FLLMMCIS: A Web-based Database-driven Inventory System

Russell Pierce

Major Professor: Dr. Homer Carlisle

Auburn University

Samuel E. Ginn College of Engineering

Department of Computer Science and Software Engineering

Master's of Software Engineering (MSWE) Non-thesis Project

Abstract

This project has produced a working inventory system for the Auburn University Department of Foreign Languages and Literatures Multimedia Center. The produced system is called the Foreign Languages and Literatures Multimedia Center Inventory System, abbreviated as "FLLMMCIS" or simply "IS" within the context of this document for brevity. The IS facilitates the Multimedia Center's inventory lending activities. The system is a database-driven web site that has been written in ColdFusion backed by a MySQL database. This paper introduces readers to the system and details its development. In this paper an overview of the IS from a functional perspective is offered followed by the details of the project and a brief conclusion.

Table of Contents

List of Figures

1. System Introduction and Walk-through

The inventory system (IS) facilitates the operation of the Department of Foreign Languages and Literatures (FLL) Multimedia Center (MMC) library. The primary goal of the library is to provide lending services of multimedia assets to students, faculty, and staff. The system facilitates lending by tracking the status of the library inventory and enables users to perform related business activities. The users of the inventory system are MMC employees. They use the system to maintain an accurate inventory, document lending activities, and administer MMC library policy.

The FLLMMCIS is a database-driven web site. The site is organized into two sections. The library section provides the user with all of the primary functionality for which the system exists. The administration system exists to support the operation of the library section. Users operate within the library section for all day-to-day activities, and use the administration section for special-purpose and occasional activities.

1.1 Library Section

The IS accessed through a web browser. The login screen prompts users for their username and password. Upon login the users are taken to the main page of the library section. The library section is where “normal” system functions are accessed. Checking in and out items and maintaining the inventory are the most common activities. For these activities, large links are provided in the main content area. On the left, a menu allows for direct access to all library activities, and at the top left, a tabbed menu allows users to navigate between different areas of the system.

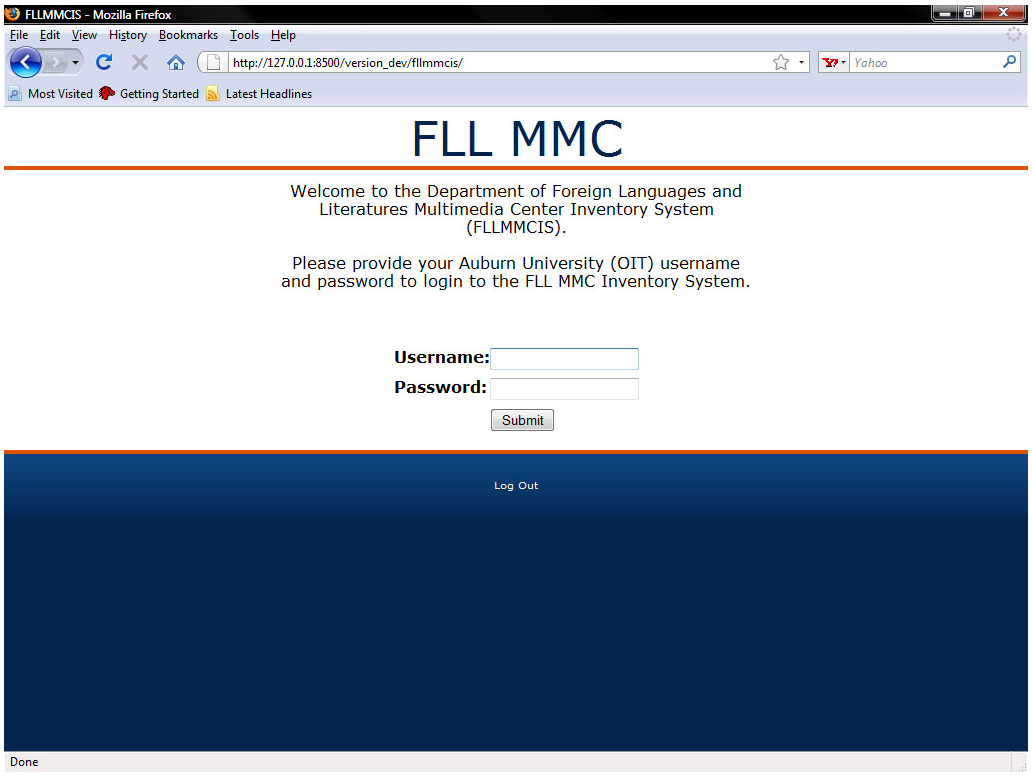

Figure 1.1 Login

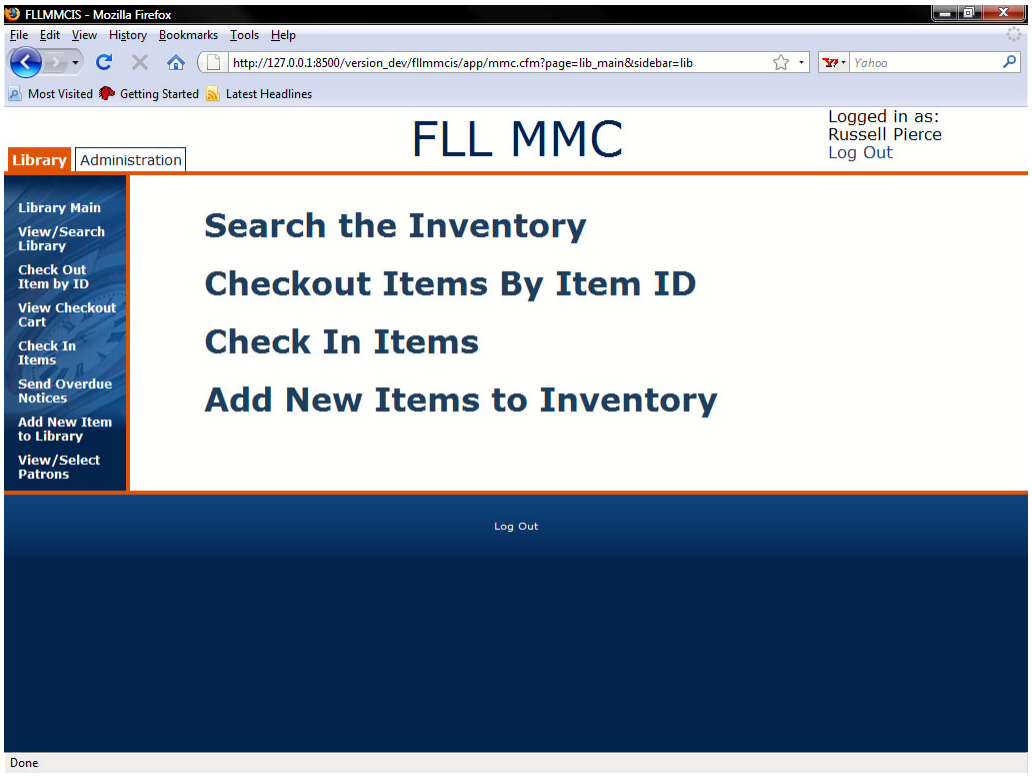

Figure 1.2 Library Main

1.1.1 Checking Out Items

There are several common scenarios for checking out an item. A number of different functions are provided to facilitate each scenario. However the checkout process is executed, the process requires that the desired items be added to the checkout cart.

During the checkout process the patron will sometimes provide only the title of the item they wish to check out. The user can search for the item using the "View/Search Library" page. On this page, users can search for items and view matching results. When the desired item has been found, it can be added to the checkout cart by clicking corresponding cart icon. If the patron supplies the item ID, the user can use the "Check Out an Item by ID" page. The user enters the item ID and clicks the "Add Item to Cart" action button. The item is then added to the checkout cart.

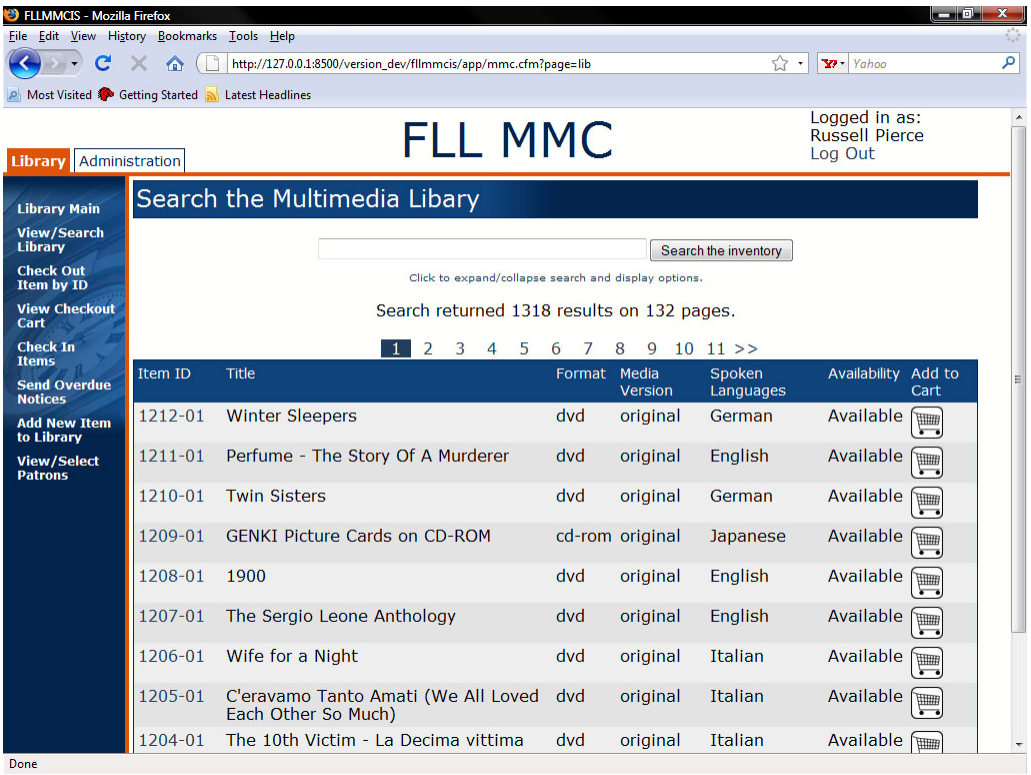

Figure 1.3 Searching for an Item

After all the desired items are in the checkout cart, the user selects the patron by navigating to the “View/Select Patrons” page or by taking the “Select a Patron” action found on the checkout cart page. On the “View/Select Patrons” page as seen in Figure 1.4, the user selects the patron from the list. If the user has taken the “Select a Patron” action from the checkout cart, then they will automatically be returned to the checkout cart. After selecting the duration for the loan, the checkout is completed with the “Check Out” action.

