Abstract—This paper presents a detailed description and explanation of the Scots Library System, a fully featured Library Management System designed and developed for the Scots School Bathurst.

The Scots Library System was developed primarily to illustrate that a fully featured, flexible, and scalable library management system, able to provide distributed and concurrent access to a centralized database, could be developed in a short time period and with little cost to client. This paper illustrates that there are very feasible, cheap alternatives to the highly expensive commercial library management systems, which are often too expensive for small libraries.

This system utilizes three major Java technologies – Enterprise Java Beans (EJB), Servlets, and Swing – and two pieces of supporting software – the JBoss Application Server and the MySQL relation database management system.

The Scots Library System is based primarily upon the Enterprise Java Beans (EJB) architecture. This architecture provides a component-based, modular, and highly flexible distributed model for the system. Using this flexible architecture allows minor modifications to the structure of the database and user interface to enable the system to be either scaled up to cater for multiple libraries in various locations, or scaled down to provide a “personal” edition of the system, suitable for private collections.

The main focus of this paper is to discuss the architectural structure of the Scots Library System, discussing the layered architecture used to provide throughput from the user interface to the database, and vice versa.

Index Terms—Object-Oriented programming, Java, Enterprise Java Beans, database interface.

I. INTRODUCTION

Providing distributed and concurrent access to a centralized database is a complex, time-consuming, and often very expensive, task. There are numerous ways of achieving this, ranging from utilizing a distributed database management system, such as Oracle, to purchasing a commercial system, such as ALICE, to developing a custom system using a language which supports network communications and concurrency, such as Java. Each of these approaches have their advantages and disadvantages. For example, a distributed database management system like Oracle provides a wide range of options for distributing the data and providing concurrent access, but these systems are usually very expensive, making them suitable only to large scale systems, and require very specialized knowledge. Similarly, commercial packages are often very expensive, and usually only provide set, non-customizable, features. Programming a custom distributed system allows the designer to build in all required functionality, but requires advanced programming skills, especially where concurrency is concerned.

There is, however, an easier way: Sun Microsystems’ Enterprise JavaBeans (EJB) specification. This technology provides a relatively simple mechanism by which a distributed application can be structured. Even better, the specification states that enterprise beans (the deployable components which form an EJB system) “must not attempt to manage threads,” as these “functions are reserved for the EJB container” [1] (all enterprise beans must be deployed to an Application Server, which serves as an EJB Container, in order to run), meaning that the developers do not have to explicitly handle concurrency in their enterprise beans, thus reducing the complexity of the task.

Therefore, when Scots School Bathurst approached us early in 2003 with a request for a custom Library Management System which could provide distributed, concurrent access to a central database, it was an easy choice to design the system using Enterprise JavaBeans.

II. A BRIEF HISTORY OF THIS DEVELOPMENT

The justification for the new library management system, dubbed “Scots Library System” came from the inability of the original library management system (a system called OASIS) to support key functionality such as networkability, concurrent access, a web-based search engine, and a secure master client application.

Apart from the functional deficiencies of the OASIS, there were several system attributes which were seriously lacking or not existent, such as security, robustness, data validation and
Verification, portability, scalability, and efficiency.

Furthermore, proprietary solutions, such as ALICE, an upgraded version of the OASIS system, are quite expensive, too expensive for the Scots School Bathurst.

Work for the Scots Library System officially commenced on February 23, 2003, and has seen the development of innovative solutions to both the functional deficiencies and the requisite system attribute areas, in an innovative cost effective manner.

At the time of writing this paper, development is scheduled for completion on November 7, 2003.

III. SYSTEM REQUIREMENTS

The Scots Library System is comprised of two core functional areas: a Master Client Application, and an Enquiry Terminal.

A. Master Client Application

The client required that the system be able to perform the following functions:

- Adding, removing and modifying the following objects in the system: Library items, Library item specifications, Suppliers, Authors, Publishers, Categories, Keywords, Publication locations, Staff, Students, Authorised personnel and Departments;
- Searching the database for any of the above objects based on select attributes of those objects;
- Loaning out and returning in items, checking and/or printing overdue lists and printing out barcode labels for items.

