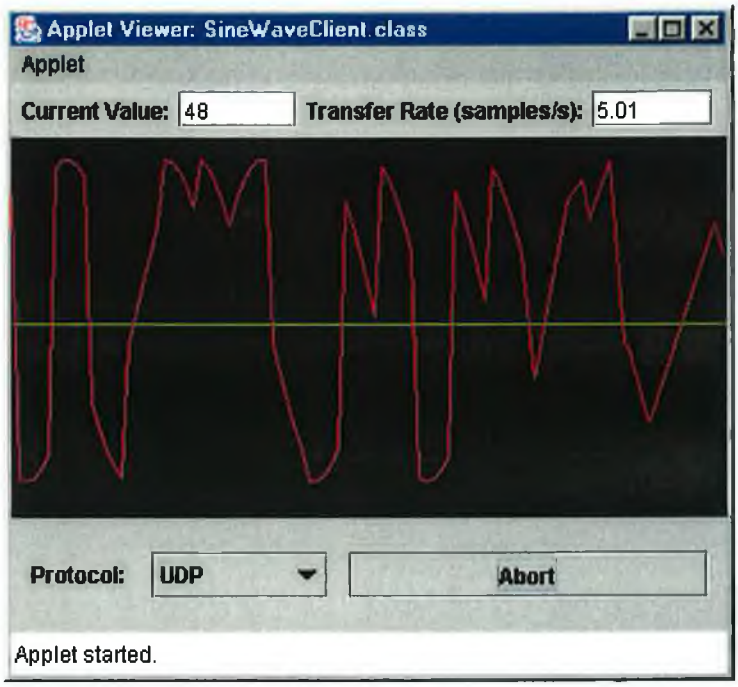

Figure 5.4: Data Packets Being Lost in Unicast UDP Test Mode

The IP multicast protocol showed consistently good performance throughout the short test. In addition, despite the fact that the protocol is based on UDP, the reliability was significantly better then the UDP unicast test. Figure 5.5 shows how the sine waveform may still be determined albeit with the loss of some data. The transfer rate was almost three times the average rate measured for UDP. This can be attributed to the fact that no connection management was necessary and also no delivery acknowledgement data packets were used.

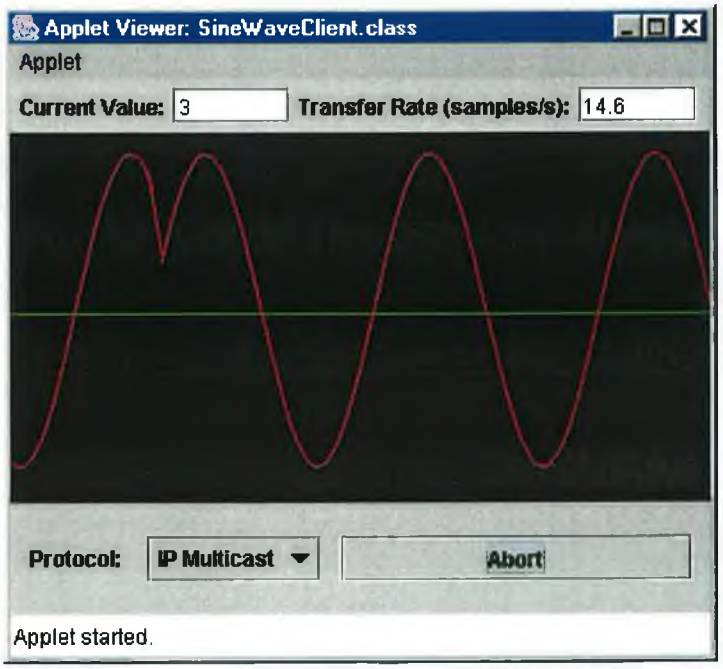

Figure 5.5: IP Multicast Sine Wave Test

Figure 5.5 shows transfer rate versus time for the four protocols under consideration. It is important to note that the transfer rate refers to the rate at which the server can dispatch the sample data and does not guarantee the delivery on the client side.

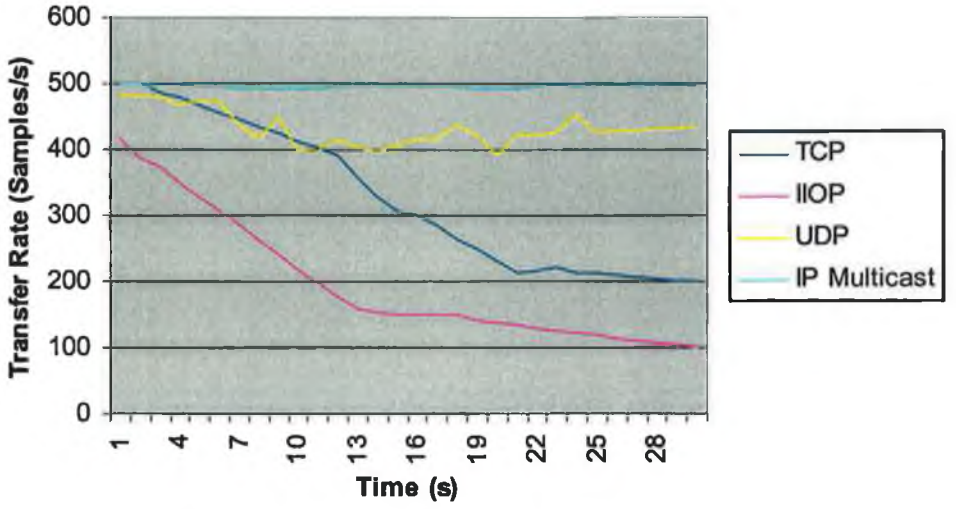

Figure 5.5: Transfer Rates for the Four Network Protocols Tested

5.4.4 Client Monitoring Protocol Evaluation Conclusion

The IP multicast protocol is clearly the best protocol of the four for continuous transfer of data under these conditions. There is no apparent degradation in delivery performance on the server. This is due to the fact that there is no data packet acknowledgement system in place such as those used by TCP and IIOP. Also there is no connection for the server to manage. This permits the number of clients to increase without adversely affecting performance on the server. This characteristic is useful for the intended test server application as there is a good deal of other processing work performed and eliminating connection management reduces the overhead on the application's resources.

From the evaluation carried out, it was evident that both TCP and IIOP were not appropriate for real-time monitoring. UDP has good characteristics for long duration remote monitoring by offering a fast transfer rate. Ko *et al.* [26] improved on their Frequency Modulation experiment by using IP multicast which itself uses UDP. The client protocol evaluation tests confirmed that this was indeed the best protocol for 'streaming' experimental test data. It was decided to adopt this approach as it solved the issue of having a potentially large number of experiment observers without

requiring the data to be carried over the network multiple times. In contrast to the control protocol, IIOP, discussed earlier, reliability, in this case, was not a major issue as the monitoring facility was only intended as an approximate indicator of the current status of experimental readings. Unlike the sine wave test, continuous displays such as strip charts are not feasible so display readouts are used instead.

5.4.5 Using the Real-Time Client Applet

The real-time applet is divided into four tabbed panels designed to be used sequentially in the Stirling Engine test procedure. Upon loading the applet, all control components, such as test server connect button, are enabled if a token has been passed to the applet in the form of an applet HTML tag. Observer users will not have this token and consequently will not be able to issue test server control commands. Even if they did discover a token value, tokens must match a user identity and client host address in order for the test server to permit commands to be carried out.

The first panel displayed to the client user is the access control panel. This panel contains the Connect and Disconnect buttons which allow administrators to establish a session with the test server using a previously assigned token. Once this session is established, all other users may only observe the test procedure until the controlling user disconnects from the test server.

A web cam is used to provide visual feedback for the real-time applet. Third party software is used on the test server machine, which offers image data over HTTP. A stream of constantly updating images was initially used to produce a live view of the test apparatus. However, this was found to be extremely resource intensive as the images server software did not support proper video streaming. Also when the rotational speed of the Stirling Engine flywheel was over 5 rpm, the client view was quite blurred. As a result, a simple button was provided to allow the user to refresh the image feedback display. Figure 5.6 shows the access control tabbed panel as it appears to the administrator.

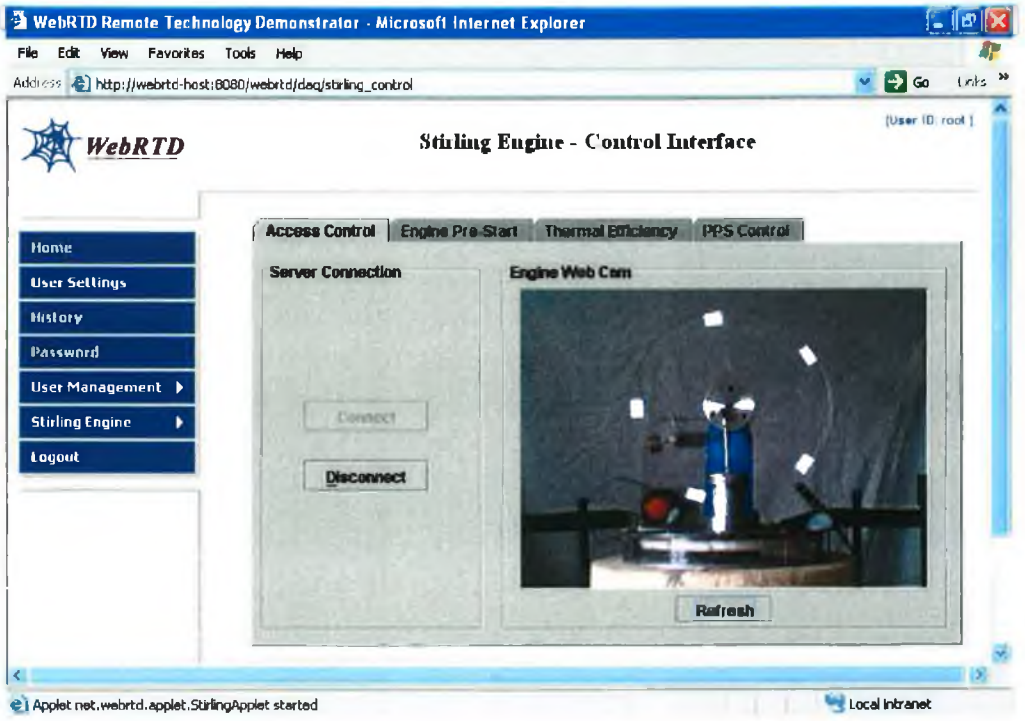

Figure 5.6: Stirling Engine Applet Access Control Panel

Up to this point we have indicated that the Stirling Engine used in the test experiment could be merely 'started' and 'stopped', but anyone who is familiar with these types of engines will know that this is not possible. The engine operates on the principle that when a confined quantity of gas is heated, the volume will increase, driving a piston upwards. As the engine reaches mid cycle, the gas cools and contracts causing the piston to drop back down. Earlier, Figure 5.1 included a fan and heater plate controlled by a Data Acquisition (DAQ) Unit connected to the test server PC. The purpose of these was to cool the cold plate and heat the hot plate of the Stirling Engine respectively. The engine will not operate unless the temperature difference between these two places is above a critical threshold temperature. As a result, the administrator must remotely heat the sealed air chamber of the engine for a period of time prior to undertaking the main experiment. The test server will ensure that the pre-heat stage is carried out before allowing the user to proceed.

Figure 5.7 shows the Engine Pre-Start panel of the applet where the user may set the trigger temperature and the starter motor duration.

The following terms are used:

Trigger Temperature:

Term used for critical threshold temperature difference.

Motor Duration:

Time allowed for the starter motor to rotate in order to impart energy to the flywheel of the Stirling Engine and overcome the initial inertia of the flywheel.

The Start button will begin a feedback loop on the test server where the PPS will supply heat to the Stirling Engine while the hot and cold plate temperature difference is monitored.

The right hand side of the Pre-Start display contains readouts of the temperatures of the engine's hot and cold plates as well as their temperature difference. This gives some feedback on the status of the system for this stage of the experiment.

On reaching the specified trigger temperature, the starter motor uses the friction of a rubber contact to impart energy to the Stirling Engine flywheel for the length of the motor duration. The motor then stops and the Stirling Engine should be then running by itself using the thermodynamic principles of gas expansion and contraction. The client will now also stop receiving broadcast values relating to the Engine Pre-Start stage.

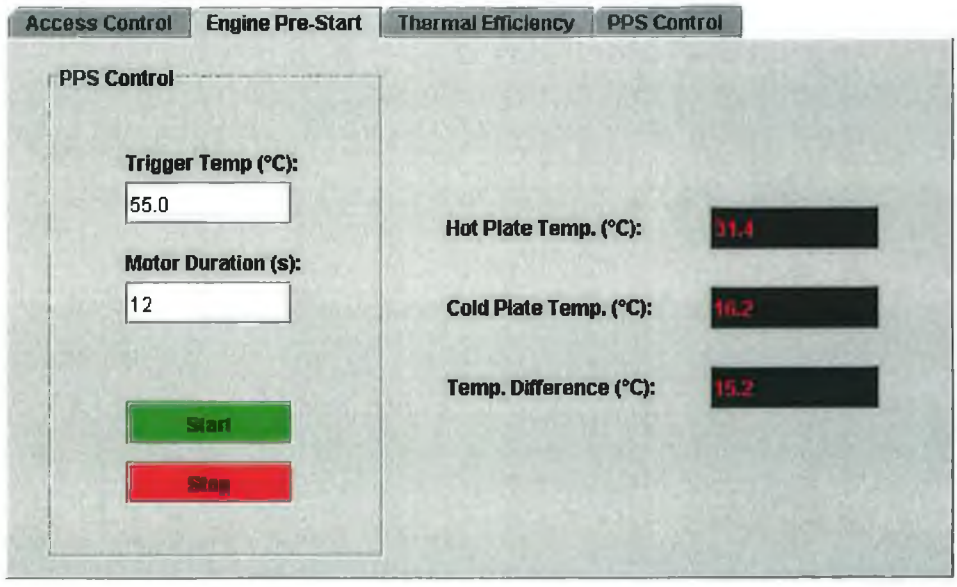

Figure 5.7: Stirling Engine Applet Pre-Start Panel

Once the threshold temperature has been reached and starter motor has been activated, the flywheel should be set in motion. The user may check that this is the case by getting an update from the web cam display. The main objective of this

experiment was to determine the thermal efficiency of the Stirling Engine. Figure 5.8 shows the Thermal Efficiency panel of the applet. The user may set the test cycle duration prior to starting the test. Clicking the Start button will cause a feedback loop to start on the test server where the relevant sensor data is stored locally to file while the test duration is monitored. A safety mechanism in the test server prevents the experiment from being initialised when the temperature difference is below the trigger temperature as set in the Engine Pre-Start stage. Also if the rotational speed indicates that the Stirling Engine flywheel is static for approximately ten seconds, then the test is aborted and the user will be required to return to the Pre-Start stage.

When the test is in progress, readings related to the efficiency such as rotational speed of the flywheel and mechanical power output are displayed on the right hand side of the panel. When the test duration has elapsed the test server processes the test data and relays the results back to the core J2EE system so that they may be saved to the database.

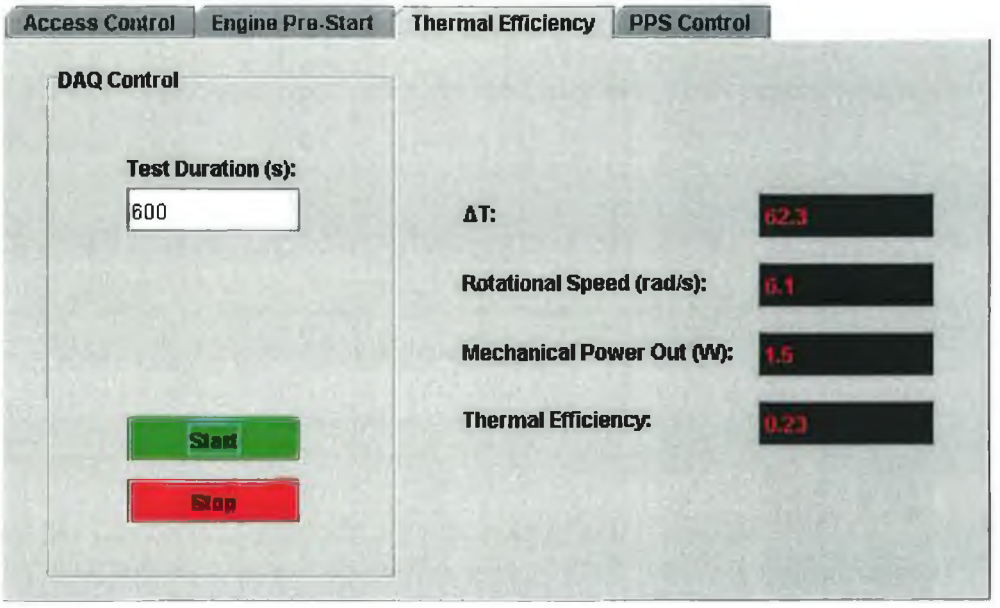

Figure 5.8: Stirling Engine Applet Thermal Efficiency Panel

The level of heat must be maintained in the Stirling Engine air chamber between the pre-start and efficiency testing stages. Indeed the PPS continues to supply power to the heater pad after the test cycle so that multiple tests to be carried out in succession. A safety mechanism on the test server ensures that power from the PPS is cut after an idle timeout. The user should, however, use the PPS control panel in order

to stop the power output from the PPS manually. Figure 5.9 shows how the voltage and current may also be set on this panel.

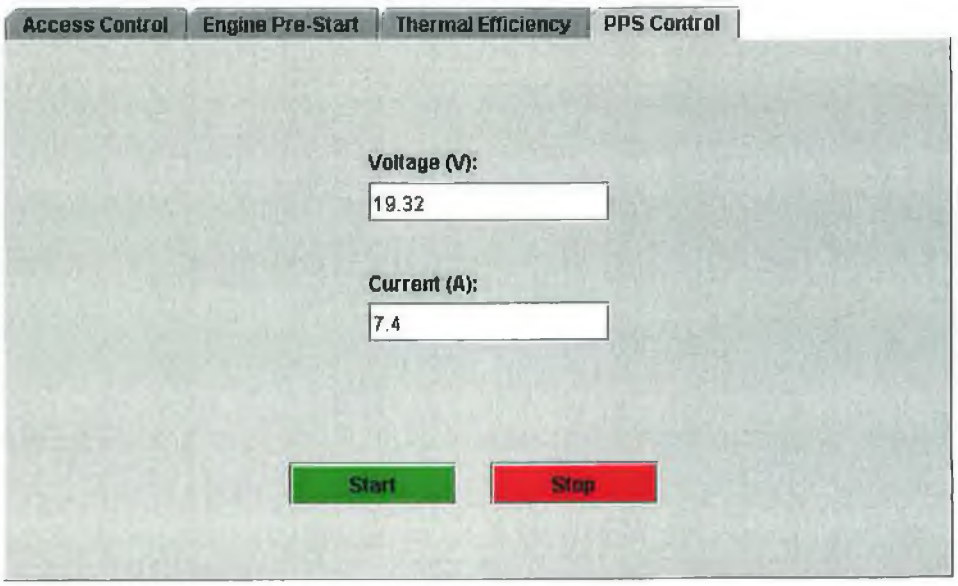

Figure 5.9: Stirling Engine Applet PPS Control Panel

Figure 5.10 shows an updated sequence diagram for the real-time test system.