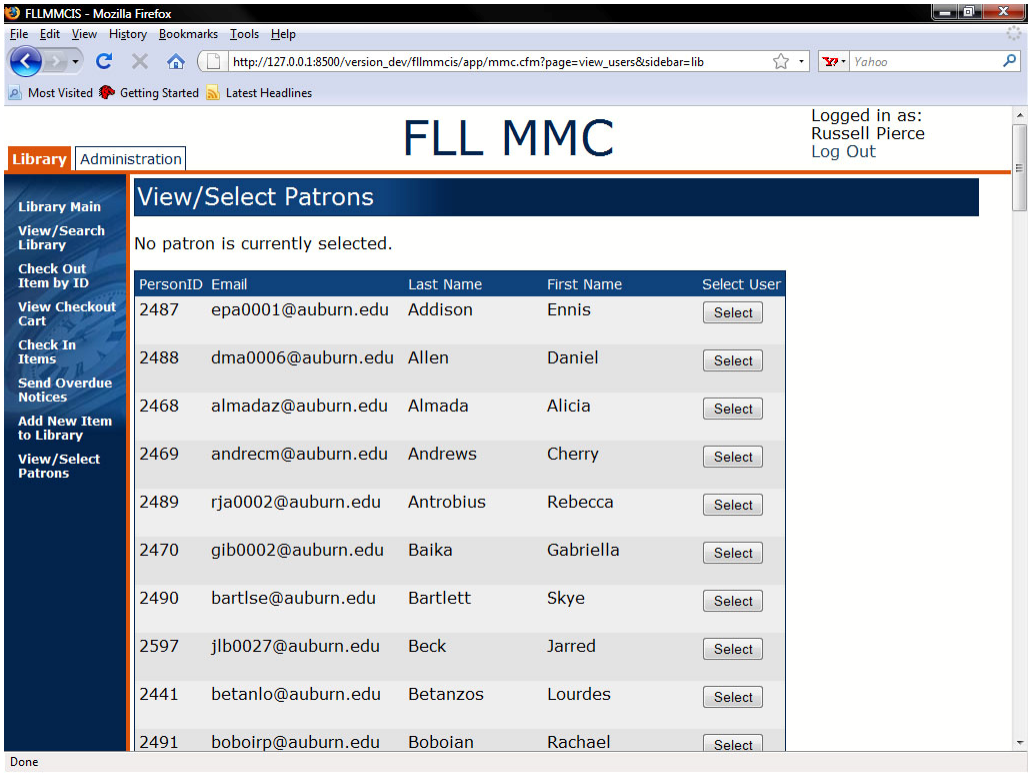

Figure 1.4 Selecting a Patron

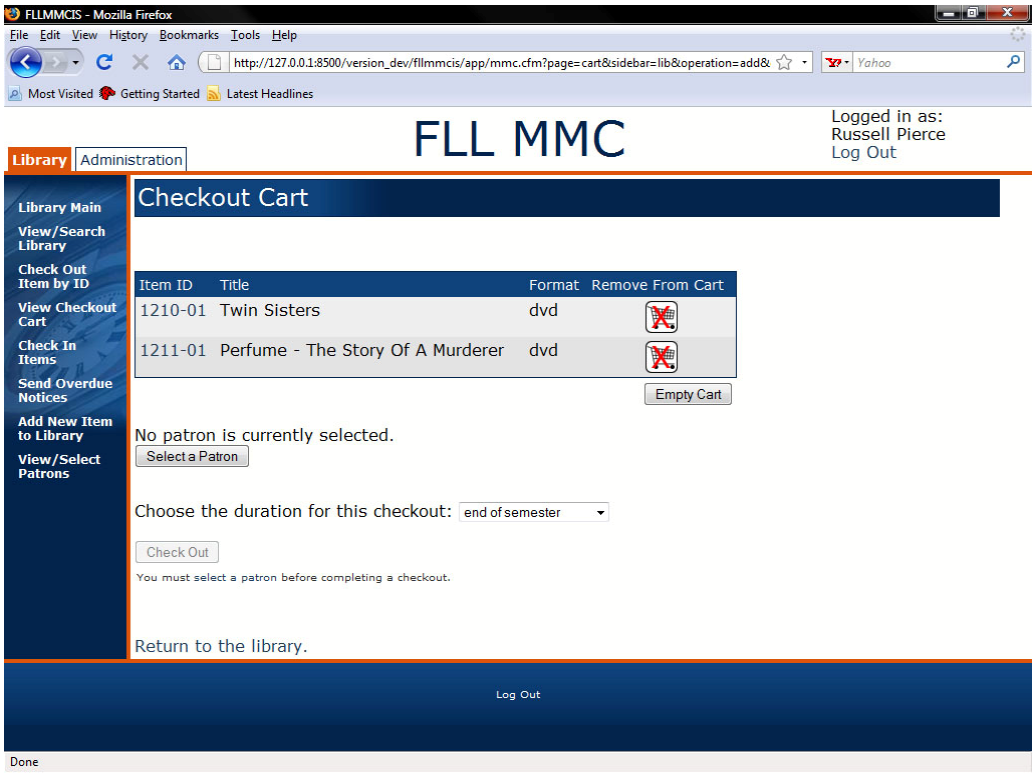

Figure 1.5 The Checkout Cart

1.1.2 Viewing Item Details

From several pages where item IDs are displayed, users can access the Item Details page by clicking on the item ID. This page provides a complete description of a single item. From this page, users can select to edit or delete the item they are viewing. If available the item can be added to the checkout cart. In Figure 1.4, the item being view is checked out, so the "Add to Checkout Cart" button is disabled.

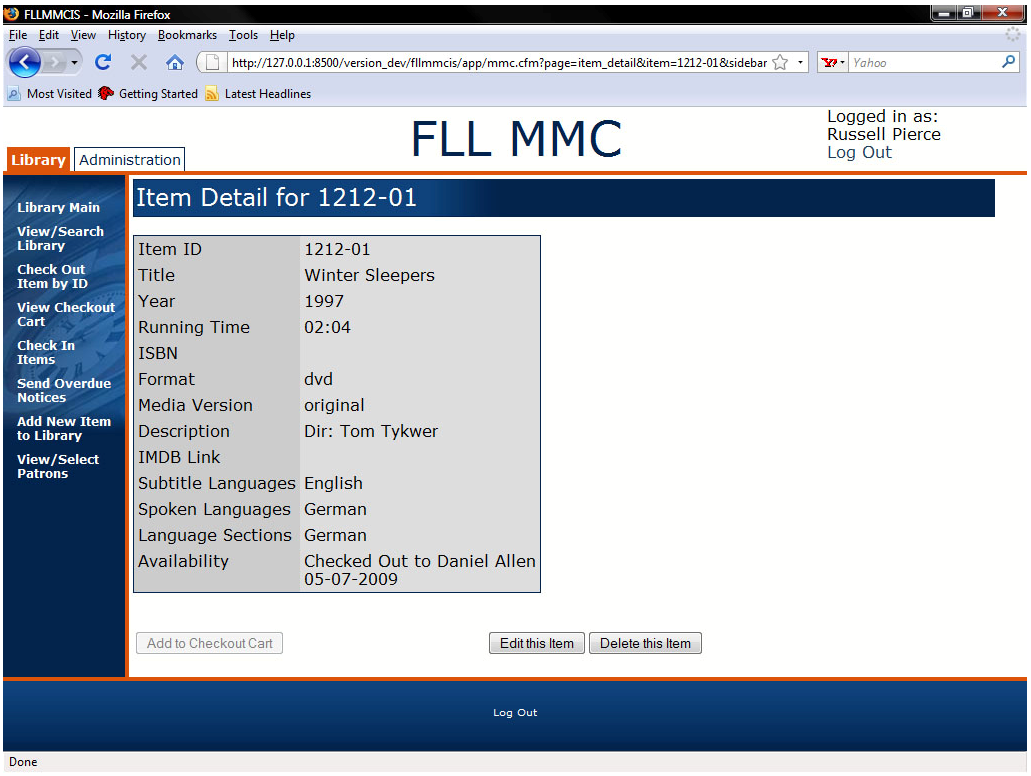

Figure 1.6 View Item Details

1.1.3 Checking In Items

The “Check In Items” page displays items that are currently checked out. Users select the item or items they wish check in by selecting the corresponding check box. The user then takes the “Check In Marked Items” action to complete the check-in.

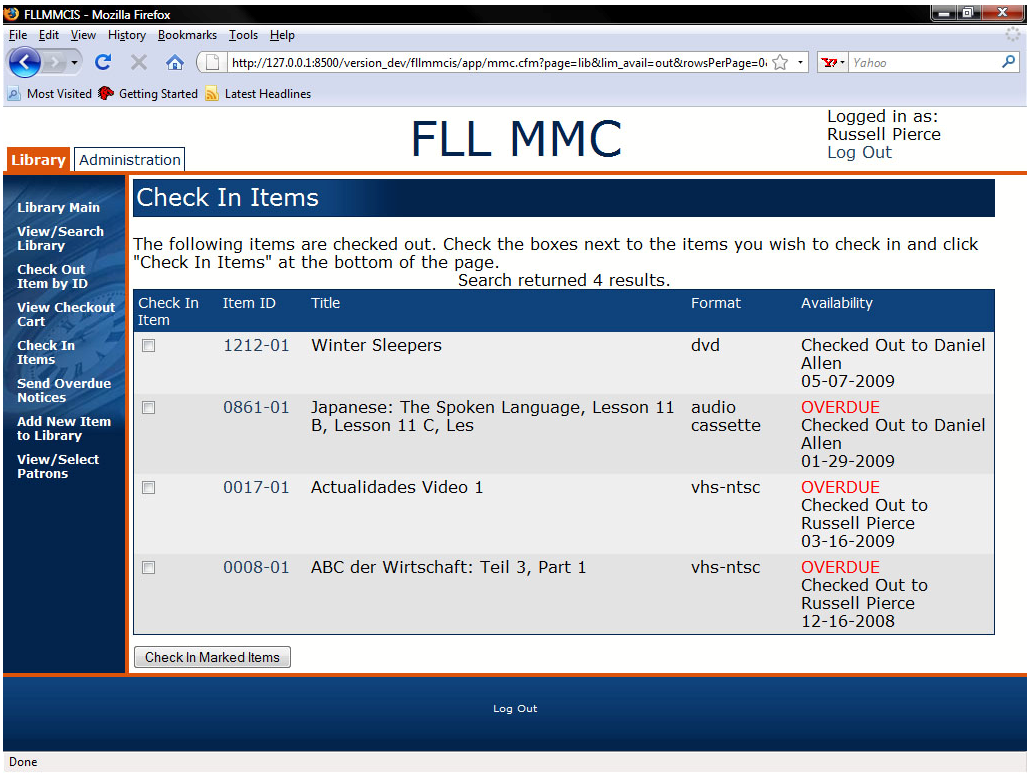

Figure 1.7 Check in Items

1.1.4 Sending Overdue Notices

Overdue notices can be sent to patrons who have exceeded the period of their loan. On the “Send Overdue Notices” screen, a list of users who are borrowing overdue items is displayed. The items that are overdue are also displayed. The user can select to send an overdue notice to some or all patrons by selecting the corresponding checkboxes and pressing the “Send Overdue Notices” button.

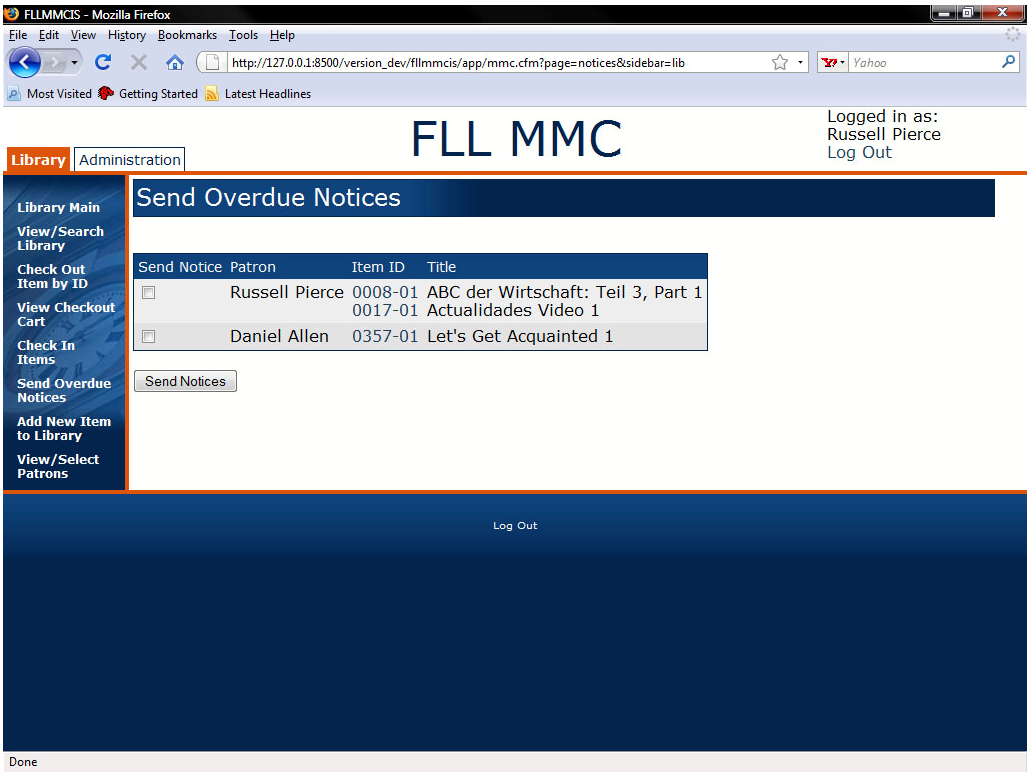

Figure 1.8 Send Overdue Notices

1.1.5 Adding, Editing, and Deleting Items

New items are added to the MMC library regularly. The "Add Item New Item to Library" link takes users to page where they can add one item to the inventory. The user completes desired text fields and makes appropriate selections before using the "Add Item" button to add a new item.

From an item's detail page, a user can choose to edit the displayed item. The user is taken to the "Edit Item" page which uses the same form as the "Add Item" page. The user can change any field appropriately and apply the changes using the "Edit Item" button.

On the "Item Detail" page users can also delete the displayed item from the IS. When the "Delete this Item" action is taken, the user is taken to a conformation screen. This screen displays a warning and asks the user to confirm the delete action.