From these specifications we began to develop the core components of the Master Client Application, which would be secure and accessible only by library staff and authorised personnel.

B. Enquiry Terminal

The client also required that students, staff and other library customers be allowed to browse the selection of library items, from anywhere on the Scots School’s network, without being able to modify the data contained in the system in any way.

For this specification a web-based search engine was developed, which could be used without any form of authentication while not compromising the sensitive data of the system, and which students, while using a minimum of network communication, could access remotely.

IV. TECHNOLOGY AND SOFTWARE USED FOR THIS DEVELOPMENT

One of our primary goals when designing the Scots Library System was to make it as platform-independent, as portable, as we could. This was due in part to the requirements of the client: the client had originally wanted to install the server component of the system on the school’s FreeBSD internet and email server, the master client application onto a windows based machine, and have the enquiry terminal accessible from any platform, whether it be UNIX-based, Windows, or MacOS.

Therefore, our first technology decision was what programming language, or languages, to use for the development of the system. Since the main requirement of the language was that it could provide platform independence, we decided to use Java, by Sun Microsystems. Java is an extremely well supported, secure, and platform independent language. It is also free, thus reducing the cost of the system. As time progressed, it became evident that Java could provide all necessary functionality for the system.

Therefore, the Scots Library System is built upon three key Java technologies: Enterprise Java Beans (for the Server), Servlets (for the Enquiry Terminal), and Swing (for the Master Client Application). In addition to these technologies, two pieces of supporting software were required to run the system.

A. Enterprise Java Beans

Enterprise Java Beans (EJB) refers to both a specification (the EJB Specification) and a set of Java interfaces. The EJB Specification describes Enterprise JavaBeans as “an architecture for component-based distributed computing” [1]. This component based architecture provides a flexible, scalable basis for the design and construction of enterprise solutions.

Enterprise JavaBeans is one of the core components of Sun Microsystems’ Java 2 Platform, Enterprise Edition (J2EE).

EJB was chosen as the architecture for the Scots Library System server primarily due to its flexible and scalable nature. As we shall see later in this paper, this flexibility provides us with the ability to scale the Scots Library System either up, enabling it to support multiple libraries from various locations, or down, to produce a ‘personal edition’ of the application, suitable for keeping an inventory of a personal library.

Enterprise JavaBeans are deployable components, rather than standalone applications. Therefore, in order to run an EJB-based server, the server must be deployed to an Application Server, or EJB Container, which handles such things as network communication, concurrency control, resource pooling, and load-balancing. EJB also enables the bean provider to specify whether the Application Server handles such things as database transaction control and persistence control. For the Scots Library System server, we chose the JBoss Application Server, version 3.2.2, and decided to allow JBoss to handle all transaction control, while we provided the persistence control.

J2EE applications, which is essentially what the Scots Library System is, are multi-tiered applications, composed of multiple enterprise beans. An enterprise bean is a software component which can be deployed in a distributed J2EE application. The EJB Specification requires that enterprise beans expose several required methods, which allows the EJB Container to manage the beans in a uniform manner, regardless of what the beans actually do, or which application server is hosting them.

There are three types of enterprise beans: session beans, which model business process, and perform the actual business logic of the system (there are two forms of session beans –
stateless and stateful); entity beans, which model the business data; and message-driven beans, which are similar to session beans except that they are only called by sending messages to them [2]. The Scots Library System server component makes use of stateless session beans to perform the business logic of the system, and entity beans to store and retrieve data from the underlying database.

B. Servlets

One of the requirements for the Scots Library System was that it should provide an easily-to-use search engine for library customers, which would be accessible from any platform. After some careful research, we decided to use Java Servlets.

Java Servlets are server-side software components which dynamically extend the functionality of a server, usually an HTTP server [3]. Unlike Java Applets (which are executed within the client’s Java-enabled web browser), Java Servlets are executed on a Java-enabled server, such as the Jakarta Tomcat Web Server.