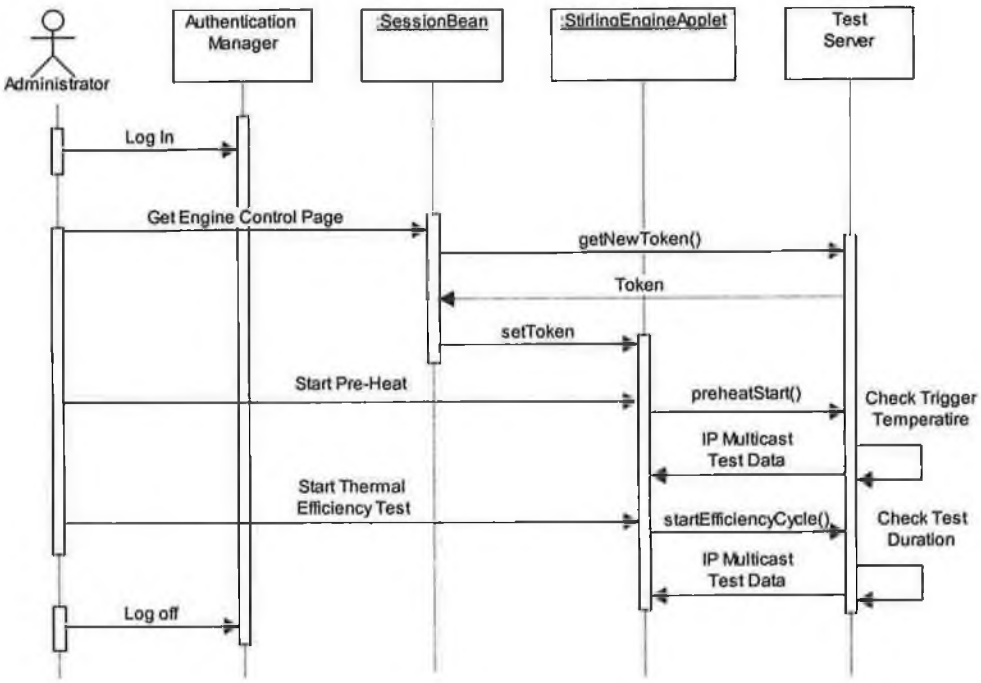

Figure 5.10: Updated Model for Real-Time Test System

5.5 Core J2EE-Based System

The core J2EE-based system introduces a new transaction management element into the area of virtual laboratories. This feature is more commonly associated with such e-

commerce enterprise applications as mobile phone subscriber portals such as Vodafone [72] and online shopping web sites such as Amazon [41].

For the RTD implementation, the web-tier is the first point of access for the user. This is built on a number of enterprise layers and is integrated seamlessly with the real-time control and monitoring system. The following sections will outline the core J2EE-based system from the database management system to the web-tier.

5.5.1 Database Management System

The application server, JBoss, allows for the integration of numerous commercial and open source databases. The open source DBMS, MySQL, was chosen over the default Hypersonic database, as it offered good administration tools as well as a JDBC 2.0 compliant interface library. Another advantage was that an automatically incrementing integer data field facility was supported between JBoss and MySQL using a special JBoss database interface deployment descriptor XML tag. A database was created for the RTD application and secured with a username and password.

5.5.2 User Management Implementation

The EJBs used in this implementation are session beans and entities beans and represent business logic and persistent application entities respectively. A more detailed description of these types of enterprise beans is presented in Chapter 3.

This section will focus on the EJBs used for user management. The application's entity beans were developed first, as they required to be referenced at a higher level by session beans.

Container Managed Persistence (CMP) was chosen to implement the entity beans as the new standard of CMP offers a realistic alternative for development of transaction based objects. Bean Managed Persistence (BMP) allows the developer more freedom when reading and writing to the database but the developer must also take responsibility for issues such as concurrent data access. CMP enables the developer to spend more time concentrating on other design issues as the EJB container handles the CMP entity beans.

The first entity bean to model is that of the generic RTD user. This entity, *RTDUserBean*, must include the required attributes: login name, password, full name and user type. For this implementation, there is only one available test, the Stirling Engine efficiency experiment, so it is permissible to have a flag here to indicate

whether or not the user wishes to receive an email update of the experiment summary. This will be known as the user's 'auto notify' attribute. Each attribute will constitute a 'getter' and 'setter' abstract method pair, which is used by the EJB container for database actions. For example, the login field will be implemented using `getLogin` and `setLogin` methods.

Before proceeding it is important to decide how entity beans in the application are going to be made available. Prior to EJB 2.0, all entity beans were required to be implemented using an RMI-based remote interface, regardless of whether or not they would be invoked directly by an external client, i.e. one outside the EJB container's JVM. This adds significant overhead for every entity bean used. Hammouda and Koskimies [55] advocate the use of the Session Façade pattern [56] as a means of loosely coupling the communication between J2EE layers. When applied to EJBs, session beans become the first point of contact for all client applications. All calls to entity beans are made though the session beans. This eliminates the need for the application's entity beans to be available remotely and makes the application itself more secure. Local interfaces were introduced in EJB 2.0 to allow EJBs in the same JVM access each other. Cecchet *et al.* [73] confirm that using local interfaces improves performance by avoiding the communication layers for local communications. It was decided, therefore, to employ local interfaces in all entity beans in this project.

Logging is provided for auditing purposes. Each user of the system has a number of associated history records. These translate into entity beans also and must include date and event description attributes. Every container-manager entity bean is required to have a primary key attribute which is used to locate the object in the database. The login name field is used in the *RTDUserBean* entity bean for example. In the case of the *HistoryRecordBean* entity bean, an automatically generated integer is used. This attribute is configured in the entity bean's XML deployment descriptor and is interpreted by the database management system.

Another entity bean, *UserRoleBean*, is associated with *RTDUserBean* and is used in identifying the user's security role in the system. The automatic primary field identifier is again used the attributes include role name and role group name. This bean, is used to enforce the permission policy system of the application server. This security feature will be studied in greater detail later in the chapter.

CMP is of little value unless the components used can operate fully inside the EJB container and not resort to embedded SQL code segments operating via JDBC. Gitzel *et al.* [74] use Container Managed Relationships (CMRs), which were recently introduced in EJB 2.0. This relationship mechanism allows the EJB container to assume full responsibility of normalisation of application data, offering one-to-one, one-to-many and many-to-many relationships. As with CMP, CMR applies only to entity beans. Figure 5.11 shows a class diagram for the *HistoryRecordBean*, *RTDUserBean*, and *UserRoleBean* entity beans.

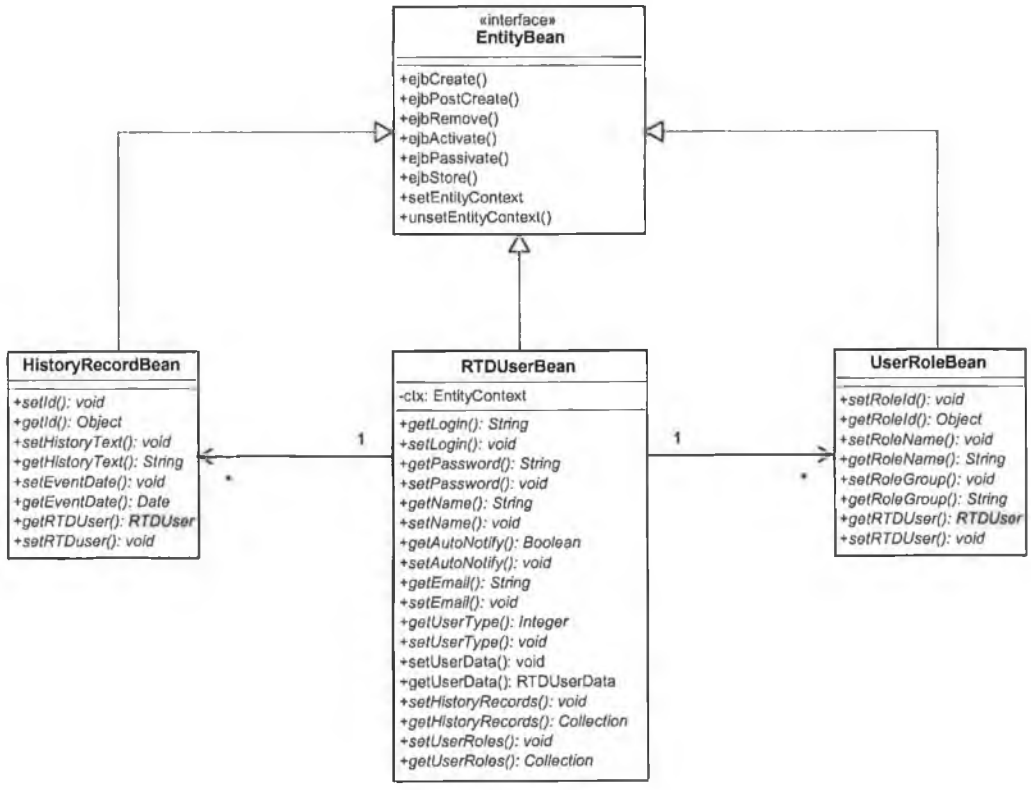

Figure 5.11: Class Diagram for RTD User Related Classes

As described in Chapter 2, session beans can be either *stateful* or *stateless*. A stateful session bean is used to access user management functions, as it is necessary to maintain current *RTDUserBean* entity bean data between method invocations. For example, when `getHistory` is called to retrieve the users history records, it is more effective to cache the entity data rather than attempting to locate it at though the *RTDUserHome* interface each time the call is made. Figure 5.12 shows a class diagram of the principal user management bean class, *VirtualLabBean*.

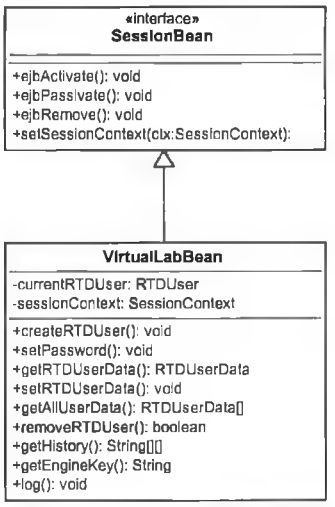

Figure 5.12: VirtualLabBean User Management Session Bean Class

This class is used for personal user management of the user's own account as well as the creation and removal of others. The Session Façade pattern is used, transforming the *VirtualLabBean* session bean into an access layer for the *RTDUserBean* entity bean interfaces.

An additional step is necessary when invoking business methods from enterprise beans from the web-tier. As discussed in the previous chapter, the Service Locator pattern is used to provide improved network and client performance through caching. Figure 5.13 shows the configuration of the Service Locator class with respect to the EJBs used.

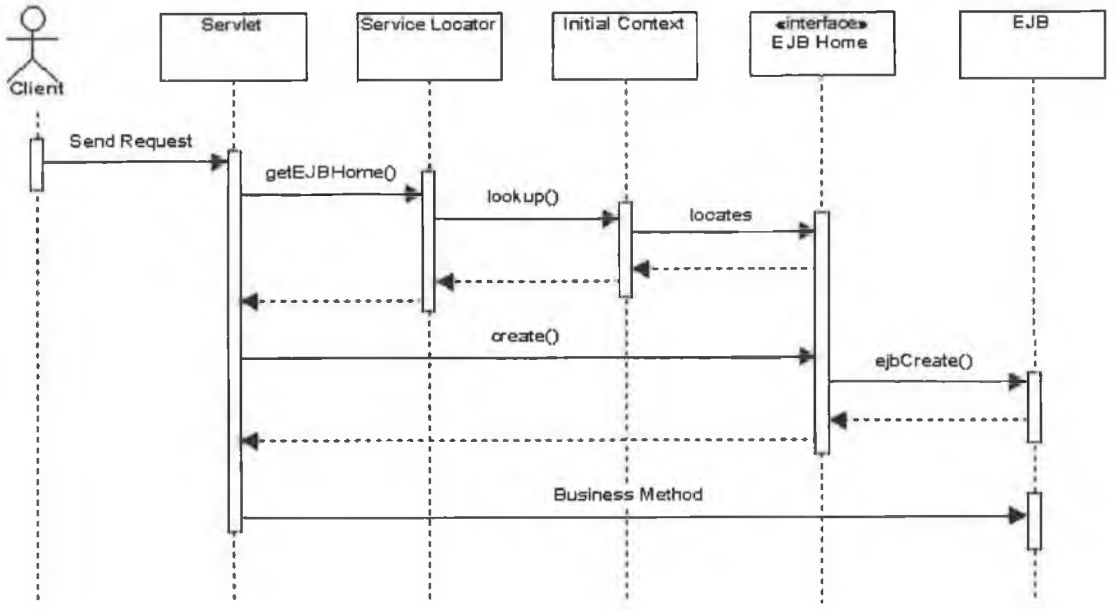

Figure 5.13: Service Locator Pattern Implementation

Refences to objects such as initial context handles used for looking up home references using JNDI may be cached for re-use at a later stage. All JBoss specific JNDI lookups may be contained within the one service locator class, thereby reducing the complexity of the client which is a Java servlet. Figure 5.13 represents the procedure involved the first time such a servlet is used to invoke an operation on an enterprise bean.

From the diagram, it may be seen that the service locator is used to get EJB home objects. Clients may use these subsequently to create enterprise beans and perform call the relevant business methods. It should be pointed out that all operations carried out on enterprise beans described in this implementation utilise the service locator mechanism even though it may not be explicitly shown.

Operations required for creating and removing system users are examples some of the more important operations that are implemented by the J2EE-based system. Figure 5.14 shows a sequence diagram depicting the creation of a new user by an administrator illustrating the interaction between the *VirtualLabBean* session bean and *RTDUserBean* entity beans.

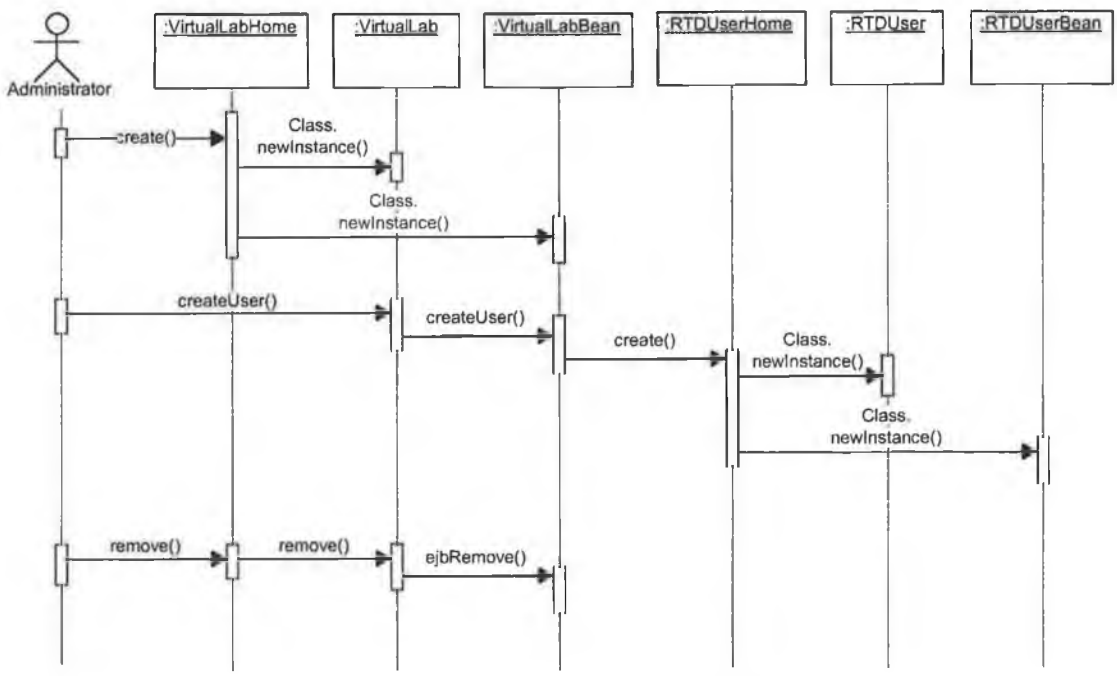

Figure 5.14: RTD User Creation Sequence

Figure 5.15 shows how a user is removed from the system. Observer users are not offered administrative features, such as create and remove user, at the presentation

layer. If any of the administrative methods were to be called by an observer user, then they would fail based on the role of the user. This restriction is enforced at the EJB layer.

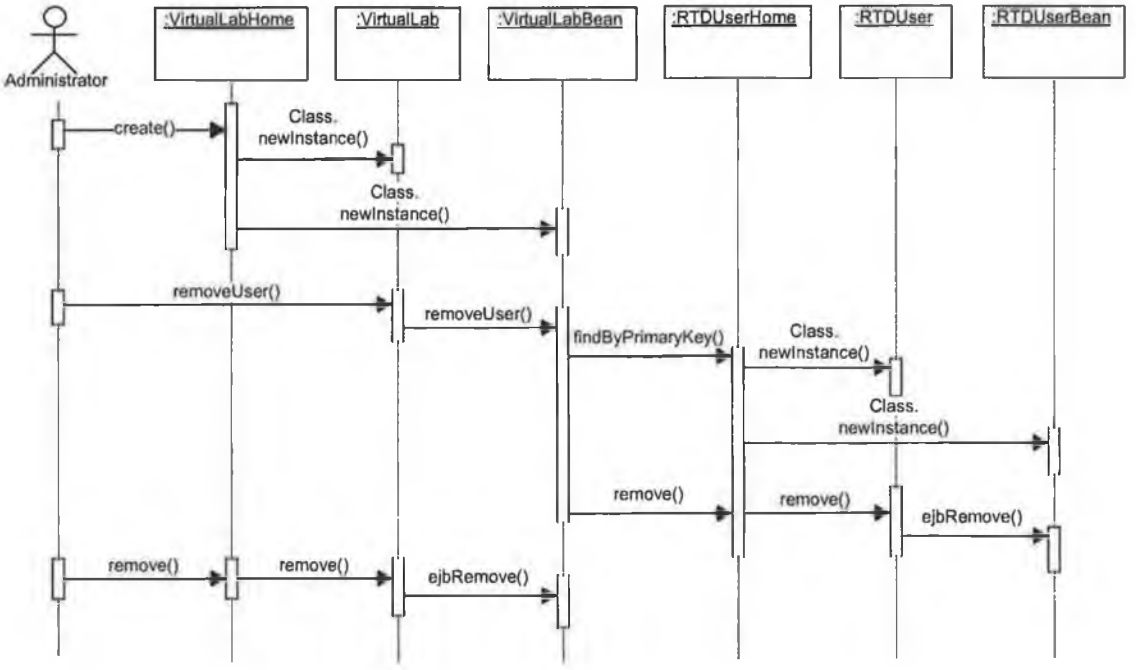

Figure 5.15: RTD User Removal Sequence

5.5.4 Web-based Presentation Layer

A simplified version of Sun's Front Controller pattern [58] is used for handling presentation-tier Java servlet requests. The application's web-based element is configured to that all requests pass through one controller servlet, *WebControllerServlet*. Figure 5.16 shows how this lightweight version of the pattern effectively handles user interaction while separating the business logic from the presentation logic.