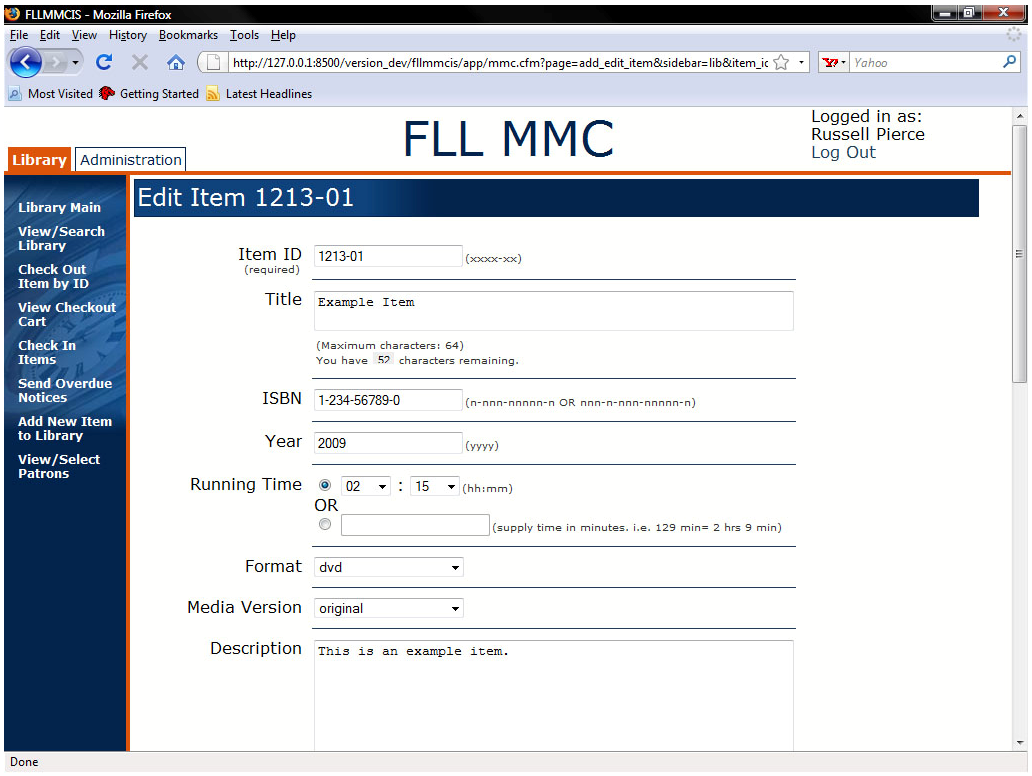

Figure 1.9 Editing an Item

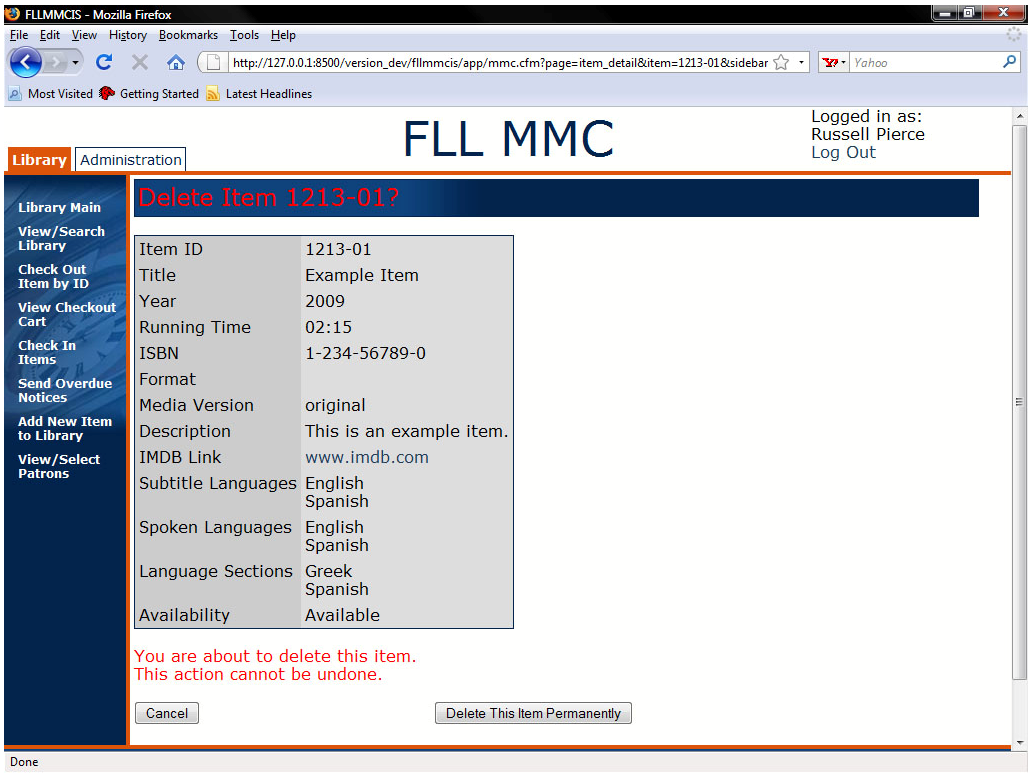

Figure 1.10 Confirming the delete action

1.2 Administration Section

The administration section provides functions that are not directly related to day-to-day operation of the library. Activities that are performed only occasionally and that are related to the operation of the system, rather than the library, are performed within this section. Such activities include editing database tables, managing system users, and viewing system reports.

1.2.1 General Database Access

In the General Database Access area, any table in the database can be viewed or manipulated through a standardized interface. Common activities in this area include adding new item formats and adding terms.

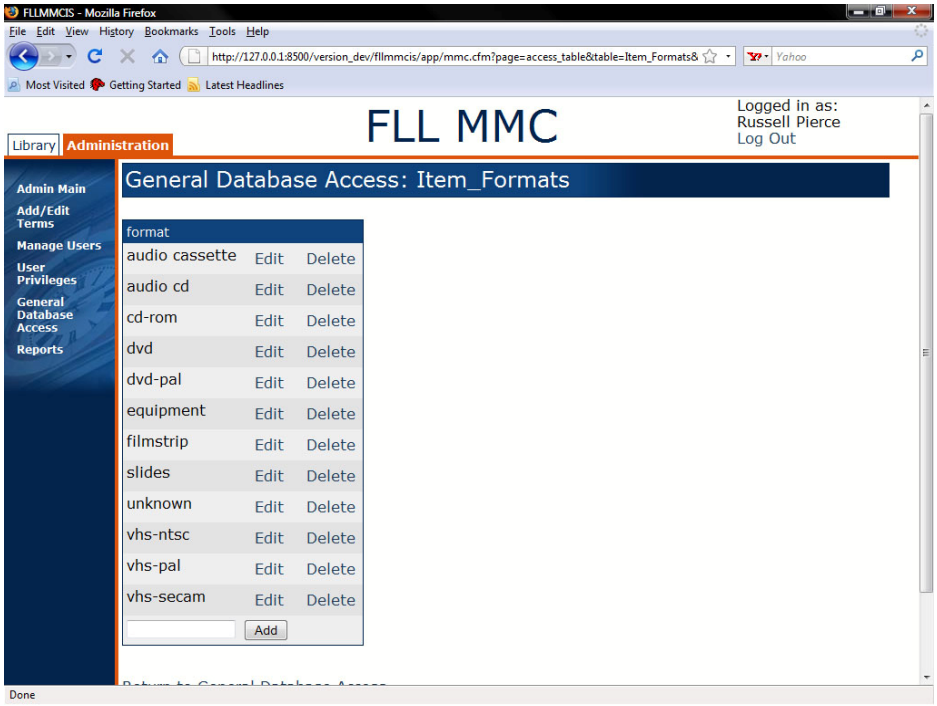

Figure 1.11 General Database Access

1.2.2 Managing Users

The “Manage Users” page allows users to be added or removed from the IS. This page also allows user classes to be assigned. These user classes are used by the access control system to determine if a user should be allowed to use the various functions of the IS. Users cannot remove themselves from the system. This prevents users from accidentally locking themselves out.

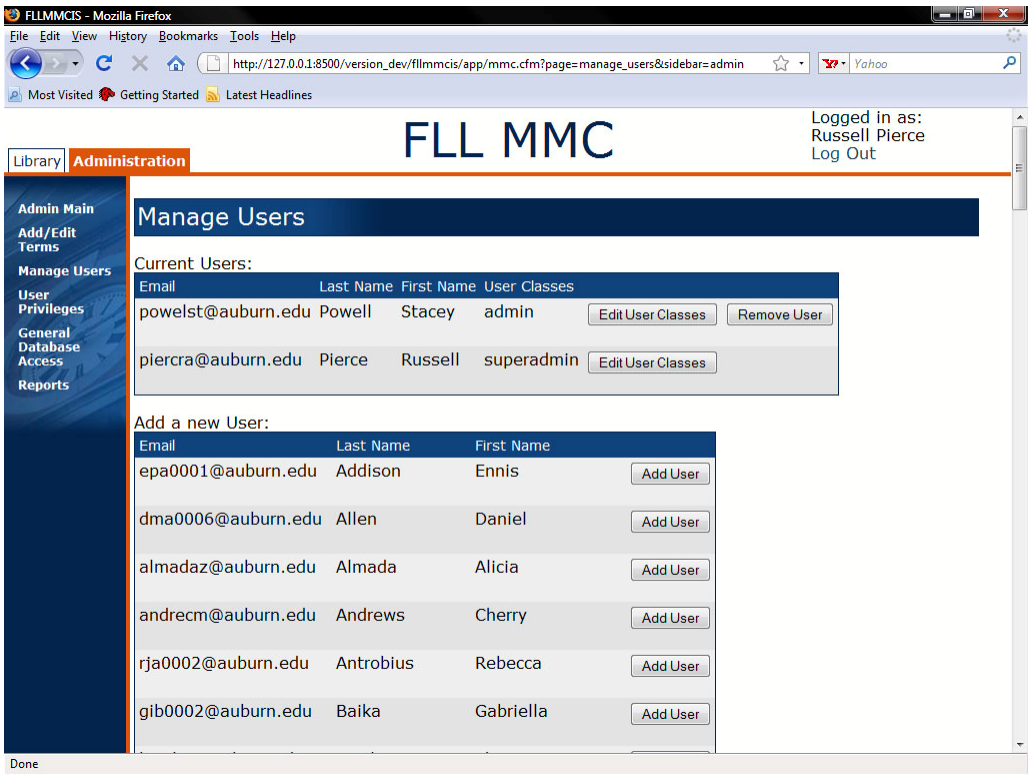

Figure 1.12 Managing Users

1.2.3 Reports

The statistics page provides information on the current status of the inventory and historical usage data. Here users can find out how many items are being borrowed and how

many items are overdue. Two graphs display the all-time usage and the usage over the course of a semester.

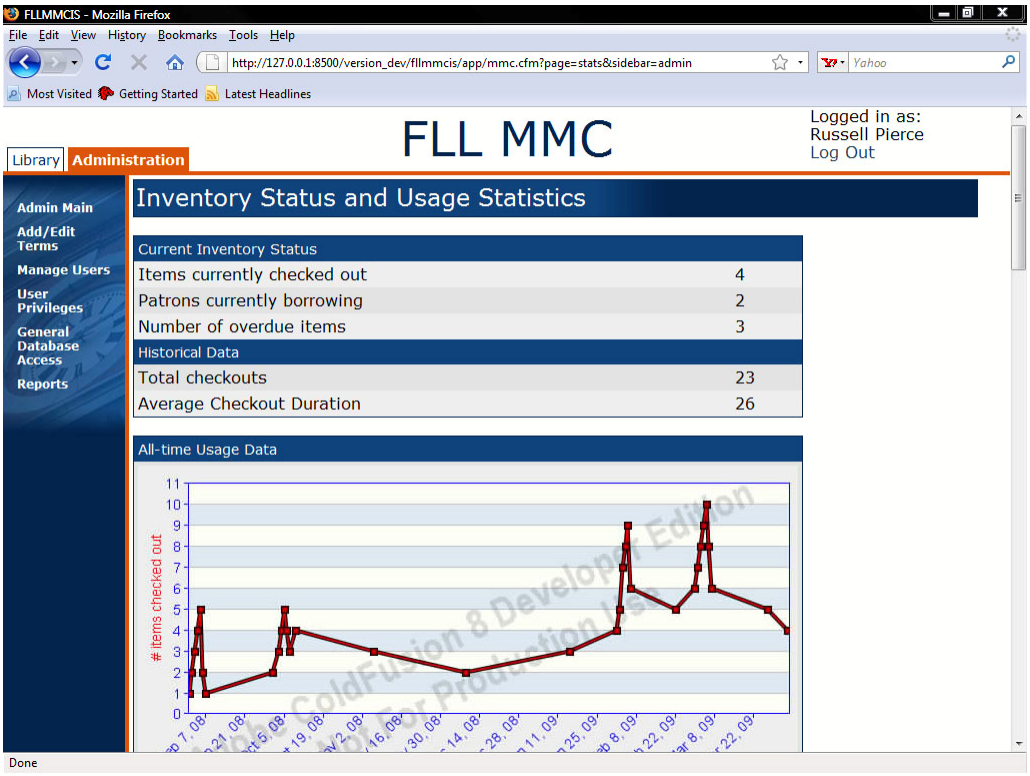

Figure 1.13 Reports

2. Project Details

In this section many details related to the project are discussed. A description of the customer and of project deliverables is offered. Engineering details including requirements collection, modeling, testing, and deployment are also addressed.

2.1 Customer Description

The Multimedia Center (MMC) Director is the primary customer. The MMC is a technology-driven learning facility provided by Department of Foreign Languages and

Literatures (FLL) to FLL students, faculty, and staff. The facility is located in the Haley Center at Auburn University. It consists of a computer lab and a computer-equipped classroom as well as areas dedicated to social language learning. An inventory of media and equipment is lent to faculty, staff, and students to provide relevant materials and technology to language students and educators. This facility is operated by the MMC Director and a staff of student workers. This group is the user-base of the IS.

2.2 Project Deliverables

There are a number of deliverable artifacts for this project. The two primary deliverables are the web site and the database. The web site consists of a directory structure containing ColdFusion files and graphics. This structure is in place on the system web server. The MySQL database exists in complete and functioning form on the MySQL server.