Using Java Servlets for the Enquiry Terminal enabled us to implement a web-based search engine which could easily interface with the main EJB-based server component, and which would be accessibly from any machine, running any operating system, on the client’s intranet (and over the Internet as well, with the appropriate connection and fire-wall permissions).

The Java Servlet API (a specification defining the classes and interfaces used to create and execute Java Servlets [3]) is one of the required components of J2EE, and therefore must be supported by all J2EE-compliant application servers [3]. Java Servlets are, therefore, supported by the JBoss application server being used for this system. In fact, JBoss actually incorporates the Jakarta Tomcat Web Server [4].

Servlets provided an ideal solution for the Enquiry Terminal for several reasons:

- Servlets can be deployed directly into the JBoss Application Server, thus reducing the amount of supporting software required by the system;
- Servlets can dynamically generate HTML pages using data retrieved from a persistent store (in this case, data retrieved from the underlying database, via the EJB-based server component of the Scots Library System), thus providing an excellent means of generating dynamic search results;
- Servlets typically provide improved performance, portability and security than many of their competitors (such as Common Gateway Interface, or CGI) – Servlets are usually run within the same process space as the server, thus reducing response time, whereas CGI scripts must create a new process for each new request (see Fig. 1). Furthermore, in addition to Java’s platform independence, Java is typically less error prone than other languages typically used for writing CGI programs (such as C).

C. Swing

Following the choice to use Java as the development language for the Scots Library System, it was a natural step to decide to use Java’s graphical user interface development package, Swing, for the development of the master client application component of the system.

While it would have been feasible to create the entire master client application using Java Servlets, it was decided that a stand-alone application would provide greater security and flexibility for the master client application component.

That left us with a choice between Java’s older AWT package, and the newer, lighter-weight, and much more flexible Swing package. This was not a difficult decision.

D. Software

Due to only a very small budget, no proprietary software could be afforded for this system. However, this was not a serious problem for this development.

The Scots Library System only required three pieces of supporting software: an Application Server for the EJB-based server component to be deployed in; a Web Server for the Java Servlet-based Enquiry Terminal to be deployed in, and a relational database management system (RDBMS), for the underlying data store.

After some investigation, we decided to use the freely available JBoss Application Server, since it was a fully J2EE compliant application server, and also incorporated the Jakarta Tomcat Web Server, which made deploying Java Servlets even easier.

As for the RDBMS, Scots School Bathurst had an existing MySQL server running on their system. We were, therefore, able to utilize the existing database server.
V. SYSTEM ARCHITECTURE

A. Enterprise JavaBeans Architecture

The architecture of Enterprise JavaBeans is far too complex to be described in detail in this paper. Of particular note to this paper, however, is the way in which EJB achieves network communications and location transparency. This ability is one of the primary reasons for choosing EJB for the Scots Library System, as the EJB Specification states that all network-related functionality is to be handled by the Application Server. This means that no explicit networking code needs to be written by the development team, thus enhancing the flexibility and portability of the system by removing the need to provide specific network control in the software.

Enterprise JavaBeans provide location transparency by using the Java Naming and Directory Interface (JNDI) to store location information for network resources, such as enterprise beans.

In order for a client application to communicate with an enterprise bean, the client needs to obtain a reference to an enterprise bean’s Home Object, which is a special EJB object that serves as a factory for the bean’s EJB Object.

The Home Object is created by the Application Server based upon the Home Interface provided by the bean developer. The EJB Object is also created by the Application Server, based upon the Remote Interface supplied by the bean developer, and handles all of the network communications. The EJB object serves as a proxy object for the actual bean implementation class, passing method calls and return values to/from the bean implementation class.