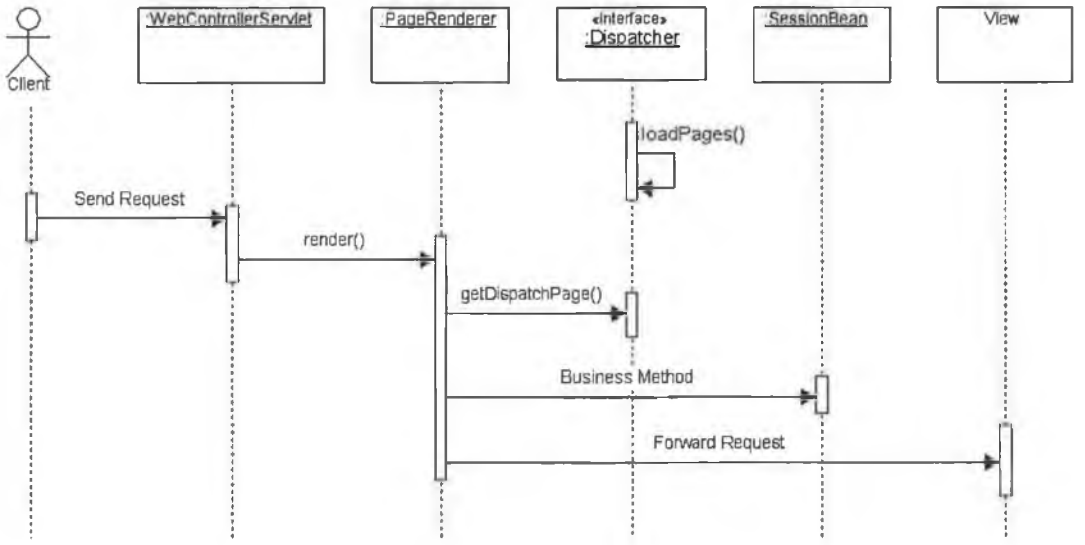

Figure 5.16: Implementation of the Front Controller in RTD Project

The Dispatcher View pattern was designed to control the flow of execution and manage access to presentation processing, responsible for generating dynamic content. In this application this is represented by the interface, *Dispatcher*. In practice the *PageRenderer* class actually implements the *Dispatcher* interface. This is shown in Figure 5.17. The principal methods in the interface are used for getting and setting the target HTML or JSP pages used in the web navigation. These dispatcher page details are stored in a property file which is read when the controller servlet is initialised. Using a property file allows the path of navigation to be modified without the need for changes in the source code.

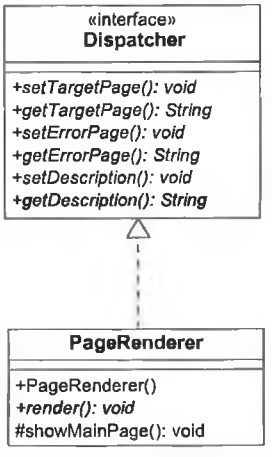

Figure 5.17: Class Diagram for PageRenderer class and Dispatcher Interface

The property file also contains logical links which are used to map web Uniform Resource Identifier (URI)[2] paths [75] to sub classes of the *PageRenderer* control class. These sub classes are retrieved from a hash map of page references as the user navigates through the web interface. The *HistoryRenderer* class is an example of one of these sub classes. Figure 5.18 shows a typical operation where a request is made to view the history records associated with the current system user.

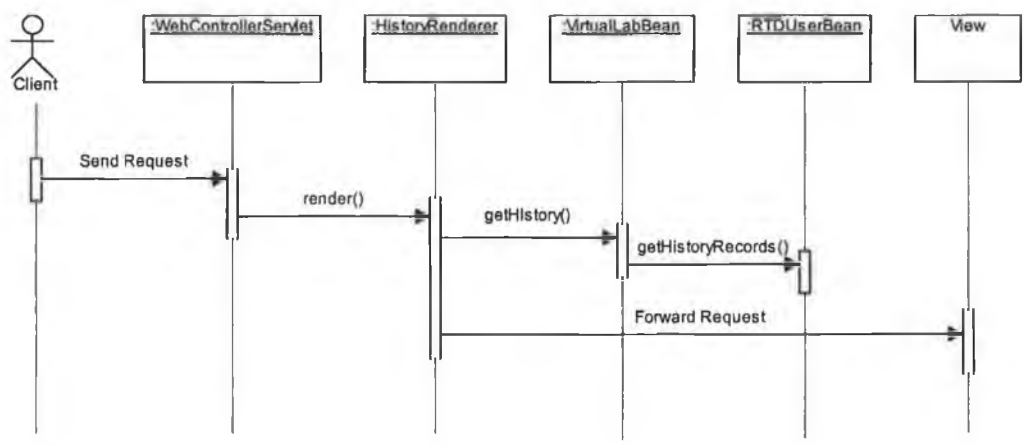

Figure 5.18: Viewing History Records

The presentation view is dynamic and may depend on the result of invoking a business method on an enterprise bean. Some page renderer objects, therefore, have an error page property, which specifies that a different JSP page be displayed if the requested operation does not complete successfully. Other page renderer objects may re-use the JSP page specified for a successful operation by setting a property on the page itself indicating that an error be displayed. This is illustrated in Figure 5.19 where the page used to change the user's password is re-used allowing for the user to make a mistake when attempting to change the account password. This design facilitates a clean separation of JSP, servlet and EJB layers allowing pages or classes to added or removed easily.

[2] A URI is a compact string of characters for identifying an abstract or physical resource.

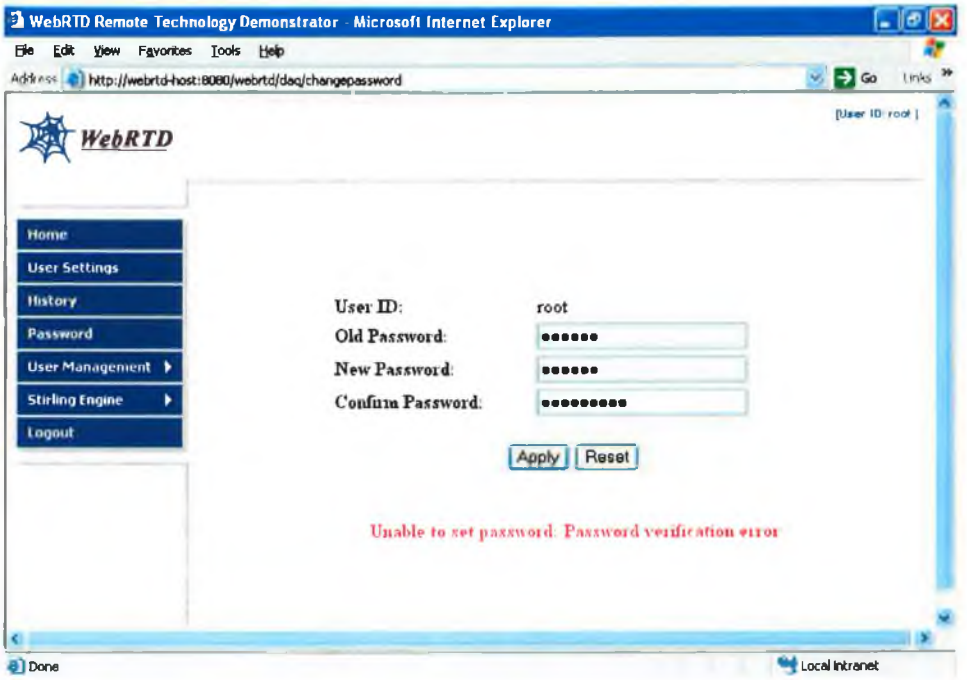

Figure 5.19: Re-use of Password JSP Page

The value object pattern is used when reading and updating the user settings to overcome the one-to-one nature of the accessor idiom used by the relevant CMP methods in the *RTDUserBean* entity bean. This combines a number of calls into one by using the *RTDUserData* object to pass data between the client and EJB layer, thereby reducing network traffic significantly. Monson-Haefel [46] describes the methods that use these objects as bulk accessors. Figure 5.20 shows how the *RTDUserData* value object is passed through the session façade to display the user settings. The *SettingsRenderer* class forwards the required user attributes to the JSP page. From the diagram, it is evident that the value object itself is only created when the bulk accessor method is invoked.

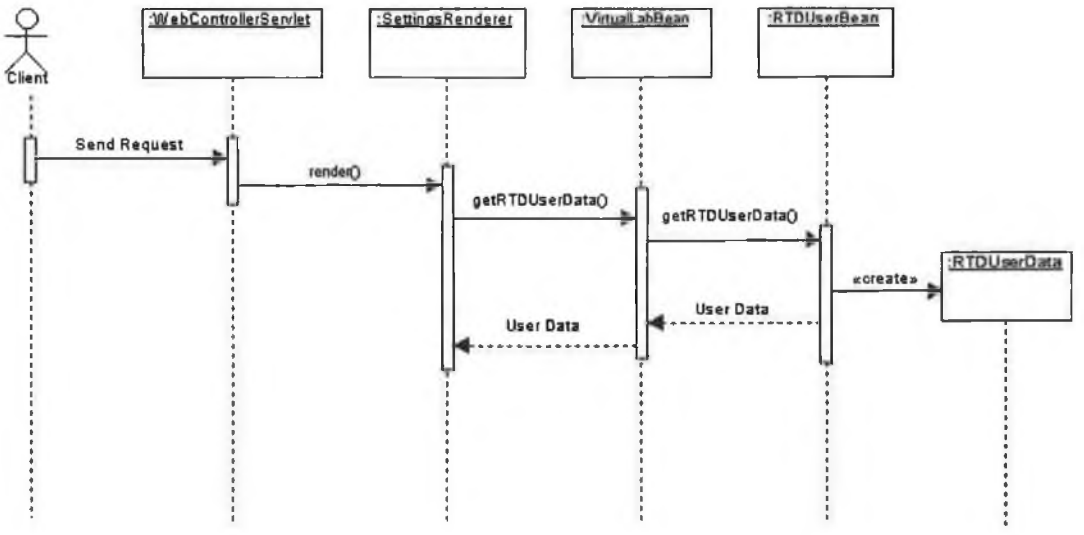

Figure 5.20: Sequence Diagram for Displaying User Settings

In the case of updating the user settings, the *RTDUserData* value object is created and populated with user attribute information before passing it through the bulk accessors of both the *VirtualLabBean* and *RTDUserBean* of the current user (Figure 5.21).

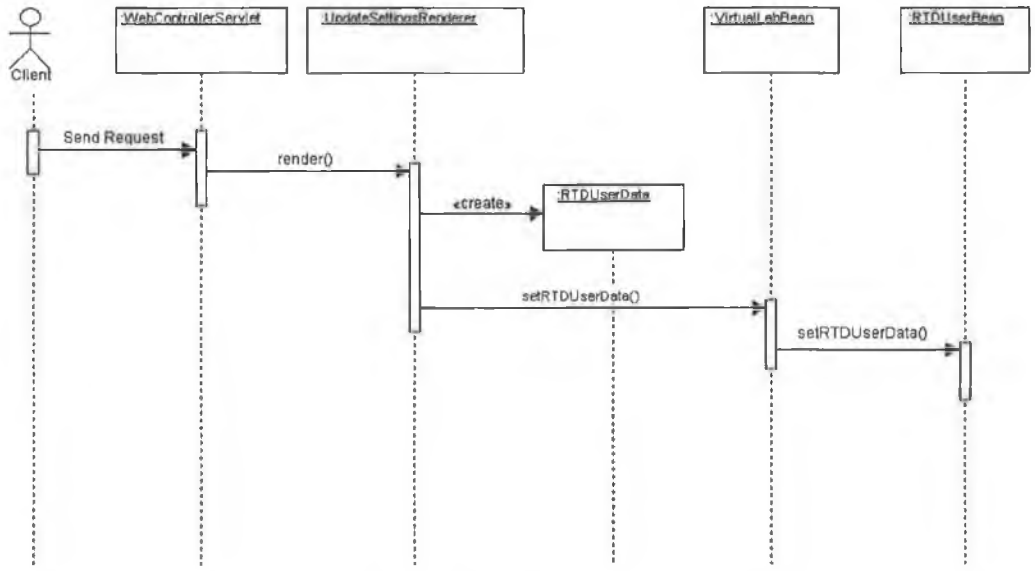

Figure 5.21: Sequence Diagram for Updating User Settings

5.5.5 Web User Interface Design

In Chapter 1, the design of the user interface was highlighted as an important aspect of the system. According to Nielsen [76], the two most important keys to a successful web site are content and usability. Hübscher *et al.* [77] highlight the importance of content organisation to support navigation through the web site. They state that there is no one solution as there are a number of factors involved in web design including the proficiency of the end user. Bevan [78] provides a useful template for addressing usability issues in web design. He asks a number of questions regarding the potential users of a web site, such as their purpose for accessing the site and experience and expertise. In this instance, administrator and observer users will access the web site, but may have varying degrees of expertise in web navigation.

A relatively simple and consistent web design was employed on the user interface. No patterned backgrounds were used, as these make text difficult to read. Low-resolution graphics employed in the design as many web browsers have slow connections. Animations and flashing text were also avoided as users find this very distracting. Tables, most of them borderless, were used instead of frames, as frames can interfere with printing and bookmarking. A JavaScript menu was positioned on

the left-hand-side of every page ensuring that the user always has access to the key features provided by the web user interface. All functionality relating to the Stirling Engine demonstration was grouped under a 'Stirling Engine' menu item on the menu. This allows for the future demonstrations to be added in a similar manner. HTML ALT tags used to describe links associated with graphics. These tags define web page text that is displayed when the image is loading or missing. The Web interface was tested in Netscape Navigator and Internet Explorer to ensure consistent behaviour and appearance of the user interface. A minimum operating resolution of 800 x 600 pixels was used during the user interface design. All web content is resizable and care was taken to ensure that all controls in the embedded real-time applet would be clearly seen at this resolution.

Another key aspect of the system, indicated in Chapter 1, was the effective communication of the principles and background information on the remote demonstration. To this end, a brief online help is provided to familiarise the user with the operation and underlying thermodynamic principles of the Stirling Engine. Figure 5.22 shows the initial section of the help page as it appears in the web user interface.

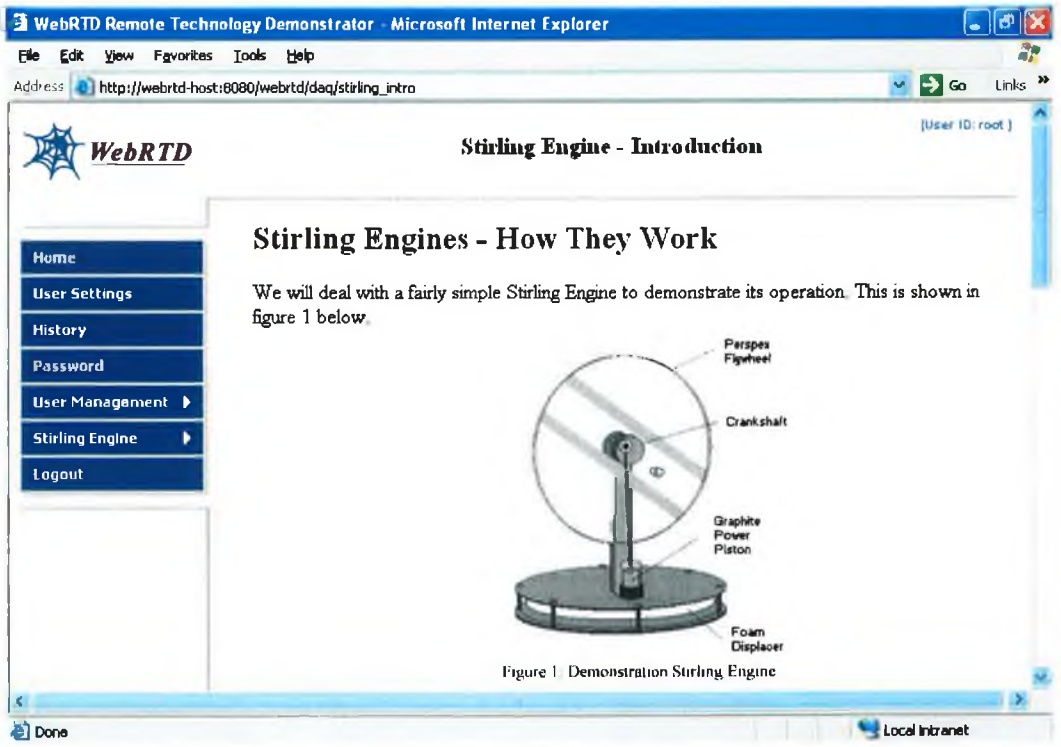

Figure 5.22: Online Help Web Page for Stirling Engine Demonstration

5.5.6 User Type Based Content Generation

The Servlet Intercepting Filter Pattern [61] is used to mange the content based on the user type. A filter is attached to the *WebControllerServlet* for modifying the forward requests received by JSP pages. JSP pages then allow access to additional system functionality by including HTML elements with links to administrator specific features such as user management. Figure 5.23 shows the difference between the main web page after both an administrator and an observer user type has been authenticated.

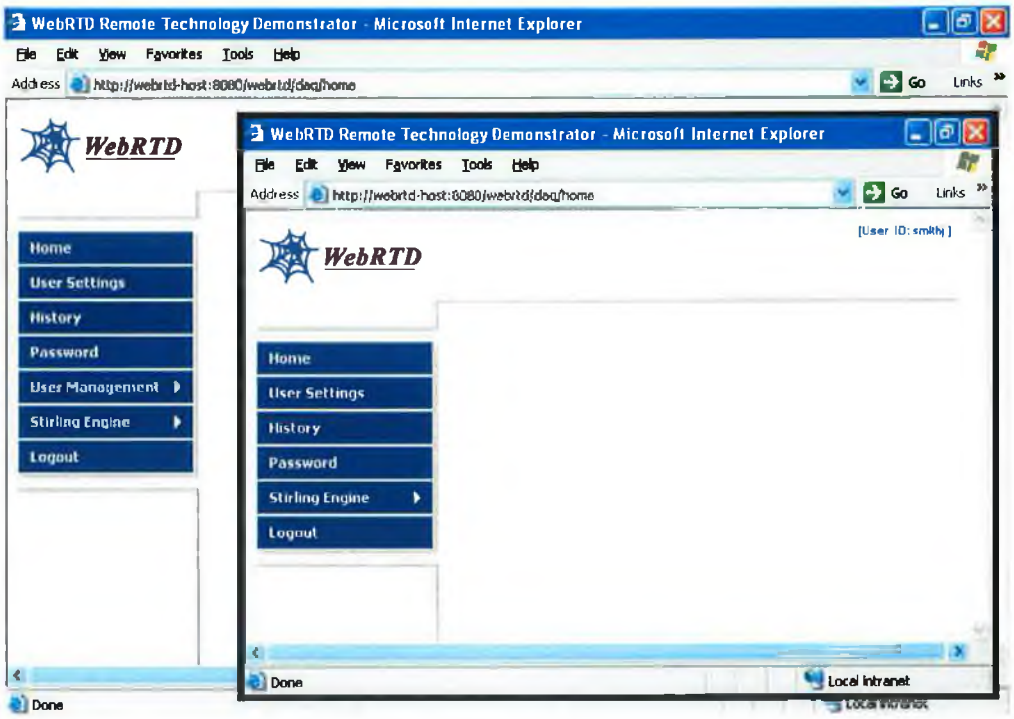

Figure 5.23: Main Web Page for Administrator User (Inset: Main Web Page for Observer User)

All resource management facilities should purposely omitted using this approach. However, in the case of the Stirling Engine demonstration, the same applet code serves both administrator and observer users alike. Token requests are not made to the test server on behalf of observer users. This results in the client applet being used minus the token required for control. In this case, control features on the applet will be disabled for observers.

5.5.7 Data Callback Service

The data callback service is required to facilitate the reliable sending of processed test information and results to the application server so that the data may be written to the database. If this is achieved, then clients will not require test servers to be active in

order to retrieve and analyse results from previous test cycles. A Java-only option such as RMI was not feasible as the test server itself was written in C++.

CORBA presented the possibility of sending the data between the test and application servers due to the fact that, in recent releases, JBoss has enabled the configuration of IIOP interfaces in the application server. However, in order for clients to take advantage of this new feature they must support some of the newer features of the CORBA standard such as `WStringValue` objects and the CORBA `valuetype` keyword.

As discussed in Chapter 3, Web Services was one of the most recent additions to the J2EE specification. One of these services, namely SOAP, provided the solution for the callback requirement. SOAP is platform and language independent and typically operates over HTTP using XML data content. Apache Axis is an open source Java SOAP implementation and is fully functional as a plug-in to JBoss. J2EE maps SOAP operations to stateless session beans methods. Figure 5.24 shows the session bean, *TestResultsBean*, used for test data callbacks. No UDDI registry is necessary as the callback service is relatively straightforward and is already known to the test server application.