Two documents are provided to support users and maintenance personnel. The user manual provides instructions for operating the IS. Common topics of interest are addressed with examples and step-by-step instructions. Users unfamiliar with the system can reference this document and begin operating the system without formal training. Users should have a working knowledge of MMC policy and be familiar with using a web browser.

Technical documentation is provided as a reference to installers and maintenance providers. This documentation describes the architecture and operation of the system at the technical level. Installation, backup, and restoration instructions are also provided in the technical documentation. This document is intended for web programmers who will service the IS.

Some artifacts from the development process will also be delivered. Among these are the scripts that create and migrate the database and several design artifacts. The database is built by a BASH script. The shell script assembles a MySQL script from a directory structure containing scripts for each table. It then executes the assembled build script which constructs the tables for the database. Another BASH script performs similar actions for migration. A similar directory structure containing migration scripts is assembled and executed to migrate the data from the old database to the new one. These scripts are not of any immediate use to system users or maintenance personnel, but provide a record of and insight to how the system was created.

Design artifacts are provided for maintenance purposes. The requirements document, testing document, the UML object model, and database model are provided in stand-alone digital format. These artifacts may be useful for updating and maintaining the system. They are also included in the technical documentation for reference.

2.3 Requirements Collection and Refinement

The old inventory system is the source for many of the core requirements. In the previous system, items could be added and removed from the inventory, updated, lent, and returned. The previous system was in service for several years, and during that time, many desirable and undesirable attributes were identified. I gained valuable experience with the old system while I worked at the MMC. During that time, I performed business activities using the system and acted as a maintenance programmer. In this regard, I am an expert on the old inventory system and am familiar with its use, strengths, and weaknesses. Through my experience, I identified many shortcomings of the old system. Consequently, many requirements were identified as changes that should be made to the old system.

The MMC director also used the old system frequently. The director formulated a list of requirements for the IS based upon her experience and provided me with that list. The system was conceptualized through an early prototype which was discarded. Through prototyping, further requirements were identified and added to the growing collection. The requirements were expressed in a hierarchical outline and refined incrementally. The MMC director periodically reviewed the requirements offering suggestions, clarifying requirements, and identifying new requirements.

2.4 Modeling and Design

Modeling was an important part of developing the IS. The database model was developed early in the project and was used for reference repeatedly during interface design. The object library originated as a prototype to examine the possibility of producing a reusable subsystem for database interaction. That prototype and the reuse concept were abandoned when the complexity of the subsystem became overwhelming. After attempting a page-by-page approach it became clear that the system produced would have serious maintainability issues. At that point I decided to reconsider a reusable system and began modeling the object library in UML. The object library is of sufficient complexity that I do not believe it could have been produced successfully without the model.

2.5 Testing, Deployment, and Maintenance

Testing was performed on a per-requirement basis. For each requirement, one more tests were written and conducted. These tests and their associated results are formalized in the test

document. This document contains many specific tests that originated from bugs found during the development process. Whenever a fault was discovered that did not correspond to specific functional requirement, the requirements were revised, the problem was corrected, and a test was created to verify the correction.

To deploy the IS, first the user interface was put in place on the web server. Next, the database was constructed. Finally the data was migrated from the current version of the old database. To transition to the IS, users simply stopped using the old system and began using the new one.

The IS will be maintained by Multimedia Center and College of Liberal Arts Information Technology staff members. A technical guide gives an in-depth explanation of system operation and comment-based documentation provides an even finer grade perspective. With these aids, maintenance personnel should be adequately equipped to repair defects or extend/alter the functionality of the system as requirements change.

3. Implementation

The IS has been implemented using a number of technologies. ColdFusion and various web standards were used to develop the user interface and the database has been created in MySQL. In this section, some of the technologies used are introduced, the architecture of the IS is described briefly, and the implementation of the user interface and database are discussed.

3.1 Technology Overview

The IS employs many different technologies and standards. ColdFusion is a relatively new system for creating dynamic web pages. It is a competitor to PHP and ASP. Cascading Style Sheets (CSS) is a standard by which web pages can be structured and styled. CSS offers many advantages over standard HTML. The primary benefit of CSS is that it allows style elements to be separated from content elements. CSS also facilitates reuse of style information. JavaScript is a scripting language that enables extended interaction between users and web pages on a client-side basis. MySQL is a powerful database management system that provides data storage, retrieval, and manipulation. The Lightweight Directory Access Protocol (LDAP) serves as a user authentication protocol for network systems.

3.2 Architecture

The IS is deployed across two platforms: a web server and a database server. The web server is a PC running the Windows Server 2003 operating system and Microsoft's Internet Information Services (IIS) web server. The implementation of the IS is not tied to either this operating system or to the IIS web server. The web server also runs Adobe's ColdFusion 8. The IS is written for ColdFusion 8 specifically and will run on any web server that is compatible with that system. The MySQL database server is run by OIT as a service for official university applications.

3.3 User Interface

The user interface has been implemented in ColdFusion. ColdFusion has several similarities to Java and supports many concepts of Object Oriented design. Inheritance is accomplished via the "extends" keyword, and functionality mimics that of the Java implementation. In the newest version of ColdFusion, the Java construct "interface" is available using the "implements" keyword. These and other similarities to Java afford simple translation of object oriented principles to the domain of web programming.

The IS is composed in part by a collection of ColdFusion scripts. These scripts are executed by the ColdFusion server which produces HTML as output. That HTML is delivered by the IIS web server to the client's browser. The structure of the user interface is realized in a single ColdFusion template. This template is used to produce all of the pages in the IS. It loads the appropriate components which are specified in the Uniform Resource Locator (URL) of the page being accessed.

The various pages of the IS web site are created by ColdFusion components. These components produce the page-specific content of the web site. Some of that content is hard-coded, but the majority of it is dynamic in nature. Dynamic content is produced through a library of functions and a collection of objects that have been written for reuse. These functions and objects are accessible to all of the pages of the IS. They provide database interaction, display generation, and many ancillary operations.

Component pages also use ColdFusion "custom tags." These tags are used for interface elements and are used throughout the site. Custom tags provide a way for application programmers to create reusable elements as individual files. ColdFusion has special syntax for

custom tags. The syntax mimics HTML and allows the programmer to produce ColdFusion code that incorporates neatly with HTML.

```
<cfmodule
template="/fllmmcis/customtags/section_header.cfm">
     Search the Multimedia Library
</cfmodule>
```

Figure 3.1 Calling a Custom Tag

Figure 3.1 is an example of two calls to a custom tag. Both of the bracketed “cfmodule” elements call the custom tag. The opening tag states explicitly the file which should be executed. ColdFusion determines which file should be executed by the closing tag element by parsing the script and pairing matching tags. Figure 3.2 shows the contents of the custom tag being called. The “thisTag” scope is provided by the ColdFusion server automatically. The server sets the “executionMode” variable based upon the context of the call. So, this tag’s execution is conditional upon the context in which it was called.

```
<cfif thisTag.ExecutionMode is 'start'>
     <!--- Start tag processing --->
     <div class="section_header_container">
     <div class="section_header_text">
<cfelse>
     <!--- End tag processing --->
        </div>
        <div class="section_header_gradient">
        </div>
    </div>
</cfif>
```

Figure 3.2 Contents of Custom Tag

Another more powerful construct in ColdFusion is that of the object. There are several reasons why a developer might use ColdFusion objects. Using ColdFusion's session management, an application can have persistent objects associated with each client. This allows development of state-based applications even though HTTP is a stateless protocol. Another reason for using ColdFusion objects is to capitalize on the benefits of Object Oriented Programming (OOP). Even if the objects are not used for state-based applications, OO design can make a complex system manageable. The IS uses a collection of ColdFusion objects to produce database interaction elements for the user interface. These components communicate with the database and produce appropriate data output and interaction controls.

The user interface makes use of some web-related technologies besides ColdFusion. Cascading Style Sheets are the preferred method for styling web sites. Using CSS, the developer can remove style information from application code and place it in its own file. This makes application code more readable and allows the developer to reuse styles. Using CSS, site-wide style changes can be made in only one place: the style sheet.

JavaScript is another technology employed by the IS. JavaScripts are used to provide dynamic client-side elements. For the IS, JavaScripts assist the user in providing valid form input by displaying information about the input field such as how many more characters can be input. JavaScripts also produce interactive elements such as collapsible site components.

3.4 Database

MySQL is an industrial-grade database management system (DBMS). The IS is driven by a MySQL database hosted on an OIT MySQL server. It is defined by a collection of MySQL scripts. These scripts contain MySQL statements that, when executed by the DBMS, create the

tables required by the IS. After the database is created, another collection of MySQL scripts migrate the data from the old database to the new. The database is solely responsible for representing the state of the library and is the only way in which data is stored within the IS.

4. System Composition

The IS can be considered to consist of three major components: the user interface, the object library, and the database. The user interface and object library are closely related. The user interface is specific to the IS while the object library has been designed to exist independently. For this reason, they are addressed separately here.

4.1 User Interface

The organization of the IS web site follows a familiar format. The sections of the web site are accessible via a tabbed navigational bar that runs horizontally just below the page header. A contextual menu relevant to the current section appears below the navigation bar on the left-hand side of the page. These elements are present throughout the site and provide a consistent means for navigating the IS.

The web site is divided into two sections. These sections group related functions. The entry page to each section displays options that correspond to the most frequent activities users perform. The aesthetics of the interface come from the Auburn University web site. Departments have been encouraged to update their sites to look like main university site. In keeping with this initiative, the system has been designed to mimic that design. Beyond simple

aesthetics, interface design activities for this project pertain mostly to the many forms used throughout the system.

Form design is addressed carefully throughout the IS. User-friendly forms can make the difference between a system that is embraced or abandoned. Best practices applied to forms can increase completion rates and overall success. Forms should require the minimum input from the user to achieve the business goal. Consider the forms in figure 4.1. On the left, the traditional solution is a set of radio buttons and a submit button. This form requires both selection and submission actions from the user. With the form the right, the user can make the selection with a single action. Where possible, users can interact with the IS using single actions.

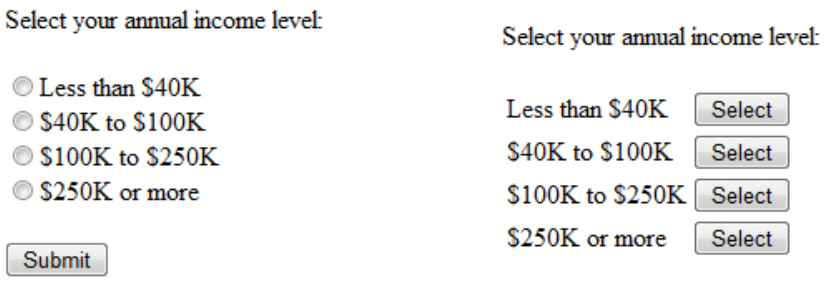

Figure 4.1 Form Interaction Comparison

A well-designed form provides a clear and intuitive path to completion. The user should be able to identify what actions must be taken to complete a form easily. The alignment of field labels and placement of input fields can assist users to that end. In figure 4.2 “Add Item” page uses right-aligned labels and left-aligned fields. This makes the correlation of a label to its field clear. This also produces a straight and vertical path to completion that guides the user through the form to the “Edit Item” action. This form uses minimalist visual elements to separate input elements. A simple horizontal rule provides separation without creating distractions.

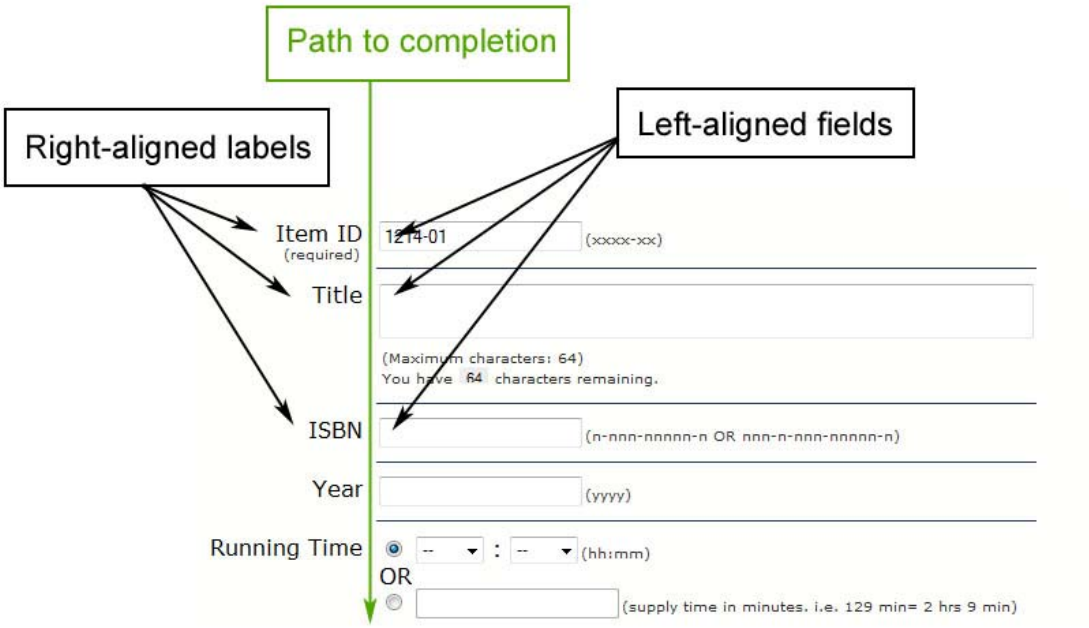

Figure 4.2 Path to Completion

Special attention is paid to addressing errors generated during form submissions. Fields in error in figure 4.3 are redundantly marked so that the problem can be quickly identified and addressed by the user. These fields are marked by a red error message describing the problem, the field labels are change to red, and an alert icon appears by the field label. The alert icon allows colorblind users to identify error fields. This redundant marking reduces user frustration by directing the user's attention toward resolving the errors and completing the form with minimal time and effort [Wroblewski 2008].