Fig. 2. Enterprise JavaBeans invocation model, illustrating how EJB uses JNDI and remote objects to supply location transparency. Client code obtains a reference to the remote Home Object by looking up the object reference via JNDI, and calls the create() method, via the enterprise bean’s Home Interface, which returns a reference to the remote EJB Object. The client then calls business methods on the EJB Object (these methods are specified in the bean’s Remote Interface), which are then passed to the bean’s implementation class to be run. The EJB Object handles all network related functionality, such as marshalling and unmarshalling parameters, and so forth. (Source: E. Roman, S.W. Ambler, T. Jewell, Mastering Enterprise JavaBeans, 2nd ed. John Wiley & Sons, USA, 2002, pg 73) [2]

Fig. 2. shows the EJB invocation model, which illustrates how these objects interact with JNDI to provide EJB’s location transparency and portability.

B. Scots Library System Architecture

As stated previously, the Scots Library System was designed using an architecture based upon the Enterprise JavaBeans specification. We have adapted this architecture to provide a layered model for the system. This model consists of the following layers (see Fig. 3):

- **Data Layer** – this layer consists of the actual database containing all of the library information, from borrowers, to loans, to items. This layer consists of a single MySQL relational database, stored on the Scots School’s FreeBSD email and web server (JBoss and the server component of the Scots Library System are installed on the school’s Windows 2000 domain server)

- **Entity Layer** – this layer sits directly atop the data layer and provides the data access logic required to retrieve data from, or send data to, the database. This layer is comprised solely of Entity Beans, which model the data stored in the database [2] (for the Scots Library System, one Entity Bean models one table in the database). Relationships between tables are enforced programmatically by the Entity Beans themselves. Database transactions are handled by the EJB Container, while persistence is handled explicitly via Java Database Connectivity (JDBC) and SQL.

- **Session Layer** – this layer provides the business logic which manipulates the data to produce meaningful results which can either be displayed on the screen for the user or stored in the database. This layer is composed solely of Stateless Session Beans, which model the business functions [2] of the system. We chose to use Stateless Session Beans as opposed to Stateful as there was no real need to persist any of the data handled by/incorporated in these beans.

- **Transport Layer** – this layer is provided by the Application Server, JBoss. The transport, or network, layer handles all network communications between the session layer and the business layer. This is a part of the EJB Specification, which states that enterprise beans should not perform any network-related functionality (such as creating sockets, or setting socket factories, etc)[1]. These functions are required of any J2EE-compliant Application Server. The only network-related functionality that an enterprise bean should use is to look up beans via JNDI, Java’s standard means of looking things up over a network [2]. JNDI allows the enterprise bean programmer to obtain references to other enterprise beans that have been registered with the JNDI.

- **Business Layer** – The data, entity and session layers all reside on the server side of the Scots Library System. The business layer, however, resides on the client side. This layer is essentially the client-side
counterpart of the session layer. It is responsible for looking up remote objects to obtain a remote reference to appropriate server-side session bean, receiving data from that session bean and formatting it so that it can be displayed on the screen, and for retrieving information from the screen and sending it to the session layer. Like the session layer, the business layer must use JNDI to obtain a reference to the server objects (see below for more details).

- Presentation Layer – the presentation layer is composed of the actual user interfaces. This layer is what the user interacts with, and is where the business layer displays information for the user. The Scots Library System has two separate user interfaces: the Master Client Application and the Enquiry Terminal. Each of these has its own associated business layer.

Fig. 3. Layered architecture used for the Scots Library System. The Master Client Application and the Enquiry Terminal comprise the Presentation Layer (what is displayed to the user) and the business layer (where data is displayed in the presentation layer or retrieved from the presentation layer). The business layer sends data to the transport layer, which passes it on to the session layer, where the actual business logic is performed (i.e., the data is manipulated in such a manner as to produce meaningful results). The session layer sends the information to the entity layer, or data access layer, which performs the actual database communications with the data layer.

Fig. 4 provides an illustration of how the various layers are structured, and what technology is used to create and communicate between the client-side and server-side layers. As can be seen, the Master Client Application’s business layer communicates with the session layer via a combination of Java’s native mechanism for providing network communications, Remote Method Invocation (RMI), and JNDI, over a TCP/IP network. The Enquiry Terminal, which was developed using Java Servlets, actually resides on the server, and so does actually need to communicate over the network. Instead, the Servlet generates dynamic HTML web pages with the results of the search, which are then accessible via the Scots School’s private intranet.

