An Intramural Sport Management System

A Manuscript
Submitted to
the Department of Computer Science
and the Faculty of the
University of Wisconsin-La Crosse
La Crosse, Wisconsin

by
Christopher M. Johnson
in Partial Fulfillment of the
Requirements for the Degree of
Master of Software Engineering
May, 2004
UNIVERSITY OF WISCONSIN-LA CROSSE
La Crosse, Wisconsin

COLLEGE OF SCIENCE AND ALLIED HEALTH

Candidate: Christopher Michael Johnson

We recommend acceptance of this manuscript to the College of Science and Allied Health in partial fulfillment of this candidate’s requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. J. Tillman
Oral Examination Committee Chairperson

Dr. D. Riley, Examination Committee Member

Dr. K. Periyasamy, Advisor

Dr. K. Hunt, Advisor

Dr. G. Tymeson,
Director, University Graduate Studies

April 18, 2004
Date

April 28, 2004
Date

April 28, 2004
Date

5-18-04
Date
ABSTRACT

In today's world, software development can be complex and highly competitive. Several factors add to the complexity, particularly if the software has a web-based user interface and is a distributed application. In addition to the complexities involved during development, additional complexities arise in maintenance. For example, the software product may need to be modified to match with changing technology such as newer versions of operating systems and hardware. Software developers are thus forced to choose the right development approach while designing and implementing the software. This document describes the development of a web-based intramural sport management system. The product will be used by the Recreational Sports Department at the University of Wisconsin-La Crosse and hence has some features specific to the Recreational Sports Department. However, the product is easily extendible and modifiable in order to suit other sports management activities. This report describes the various activities in the life cycle of this software. It also includes the challenges faced during the development, the current status, and future work on the software.
ACKNOWLEDGEMENTS

I would like to thank my project advisors Dr. Kasi Periyasamy and Dr. Kenny Hunt for their profound wisdom, insight, and guidance. I would also like to thank my project sponsor, the University of Wisconsin-La Crosse Recreational Sports Department, for their initiative in formulating this project and their support during the project. I would also like to express my thanks to the Computer Science Department and the University of Wisconsin-La Crosse for providing the computing environment and necessary technologies to complete my project. I would also like to express my gratitude to the Saint Mary's University of Minnesota community for providing me the opportunity to pursue this degree. Finally, I want to express my appreciation to my wife and kids for their patience, understanding, and encouragement during the tenure of this project.
# TABLE OF CONTENTS

1. Background Information ........................................................................................................... 1

2. A Brief Introduction to Software Life Cycle ................................................................................. 4

3. The Development of IMSMS ....................................................................................................... 8

   3.1. Requirements Gathering, Analysis, and Specification ....................................................... 8

   3.2. Design .................................................................................................................................. 8

   3.3. Implementation, Testing, and Integration .............................................................................. 9

4. The IMSMS Application ............................................................................................................... 10

   4.1. General Architecture of IMSMS .......................................................................................... 10

   4.2. User Classifications ............................................................................................................. 11

   4.3. Detailed Architecture of IMSMS ......................................................................................... 12

   4.4. Database Design of IMSMS ................................................................................................ 17

   4.5. Graphical User Interface of IMSMS ................................................................................... 19

   4.6. Deploying IMSMS ............................................................................................................... 20

   4.7. Design Decisions of IMSMS ............................................................................................... 21

5. Limitations .................................................................................................................................. 24

6. Continuing Work ......................................................................................................................... 28

7. Conclusion ................................................................................................................................... 30

8. Bibliography ................................................................................................................................. 32

9. Appendices

   A. Sample IMSMS Screen Shots ............................................................................................. 33

   B. Sample IMSMS Web Interaction Maps ............................................................................... 45

   C. Sample IMSMS Class Diagrams ......................................................................................... 48
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1. The general architecture of IMSMS</td>
<td>10</td>
</tr>
<tr>
<td>4.2. Modules of IMSMS</td>
<td>12</td>
</tr>
<tr>
<td>4.3. Detailed architecture of the administrative module</td>
<td>14</td>
</tr>
<tr>
<td>4.4. Interaction map for sports administration</td>
<td>15</td>
</tr>
<tr>
<td>4.5. Database entity-relationship model</td>
<td>18</td>
</tr>
<tr>
<td>4.6. Create league JSP</td>
<td>20</td>
</tr>
</tbody>
</table>
GLOSSARY

ASP
Active Server Pages. A web technology developed and supported by Microsoft Corporation.

API
Application Program Interface. An interface whereby an application can access services provided by lower level modules such as an operating system or virtual machine.

HTML
Hypertext Markup Language. A markup language used to format text and link documents that are commonly presented on the World Wide Web.

IMSMS
Intramural Sport Management System. The web based application that supports the administration of an intramural sports program.

JavaScript
A World Wide Web scripting language developed by Netscape and allows for data passing between Java and JavaScript.

JSP
Java Server Pages. A web based technology that supports the development of dynamic web pages.
PHP
A World Wide Web scripting language.

SMTP
Simple Mail Transfer Protocol. A protocol established to define how to structure and transfer email between computers and servers.

Verification
A process that confirms a development process or activity or task to be correct.