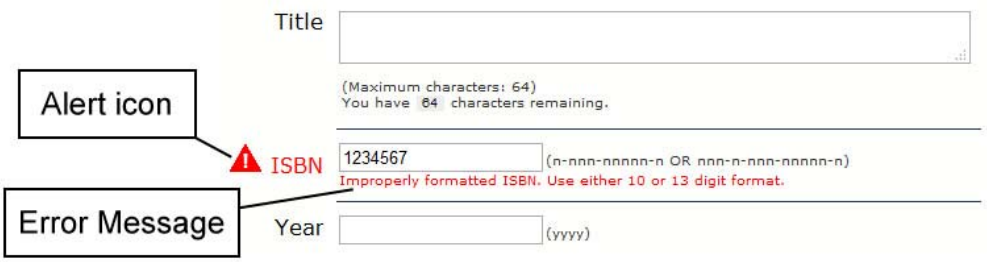

Figure 4.3 Redundantly Marked Errors

Some pages in the IS contain large amounts of textual information. Displaying this information requires considerable screen real estate. For this reason, navigational elements are of modest size. The IS is designed to be viewed at a screen resolution of 800x600 or higher. No specific support for portable or low-resolution devices has been incorporated. Small-screen support is not a requirement for the IS. JavaScript support is required by the website, but no specialized plug-ins are required, and most standard web browsers are supported.

4.2 Database

The MySQL database represents the inventory and is the repository for all IS data. In this section, the database development is discussed in three sections. First the design process is described followed by a discussion of the migration procedure. Finally, the composition of the database is examined.

4.2.1 Database Design

Database design was performed by constructing an Entity-Relation (ER) model. This model was incrementally constructed and refined until it was satisfactorily complete.

Eventually, the ER diagrams were abandoned in favor of a Microsoft Visio database diagram. This diagram is provided for reference in Figure 4.5.

Special attention has been paid to the database design with respect to referential integrity. The use of normal forms can give assurances regarding referential integrity. First Normal Form is guaranteed by the exclusion of set-type values throughout the database. The database has been constructed to adhere as closely as possible to Third Normal Form. In an attempt to approximate Third Normal Form, relations have been decomposed as far as practical, many into trivial relations. In all cases, the attributes of relations are defined by only the key. The database as designed fulfills all related requirements and minimizes redundancy.

The database uses the InnoDB storage engine. This engine supports foreign keys which are used by many of the relations in the database. The use of foreign keys allows much of the business logic to be abstracted from the web application and placed into the database itself. This reduces demands on the application developer and simplifies application code. Consider the case of an update to the Item_Formats table. If a user changes the item format from “Digital Video Disc” to “DVD,” the system should update all items with the format “Digital Video Disc.” Without foreign keys, the application programmer would have to enforce this constraint at the application level. With foreign keys, the appropriate updates are made to all items by the DBMS.

```
CREATE TABLE Items(
…
FOREIGN KEY (format) REFERENCES Item_Formats ( format )
     ON DELETE SET NULL ON UPDATE CASCADE,
…
) Type=InnoDB;
```

Figure 4.4 Foreign Key Constraints in MySQL

Figure 4.1 is part of the create statement for the Items table. The "ON DELETE SET NULL" clause tells the DBMS that if an item format is deleted, all rows in the items table that refer to that format should be set null. Similarly, if a format is updated, the update is performed all items referring to that format. This situation is representative of many such cases. Programming this kind of database maintenance at the application level is tedious, increases code complexity, and potentially introduces errors.

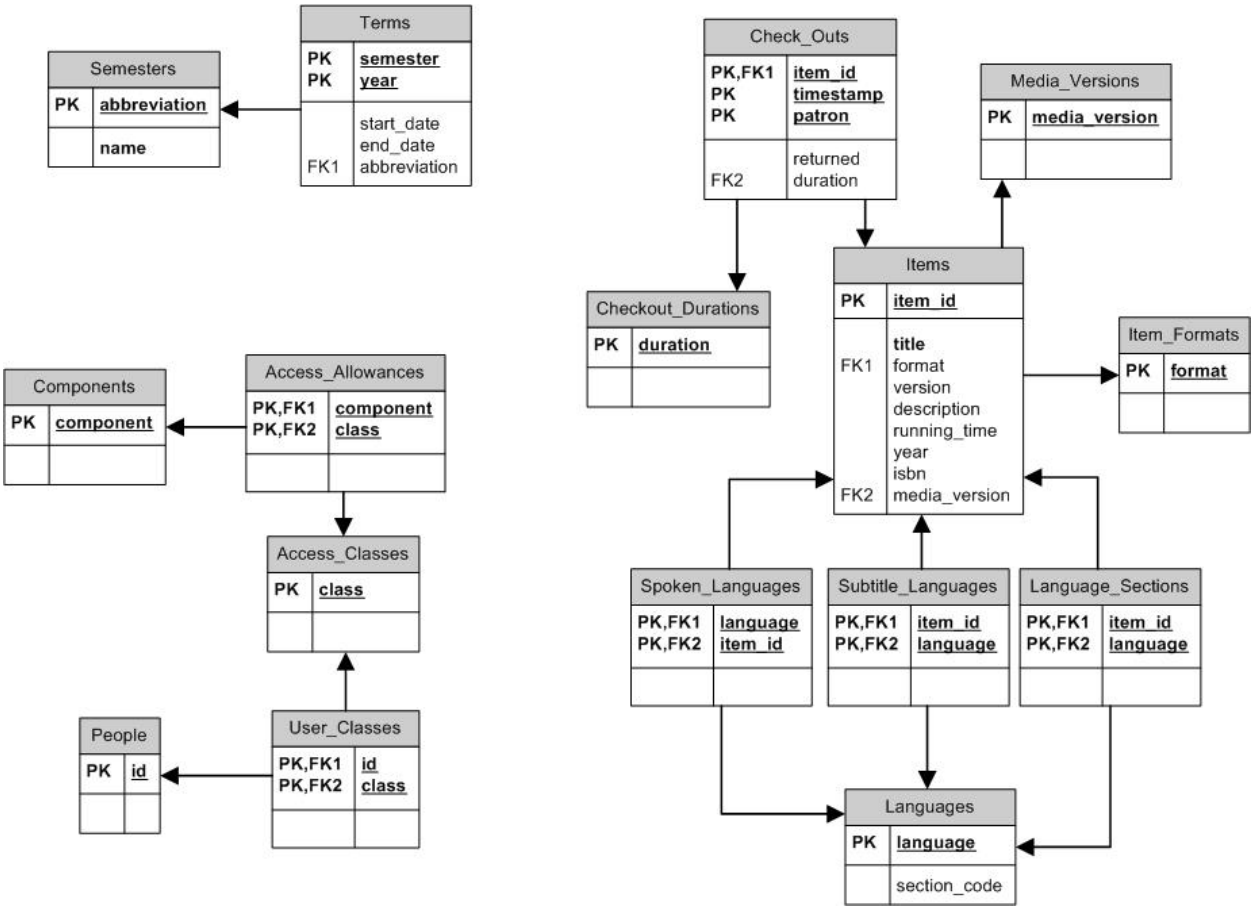

Figure 4.5 Database Diagram

4.2.2 Database Migration

The majority of data which comprises the inventory system was previously stored in another MySQL database. That database was created by the authors of the original inventory system. It did not make use of foreign key constraints. It also contained data which lay outside the scope of the new system. It was necessary to move the data from the old representation into the new representation. This was done with a table-by-table approach. The order in which the migration could occur was dictated by the structure of the new database. Tables with foreign keys had to be created after the tables to which their keys referred. Many of the tables in the old database mapped logically onto their new counterpart tables in a one-to-one fashion. Other tables in the old database mapped as one-to-many into the new database. The old representation of items compressed much of the data pertaining to a given item into a single record. This caused some inherent limitations. Consider the example of a DVD. Many DVDs have multiple audio tracks and multiple subtitle options for various languages. In the old system, there was no facility for representing this information. In the new system, the Spoken_Languages and Subtitle_Langauges tables allow for an arbitrary number of associations between an item and the relevant languages.

4.2.3 Database Tables

The database contains sixteen tables. They can be logically divided into three groups. Authorization and access control tables are used for identifying users to the system and restricting or allowing those users access to the various components. The inventory tables represent and describe the items in the inventory, and the inventory tracking tables are used to determine the status of the inventory.

4.2.3.1 Authorization and Access Control Tables

The Components table contains entries that represent components of the web site. Tuples in the Access_Allowances table indicate which Access Classes are authorized to use website components. Access_Classes are the classes to which system users belong, and User_Classes tuples pair users with classes. The users IDs are stored in the People table. A relatively simple query can test to see if a user is authorized to access a component.

```
SELECT COUNT(*)
FROM Access_Allowances
WHERE component="componentName" AND class= ANY
     (    SELECT class
          FROM User_Classes
          WHERE id="userID"
     )
```

Figure 4.6 Authorization Query

This query will return the number of corresponding Access_Allowances rows that correspond to the given component and user. If the result is greater than zero, the user is of at least one class that is allowed to access the component and will be granted access.

4.2.3.2 Inventory Tables

The Items table contains an entry corresponding to each item in the inventory. Some of the fields of the Items table are foreign keys that reference simple tables, but most fields are text. There are three tables that associate items with languages. They are Spoken_Languages, Subtitle_Languages, and Language_Sections. These tables allow for any number of associations of these types.

4.2.3.1 Inventory Tracking Tables

The Check_Outs table stores records of item checkouts and has two foreign keys. One foreign key relates the checkout to the item and the other references the checkout duration in the Checkout_Durations table.

4.3 Object Library

The object library is a collection of ColdFusion components that is used to produce a uniform method for interacting with database tables. These objects automate the process of producing forms and displays for database input and output. This collection of objects allows any table in the database to be accessed and altered. The objects in this collection are generic. They are not database specific. These objects can be used outside the context of this project to produce database-driven websites rapidly.

There are several classes in the object library. The three core classes of the collection are FormTable, DataElement, and WrappedQuery. Other classes are not discussed here except in relation to these main three.

4.3.1 FormTable

The FormTable class represents the interface object that allows users to interact with database tables. It is the gateway to accessing this collection of objects. To create the interactive tables found in the “General Database Access” section of the system, the programmer need only call one function and supply the target table’s name.

```
<cfset myFormTable = establishFormTable(tableName)>
```

Figure 4.7 Creating a FormTable

The FormTable is created as well as the corresponding WrappedQuery objects and all other required objects as needed. The FormTable generates the display with which the user will interact. It is composed of Column objects, which are composed of DataElement objects.

4.3.2 DataElement

The DataElement object represents the contents of the table cells. DataElement objects are never directly instantiated. Since the class is never instantiated, it might have been created as an interface, which ColdFusion8 supports with Java-like syntax and semantics. However, many descendants of this class benefit from inheriting non-abstract methods. So, DataElement serves as a parent class for the many type-specific classes by which it is extended. One such class is DateElement. This class is created whenever a database table contains a column of the MySQL data type DATE. Other similar classes exist for other MySQL data types. Some DataElement classes produce controls like buttons for the user interface.

4.3.3 WrappedQuery

The WrappedQuery class acts as a wrapper for a ColdFusion query object. It also retrieves and parses MySQL table information. The information derived from "SHOW CREATE TABLE" MySQL queries is parsed to determine table structure and composition. This information is used by the FormTable class to create the necessary elements for the table. Other classes associated with the WrappedQuery class realize database concepts. Within

WrappedQuery, a collection of KeyElement objects represents the MySQL primary key, if present, for the target database table. Similarly, ForeignKeyElement objects identify columns that makeup MySQL foreign key constraints and AggregateElement objects handle foreign keys that are composed of more than one column.

4.3.4 Object Model

Figure 4.8 is a UML class interaction diagram that was produced during development of the object library. It shows the classes in the library and the important relationships between them. The model of these objects was kept simple intentionally. The produced model provides adequate information about the objects. UML models with excessive detail confuse the reader and though they may be more complete, they can be less useful as tools [Arlow 2005].