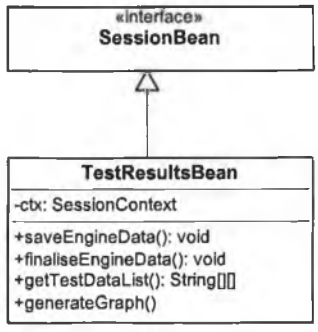

Figure 5.24 Test Results Bean Used for Test Data Callbacks

When an experiment has completed, this class is called repeatedly by the test server to save all the processed test data. Again container-manager entity beans with local interfaces are used to store the information. The *TestResultsBean* is responsible for sending summary email updates once all the data has been saved successfully. Figure 5.25 shows how the test server interacts with the SOAP provider and in turn the EJBs used to store the information.

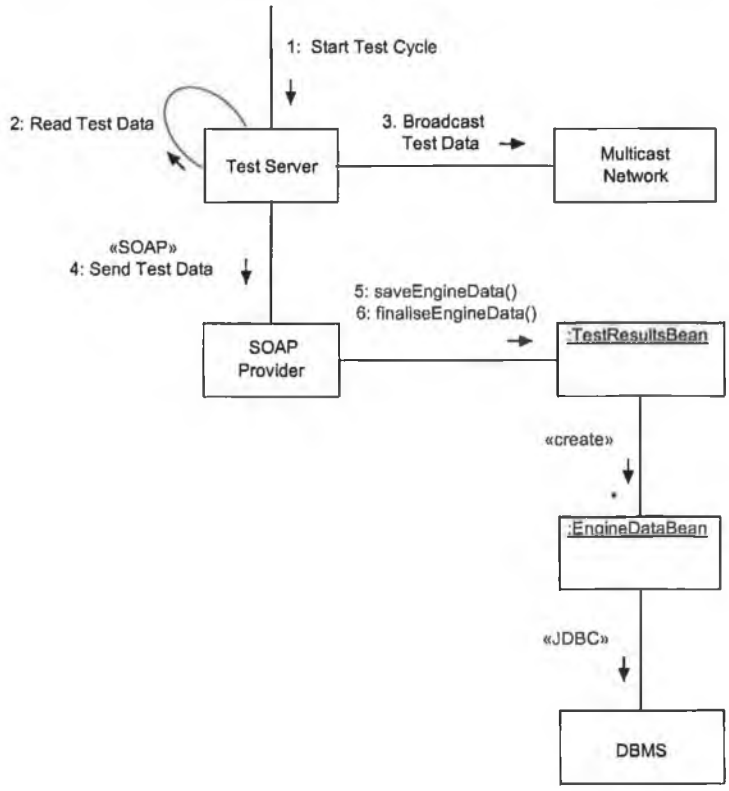

Figure 5.25: Communication Diagram for Callback System

All operations to the SOAP provider require a base 64 encrypted username and password, which is integrated into the application's authentication system.

5.5.8 EJBs for Post Experiment Analysis

The *TestResultsBean* is re-used by a servlet client in order to plot and save test results for completed experiments. Llambiri *et al.* [79] indicate that EJB Query Language (EJB-QL) may be used for complex finder methods in entity bean home interfaces. The *EngineDataBean* shown in Figure 5.26 is an entity bean that employs EJB-QL to locate a list of unique test identifiers and dates to be shown on the presentation tier.

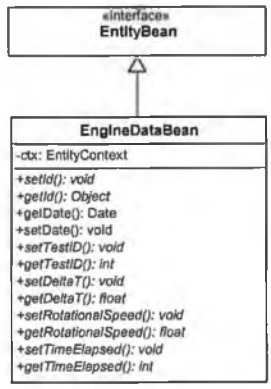

Figure 5.26: EngineDataBean Entity Bean

This entity bean enables the user to select an experiment based on the date on which it was carried out and proceed to plot the result data. Another EJB-QL finder method is used to find all the test data associated with the test date. No two test cycles will have the same test date and ID attributes.

Figure 5.27 shows the sequence of operations for a user plotting a graph from the test data.

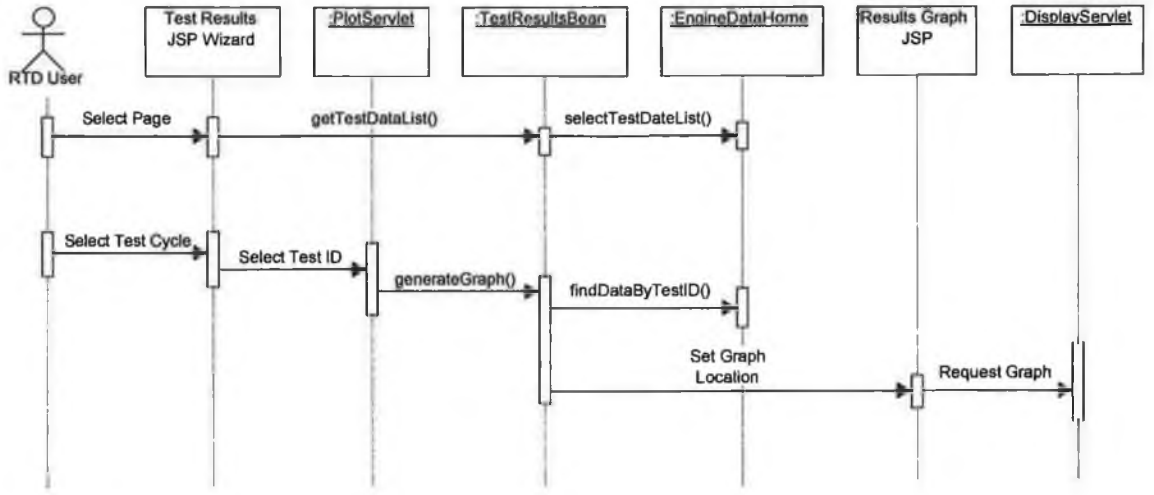

Figure 5.27: Sequence Involved in Plotting Test Data

Figure 5.28 show a dynamic web plot of Stirling Engine efficiency versus time.

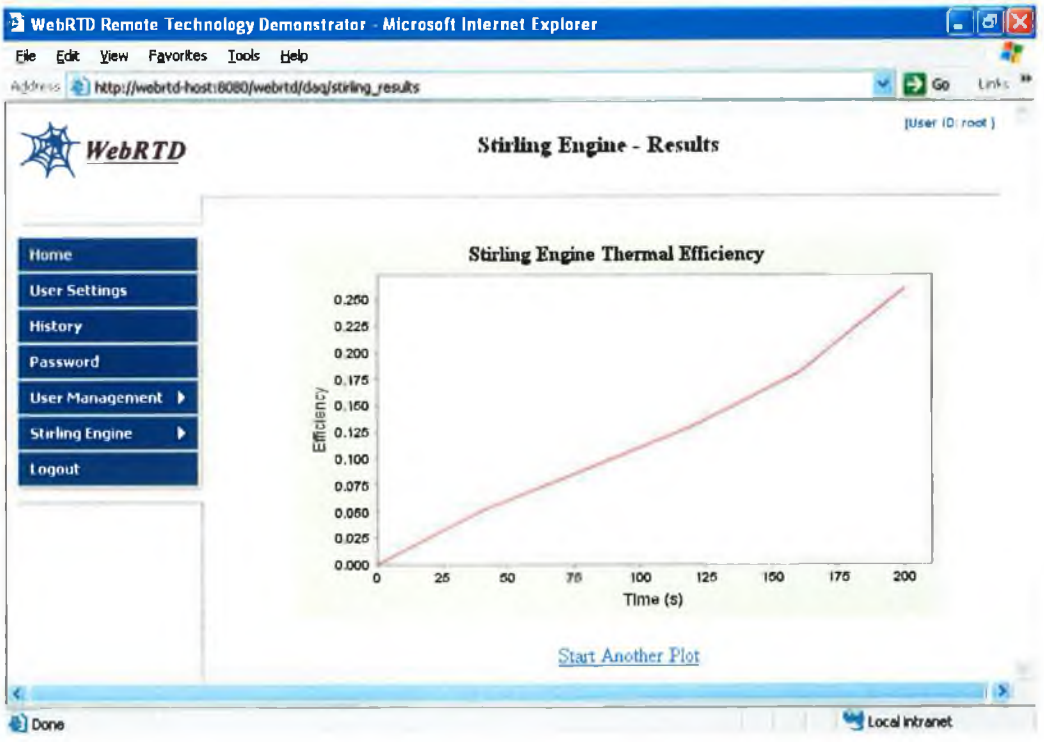

Figure 5.28: Plot of Efficiency vs. Time for Stirling Engine Test Cycle

5.6 Security

Security is a fundamental part of any enterprise application. Every layer of the new system that offers a remote interface has some level of security protection in place. Security issues for the real-time applet and web and enterprise-tiers were addressed.

5.6.1 Real-Time Applet Client Security

A token-based system similar to that used by Ko *et al.* [10] was devised to provide a secure exchange link between the core J2EE-based system and the real-time applet. The test server was configured only to accept token requests from the application server host address. Once verified, the test server would reply by returning a token to the J2EE system via IIOP. Figure 5.29 shows a sequence diagram representing the concept of token passing to the applet.

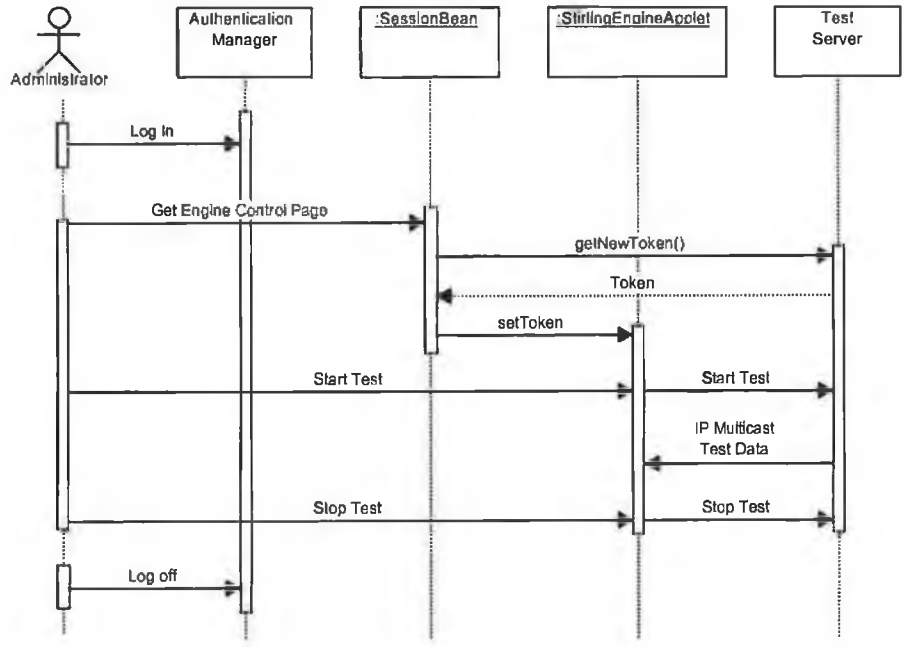

Figure 5.29: Operation of Applet Token-Based System

The test server not only restricts limits access from certain hosts, but logs all remote access to a local log file. It also displays the name of the user who has made the request. Logging is also done from the J2EE side, storing information access request information made from the J2EE system to the test server.

Due to the fact that applets operate from a security "sandbox", the real-time applet must be digitally signed to allow it to connect to a server other than the web server from which it was downloaded. Lindquist [80] highlights the security issues,

such as authenticity and integrity, for interacting with remote entities using Java. Garfinkel and Spafford [81] discuss Public Key Infrastructure (PKI) as an effective means of securely transferring data over the Internet where encryption and decryption keys are generated in pairs. The private key is used to generate the signature and is kept confidential to whoever is doing the signing. The public key is used by the receiver to verify authenticity of the message. The signer should distribute the public key to anyone who will receive signed information.

The issue as to whether the public key corresponds to the sender is resolved with certificates. A certificate represents a chain of trust leading from the sender to the receiver, indicating that the public key belongs to whom you want to believe it belongs. Certificate Authorities (CAs), such as Verisign and Entrust, offer commercial certificates which are used in web browsers.

In the case of the real-time applet, a custom made certificate is used to sign the Java Archive (JAR) file used. As the web browser plug-in will not recognise this as a standard certificate, the user will be requested whether to accept or reject the certificate before proceeding. Figure 5.30 shows the security certificate dialog for the applet.

Using this dialog, the user may view the digital certificate details and choose whether to accept or reject the certificate depending on its credentials. Rejecting this will deny the client accessing the test server. Both administrators and observers will see this dialog, as both will be connecting to a different address for reading data from the IP multicast class D host address.

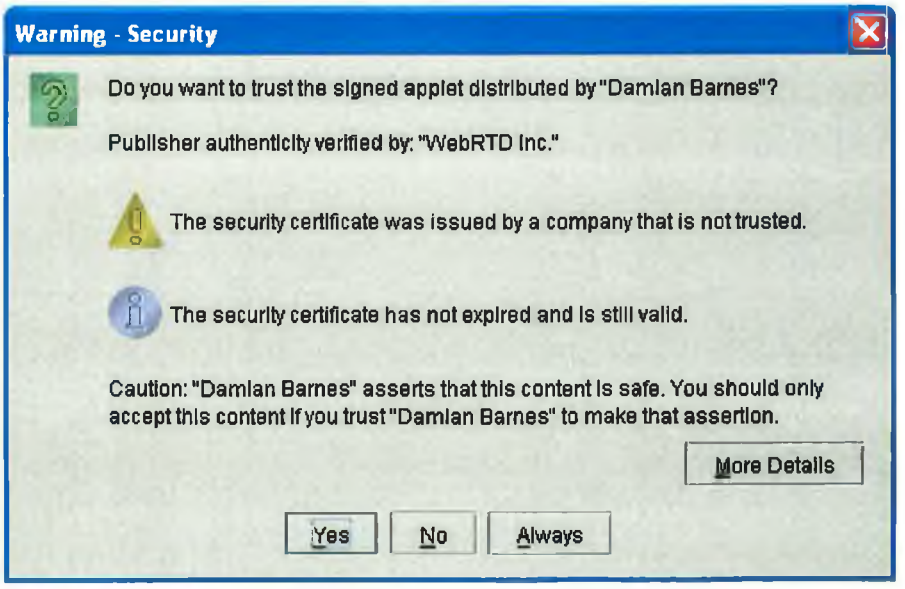

Figure 5.30: Real-Time Applet Security Certificate Dialog

5.6.2 Web and Enterprise-Tier Security

Grubb and Carter [82] acknowledge the fact that a single sign-on facility is a ubiquitous requirement in distributed systems. It is intended that the new system be extensible in order to allow access to multiple demonstrations without the need for authentication before accessing each demonstration in the same session. Ellison *et al.* [83] highlight the need for a simple and secure identity mechanism for web-based applications. In the instance of the remote technology demonstrator, form-based authentication is used to authenticate the user before allowing access to any services. The login page, shown in Figure 5.31, uses a HTTP POST request to submit the username and password for authentication. The web.xml XML deployment descriptor is configured to manage the POST request and permit authenticated users to access the application's web pages. The functionality for handling the actual authentication resides within the Tomcat web server.

Figure 5.31: Login Page for WebRTD©

Tomcat's form-based authentication mechanism is integrated into the JBoss to allow it to be seamlessly integrated into the application server's Java Authentication and Authorisation Service (JAAS). This makes it possible to restrict access to the application and enforce fine-grained access to the enterprise bean operations. Every *RTDUserBean* will be assigned at least one security role, which is used by JAAS to

assign a role identifier, before the user may attempt to invoke enterprise bean operations. Fine-grained access is achieved by assigning method permissions to the security roles themselves in the bean's ejb-jar.xml XML deployment descriptor. This ensures that no security functionality is necessary in the bean code and allows the developer to concentrate on the business logic.

A helper entity bean is added to provide security role information for the application's users. JBoss includes a Service Provider Interface (SPI) security class used to read a user's role list from a table using JDBC. In order to continue using a container-managed approach, an entity bean, *UserRoleBean*, was associated with every user using CMR.

5.7 System Deployment

The enterprise code components implement a significant portion of the application logic but these alone do not represent available services or operational behaviour. XML deployment descriptors are required to indicate to the application server how the various enterprise beans, web-components and other distributed components interact with clients as well as with each other. There are numerous publications such as Roman [84], which describe the syntax of these deployment descriptors. A brief overview is given of each of the descriptor files used in the remote technology demonstrator application.

ejb-jar.xml

The ejb-jar.xml deployment descriptor file was introduced in EJB 1.1 when indeed all XML deployment descriptors were introduced. The functionality for this file is defined in the Enterprise Javabeans specification and includes key enterprise definitions such as bean declaration types (i.e. session, entity or message-driven), entity bean persistence types (CMP or BMP), entity bean relationships and security roles. The ejb-jar.xml descriptor file for this application is included in Appendix E.

jboss.xml

Due to the fact that J2EE is a standard rather than an implementation, there are a number of areas where application server implementers must provide their own mappings. The jboss.xml deployment descriptor is used mainly for mapping

enterprise beans and other resources to JNDI names. These will allow clients to locate these objects after the application has been deployed.

jbosscmp-jdbc.xml

This deployment descriptor allows the developer to define data types for CMP entity bean fields to be used in the database. There are default data type mappings already on the application server, but these may be overridden for performance reasons. The jbosscmp-jdbc.xml file also contains the relationship field details for CMR functionality.

web-service.xml

The web-service.xml deployment descriptor defined the available Web Service interfaces as well as their permitted operations. The Web Service in the remote technology demonstrator allows post experiment data to be sent from the test server to the application server so that it may be stored in the database for later analysis.

jboss-web.xml

The jboss-web.xml may be used to define components for the integrated JBoss-Catalina applications. For this application, this descriptor file was used to defined the JAAS security domain with a view of having seamless integration between the application's form-based authentication and the enterprise bean layer.

web.xml

The web.xml file is used to define available Java servlets as well as their permitted operations such as HTTP GET and HTTP POST. This file was also used for defining the application's form-based authentication. This descriptor covers web applications which may include HTML pages, servlets and/or JSPs.

application.xml

The application.xml descriptor file is used to define the complete set of available enterprise services consisting of EJBs, web applications and Web Services.

JBoss operates what is known as a 'hot deploy' feature where enterprise applications may by updated 'on the fly'. This allows code or configuration updates and existing database information can be preserved also if necessary. On deployment, the

application's XML deployment descriptors are read and the enterprise interfaces (home, remote and local) are interrogated using Java reflection API. This enables the EJB container to create bean instances as well as invoke methods calls on behalf of the client. The real-time and J2EE elements of the remote technology demonstrator are deployed as shown in Figure 5.32.