The Entity Layer communicated with the data layer via JDBC and SQL.

Fig. 4. Technologies used to create, and communicate between, the various layers composing the Scots Library System’s architecture.

VI. RESULTS

The use of the architecture described in the previous section, combined with the technology and software described in Section IV, has enabled us to create a highly flexible, scalable solution to a difficult problem with a minimum of costs for the client.

Since all of the software and technology being used for this development is freely available for both development and deployment, the client was not required to spend large amounts of money on this system, as they would have had they gone with a proprietary package. The costs incurred during this development totaled to $250 (this did not including labor costs, as this was essentially a research project).

The use of EJB’s component-based architecture, combined with the layered architecture designed for the Scots Library System, has made this system easily scalable. By breaking the major areas of data access and business logic into two separate layers, it is relatively simple to add new functionality, in the form of additional Entity Beans and/or Session Beans, or to remove existing functionality, by removing Entity Beans and/or Session Beans. There is very little interaction between beans in each of these layers, meaning that beans can be removed without affecting the functionality of other beans.

The following section outlines some ways in which the Scots Library System could be modified to take advantage of this flexibility and scalability.

VII. FUTURE MODIFICATIONS AND APPLICATIONS

Due to the flexibility and scalability of Enterprise JavaBeans, the Scots Library System could easily be adapted for a number of situations with only a few very minor modifications.

For example, at the moment this system is suitable for a single library, such as a school or public library. There is no notable limit on the number of library items that can be stored...
in the system, or the number of registered borrowers or authorized personnel. Therefore, in its current state, this system would be suitable for a library of any size, as long as it only had a single branch or location.

However, with the simple addition of a ‘location’ field into the information stored on library items (both in the database and in the user interfaces) the Scots Library System could be expanded to cater for larger libraries with multiple branches, or for a coalition of cooperating libraries. This additional field would simply indicate at which location/branch a particular item is stored. Items could then be borrowed from Location A, and returned at Location B (providing there was some means of physically transporting items between locations).

The addition of a location field would make this system suitable for almost any library or group of libraries.

Or, we could scale the system the other way: remove information regarding borrowers, loans, reservations, etc, and we would have an effective ‘personnel edition’ of the system, suitable for any private collection. We could even make the system modular, and allow the user to choose what data/functionality they want at install-time. Then it would be a matter of simply not installing the components (enterprise beans) for the unwanted data/functionality.

Furthermore, additional data/functionality could be added in a relatively simple manner: only relatively small, and mostly isolated, portions of the overall system would need modification, rather than having to re-write and re-compile the entire system.

VIII. CONCLUSION

Scots Library System is a fully featured library management system designed and developed for the Scots School Bathurst, to provide distributed, concurrent, and secure access to the Library’s central database. The Scots Library System utilizes an Enterprise JavaBeans-based, layered architecture to provide the distribution of access.

Through the careful design of the system’s architecture, combined with EJB’s component-based design, the Scots Library System is able to provide a flexible, modular, and scalable library management system, which can be easily modified to become suitable for much larger organizations, such as a coalition of cooperating libraries, or for much smaller systems, such as a personal collection.

The inherent platform independence of the technologies used carries over into the Scots Library System, thus making it suitable for almost any environment, from UNIX to Windows.

The Enterprise JavaBeans specification states that all concurrency control must be handled by the Application Server in which the enterprise beans are deployed, thus removing the responsibility of providing thread-based concurrency control from the developers. This enables the development of a system which can be used by multiple users concurrently, without suffering from such problems as read/write conflicts or deadlock, and without requiring complex concurrent programming abilities from the developers.

Finally, Enterprise JavaBeans, as well as the other technologies and software used in this development, provide a very inexpensive and relatively simple alternative to creating a system which manually handles complex network communications and concurrency control, or using an expensive distributed database management system, or commercial product.

REFERENCES