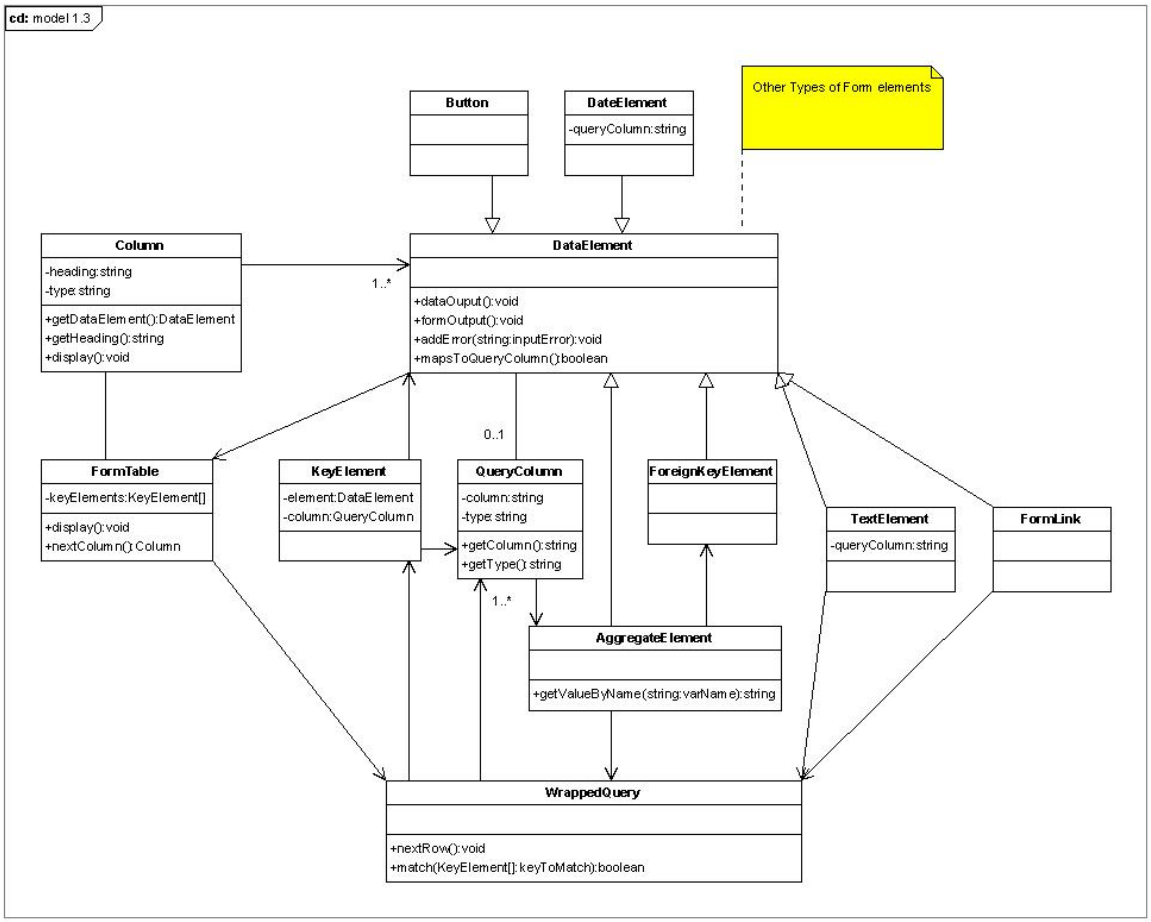

Figure 4.8 UML Model of Object Library

5. System Requirements and Solutions

The IS is replacing an existing system. The primary reason for the replacement is due to a change in available resources. The Office of Information Technology (OIT) plans to discontinue operation of their PHP web sever. The old system was written in PHP and hosted on that server. The College of Liberal Arts has chosen Adobe's ColdFusion as their dynamic web server technology. The redesigned inventory system is written in ColdFusion as it is the only dynamic web environment available to the customer. The use of ColdFusion as a development platform was the first requirement identified for the new system.

In this section, selected system solutions are presented. These solutions are categorized as belonging to five groups: inventory tracking, inventory management, database and system management, business activities, and security.

5.1 Inventory Tracking

The operation of MMC library is similar to a conventional book-lending institution. Materials are given to patrons for a period, with the expectation that they will be returned by a pre-determined date. The status of the library inventory must be maintained so that materials can be accounted for whether they are physically present in the library or not. When a patron borrows from the library, the system must record this event. Similarly, when the items are returned, this must be reflected in the system. In this manner, the status of the inventory is recorded. These behaviors and capabilities are requirements for inventory tracking. These requirements are realized as checkout and check-in functionality.

5.1.1 The Checkout Process

Checking out an item to a patron is one of the most common activities for a user. The checkout system works on the common paradigm of a shopping cart. Items to be checked out are logically "put in the cart" before the checkout action is taken. A checkout event is constituted by recording the fact that an item has been lent to a patron for a duration. The system provides several ways to accomplish a checkout task.

Patrons wishing to borrow items frequently supply the user with a list of item IDs. In this case, the user can easily and quickly accomplish the checkout using the "Add Item to Cart By

ID” page as seen in Figure 5.1. Because it will be used frequently, this page has been kept very simple to maximize efficiency by minimizing mistakes and distractions. The page uses a simple form to allow the addition of items to the checkout cart.

Figure 5.1 Adding an Item by ID

Less frequently, patrons supply the user with other identifying information for the desire items. The patron may give a partial or complete title or a description of the item for which they are looking. In this case, the user must identify the item before it can be added to the cart. To facilitate identification of items, a search feature has been implemented on the “View/Search Library” page. A text search allows users to look for items matching the patron-supplied criteria. The text search attempts to match against item numbers, titles, and descriptions. Usually, the title is known. This being the case, a simple text search usually reveals the item in question, and more elaborate searches requiring changes to the search criteria are unnecessary. Considering this, the search and display options are hidden by default as in Figure 1.3. This frees screen real estate and reduces distractions for the user [Wroblewski 2008]. Should the user need to customize the search, many useful options are offered as seen in in Figure 5.2. The output from the search can be customized so that particular attributes are displayed or not. Results can be sorted by any field in ascending or descending order, and the number of results per page can be adjusted. These options can assist a user in identifying an item when limited information is available.

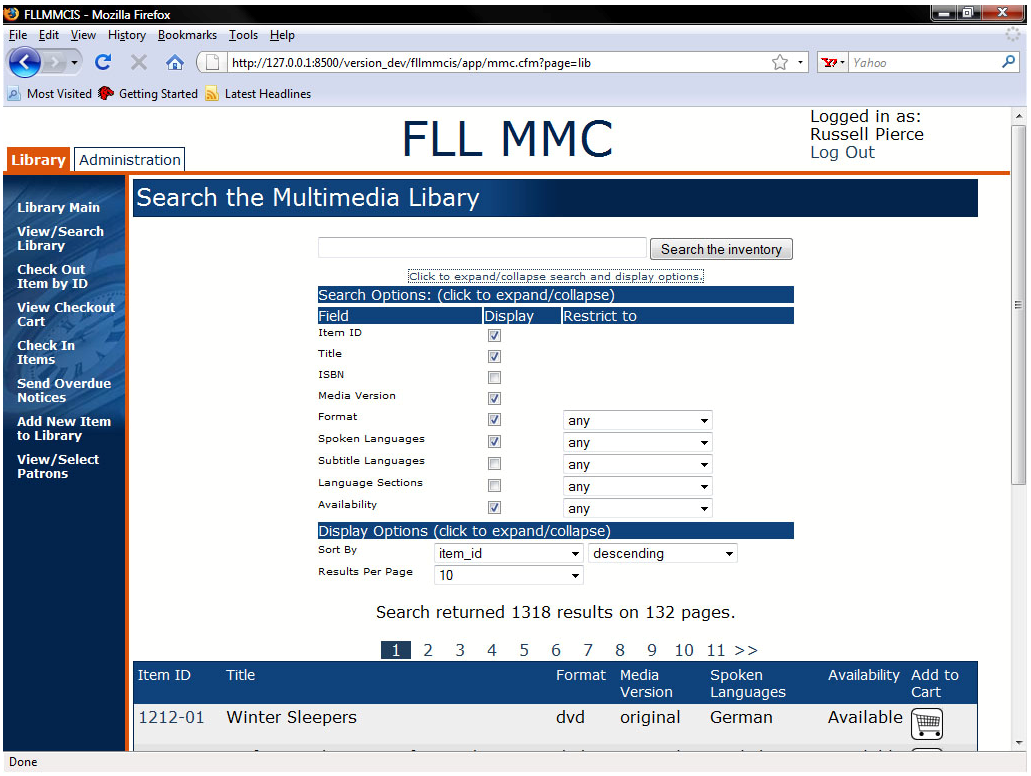

Figure 5.2 Expanded Search Options

Prior to checkout, the user must identify the patron. To do this, the user selects the patron using the "View/Select Patrons" page shown in Figure 1.4. On this page, patrons are listed alphabetically. A patron is selected by clicking the corresponding "Select" button. The patron can be selected at any time during the checkout process, but must be selected before a checkout can occur.

Most frequently, the user will add the desired items to the checkout cart before selecting the patron. On the "View Checkout Cart" page, a status message indicates whether a patron has been selected and identifies the selected patron if able. If no patron is selected or the selected patron is not correct, the user can use the provided action "Select a Patron" to navigate to the "View/Select Patrons" page. If the user takes this action, upon selecting a patron, the user is returned to the checkout cart.

Before completing the checkout the user should select the duration of the checkout. By default, checkout durations are until the end of the current semester. Due to MMC checkout policy, this is by far the most common duration. This use of a "smart default" as seen in Figure 1.5 should reduce user effort in the majority of cases [Wroblewski 2008].

After the items, patron, and duration are chosen, the user takes the "Check Out" action to complete the checkout. Appropriate entries are added to the Check_Outs table, the checkout cart is emptied, and the patron is deselected. The system is then in an ideal state to handle another checkout process.

The checkout process is modeled by an informal process flow chart in Figure 5.3. This model helps clarify the process and aids in the design of the interface. Only user actions and not system states are modeled. Initially, only high-level user actions were modeled. Through refinement, the precise actions required by the user were discovered. These actions correlate to

specific system requirements. Understanding how the user will interact with the system provides insight into how to create a user interface that meets those requirements.

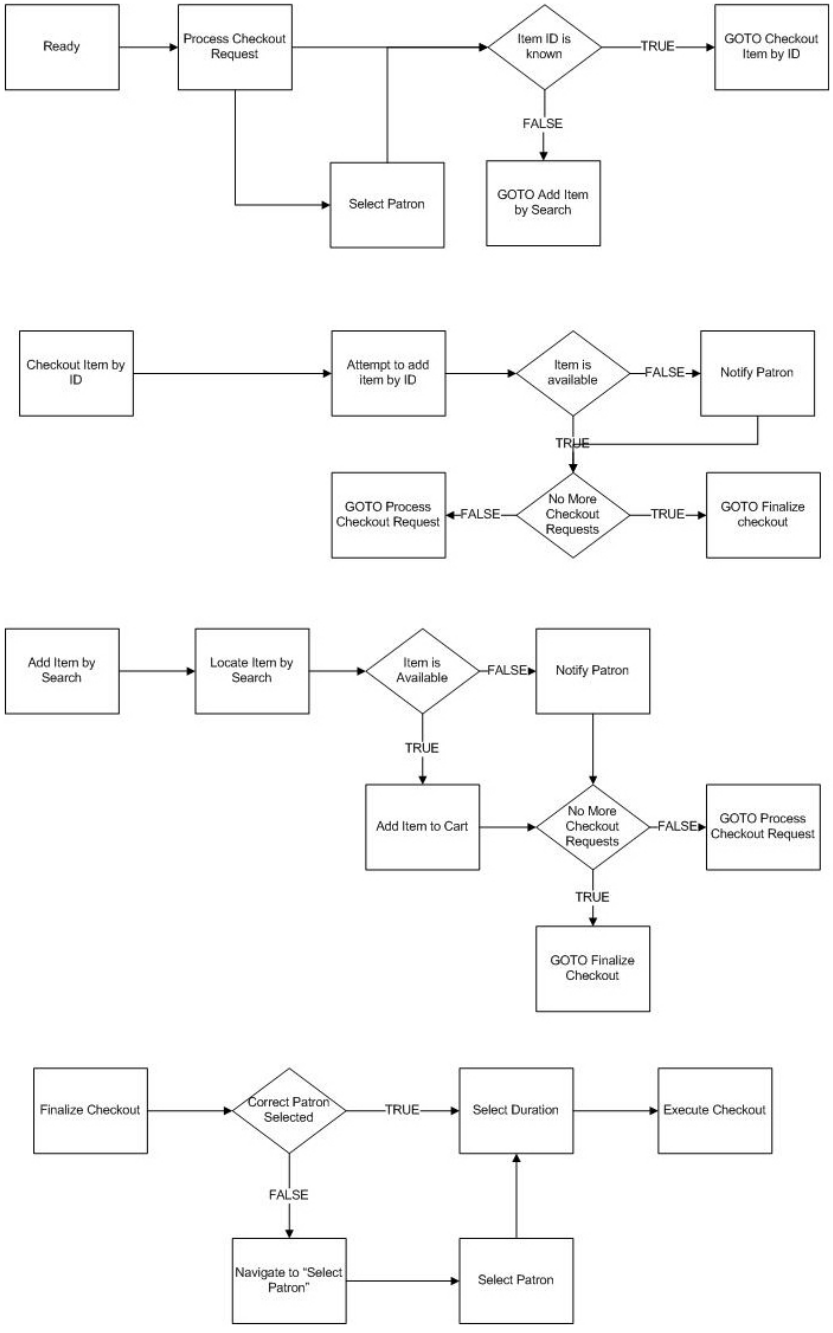

Figure 5.3 Process Flow Chart

5.1.2 Check-ins

The second most frequent activity for IS users is checking in items. The "Check In Items" page seen in Figure 1.7 allows users to indicate to the IS that an item has been returned. On this page, the user can see all of the items that are currently checked out. The item id, title, format, and item availability are provided to assist users in locating the item to be checked in from the list. To check in items, the user selects the checkboxes corresponding to the items and chooses the "Check In Marked Items" action. The appropriate entries in the Check_Outs table corresponding to these items are updated, indicating that the items have been checked in.