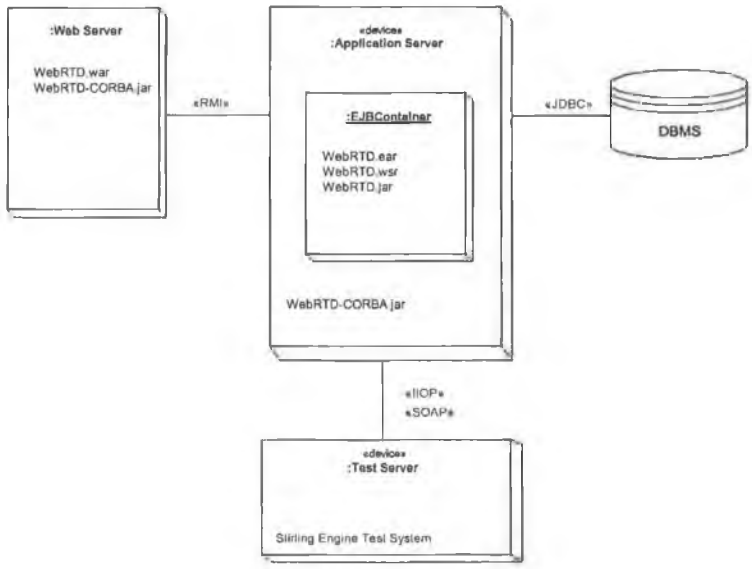

Figure 5.32: Deployment Diagram for Main Components of Remote Technology Demonstrator

5.8 Summary of the System Implementation

Prior to the implementation, it was necessary to select the software which would be employed in the new system. A high proportion of this software was open source and the application server used, JBoss, was no different. A concise description of the hardware architecture is presented for the Stirling Engine demonstration. A test server application is responsible for allowing users to control and monitor the demonstration. Safety and security issues are addressed for the test server itself. On the client side, a soft real-time Java applet is used to administer and observe demonstrations. The applet includes tabbed panels for access control, Stirling Engine pre-heat functionality, thermal efficiency experimentation and PPS control. A web-cam facility is embedded in the applet to provide visual feedback over HTTP.

The real-time applet is intended to be used sequentially from left to right during the Stirling Engine demonstration. Access control is first established over IIOP using the first tabbed panel. The Engine Pre-Heat panel is used to achieve the required

temperature difference in the Stirling Engine before the Thermal Efficiency panel may be used to run a test cycle to determine the thermal efficiency. Test data is relayed back to all clients via IP multicast. On completion of the test cycle, the collected test data is sent via SOAP to the application server so that it may be written to the database for later analysis. Email summaries are dispatched to users who wish to be informed of the test cycle results.

The core J2EE-based system employs entity beans which used the CMP model for data persistence. CMR is used to normalise persistent data enabling one-to-many relationships between system user entity beans and their associated history records and security roles. The Session Façade pattern is used to provide indirect access to entity bean data, enabling local rather than remote interfaces to be utilised for all entity beans, thereby enhancing overall performance. Another performance improving measure is the use of value objects in bulk accessor methods of EJBs, e.g. creating a bulk accessor method getting and setting user details.

The service locator pattern is used by the web-tier to locate components in the EJB layer using JNDI. The Front Controller and Dispatcher View patterns are combined to manage navigation through the web user interface. This allows business logic to be separated from the presentation logic. The Servlet Intercepting Filter Pattern is used to mange the content based on the user type.

A simple web user interface design with low-resolution graphics is used to provide easy navigation and fast loading over the web. A JavaScript menu was positioned on every page ensuring that the user always has access to the key features provided.

Security issues for the real-time applet and web and enterprise-tiers were addressed. A token-based system is used to request control of the Stirling Engine test server demonstration. The initial request is made through a session bean and is authenticated by the test server application. The returned token is passed directly from the real-time applet to the test server when requesting control. The applet itself is digitally signed allowing the user to view the security credentials of the applet.

A single sign-on security mechanism is achieved using form-based authentication to provide access to one or more remote demonstrations. This is seamlessly integrated into the application server's JAAS mechanism, enabling fine-grained access to be enforced for enterprise bean operations.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The principal objective of this research aimed to show how e-commerce principles could be applied to virtual laboratories and to explore possible benefits. At least, three areas were identified as being of critical interest in the new system, namely transaction management, extensibility and security.

The J2EE-based system of the remote technology demonstrator utilises the Java Transaction API (JTA), which is not used directly by the application developer, but is employed by entity beans in the EJB container. In this instance, Container Managed Persistence (CMP) was used, greatly simplifying data persistence tasks by allowing the application server to manage all database transactions using the Java Transaction API (JTA). This approach was extended to the data normalisation of entity beans by using Container Managed Relationships (CMRs), removing the need for using raw Structured Query Language (SQL) in the entity bean code. Even when SQL was required in a complex finder method in the *TestResultsBean*, it was possible to use Enterprise Javabeans Query Language (EJB-QL). EJB-QL is defined in the ejb-jar.xml deployment descriptor (see Appendix E) and may be modified without requiring any source code to be changed.

The J2EE-based system is extensible in so far as new components may be added to the application. The soft real-time element of the system enables varying numbers of observers who may participate in the active demonstration. This was realised by adopting the IP multicast approach used by Ko *et al.* [26].

Security for this application was achieved using J2EE and non-J2EE solutions, all of which were Java-based and commonly found in e-commerce applications. Digitally signing the real-time applet provided a good means of associating security credentials with the bytecode and also allowed an IIOP network connection outside the applet sandbox. A single sign-on facility was achieved using form-based authentication. The authentication facility and the application's JAAS functionality, were integrated using only XML deployment descriptors, allowing security credentials to be passed directly from the web-tier to the EJB-tier, without the need to write any code.

The secondary objective of this project was to minimise the overall project cost by employing open source software. The open source application server, JBoss, was used to deliver the required J2EE functionality, along with a number of other open source software products such as Apache Tomcat and MySQL. The result was a system that allowed the remote technology demonstrator to perform without any obvious bottlenecks while enforcing the required security restrictions. Initial test trials showed that the overall system was stable and performed well during active demonstrations.

Another objective of this research was to develop an intuitive user interface which would promote e-learning by effectively communicating to the learner the principles and any background information related to the remote demonstration. To this end, a simple web user interface was used and brief, but informative, online help was added to outline the thermodynamic concepts behind the operation of a Stirling Engine.

Other findings from this research are presented here in brief:

- From the literary review conducted in Chapter 2, it is evident that there is currently no template for the design of virtual laboratories. UML provides an excellent means of communicating system requirements and modelling the proposed sequence of operations for new software systems.
- The pattern-based design and development approach helped solve many of the problems associated with e-commerce applications. One of the most useful of these was the combined use of the Front Controller and Dispatcher View patterns to manage web navigation and to separate business logic from presentation logic. Using the Session Façade pattern allowed loose coupling between session and entity bean layers and allowed local interfaces to be used for all entity beans, thereby enhancing performance by avoiding the communication layers for local communications.
- The token-based system, adopted from Ko *et al.* [10], provided an effective means of providing authentication between the J2EE-based system, test server and real-time applet.
- The CORBA implementation used in the test server, OmniORB, did not support some of the newer features of the CORBA standard required for J2EE callbacks.

SOAP was used to solve the problem of transferring data between the C++ test server and J2EE-based system was solved using SOAP.

6.2 Future Work

This project successfully integrated a real-time system with a J2EE-based one to create a new type of virtual laboratory. Specialists in these two areas will recognise that there is possible scope for improvement.

In order to improve security throughout the application SSL could be used where sensitive data is transmitted across the network. This could be implemented on the web page where the user logs in and when transmitting control commands to the test server.

It was deemed adequate to provide the user with a button to request update for visual feedback from the web-cam. For future experiments, a streaming feedback may be required. This would involve a new streaming video server being used as well as significant modifications being made to the real-time applet.

Patterns such as the session façade and value object were used to good effect in this project. Another pattern which could be used in future, is the Data Access Object (DAO) pattern. This pattern transparently maps bean operations to the underlying data store, thereby allowing the data store to be changed without any change to the application itself. DAO instances can not only map to relational databases, but also to object databases, file systems and XML documents. This pattern will be supported in JBoss in future releases.

An even more substantial upcoming feature in JBoss is support for Aspect-Oriented Programming (AOP). AOP [85] offers extended mechanisms to Object-Oriented Programming (OOP) for decomposing problem domains into cleanly encapsulated entities. The result is code that is more reusable than that produced by OOP.

Possibilities still exist for expanding the capabilities of the Stirling Engine demonstration. From an educational perspective, it might be possible to integrate both a pressure and volume sensor onto the engine which would enable PV diagrams to be plotted directly. This would show how the pressure and volume changes in relation to each other for the thermodynamic cycle employed.

In terms of future applications for the WebRTD© virtual laboratory system, the possibilities are limitless. Other areas such as electronic engineering and process control, which have already been influenced by this approach, could be enhanced further with new and innovative applications.

References

[1] Martin, W., and Haque, Mohammed E., "*Distance Learning In Engineering And Construction*", International Conference IT in Construction in Africa, Mpumalanga, South Africa, 30 May-June 1, 2001.

[2] Knight, C.D., DeWeerth, S.P., "*A Distance Learning Laboratory for Engineering Education*", Proceedings of the 1997 ASEE Annual Conference, Milwaukee, WI., June 15-18, 1997.

[3] World Wide Web Consortium (W3C) web site, http://www.w3.org/

[4] Crisp, G., "*Using Java Applets To Help Make Online Assessment Interactive*", Proceedings of Ascilite 2002, Auckland, New Zealand, December 8-11, 2002.

[5] Reisman, S., Carr, W.A., "*Perspectives on Multimedia Systems in Education*", IBM Systems Journal, 30, 3, 280-295, (1991).

[6] Tuttas, J. and Wagner, B., "*Distributed Online Laboratories*", International Conference on Engineering Education August 6-10, 2001 Oslo, Norway.

[7] Guggisberg, M., Fornaro, P., Gyalog, T., and Burkhart, H., "*An Interdisciplinary Virtual Laboratory on Nanoscience*", 29th Speedup Workshop, Bern, Switzerland, March 22-23, 2001.

[8] Virtual Physics Laboratory, http://www.phy.ntnu.edu.tw/java/index.html.

[9] Hsu, S., Alhalabi, B., and Ilyas, M., "*A Java-based Remote Laboratory for Distance Learning*", International Conference on Engineering Education, Taipei, Taiwan, August 14-16, 2001.

[10] Ko, C. C., Chen, B. M., Hu, S., Ramakrishnan, V., Cheng, C. D., Zhuang, Y., and Chen, J., "*A Web-Based Virtual Laboratory on a Frequency Modulation Experiment*",
IEEE Transactions On Systems, Man, And Cybernetics, Part C: Applications And Reviews, Vol. 31, No. 3, August 2001.

[11] Amaratunga, K. and Sudarshan, R., "*A Virtual Laboratory for Real-Time Monitoring of Civil Engineering Infrastructure*", Proceedings of the International Conference on Engineering Education, UMIST, Manchester, UK, August 2002.

[12] Röhrig, C. and Jochheim, A., "*The Virtual Lab forcontrolling real experiments via Internet*", Proceedings of 1999 IEEE International Symposium on computer-aided control system design, Hawaii, USA, August 8-12, 1999.

[13] Macías, M. E., Cázares, V. M. and Ramos, E. E., "*A Virtual Laboratory For Introductory Electrical Engineering Courses To Increase The Student Performance*", Proceedings from Frontiers in Education Conference, Reno, Nevada, October 2001.

[14] Baher, J., "*How Articulate Virtual Labs can Help in Thermodynamics Education: A Multiple Case Study*", Proceedings from Frontiers in Education Conference, Tempe, Arizona, November 4-7, 1998.

[15] Aktan, B., Bohus, C., Crowl, L., and Shor, M. H., "*Distance learning applied to control engineering laboratories*", IEEE Trans. Educ., vol. 39, pp. 320-326, August 1996.

[16] GNU General Public License, http://www.gnu.org/copyleft/gpl.html.

[17] Free Software Foundation, http://www.fsf.org.

[18] Apache open source web server, http://www.apache.org.

[19] SourceForge open source software development website, http:// sourceforge.net.

[20] Mockus, A., Fielding, R. T. and Herbsleb, J., "*A case study of open source software development: the Apache server*", Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, June 5-7, 2000.

[21] Sun Microsystems' blueprint design patterns, http://java.sun.com/blueprints/patterns/.

[22] Chen, S. H., Chen, R., Ramakrishnan, V., Hu, S. Y., Zhuang, Y., Ko, C. C. and Chen, B. M., "*Development of remote laboratory experimentation through Internet*", Proceedings of the IEEE Hong Kong Symposium on Robotics and Control, Hong Kong, July 12-16, 1999.

[23] Comer, D. E., *Internetworking with TCP/IP Volume 1 - Principles, Protocols, and Acrhitecture*, 3rd Ed., pp. 291-295, Prentice Hall, 1995.

[24] Wang, G., Robinson, R., "*An Architecture for Web-Enabled Engineering Applications based on Lightweight High Performance CORBA*", 6th International Enterprise Distributed Object Computing Conference (EDOC'02), Lausanne, Switzerland, September 17-20, 2002

[25] Orfali, B. and Harkey, D., Client/Server Programming with Java and CORBA, pp. 185-230, Wiley, 1998.

[26] Ko, C.C., Chen, Ben M., Chan, K.P., Cheng, C.D., Zeng, G.W. and Zhang, J., "*A Webcast Virtual Laboratory on a Frequency Modulation Experiment*", IEEE Conference on Decision and Control, Orlando, FL, December 4-7, 2001.

[27] EBay online auction web site, http://www.ebay.com.

[28] Soldar, G., Spencer, Brian, and Smith, Dan, "*Impact of New Technologies on the Expansion of Electronic Commerce*", Workshop Proceedings of the 1st Panhellenic Conference with International Participation in Human Computer Interaction (PC-HCI 2001), Patras, Greece, December 7-9, 2001.

[29] Pour, G., "*Enterprise JavaBeans, JavaBeans & XML Expanding the Possibilities for Web-Based Enterprise Application Development*", 31st International Conference on Technology of Object-Oriented Language and Systems, Nanjing, China, September 22-25, 1999.

[30] Pour, G., "*Integrating Component-Based & Reuse-Driven Software Engineering Business Into Software & Information Engineering Curriculum*", Proceedings from Frontiers in Education Conference, Kansas City, Missouri, October 18-21, 2000.

[31] Java Pet Store Demo application, http://java.sun.com/developer/releases/petstore/.

[32] Rice University Bidding System (RUBiS), http://demos.objectweb.org/rubis/.

[33] Candea, G., Delgado, M., Chen, M., Fox, A., "*Automatic Failure-Path Inference: A Generic Introspection Technique for Internet Applications*", Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP), San Jose, CA, June 23-24, 2003.

[34] JBoss Application Server, http://www.jboss.org.

[35] Chambers, M., Sindel, M., Thompson, J. and Tien, D., "*Using Enterprise JavaBeans to Provide Distributed, Concurrent Access to a Centralized Database*", Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004), Harbin, China, January 9-11, 2004.

[36] Borges, A.P., Lisboa, M.O.P., Fernandez, F.J.R., "*A Graphical Interface to Link Virtual Instruments through a Web Browser*", 13th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'00), Gramado, Brazil, October 17-20, 2000.

[37] Pasquarette, J., "*Activating the Internet for Virtual Instrumentation*", Available online: http://www.evaluationengineering.com/pctest/articles/e702pcni.htm.

[38] Allamaraju *et al.*, *Professional Java Server Programming J2EE Edition*, p. 341, Wrox Press, 2000.

[39] Schattkowsky, T. and Müller, W., "*Distributed Engineering Environment for the Design of Electronic Systems*", Proceedings of the IEEE Workshop on Design and Diagnostics of Electronic Circuits (DDEC), Poznan, Poland, April 15-16, 2003.

[40] Pour, G., Guo, Y., "*Web Service-Oriented Architecture for Supply Chain Management*", Proceedings of the International Conference on Internet Computing, Las Vegas, Nevada, USA, June 23-26, 2003.

[41] Amazon online bookstore, http://www.amazon.com.

[42] Java 2 Platform, Enterprise Edition (J2EE), http://java.sun.com/j2ee/.

[43] Java Management Extensions (JMX), http://java.sun.com/products/JavaManagement/.

[44] Enterprise Javabeans Specification, http://java.sun.com/products/ejb/.

[45] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, pp. 107-116, 185-193, Addison-Wesley Publishing Company, 1995.

[46] Monson-Haefel, R., Enterprise JavaBeans, pp. 153-171, 3rd Ed., O'Reilly, 2001.

[47] Extensible Markup Language (XML) Specification, http://www.w3.org/TR/2000/REC-xml-20001006.

[48] Object Management Group (OMG), http://www.omg.org.

[49] CORBA Specification, http://www.omg.org/technology/documents/formal/corba_iiop.htm.

[50] Lai, C., Gong, C. L., Koved, L., Nadalin, A., Schemers, R., "*User Authentication and Authorization in the Java Platform*", Proceedings of the 15th Annual Computer Security Applications Conference, Phoenix, AZ, December 6-9, 1999.

[51] Booch, G., Jacobson, I., Rumbaugh, J., *The Unified Modeling Language User Guide*, Addison-Wesley Professional, September 30, 1998.

[52] Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 2nd Ed., p. 1, Addison-Wesley, September 19, 2003.

[53] Louis, S. J., McDonnell, J., Hohmeyer, D., Heinselman, L., Walker, A., "*A Case Study in Object Oriented Modeling, Architecting, and Designing an Enterprise Monitoring Application*", Proceedings of the International Conference on Software Engineering Research and Practice (SERP'03), Las Vegas, Nevada, June 23-26, 2003.

[54] Mos, A. and Murphy, J., "*New Methods for Performance Monitoring of J2EE Application Servers*", Proceedings of IEEE 8th International Conference on Telecommunications (ICT-2001), Vol I, p. 423-427, Bucharest, Romania, June 2001.

[55] Hammouda, I. and Koskimies, K., "*Generating a Pattern-Based Application Development Environment for Enterprise JavaBeans*", Proceedings of the 26th Annual International Computer Software and Applications Conference, Oxford, England, August 26-29, 2002.

[56] Session Façade pattern, http://java.sun.com/blueprints/corej2eepatterns/Patterns/SessionFacade.html.

[57] Service Locator Blueprint Pattern, http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceLocator.html.

[58] Front Controller Pattern, http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html.

[59] Dispatcher View Pattern, http://java.sun.com/blueprints/corej2eepatterns/Patterns/DispatcherView.html.

[60] Webmacro Java Template Language, http://www.webmacro.org.

[61] Intercepting Filter Pattern, http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html.

[62] Apache Tomcat Web Server, http://jakarta.apache.org/tomcat/.

[63] Apache Axis SOAP Implementation, http://ws.apache.org/axis/.

[64] MySQL Database Management System, http://www.mysql.org.

[65] JFreeChart Java Charting/Graphing Library, http://www.jfree.org/jfreechart.

[66] Omniorb open source C++ CORBA Implementation, http://omniorb.sourceforge.net/.

[67] EasySoap++ C++ SOAP library , http://easysoap.sourceforge.net.

[68] Python Language, http://www.python.org.

[69] Expat XML parser, http://www.jclark.com/xml/expat.html.

[70] Srinivasan, A. and Anderson, J. H., "*Efficient Scheduling of Soft Real-time Applications on Multiprocessors*", Journal Of Embedded Computing, Vol. 1, No. 3, June 2004.

[71] Puschner, P., Bernat, G. and Wellings, A., "*Making Java Hard Real-Time*", Annals of the Marie-Curie Fellowship Association, Vol. 2, 2002.

[72] Vodafone Mobile Phone Subscriber Portal site, http://www.vodafone.ie.

[73] Cecchet, E., Marguerite, J., Zwaenepoel, W., "*Performance and Scalability of EJB Applications*", 17th ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (Oopsla 2002), Seattle, WA, USA, November 4-8, 2002.

[74] Gitzel, R., Korthaus, A., Mazloumi, N., "*Handling Huge Data Sets in J2EE/EJB 2.1 with a Page-By-Page Iterator Variant for CMP*", Proceedings of the International Conference on Software Engineering Research and Practice (SERP'03), Las Vegas, Nevada, USA, June 23-26, 2003.

[75] Uniform Resource Identifier Definition, http://www.ietf.org/rfc/rfc2396.txt.

[76] Nielsen, J., *Designing Web Usability*, New Riders Publishing, 1999.