5.2 Inventory Management

A core requirement for the IS is the ability to manage a dynamic inventory. Changes to the inventory are frequent and should require minimal effort. Users must be able to add, remove, and update items to reflect an accurate inventory.

5.2.1 Adding/Editing Items

The most common inventory management activity is adding items to the inventory. Items are added one at a time through the "Add Item to Library" page. The form on this page has been designed using best practices from industry experts. The number of fields is substantial, but does not warrant a multi-page solution. All of the fields ask questions that must be answered, if they apply. Right-aligned labels and left-aligned input fields provide a clear path to completion. Help text is provided where appropriate to assist the user in providing proper inputs. For the large text field inputs, the limitations of length are shown as well as how many more characters can be supplied. When the available space in one of the fields is exhausted, the

counter turns red. This draws the attention of the user to the counter. This helps the user understand why they cannot supply additional input and reduces user frustration. The “Running Time” field provides users with two ways to supply this input. Users may provide the time in hours and minutes or in minutes only. This capability removes the tedious task of manually converting a time in one format to another. It also removes the chance of an error introduced through conversion by the user. In the old inventory system, only one spoken language, one subtitle language, and one language section could be associated with an item. This limitation was imposed by the database structure used in that system. This was not a significant problem before DVDs became common. The DVD introduced multiple language and subtitle options, and the new database and system allow for any number of language associations for a given item.

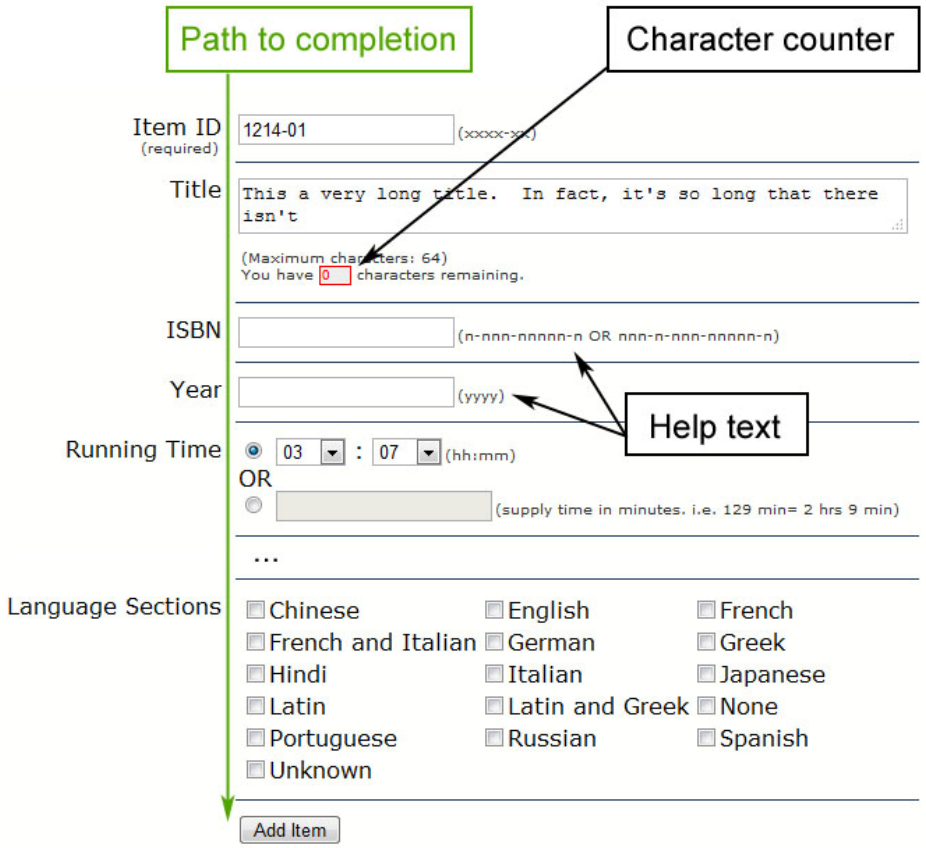

Figure 5.4 Examination of Add/Edit Item Page

Items can be edited by clicking the “Edit this Item” action under the item detail view. The “Edit Item” page uses the same ColdFusion component as the “Add Item to Library Page.” Using the same form for both adding and editing items means that users perform familiar actions to accomplish either task.

When adding or editing items, extensive input validation is performed. The “Item ID,” “ISBN,” and “Year” fields use regular expression matching for validation. When errors occur, rather than receiving a completion message, users are returned to the form.

5.2.2 Deleting Items

Rarely, it is necessary to delete an item. Users can delete an item from its "Item Detail" page. The "Delete this Item" action brings users to a confirmation screen version of the item detail page see in Figure 1.10. The user is asked if they want to delete the displayed item and warned that the operation is not reversible. This form contains two actions. The primary action is to delete the item, and the secondary action is to cancel the operation. Normally, the primary action would be aligned in the vertical, left-hand path to completion. However, since accidental deletions are of greater concern than failed deletions, the primary action is moved off the path to completion, forcing users to take longer and positively confirm the action.

5.3 Database and System Management

The user interface exists to provide a way for users to interact with the database. Those interactions realize the main goals of the system. However, there are other requirements that are fulfilled through database interaction. These requirements are related to maintaining the system, rather than supporting business activities. Almost every table in the database may require alterations and the user must be able to accomplish this through the user interface. These activities are called database management. For the IS, system management refers to adding or removing users from the system and setting user classes for access control.

5.3.1 General Database Access

One shortcoming of the previous inventory system was the general inaccessibility of database tables. Only a few tables could be altered through the web interface. This meant that if

such updates were required, a user with direct database access and a working knowledge of MySQL was required to perform a simple update. One example is adding a language. On repeated occasions, materials would be added to the database, but their corresponding languages could not be found in the languages table. Non-technical staff could not make the required addition to the table, and the task would remain incomplete until the request could be submitted to a technical staff person who could perform the required database update.

The ability of non-technical staff to make unforeseen updates to the database is even more critical now. The MMC no longer employs technical staff and instead relies upon College of Liberal Arts staff for technical assistance and maintenance. The new system provides a uniform method for database access. Any table can be added to, deleted from, or updated through the General Database Access section of the IS Administration area.

5.3.2 Managing Users

The MMC staff changes frequently. Students work at the MMC on a semesterly basis, so changes to the IS user base are common. While this requirement could be fulfilled using the general database access functionality, a specialized interface makes the task easier. The "Manage Users" page seen in Figure 1.12 provides an interface through which users can be added to or removed from the system. This page also allows assignment of user classes. When users are added to the system, they have no user class. One or more user classes must be assigned so that the new user can access the various parts of the IS. Since these operations are so closely related, it is convenient for the user to find both on the same page.

5.4 Business Activities and Requirements

There are two business related requirements for the IS. The IS must allows users to send overdue notices and produce reports useful for making business decisions.

5.4.1 Overdue Notices

MMC library policy dictates that patrons who have failed to return borrowed items in the time allotted should be notified via email of the situation and prompted to return those items. The IS provides a simple way to send overdue notices. Users can elect to send notices to any or all patrons, and the notification emails are generated automatically with pertinent information.

Before overdue notices can be sent, the system must determine what items are overdue. It is of particular significance that the due dates of items are not stored in the database. Due dates are calculated by the system when needed based upon the checkout_date and duration. For checkouts with a duration of two weeks, a MySQL query calculates the due date using the MySQL function ADD_DATE. For checkouts lasting until the end of semester, the IS determines the Term in which the item was lent and returns the end_date of that Term. The advantage to this strategy is that if the end of a term need to be adjusted, the due date of the item will automatically reflect that change. Alternatively, the due date could be calculated at checkout time and stored in the checkout record. However, since most checkout durations are until the end of the semester, that date would be stored redundantly for each checkout of this type. Calculating the due date avoids this redundancy.

5.4.2 Reporting

One capability not realized by the previous system is the ability to generate reports relating to the inventory status or from historical usage data. The customer requires that reports, comprised of compiled statistics and figures, be produced on-demand. Such reports can be used to provide metrics for success, and such metrics may assist in the process of making business decisions regarding the MMC. The IS produces reports in the form of charts and figures that are collected and calculated from database records. Graphs of lending activity are generated for all-time usage and on a semesterly basis. The IS also produces charts displaying the composition of the inventory.

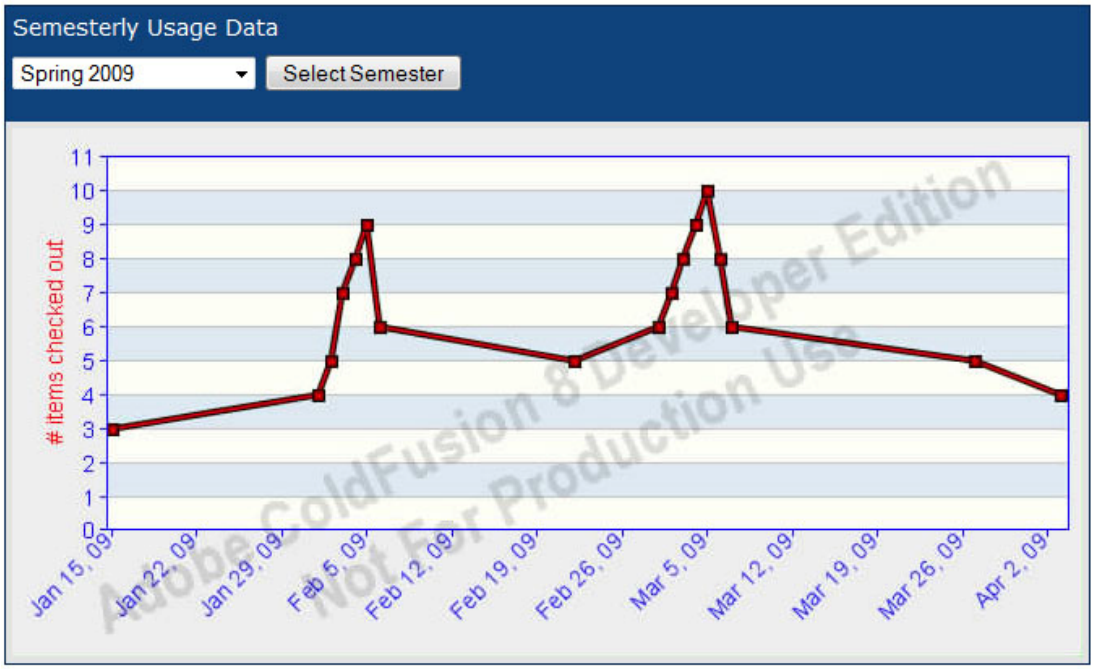

Figure 5.5 Graph of Usage by Semester

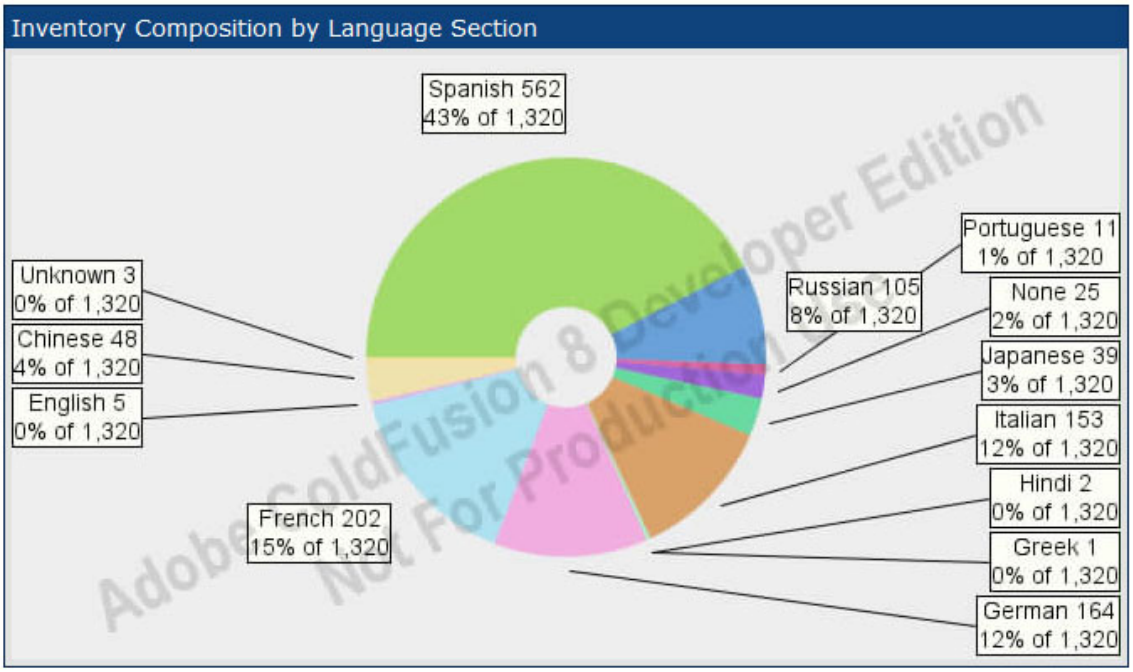

Figure 5.6 Library Composition by Language Section

5.5.2 Security

The two main security issues for the IS are authenticating users and controlling user access. User authentication is the process of verifying the claimed identity of a user. Access control refers to restricting users to particular system functions.