[77] Hübscher, R., Pittarese, T., Lanford, P., "*Navigation in e-Business Sites*", Architectural Issues of Web-Enabled Electronic Business, Idea Group Publishing, 2002,

[78] Bevan, N., "*Usability Issues in Web Site Design*", Proceedings of Usability Professionals' Association (UPA) '98, Washington, DC, April 18-23, 1998.

[79] Llambiri, D., Totok, A., and Karamcheti, V., "*Efficiently Distributing Component-based Applications Across Wide-Area Environments*", 23rd International Conference on Distributed Computing Systems, Providence, Rhode Island, USA, May 19-22, 2003.

[80] Lindquist, T., "*Security Considerations for Distributed Web-Based E-Commerce Applications in Java*", Proceedings of the 35th Hawaii International Conference on System Sciences, Hawaii, USA, January 7-10, 2002.

[81] Garfinkel, S. and Spafford, G., *Web Security & Commerce*, pp. 207-208, O'Reilly, 1997.

[82] Grubb, M. and Carter, R., "*Single Sign-On and the System Administrator*", Proceedings of the Twelfth Systems Administration Conference (LISA '98), Boston, MA, December 6-11, 1998.

[83] Ellison, G., Hodges, J., Landau, S., "*Security and Privacy Concerns of Internet Single Sign-On*", Available online: http://research.sun.com/Liberty/SaPCISSO/sapcil.pdf.

[84] Roman, E., *Mastering Enterprise JavaBeans, 2nd Ed.*, pp. 546-568, Wiley Computer Publishing, 2002.

[85] Aspect-Oriented Programming Homepage, http://www2.parc.com/csl/projects/aop/.

[86] American Stirling Company, http://www.stirlingengine.com

Appendix A

Glossary

3-Tier Architecture

This is a common term used for distributed object systems. In a three-tier architecture the presentation logic resides on the client (first tier), the business logic resides on the middle tier (second tier), and other resources, such as the database, reside on the backend (third tier).

ActiveX

A loosely defined set of technologies developed by Microsoft. ActiveX is an outgrowth of two other Microsoft technologies called OLE (Object Linking and Embedding) and COM (Component Object Model). OLE enables you to create objects with one application and then link or embed them in a second application. COM is a software architecture developed by Microsoft to build component-based applications.

Application Server

An application server is a server program in a distributed network that provides the business logic for an application program. The application server is frequently viewed as part of a three-tier application, consisting of a graphical user interface (GUI) server, an application (business logic) server, and a database and transaction server.

Aspect-Oriented Programming (AOP)

Aspect-Oriented Programming (AOP) complements OO programming by allowing the developer to dynamically modify the static OO model to create a system that can grow to meet new requirements. AOP allows new functionality to be added to existing code to be without having to modify the original code.

Atomicity, Consistency, Isolation, Durability (ACID)

A term used to describe transactions with ACID properties.

Atomic – Transactions are atomic (all or none)

Consistency – A consistent state of the database may be expected at all times.

Isolation – Transactions are isolated from another; race conditions ensure that multiple transaction instances attempts do not collide.
Durable – Once a transaction commits, it's updates survive even, if the system goes down.

Axis

Apache Axis is an Open Source SOAP implementation and it available for Java and C++.

Bean-Managed Persistence (BMP)

Bean-managed persistence (BMP) occurs when the entity object manages its own persistence. The enterprise bean developer must implement persistence operations (e.g. JDBC) directly in the enterprise bean class methods.

Certificate Authority (CA)

An individual wishing to send an encrypted message applies for a digital certificate from a Certificate Authority (CA). The CA issues an encrypted digital certificate containing the applicant's public key and a variety of other identification information. The CA makes its own public key readily available through print publicity or perhaps on the Internet.

Commercial Off-The-Shelf (COTS)

Adjective that describes software or hardware products that are ready-made and available for sale to the general public. For example, Microsoft Office is a COTS product that is a packaged software solution for businesses. COTS products are designed to be implemented easily into existing systems without the need for customization.

Common Gateway Interface (CGI)

A specification for transferring information between a World Wide Web server and a CGI program . A CGI program is any program designed to accept and return data that

conforms to the CGI specification. The program could be written in any programming language, including C, Perl, Java, or Visual Basic.

Common Object Request Broker Architecture (CORBA)

An architecture that enables pieces of programs, called objects, to communicate with one another regardless of what programming language they were written in or what operating system they're running on. CORBA was developed by an industry consortium known as the Object Management Group (OMG).

Component-Based Enterprise Software Engineering (CBESE)

Component-Based Enterprise Software Engineering (CBESE) is concerned with the development of software intensive systems from reusable parts (components), the development of such reusable parts, and with the maintenance and improvement of systems by means of component replacement and customisation.

Container-Managed Persistence (CMP)

Container-managed persistence (CMP) occurs when the entity object delegates persistence services. With CMP, the EJB container transparently and implicitly manages the persistent state. The enterprise bean developer does not need to code any database access functions within the enterprise bean class methods.

Container-Managed Relationships (CMR)

Relationship model introduced in EJB 2.0 supporting one-to-one, one-to-many and many-to-many relationships between entity beans.

Computer Supported Collaborative Work (CSCW)

Computer-assisted coordinated activity carried out by a group of collaborating individuals.

Data Access Object (DAO)

Pattern designed to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and store data.

Data Acquisition (DAQ)

Term used to define the process of collecting information, usually in an automated fashion, on a broad variety on variables. Data acquisition is heavily used in manufacturing processes that have a multitude of variables to be measured. Data acquisition uses a combination of hardware, software and specialized instruments with a variety of specialized term like PCI, PCMCIA, USB, RS-232, FPGA, ADC, SCSI, GPIB, etc.

Database Management System (DBMS)

A collection of programs that enables you to store, modify, and extract information from a database.

Deployment Descriptor

An XML file provided with each module and application that describes how they should be deployed.

Design Pattern

A description of an object-oriented design technique which names, abstracts and identifies aspects of a design structure that are useful for creating an object-oriented design. The design pattern identifies classes and instances, their roles, collaborations and responsibilities. Each design pattern focuses on a particular object-oriented design problem or issue. It describes when it applies, whether it can be applied in the presence of other design constraints, and the consequences and trade-offs of its use.

Digital Certificate

An attachment to an electronic message used for security purposes. The most common use of a digital certificate is to verify that a user sending a message is who he or she claims to be, and to provide the receiver with the means to encode a reply.

The recipient of an encrypted message uses a Certificate Authority's (CA) public key to decode the digital certificate attached to the message, verifies it as issued by the CA and then obtains the sender's public key and identification information held within the certificate. With this information, the recipient can send an encrypted reply. The most widely used standard for digital certificates is X.509.

Dispatcher View

Pattern used to manage direction of enterprise components. This applies particularly to abstracting user navigation functionality for Java servlets and JSPs.

Domain Name Service (DNS)

An Internet service that translates domain names into IP addresses. Because domain names are alphabetic, they're easier to remember. The Internet however, is really based on IP addresses . Every time you use a domain name, therefore, a DNS service must translate the name into the corresponding IP address. For example, the domain name www.example.com might translate to 198.105.232.4.

E-Commerce

Business that is conducted over the Internet using any of the applications that rely on the Internet, such as e-mail, instant messaging, shopping carts, Web services, UDDI, FTP, and EDI , among others. Electronic commerce can be between two businesses transmitting funds, goods, services and/or data or between a business and a customer.

EasySoap++

Open Source C++ SOAP implementation.

Enterprise JavaBeans

The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification. Enterprise JavaBeans server-side components come in three fundamentally different types: *entity, session*, and *message-driven beans*

Enterprise JavaBeans (EJB) Container

A container that implements the EJB component contract of the J2EE architecture. This contract specifies a runtime environment for enterprise beans that includes security, concurrency, life-cycle management, transactions, deployment, naming, and other services. An EJB container is provided by an EJB or J2EE server.

Enterprise JavaBeans Query Language (EJB-QL)

Defines the queries for the finder and select methods of an entity bean having container-managed persistence. A subset of SQL92, EJB QL has extensions that allow navigation over the relationships defined in an entity bean's abstract schema.

Entity Bean

An enterprise bean that represents persistent data maintained in a database. An entity bean can manage its own persistence or can delegate this function to its container. An entity bean is identified by a primary key. If the container in which an entity bean is hosted crashes, the entity bean, its primary key, and any remote references survive the crash.

Expat

Open Source XML parser written in C.

Extended Hypertext Markup Language (XHTML)

A hybrid between HTML and XML specifically designed for numerous networked devices from anything from Personal Digital Assistants (PDAs) to toasters. XHTML is a stricter, tidier version of HTML. It is a markup language written in XML and therefore, is an XML application.

Extensible Mark-Up Language (XML)

Specification developed by the W3C. XML is a subset of SGML, designed especially for Web documents.

File Transfer Protocol (FTP)

Protocol for exchanging files over the Internet.

Frequency Modulation

A form of modulation in which the frequency of the modulated carrier wave is varied in proportion to the amplitude of the modulating wave. In this case the phase of the carrier varies with the integral of the modulating wave.

Front Controller

Pattern used to enforce separation of business logic from presentation logic in the same components.

General Purpose Interface Bus (GPIB)

Interface system designed to create a reliable bus system especially designed for connecting computers and instruments.

GNU

Self-referentially, short for "GNU's not UNIX", a UNIX -compatible software system developed by the Free Software Foundation (FSF). The philosophy behind GNU is to produce software that is non-proprietary. Anyone can download, modify and redistribute GNU software. The only restriction is that they cannot limit further redistribution. The GNU project was started in 1983 by Richard Stallman at the Massachusetts Institute of Technology.

General Public License (GPL)

The license that accompanies some open source software that details how the software and its accompany source code can be freely copied, distributed and modified. The most widespread use of GPL is in reference to the GNU GPL, which is commonly abbreviated simply as GPL when it is understood that the term refers to the GNU GPL. One of the basic tenets of the GPL is that anyone who acquires the material must make it available to anyone else under the same licensing agreement.

Hypertext Markup Language (HTML)

The authoring language used to create documents on the World Wide Web. HTML is similar to SGML, although it is not a strict subset.

Hypertext Transfer Protocol (HTTP)

The underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.

Intercepting Filter

Pattern used to pre-process and post-process client web requests and responses.

Interface Definition Language (IDL)

Language used to define a protocol between client and server processes so that they can communicate with each other at a level higher than simple byte strings in a heterogeneous networking environment.

Internet Inter-ORB Protocol (IIOP)

A protocol developed by the Object Management Group (OMG) to implement CORBA solutions over the World Wide Web. IIOP enables browsers and servers to exchange integers, arrays, and more complex objects, unlike HTTP, which only supports transmission of text.

Internet Message Access Protocol (IMAP)

A protocol for retrieving e-mail messages.

Java

A high-level programming language developed by Sun Microsystems. Java was originally called OAK, and was designed for handheld devices and set-top boxes. Oak was unsuccessful so in 1995 Sun changed the name to Java and modified the language to take advantage of the burgeoning World Wide Web. Java is an object-oriented language similar to C++, but simplified to eliminate language features such as memory management that cause common programming errors.

Java 2 Enterprise Edition (J2EE)

An environment for developing and deploying enterprise applications. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, Web-based applications.

Java 2 Standard Edition (J2SE)

A complete environment for applications development on desktops and servers which includes a bytecode compiler and interpreter.

Java API For XML Messaging (JAXM)

JAXM enables applications to send and receive document-oriented XML messages using SOAP.

Java API For XML Parsing (JAXP)

JAXP is a Java API that enables applications to parse and transform XML documents independent of a particular XML processing implementation.

Java Applets

A program designed to be executed from within another application. Unlike an application, applets cannot be executed directly from the operating system. A well-designed applet can be invoked from many different applications.

Web browsers, which are often equipped with Java virtual machines, can interpret applets from Web servers. Because applets are small in files size, cross-platform compatible, and highly secure (can't be used to access users' hard drives), they are ideal for small Internet applications accessible from a browser.

Java Authentication And Authorisation Service (JAAS)

The Java Authentication and Authorization Service (JAAS) is a set of APIs that enable services to authenticate and enforce access controls upon users. It implements a Java technology version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization.

Java Message Service (JMS)

JMS is a standard vendor-neutral API that is part of the J2EE platform and can be used to access enterprise messaging systems. Enterprise messaging systems (a.k.a. message-oriented middleware) facilitate the exchange of messages among software applications over a network. Many commercial enterprise-messaging products currently support JMS, including IBM's MQSeries and BEA's WebLogic JMS service.

Java Naming And Directory Interface (JNDI)

The Java Naming and Directory Interface (JNDI) provides an interface for accessing name and directory services such as Lightweight Directory Access Protocol (LDAP) directory services and Domain Name Service (DNS).

Java Server Page (JSP)

JSPs are also used to dynamically generate web content such as HTML, XHTML, and XML in web applications. JSP technology enables easy authoring of web pages, which allows web designers rather than programmers focus on the presentation. Java code may also be embedded in JSPs.

Java Transaction API (JTA)

API used for handling commit and the rollback of transactions as well as ensuring ACID properties of a transaction.

Java Virtual Machine (JVM)

An abstract computing machine, or virtual machine, JVM is a platform-independent execution environment that converts Java bytecode into machine language and executes it. Most programming languages compile source code directly into machine code that is designed to run on a specific microprocessor architecture or operating system, such as Windows or UNIX.

JavaIDL

A technology that provides CORBA interoperability and connectivity capabilities for the J2EE platform. These capabilities enable J2EE applications to invoke operations on remote network services using IIOP.

JavaMail

An API for sending and receiving email.

JBoss

An open source J2EE compatible application server.

JDBC

An API for database-independent connectivity between the J2EE platform and a wide range of data sources.

JFreeChart

An open-source, graphing and charting library.

Lightweight Directory Access Protocol (LDAP)

A set of protocols for accessing information directories.

Message Driven Bean (MDB)

Message-driven beans (MDBs) are stateless, server-side, transaction-aware components for processing asynchronous Java Message Service (JMS) messages.

Microsoft Transaction Server (MTS)

MTS is a software application that mainly consists of a Transaction Processing (TP) monitor and an object request broker. MTS, based on the Component Object Model (COM) which is the middleware component model for Windows, is used for creating scalable, transactional, multi-user and secure enterprise-level server side components.

Middleware

Software that connects two otherwise separate applications. For example, there are a number of middleware products that link a database system to a Web server. This allows users to request data from the database using forms displayed on a Web browser, and it enables the Web server to return dynamic Web pages based on the user's requests and profile. The term middleware is used to describe separate products that serve as the glue between two applications. It is, therefore, distinct from import and export features that may be built into one of the applications.

Multipurpose Internet Mail Extensions (MIME)

A specification for formatting non-ASCII messages so that they can be sent over the Internet. Many e-mail clients support MIME, which enables them to send and receive graphics, audio, and video files via the Internet mail system.

MySQL

MySQL is an open source RDBMS that relies on SQL for processing the data in the database. MySQL provides APIs for the languages C, C++, Eiffel, Java, Perl, PHP and Python.

Object Management Group (OMG)

A consortium with a membership of more than 700 companies. The organization's goal is to provide a common framework for developing applications using object-oriented programming techniques. OMG is responsible for the CORBA specification.

Object-Oriented Programming (OOP)

A type of programming in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions.

OmniOrb

Open source C++ CORBA implementation.

Open Source Software (OSS)

Open Source Software is software in which the source code is available to the general public for use and/or modification from its original design free of charge.

Passivation

The process of transferring an enterprise bean from memory to secondary storage.

Post Office Protocol 3 (POP3)

A protocol used to retrieve e-mail from a mail server.

Programmable Power Supply (PPS)

Power supply unit that may be controlled externally. This is normally achieved using a GPIB or RS-232 hardware interface.

Public Key Infrastructure (PKI)

A system of digital certificates, Certificate Authorities, and other registration authorities that verify and authenticate the validity of each party involved in an Internet transaction.

Python

An interpreted, object-oriented programming language developed by Guido van Rossum. Python is very portable since Python interpreters are available for most operating system platforms. Although Python is copyrighted, the source code is open source, and unlike GNU software, it can be commercially re-sold.

Real-Time

Pertains to a data collecting system that controls an on-going process and delivers its outputs (or controls its inputs) not later than the time when these are needed for effective control. This is also known as *hard* real-time. In contract *soft* real-time may be non-deterministic and may be allowed to miss its deadlines.

Remote Procedural Call (RPC)

A type of protocol that allows a program on one computer to execute a program on a server computer.

Rice University Bidding System (RUBiS)

RUBiS is an auction site prototype modelled after the popular e-commerce web-portal, eBay.

Remote Method Invocation (RMI)

A set of protocols being developed by Sun that enables Java objects to communicate remotely with other Java objects.

RMI/IIOP

Java Remote Method Invocation (RMI) technology run over Internet Inter-Orb Protocol (RMI-IIOP) delivers Common Object Request Broker Architecture (CORBA) distributed computing capabilities to the Java 2 platform. Java RMI over IIOP was developed by Sun and IBM.

RS-232

A standard interface approved by the Electronic Industries Alliance (EIA) for connecting serial devices.

Secure Socket Layer (SSL)

A protocol developed to transmit private documents via the Internet. SSL works by using a private key to encrypt data that's transferred over the SSL connection.

Service Locator

Design pattern used to locate enterprise services while hiding complexity involved.

Service Oriented Architecture (SOA)

An application architecture in which all functions, or services, are defined using a description language and have invokable interfaces that are called to perform business processes. Each interaction is independent of each and every other interaction and the interconnect protocols of the communicating devices (i.e., the infrastructure components that determine the communication system do not affect the interfaces). Because interfaces are platform-independent, a client from any device using any operating system in any language can use the service.

Servlets

Servlets are used to dynamically generate Hypertext Markup Language (HTML), Extended HTML (XHTML), and XML output using Java technology.

Session Bean

Session beans are enterprise beans used to implement business objects that hold client-specific business logic. A session bean typically executes on behalf of a single client and cannot be shared among multiple clients. There are two types of session beans; *stateful* and *stateless*. Stateful session beans maintain state information between method invocations whereas stateless session beans do not.

Session Façade

Design pattern gateway mechanism which manages business objects, and provides a uniform coarse-grained service access layer to clients. The Session Façade reduces

network overhead between the client and the server because its use eliminates the direct interaction between the client and the business data and business service objects.

Simple Mail Transfer Protocol (SMTP)

A protocol for sending e-mail messages between servers. Most e-mail systems that send mail over the Internet use SMTP to send messages from one server to another; the messages can then be retrieved with an e-mail client using either POP or IMAP.

Simple Object Access Protocol (SOAP)

A lightweight XML-based messaging protocol used to encode the information in Web service request and response messages before sending them over a network. SOAP messages are independent of any operating system or protocol and may be transported using a variety of Internet protocols, including SMTP, MIME, and HTTP

Stirling Engine

Extremely efficient engine which converts heat energy into mechanical work using only thermodynamic principles of expanding and contracting volume of gas.

Supply Chain Management (SCM)

The control of the supply chain as a process from supplier to manufacturer to wholesaler to retailer to consumer. Supply chain management does not involve only the movement of a physical product through the chain but also any data that goes along with the product (such as order status information, payment schedules, and ownership titles) and the actual entities that handle the product from stage to stage of the supply chain.

There are essentially three goals of SCM: to reduce inventory, to increase the speed of transactions with real-time data exchange, and to increase revenue by satisfying customer demands more efficiently.

Swing

Graphical User Interface (GUI) API for creating Java-based applications.

Thermocouple

A temperature sensor formed by the junction of two dissimilar metals which has a voltage output proportional to the difference in temperature between the hot junction and the lead wire (cold) junction.

Transistor

A basic solid-state control device, which allows or disallows current flow between two terminals, based on the voltage or current delivered to a third terminal.

Transmission Control Protocol (TCP)

TCP is one of the main protocols in TCP/IP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.

Unified Modelling Language (UML)

A general-purpose notational language for specifying and visualizing complex software, especially large, object-oriented projects.

Uniform Resource Identifier (URI)

The generic term for all types of names and addresses that refer to objects on the World Wide Web. A Uniform Resource Locator (URL) is one kind of URI.

Uniform Resource Locator (URL)

The global address of documents and other resources on the World Wide Web.

Universal Description Discovery And Integration (UDDI)

A Web-based distributed directory that enables businesses to list themselves on the Internet and discover each other, similar to a traditional phone book's yellow and white pages.

User Datagram Protocol (UDP)

A connectionless protocol that, like TCP, runs on top of IP networks. Unlike TCP/IP, UDP/IP provides very few error recovery services, offering instead a direct way to send and receive datagrams over an IP network.

Value Object

Java object with multiple public attributes, used for setting and retrieving a number of values in one method invocation, thereby reducing network overhead.

Virtual Laboratory

A virtual laboratory is described as either a simulation of laboratory infrastructure or a system that provides remote access real-world laboratory equipment.

Web Service Description Language (WSDL)

An XML-formatted language used to describe a Web service's capabilities as collections of communication endpoints capable of exchanging messages. WSDL is an integral part of UDDI, an XML-based worldwide business registry

Web Services

The term Web services describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available.

Webmacro

WebMacro is a 100% Java open-source template language which may be used to separate presentation content from Java servlets.

World Wide Web Consortium (W3C)

An international consortium of companies involved with the Internet and the Web . The W3C was founded in 1994 by Tim Berners-Lee, the original architect of the World Wide Web. The organization's purpose is to develop open standards so that the

Web evolves in a single direction rather than being splintered among competing factions.

Appendix B

Overview of Stirling Engines

A Stirling Engine is a mechanical device which converts heat energy into mechanical work. It can be used for pumping water, generating electricity or turning industrial machinery. It does not need to use high quality refined fuels such as petrol or diesel to make it run, but can work on any source of heat.

The Stirling engine is a heat engine that is vastly different from the internal-combustion engine. Invented by Robert Stirling in 1816, the Stirling engine has the potential to be much more efficient than a petrol or diesel engine. But today, Stirling engines are used only in some very specialized applications, in propulsion systems for submarines for instance or auxiliary power generators for yachts, where quiet operation is important. Although there hasn't been a successful mass-market application for the Stirling engine, research into this area is ongoing.

The Stirling Engine relies on the principle that when a quantity of gas is heated (usually air, but sometimes helium or hydrogen), it will expand, and its volume will increase. If the gas is sealed in a container, then the pressure inside the container will rise. When cooled, the gas contracts, the volume decreases and thus the pressure will fall. Figure B-1 shows the operation involved in a simple displacer type Stirling Engine.

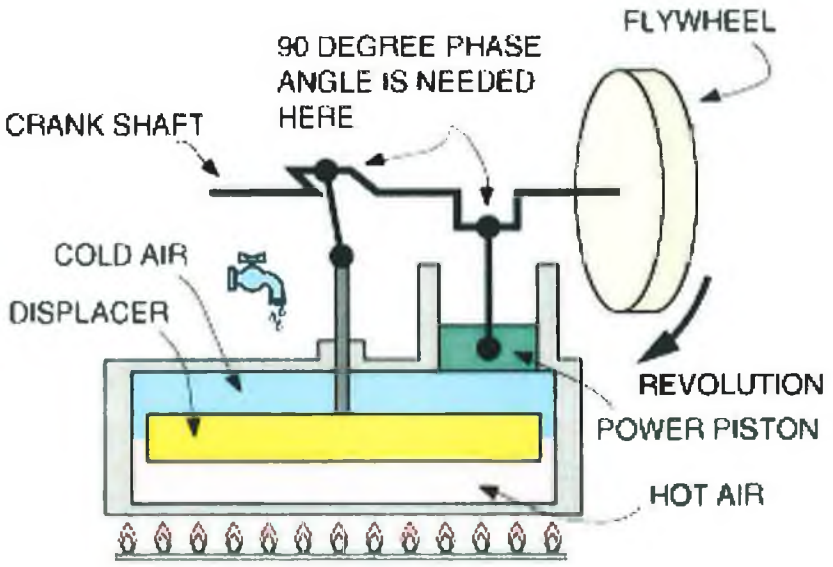

Figure B-1: Stirling Engine Operation (Source: American Stirling Company [86])

A displacer-type engine has a piston and a displacer. The displacer serves to control when the gas chamber is heated and when it is cooled. In order to run, the engine above requires a temperature difference between the top and the bottom of the large cylinder.

In Figure B-1, two pistons are shown:

The power piston

This is the smaller piston at the top of the engine. It is a tightly sealed piston that moves up as the gas inside the engine expands.

The displacer

This is the large piston in the drawing. This piston is very loose in its cylinder, so air can move easily between the heated and cooled sections of the engine as the piston moves up and down.

The displacer moves up and down to control whether the gas in the engine is being heated or cooled. There are two positions. When the displacer is near the top of the large cylinder, most of the gas inside the engine is heated by the heat source and it expands. Pressure builds inside the engine, forcing the power piston up. When the displacer is near the bottom of the large cylinder, most of the gas inside the engine cools and contracts. This causes the pressure to drop, making it easier for the power piston to move down and compress the gas. The engine repeatedly heats and cools the gas, extracting energy from the gas's expansion and contraction.

There are a couple of key characteristics that make Stirling engines impractical for use in many applications. Due to the fact that the heat source is external, it takes a significant time delay for the engine to respond to changes in the amount of heat being applied to the cylinder. This means that the engine requires some time to warm up before it can produce useful power and also cannot change its power output quickly. These shortcomings all but guarantee that it won't replace the internal-combustion engine any time soon. However, a Stirling-engine-powered hybrid engine might be feasible in the future.

Appendix C

Calculating the Efficiency of the Stirling Engine

Figure C-1 shows the configuration of the equipment used to characterise the performance of the Sterling Engine. Note that the Stirling Engine heater and cooling fan are both powered by the same power supply. Electric power delivered (P_{in}) by the programmable power supply is converted to Thermal power (Q_{in}) by the silicon rubber matt heater. Depending on the mechanical efficiency of the engine, some of this energy is converted into Mechanical Power (P_{out}) in a form that turns the output shaft and a certain speed (ω). An electrical circuit diagram for the Engine is presented in Figure 1.

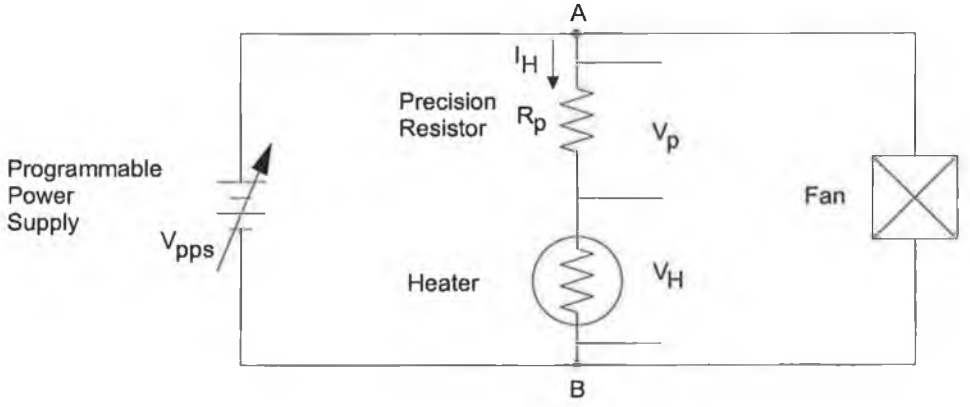

Figure C-1: Circuit diagram for Stirling Engine Heater & Fan

As the resistance of the heater varies with temperature a precision resistor is added in series to the heater circuit. This allows the current through the heater to be continually monitored and the thermal power dissipation of the heater can be established by multiply I_H by V_H.

First, the Current through the Heater: $= I_H = V_P / R_P$

The voltage drop across the heater, V_H, may be determined by:

$$V_H = V_{PPS} - V_P$$

The mechanical power out or kinetic energy of rotation may be determined by:

$$P_{Output} = T\omega = \frac{1}{2} m\omega^2 k^2$$

And

$$I_H = \frac{V_P}{R_P}$$

$$Q_{in} = I_H (V_{PPS} - V_P)$$

$$\Rightarrow Q_{in} = \frac{V_P}{R_p}(V_{PPS} - I_H R_P)$$

Where:

T = Torque required to turn the flywheel at a measured rotational speed (ω – rad/s)

m = mass of flywheel (kg)

k = radius of gyration of flywheel disk (m) (**k** = 0.71 r for a solid disk)

The resultant efficiency may now be calculated.

$$\eta = \frac{P_{Output}}{Q_{in}}$$

Figure C-2 represents the expected variation in temperature difference, rotational speed and thermal efficiency with respect to time.

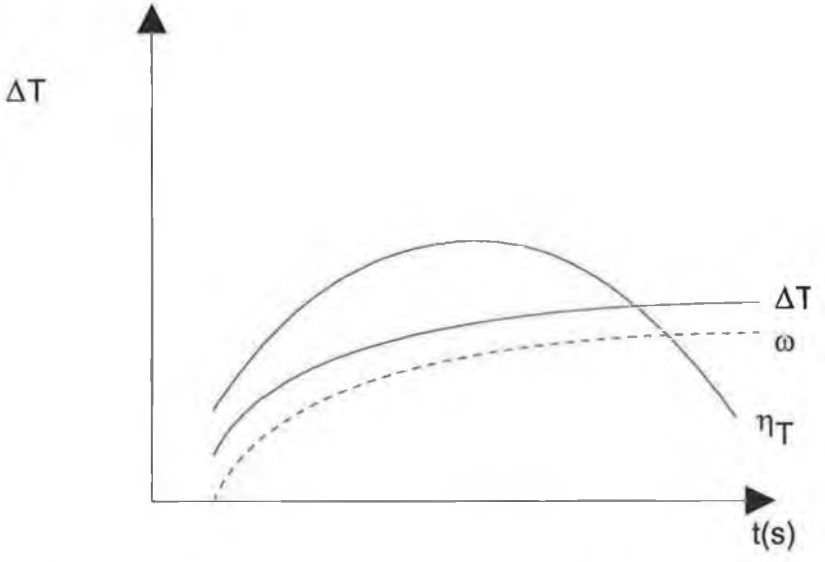

Figure C-2: Variation in Temperature Difference, Rotational Speed and Thermal Efficiency w.r.t. Time

Appendix D

Using the Stirling Engine Test Server Application

A Microsoft Windows-based application was developed to allow the application server and the real-time applet to control and monitor the test equipment used. The result, the Stirling Engine test server, is a native C++ Windows application which acts both as a local test application and as an demonstration management server. Figure D-1 shows the application on start-up where the previous settings are read and driver information is loaded into memory.

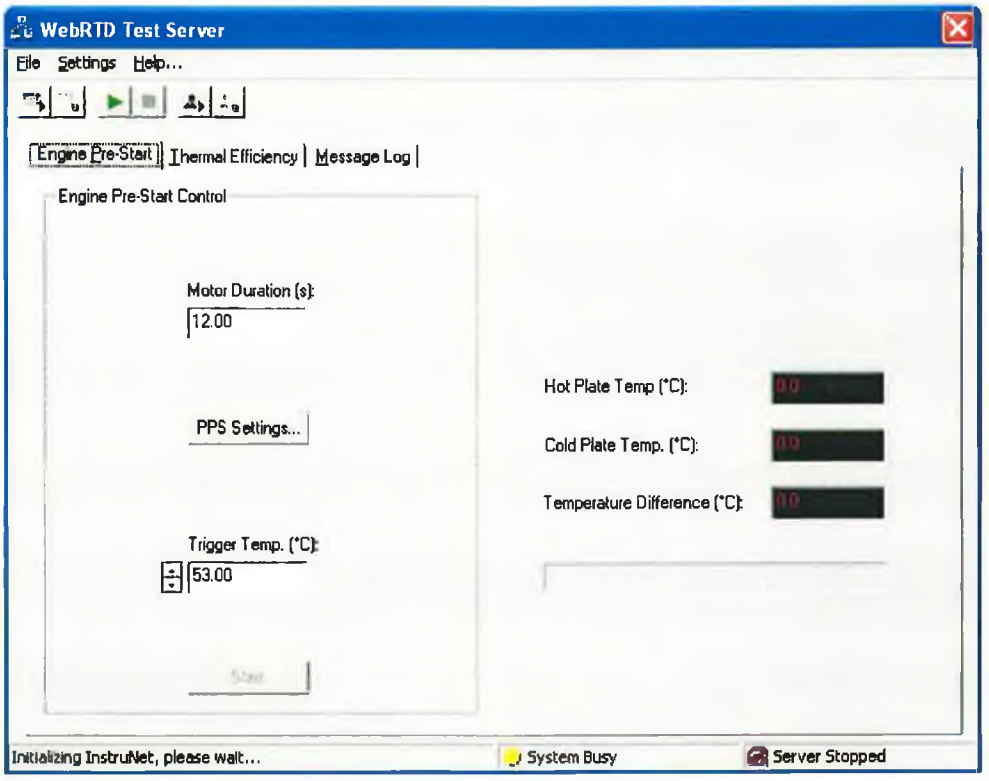

Figure D-1: Stirling Engine Test Server Initialising

The application has three pairs of buttons in the top left hand corner of the main panel area which are used for;

- Programmable Power Supply (PPS) control
- Server Control
- Web-Cam Control

There are three main tabbed panels, which include Engine Pre-Start and Thermal Efficiency sections similar to the real-time applet. A message log panel is also provided for displaying log messages.

Engine Pre-Start

The controls on the Engine Pre-Start are similar to those that appear on the remote applet. In addition, a progress bar is added on the right hand of the panel to indicate the progress made in reaching the pre-set trigger temperature difference between the hot and cold plates. Text fields are provided for specifying the starter motor duration and trigger temperature difference. The "PPS Settings..." button may be used to display the PPS settings dialog and will be described later.

The start button, as shown in Figure D-1, is used to start and stop the PPS, which should result in the attached heater pad and Stirling Engine hot plate being heated. Readouts on the right hand side will be updated in real-time, as there is no network connection with which to contend. All readings are read using the DAQ unit. When the trigger temperature difference has been reached, the application will signal the starter motor to start.

Thermal Efficiency

When the Stirling Engine is operating at the trigger temperature and its flywheel is rotating, the user may begin the thermal efficiency test procedure. This is done using the second main application tabbed panel as shown in Figure D-2.

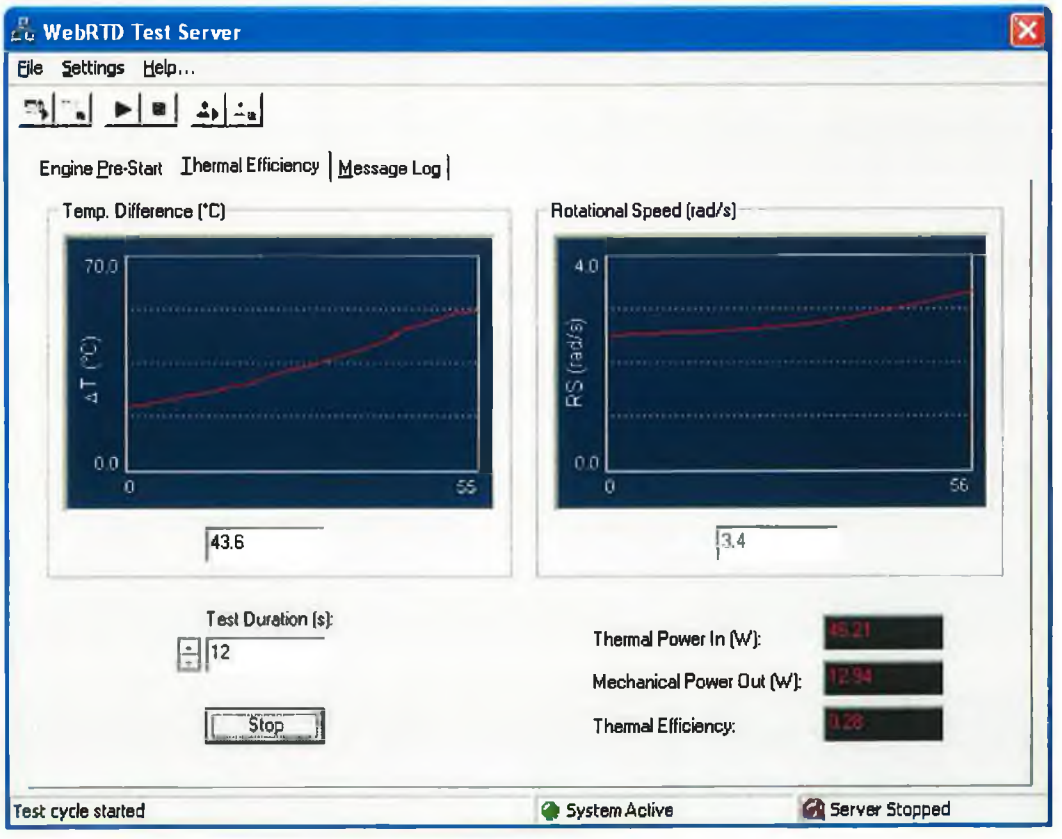

Figure D-2: Thermal Efficiency Tabbed Panel on Test Server Application

The test duration should be set prior to starting the thermal efficiency test. Again the start button is used to start and stop the test procedure. The temperature difference and rotational speed are plotted on the strip chart controls on this panel. The values of both appear just below the controls themselves, in text fields. The electrical thermal power input, mechanical power output and resultant efficiency are also displayed on this panel. If the rotational speed is detected at 0 rpm for approximately 10 seconds, then the application will display an error indicating that the flywheel is not in motion. This message is also broadcast if necessary, if the server facility is active. Power will continue to be supplied to the PPS to allow for an expedient re-start.

Elapsed Time, temperature difference and rotational speed are written locally to a file for subsequent storage in the application server database.

Server Facility

The test server application offers a server facility which may be activated if by the user if required. Figure D-3 shows how remote access may also be monitored using the Message Log display.

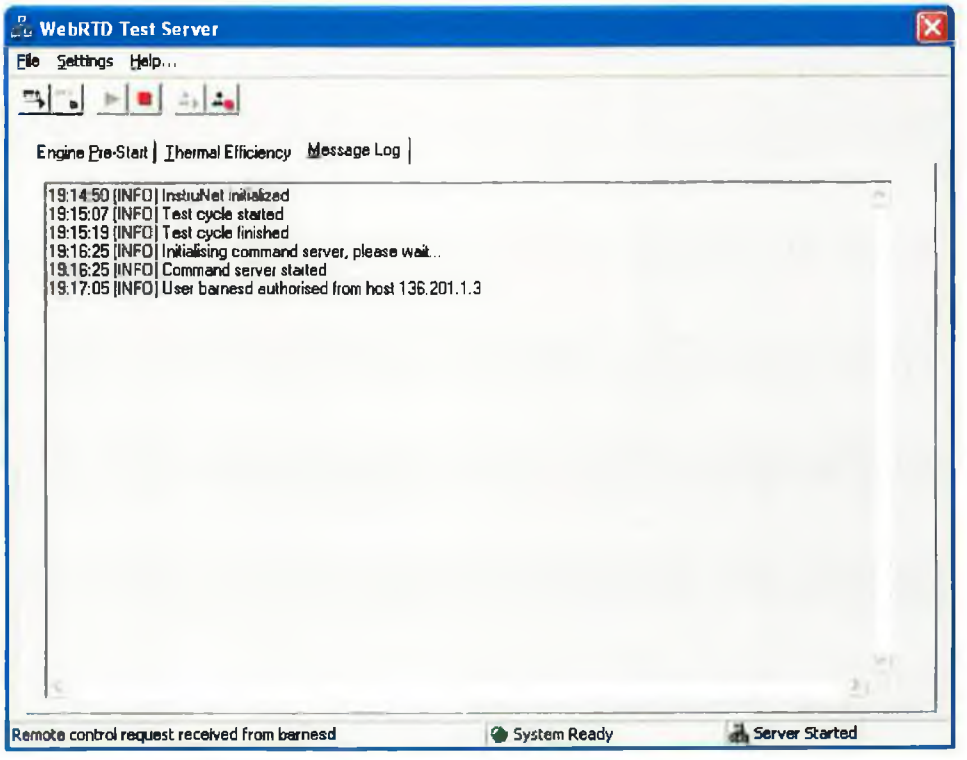

Figure D-3: Message Log Display

The middle pair of buttons in the top left-hand corner of the main application window allows the user to activate and de-activate the server facility. If this facility is enabled, then administrator users are allowed to control the application remotely depending on their remote address. Controls on the application are updated to reflect the current state of operation of the test procedure. For example, if the remote user signals the thermal efficiency test procedure to begin, the start button text will read "Stop" and the status bar will indicate that the system is active similar to that shown in Figure D-2.