5.5.1 User Authentication

Users are authenticated against the LDAP service provided by OIT. The Lightweight Directory Access Protocol (LDAP) is an authentication system provided by OIT. Auburn students, faculty, and staff are issued a username and password that can be used across many applications for authentication purposes. The use of LDAP is highly valuable to this project. By using this service, the system offloads the responsibility of username and password handling to

OIT. In doing so, the system is not required to handle lost usernames and passwords or other related issues. This translates to reduced requirements on MMC staff and reduced complexity in the system. Furthermore, since the usernames and passwords are not store within the system, the level of information sensitivity is minimized. Another substantial benefit is that users do not have to remember a separate username and password to access this system.

The Auburn Office of Information Technology (OIT) policy requires that any application receiving Auburn usernames and passwords use a secure method to protect login information. ColdFusion supports secure LDAP which satisfies this requirement [Active Directory Authentication]. The use of secure LDAP protects login information transmitted from the web server to the LDAP server. To protect login information sent from the client's browser to the web server HTTPS is used. HTTPS is supported by the Internet Information Services (IIS) web server. This protocol uses the public key encryption protocol Secure Sockets Layer (SSL). This protocol is widely used throughout the web for secure transactions. The only sensitive data handled by the system are usernames and passwords. The database does not store any sensitive information. The security guidelines in [Lee E.] have been used extensively as a guide to the creation of the user authentication system.

5.5.2 Access control

A rudimentary access control system allows the customer to define which functions of the site are accessible to different groups. Initially, the access control system was to be more complex and subtle. This was necessary when the user base would include all FLL department faculty and teaching graduate students. Since the user-base was reduced to only MMC employees, this system was simplified. With the current access-control system, user roles can be

defined and access can be granted to system functions on a per-page basis. It will sometimes be the case that an MMC employee is temporary and requires access to only the inventory maintenance functions. To reduce the chance for mistakes and prevent malicious behavior, that employee can be granted limited access.

Access to a particular page by a given class is defined in the Access_Classes table. The presence of a tuple in this table indicates that users of that class are allowed to access the page. There is no restriction on the number of roles a user many have. If any role held by a user is allowed to access a given page, then that user will be granted access. A useful capability of this system is that a user's access may be increased temporarily by simply adding an additional class with the appropriate authorization. Tuples from the Access_Allowances table are displayed on the "User Privileges" page shown in Figure 5.7. Here, access to system components can be granted or denied to the various user classes.

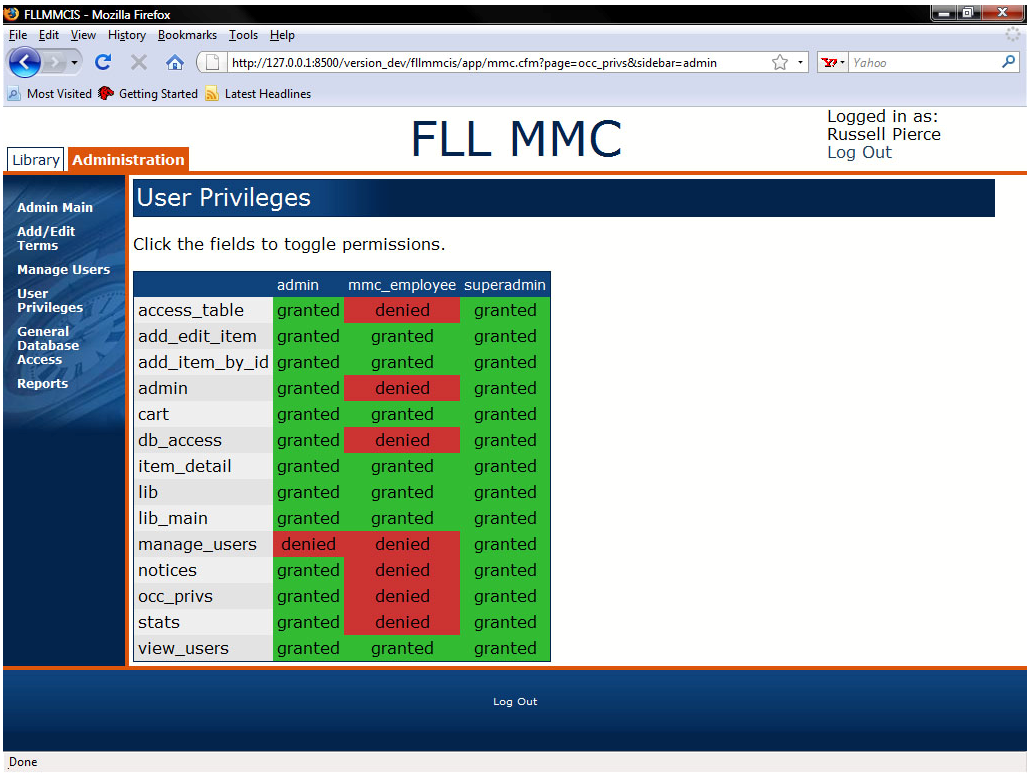

Figure 5.7 User Privileges

It should be noted that information about users is not stored in the IS. Instead, the IS accesses a department database that contains information on employees and graduate students. That access is via Web Service. This capability is a great advantage over the old system. Before, MMC employees maintained the database of user information. The burden of that maintenance that has been removed from IS users with the new system.

6. Technical Challenges

I gained some experience will ColdFusion while working at the MMC. I acted as a maintenance programmer for the department and MMC website. These site are mostly static and do not leverage the many advanced feature of ColdFusion. There are many subtleties to the

ColdFusion environment that, if not understood, can create serious barriers for a programmer. One unusual property of ColdFusion is related to variable scoping. ColdFusion will attempt to "find" variables that are accessed without a scope. Variables declared without a scope exist in the local scope. Variables accessed without scope are first looked for in the local scope. If the variable is not found, ColdFusion will look through six additional scopes in a fixed order. If a variable with the same name is found, ColdFusion will use that variable. This leads to unusual system behavior or errors. Consequently, it is highly recommended that variables be explicitly scoped in general.

Some of the challenges of this project are related to the server environment used. The web server on which the user interface resides is shared amongst all departments in the College of Liberal Arts. Due to this shared environment, developers are not allowed access to the administrative features of the ColdFusion server. Unfortunately, this means that some powerful and desirable features of ColdFusion are not available, and creative solutions must be found to deal with this limitation.

The ColdFusion "custom tag" construct can be very useful. However, due to the semantics of this construct, special considerations should be taken. There are three ways in which a custom tag can be called. Assuming the custom tag is stored in the file "./myTagDirectory/myTag.cfm," this tag may be called by any of the following methods.

```
<cf_myTag>
     This is bad style!
</cf_myTag>
```

Figure 6.1 The "cf_" Method

```
<cfimport prefix=”myTagCollection” taglib=”myTagDirectory”>
…
<myTagCollection:myTag>
     This is OK, but not preferred.
</myTagCollection:myTag>
```

Figure 6.2 The Import and Prefix Method

```
<cfmodule template=”./myTagDirectory/myTag.cfm”>
     This is good style.
</cfmodule>
```

Figure 6.3 The cfmodule Method

Custom tags exist as ColdFusion template files. When a custom tag is called ColdFusion must determine file the programmer intends to execute. Using the “cf_” method shown in Figure 6.1, ColdFusion will look in the current directory, then in any server-configured custom tag directories, and then in the main ColdFusion custom tag directory a tag file with the supplied name. This searching behavior can cause problems for developers. Consider the case that the developer cannot create custom tag directories and cannot access the main custom tag directory. This is often the case, when there is one web server for many different sites. If this is the case, the custom tag must be located in the current working directory. Often, the developer may want to access the custom tag from any page on the site. Either all pages that use the tag must be in the same directory or another method must be used. Server-configured custom tag directories could provide a solution, but they are not available due to the shared-server environment.

The import and prefix method shown in Figure 6.2 allows the programmer to specify the directory of the desired tag. The drawback is that the custom tag directory must be imported in every page where the tag is accessed. It would be preferable if the custom tags were available site-wide without having to repeat the same import statement in many places. The “cf_module”

method is explicit and does not require the import statement as seen in Figure 6.3. Many ColdFusion developers recommend using this method exclusively. I discovered the shortcomings of the other methods before learning about the benefits of “cf_module.”

ColdFusion has some unusual behaviors regarding directories and files. The <cfinclude template=”path/file.cfm”> tag allows a programmer to execute a ColdFusion template from within a ColdFusion template. The behavior is similar to PHP’s include function. In ColdFusion the path specified must be a relative path. This means that different levels within a site’s directory structure, the specified path will be different. There are two ways in which this can be avoided. The ColdFusion server can be configured with custom directory mappings, or the developer can establish directory mappings at run-time. These directory mappings are server-wide and are a shared resource for all developers on a server. So, if two developers would like to create a mapping called “includes,” only one may do so. These directory mappings are a poor choice for multi-site web servers when many developers are involved. In the case of the IS, these mappings were not available or practical.

Runtime mappings can be established on a per-site basis and do not require access to ColdFusion administrative functions. These mappings provide developers with a uniform way by which paths can be specified within ColdFusion. Run-time mappings are new to ColdFusion 8. They are not well documented and developers are just learning how useful they can be.

7 Conclusion

This project has considerable value as a starting point for a stand-alone database interface package. The object library could be elaborated upon to produce a generic database interaction system. With such a system, new database-driven websites could be constructed easily and

rapidly. To extend the object library into a generic tool, several issues would need to be addressed.

This system does not support all MySQL data types. Only data types used in this project are currently supported. Completing support for remaining types would be necessary to create a versatile tool. Also, scalability has not been considered. The system is adequate for its intended purpose and work done towards scalability would not likely benefit the customer. To produce a generic tool, scalability concerns should be addressed. In particular, database and query optimization should be considered carefully if the tool is to be used for larger systems with more complex databases.

Dynamic websites are becoming the norm on the World Wide Web. Systems like the IS are in high-demand. While it may appear to be a relatively simple tool, the subtleties of developing such a system are paramount. Good database design is crucial for creating systems that can accurately model business semantics. Reuse is of primary concern for this kind of development as well. If reuse is not employed, systems like the IS would contain large amounts of redundant code resulting in a system that is difficult to maintain.

The IS project has provided an opportunity for me to produce a fielded software product. I now have experience dealing with a dynamic set of requirements. The term “feature creep” is used to describe the phenomena of an ever-expanding requirements set. The temptation as a developer to try to incorporate features that go beyond the core of the required functionality is considerable. Often it seems as though adding a new feature will require only a small amount of effort in addition to a system that is already of considerable scale. However, since estimation is a major problem for software projects, it is important to concentrate on developing the simplest acceptable solution before elaborating upon it. Failure to control “feature creep” can be costly.

The models produced during this project have been of unexpectedly high value. Even though I am the only developer, I've found that I can sometimes forget how parts of it work. Without models for reference, I would have to spend a considerable amount of time to again familiarize myself with parts of the system that I had not worked on in some time. All of the time spent on modeling during this project was well spent.

I've gained considerable experience through this project. It has been an opportunity to apply the skills I have acquired during my education to produce a product with real world value. A working system could have been produced with less effort, but my goal has been to produce a product that can be maintained and provide years of useful service. To achieve that goal, much time and effort have been invested in reuse. The result is a system that has been carefully designed and implemented with maintenance in mind.

References

Wroblewski, L. [2008], *Web Form Design*, Louis Rosenfeld, Brooklyn

Arlow, J. and Neustadt, I. [2005], *UML 2 and the Unified Process*, Addison Wesley, Boston.

The Office of the Executive Director, OIT, 8 Dec. 2005. *Active Directory Authentication* <http://www.auburn.edu/oit/it_policies/active_directory_authentication_policy.php>

Lee E. Melven I., and Sargent S. [2007] *ColdFusion 8 developer security guidelines*, Adobe System Incorporated. 10 Nov. 2008 <http://www.adobe.com/devnet/coldfusion/articles/dev_security/coldfusion_security_cf8.pdf>