If the server facility is enabled, then values of temperature difference and rotational speed are broadcast to a class D IP network address. This allows multicast clients to read and display current values. No data is transmitted and no remote access is possible when the server facility is disabled.

Available Configuration Items

The Settings menu in the main application window contains a number of menu items which enable the user to configure a number of hardware and software settings.

Programmable Power Supply Settings

The PPS requires voltage and current values to be prior to operation. These values may be specified using the PPS Settings dialog box as shown in Figure D-4.

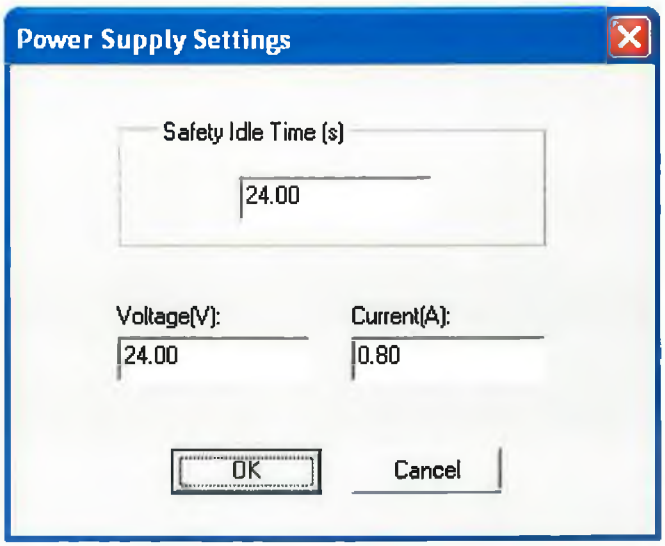

Figure D-4: PPS Settings Dialog Box

This dialog box is also used to supply what is known as the "Safety Idle Time", the idle time allowed where no local of remote access has been made, before power to the PPS is discontinued.

Server Settings

Permitted remote host addresses may be entered using the server setting dialog shown in Figure D-5. Only clients from these addresses will be given a token from the application, otherwise a request denial exception will be thrown. If necessary, this restriction may be removed by clicking on the "Allow All Hosts" radio button.

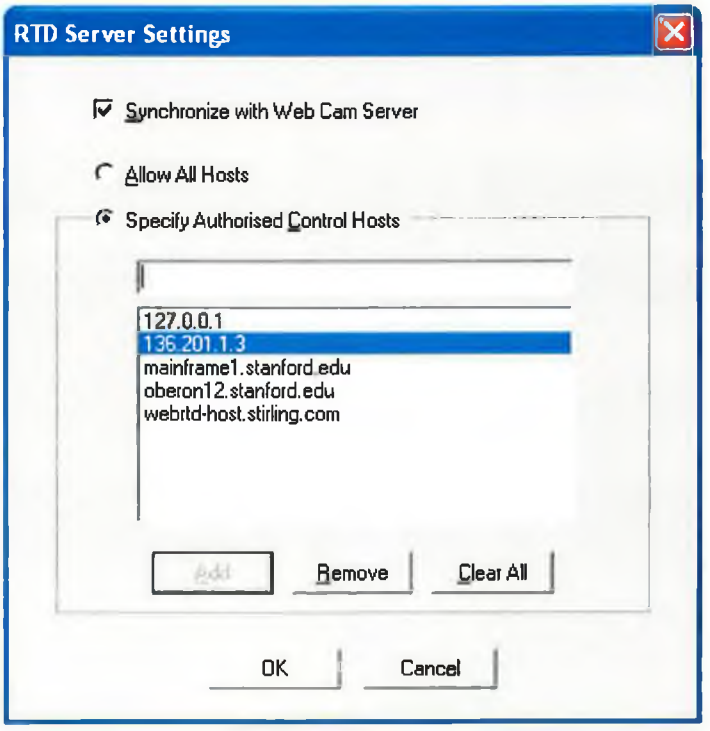

Figure D-5: Server Settings Dialog Box

A check box is also provided on this dialog box to allow the launching and terminating of the third-party web-cam image server to be synchronised with the actions of the server facility.

Data Acquisition Unit Settings

Figure D-6 shows the dialog box used to configure the input sensors used in the experiment. Not all channels were utilised but this feature allow different types of thermocouples to be used if necessary.

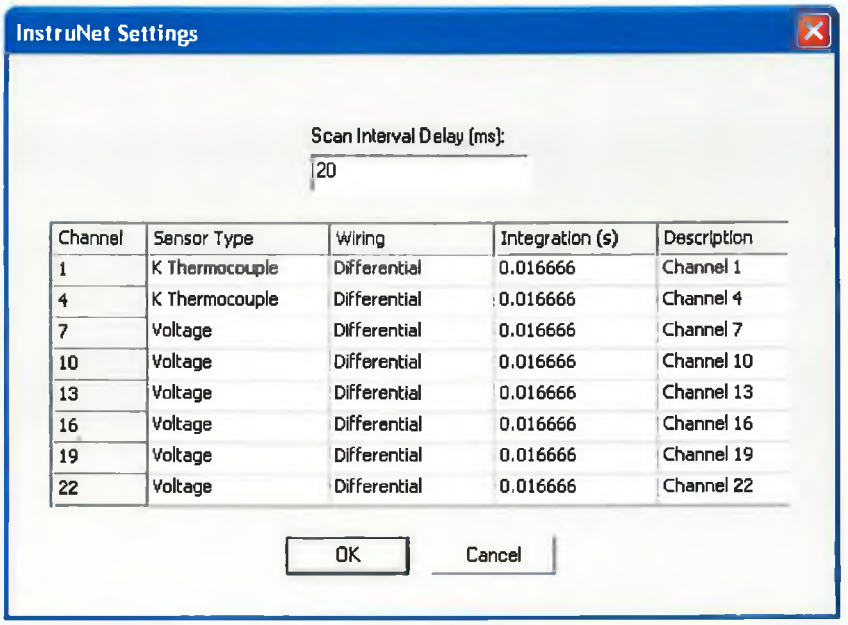

Figure D-6: DAQ Unit Settings

The scan interval delay shown here represents the delay used between sampling and processing loops in the thermal efficiency test cycle.

Web-Cam Settings

A third-party freeware application is used to transmit data to the remote web clients. The web-cam settings dialog box shown in Figure D-7 is used to supply the information necessary to launch and terminate the image server application. A check box is provided to disable the web-cam feature if, for instance, a web-cam is not available.

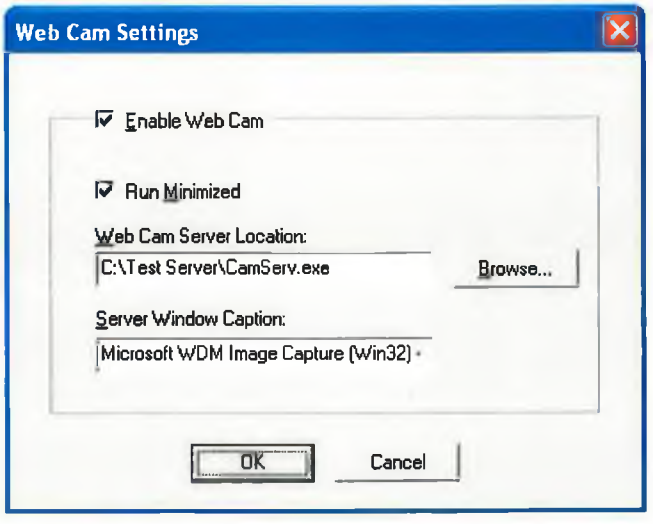

Figure D-7: Web-Cam Settings Dialog Box

Test Data Results Callback

On completion of the thermal efficiency test cycle the test data may be sent to a database for long-term storage. This setting may be made in the Result Data Settings dialog box shown in Figure D-8

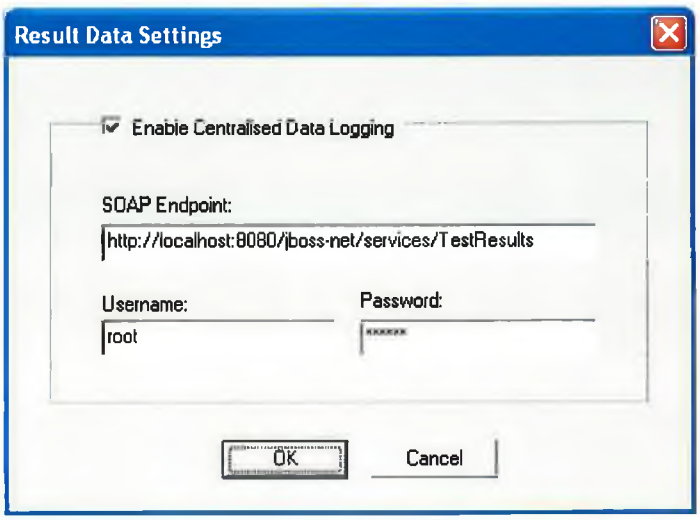

Figure D-8: Result Data Settings Dialog Box

The SOAP protocol is used to send the test data to the application server and the dialog box allows the SOAP endpoint to be specified. The application server requires a username and password to be sent as part of the SOAP message. These values can be supplied here also.

Appendix E

XML Deployment Descriptor - ejb-jar.xml

```
<?xml version="1.0" encoding="UTF 8" ?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN"  "http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb jar>
  <description>Virtual Lab Application</description>
  <display name>Virtual Laboratory</display name>
  <enterprise beans>
    <session>
      <description>
        Virtual Lab Main Access Point EJB
      </description>
      <ejb name>VirtualLab</ejb name>
      <home>net.webrtd.ejb.VirtualLabHome</home>
      <remote>net.webrtd.ejb.VirtualLab</remote>
      <ejb class>net.webrtd.ejb.VirtualLabBean</ejb class>
      <session type>Stateful</session type>
      <transaction type>Container</transaction type>
      <env entry>
        <env entry name>
          EngineReference
        </env entry name>
        <env entry type>java.lang.String</env entry type>
          <env entry value>
            corbaloc::1.2@enginehost.rtdhost.net:3030/AuthFactory
          </env entry value>
      </env entry>

      <security identity>
        <run as>
          <role name>InternalUser</role name>
        </run as>
      </security identity>
    </session>

    <session>
      <description>Test Results Callback Bean</description>
      <ejb name>TestResults</ejb name>
      <home>net.webrtd.ejb.TestResultsHome</home>
      <remote>net.webrtd.ejb.TestResults</remote>
      <ejb class>net.webrtd.ejb.TestResultsBean</ejb class>
      <session type>Stateless</session type>
      <transaction type>Container</transaction type>

      <security identity>
        <run as>
          <role name>InternalUser</role name>
        </run as>
      </security identity>
```

```
    <resource ref>
      <res ref name>mail/Mail</res ref name>
      <res type>javax.mail.Session</res type>
      <res auth>Container</res auth>
    </resource ref>
  </session>

  <entity>
    <description>Models a data acquisition user</description>
    <ejb name>RTDUser</ejb name>
    <local home>
        net.webrtd.ejb.RTDUserHome
    </local home>
    <local>net.webrtd.ejb.RTDUser</local>
    <ejb class>net.webrtd.ejb.RTDUserBean</ejb class>
    <persistence type>Container</persistence type>
    <prim key class>java.lang.String</prim key class>

    <reentrant>False</reentrant>
    <cmp version>2.x</cmp version>
      <abstract schema name>
          RTDUser
      </abstract schema name>

    <cmp field>
      <field name>login</field name>
    </cmp field>
    <cmp field>
      <field name>password</field name>
    </cmp field>
    <cmp field>
      <field name>name</field name>
    </cmp field>
    <cmp field>
      <field name>userType</field name>
    </cmp field>
    <cmp field>
      <field name>autoNotify</field name>
    </cmp field>
    <cmp field>
      <field name>email</field name>
    </cmp field>
    <primkey field>login</primkey field>
  </entity>

  <entity>
    <description>History Log Record</description>
    <ejb name>HistoryRecord</ejb name>
    <local home>
      net.webrtd.ejb.HistoryRecordHome
    </local home>
    <local>net.webrtd.ejb.HistoryRecord</local>
```

```
    <ejb class>net.webrtd.ejb.HistoryRecordBean</ejb
      class>
    <persistence type>Container</persistence type>
    <prim key class>java.lang.Object</prim key class>
    <reentrant>False</reentrant>
    <cmp version>2.x</cmp version>
      <abstract schema name>
        historyrecord
      </abstract schema name>
    <cmp field>
      <field name>historyText</field name>
    </cmp field>
    <cmp field>
      <field name>eventDate</field name>
    </cmp field>
  </entity>

  <entity>
    <description>User Role Definition</description>
    <ejb name>UserRole</ejb name>
    <local home>net.webrtd.ejb.UserRoleHome</local
      home>
    <local>net.webrtd.ejb.UserRole</local>
    <ejb class>net.webrtd.ejb.UserRoleBean</ejb class>
    <persistence type>Container</persistence type>
    <prim key class>java.lang.Object</prim key class>
    <reentrant>False</reentrant>
    <cmp version>2.x</cmp version>
    <abstract schema name>role_schema</abstract schema
      name>
    <cmp field>
      <field name>roleName</field name>
    </cmp field>
    <cmp field>
      <field name>roleGroup</field name>
    </cmp field>
  </entity>

  <entity>
    <description>
      Stirling Engine Test Data Definition
    </description>
    <ejb name>EngineData</ejb name>
    <local home>net.webrtd.ejb.EngineDataHome</local home>
    <local>net.webrtd.ejb.EngineData</local>
    <ejb class>net.webrtd.ejb.EngineDataBean</ejb class>
    <persistence type>Container</persistence type>
    <prim key class>java.lang.Object</prim key class>
    <reentrant>False</reentrant>
    <cmp version>2.x</cmp version>
    <abstract schema name>
        enginedata_schema
    </abstract schema name>
```

```
        <cmp field>
          <field name>testID</field name>
        </cmp field>
        <cmp field>
          <field name>deltaT</field name>
        </cmp field>
        <cmp field>
          <field name>rotationalSpeed</field name>
        </cmp field>
        <cmp field>
          <field name>timeElapsed</field name>
        </cmp field>

        <query>
          <query method>
            <method name>findDataByTestID</method name>
            <method params>
              <method param>int</method param>
            </method params>
          </query method>
          <ejb ql>
            <![CDATA[

                    SELECT OBJECT(data)
                    FROM enginedata_schema data
                    WHERE data.testID = ?1

            ]]>
          </ejb ql>
        </query>

        <query>
          <query method>
            <method name>ejbSelectTestIDList</method
              name>
            <method params />
          </query method>
          <ejb ql>
            <![CDATA[

                    SELECT DISTINCT data.testID
                    FROM enginedata_schema data

            ]]>
          </ejb ql>
        </query>
      </entity>
    </enterprise beans>

    <relationships>
      <ejb relation>
        <ejb relation name>
            RTDUser HistoryRecord
        </ejb relation name>
        <ejb relationship role>
```

```
        <ejb relationship role name>
                RTDUser has records
        </ejb relationship role name>
        <multiplicity>One</multiplicity>
        <relationship role source>
          <ejb name>RTDUser</ejb name>
        </relationship role source>
        <cmr field>
          <cmr field name>historyRecords</cmr field name>
          <cmr field type>java.util.Collection</cmr field type>
        </cmr field>
      </ejb relationship role>

      <ejb relationship role>
        <ejb relationship role name>
                record belongs to RTDUser
        </ejb relationship role name>
        <multiplicity>Many</multiplicity>
        <cascade delete />
        <relationship role source>
          <ejb name>HistoryRecord</ejb name>
        </relationship role source>
        <cmr field>
          <cmr field name>RTDUser</cmr field name>
        </cmr field>
      </ejb relationship role>
    </ejb relation>

    <ejb relation>
      <ejb relation name>RTDUser UserRole</ejb relation name>
      <ejb relationship role>
        <ejb relationship role name>RTDUser has roles</ejb
          relationship role name>
        <multiplicity>One</multiplicity>
        <relationship role source>
          <ejb name>RTDUser</ejb name>
        </relationship role source>
        <cmr field>
          <cmr field name>userRoles</cmr field name>
          <cmr field type>java.util.Collection</cmr field
            type>
        </cmr field>
      </ejb relationship role>

      <ejb relationship role>
        <ejb relationship role name>roles belong to RTDUser
        </ejb relationship role name>
        <multiplicity>Many</multiplicity>
        <cascade delete />
        <relationship role source>
          <ejb name>UserRole</ejb name>
        </relationship role source>
        <cmr field>
          <cmr field name>RTDUser</cmr field name>
```

```
        </cmr field>
      </ejb relationship role>
    </ejb relation>
  </relationships>

  <assembly descriptor>
    <security role>
      <role name>InternalUser</role name>
    </security role>
    <security role>
      <role name>AdminRole</role name>
    </security role>
    <method permission>
      <role name>AdminRole</role name>
      <method>
        <ejb name>VirtualLab</ejb name>
        <method name>*</method name>
      </method>
    </method permission>

    <method permission>
      <role name>AdminRole</role name>
      <method>
        <ejb name>TestResults</ejb name>
        <method name>*</method name>
      </method>
    </method permission>
    <method permission>
      <role name>InternalUser</role name>
      <method>
        <ejb name>RTDUser</ejb name>
        <method name>*</method name>
      </method>
    </method permission>

    <method permission>
      <role name>InternalUser</role name>
      <method>
        <ejb name>HistoryRecord</ejb name>
        <method name>*</method name>
      </method>
    </method permission>

    <method permission>
      <role name>InternalUser</role name>
      <method>
        <ejb name>UserRole</ejb name>
        <method name>*</method name>
      </method>
    </method permission>

    <method permission>
      <role name>InternalUser</role name>
      <method>
```

```
            <ejb name>EngineData</ejb name>
            <method name>*</method name>
          </method>
        </method permission>

        <container transaction>
          <method>
            <ejb name>RTDUser</ejb name>
            <method name>*</method name>
          </method>
          <method>
            <ejb name>HistoryRecord</ejb name>
            <method name>*</method name>
          </method>
          <method>
            <ejb name>UserRole</ejb name>
            <method name>*</method name>
          </method>
          <method>
            <ejb name>EngineData</ejb name>
            <method name>*</method name>
          </method>
          <trans attribute>Required</trans attribute>
        </container transaction>

        <container transaction>
          <method>
            <ejb name>RTDUser</ejb name>
            <method name>getUserData</method name>
            <method intf>Local</method intf>
            <method params />
          </method>

          <method>
            <ejb name>RTDUser</ejb name>
            <method name>setUserData</method name>
            <method intf>Local</method intf>
            <method params>RTDUserData</method params>
          </method>
          <trans attribute>Required</trans attribute>
        </container transaction>
      </assembly descriptor>
    </ejb jar>
```