Validation
A process that confirms that the product (or partial product) meets the expectations.
1. Background Information

Management of an intramural sport program on a college or university campus can be a daunting task irrespective of whether the institution accommodates a small student body or a large population. Administrators of such a program need to manage not just the sports activities, but also the teams and athletes that participate in the various sports as well as maintain statistics that are related to the program. In addition, coordinating the scheduling of contests, facilities, and officials as well as manipulating the large amount of data in various logical formats becomes an overwhelming task.

A typical intramural program includes a set of sports offered at various levels of competition; participatory, co-recreational, and competitive. These are further classified into several seasons of play that define the beginning and ending dates of various competitions; Fall, Winter, Spring, and Summer or possibly by semester or other designation. Often, a given sport offered during a specific season is called a league in which teams register to participate. Facility scheduling is another additional aspect to be considered to schedule a sport.

Outside of sports, seasons, leagues, contests, and facilities, an intramural department must coordinate and manage the teams and participants. An intramural department must know which athletes belong to what teams and the name of the team captain. Often, communications may flow from the intramural department to the team captains and then from the team captain to the athletes on the team.

There are two approaches that can be taken to manage an intramural sports program: do it manually using pen and paper or integrate technologies that support management's needs. The first approach requires manually recording all information with regard to all data and manually creating the contest schedules, coordinating
facility usage, and hand-registering athletes and teams. The dissemination of information would require that documents be typed, photocopied, and posted in mail to the participating individuals.

The second approach is the integration of software technologies to help manage the large amounts of data. While technologies that support sport management do exist in today’s market, the functionalities each technology supports can be limited and may vary considerably. Some products provide useful facility management tools but lack the ability to work with teams, while others tend to work well with teams but not with facilities. Other products may provide the necessary tools for both team and facility management but may be a stand alone application loaded on a specific machine. For example, the All-Pro League Scheduler 4.0 software developed by All-Pro Sports Software provides functionalities for scheduling competitions within a league and printing schedule reports, but does not provide functionalities for online registration of teams and players within a league [2]. Because the All-Pro Sports Software does not provide the online registration of teams and athletes, a department could incorporate the Active University product by Active.com to provide those functionalities [1]. There are products that do incorporate the desired functionalities, but are extremely complex or may present information in an unclear manner. For example, Recreational Solutions has developed several IMTrack modules, that when integrated, provide online league registration, contest scheduling, and facility management [9]. However, this solution utilizes several different products using different interfaces (stand-alone, web, and instant messenger based), which can cause confusion. Furthermore, the IMTrack solution may not present data in a desired format. For example, when IMTrack presents the contest schedules for a league, it does not print the team name, but instead prints the team ID number. This may not be useful to users of the system who do not know a team’s ID number.

When it comes to the management of a full intramural system, it may require the integration of several technologies where each technology provides a set of functionalities. These technologies are assumed to be separate entities that do not
interact or share data. The coordinators of the sport system would then be required to use each technology separately to support the specific tasks and copy common data between systems to achieve a complete solution.

This was the situation in the Recreational Sports Department at the University of Wisconsin-La Crosse. The department was using a combination of manual recording and different computing technologies to manage data. With regard to the technologies, the issue of specificity versus generality plagued the department with each system. One system served the needs for game management but lacked functionalities for automated team registration, while another product supported team registration but did not manage facility scheduling. Some products on the market provide most of the desired functionalities but have a high cost or are too complex to be used in a general setting. So the department decided to build its own sports management software customized to suit the department's needs.

The Recreational Sports Department desired a system that would manage the intramural registration process by providing an online interface for athlete interaction. The direct interaction by athletes would alleviate much of the pen and paper work that currently bogs down employee productivity. In addition to the registration process, the system should support the generation and presentation of contest schedules, statistics, and provide functionalities to disseminate information to administrators, athletes, and the general public.
2. A Brief Introduction to Software Life Cycle

The phrase “software life cycle” is synonymous with the utilization of a well-defined process that is applied to software development. In short, software life cycle can be defined as a set of activities or phases that occur as the software moves from its concept to retirement. There is no one specific life cycle that can be followed for all applications; nor is there one specific software application that will always be best served by a specific life cycle model, even though a model may have succeeded in previous applications. Choosing the appropriate approach can be a daunting task due to several factors such as changing requirements during the development, changing technologies, and changing management policies. However, a life cycle model is expected to drive the development process with the best solution. This should be supported by sound theory, adequate resources, experience of developers, and should ultimately lead to customer satisfaction.

There are many software life cycle models reported in literature [11, 12, 15, 16]. One of the well-known and traditional models is the \textit{waterfall model} that defines a specific set of phases that are followed in a strictly linear sequence. Other models have been developed and are modifications to the waterfall model or utilize the waterfall model but focus on the iteration of various phases. All such models include the following phases: \textit{requirements analysis and specification, design, implementation, testing and integration, and maintenance}.

The \textit{requirements analysis and specification} phase is concerned with the elicitation of customer’s needs and attempts to determine what the system will do. Requirements can be gathered in several ways: customer interviews (most common for customized software development), literature survey, and analysis of competitive market survey. The \textit{requirements analysis and specification} phase is concluded once a developer
believes all requirements have been collected from the customer. These requirements are then transcribed to a set of formal and rigid requirements, and documented in the Software Requirement Specification (SRS). The SRS formally specifies the exact functionalities of the application and what the project will do. This document will later be used as the basis for testing to ensure that the final product does what was initially conceived.

Next the development of the project moves to the design phase which is a transformation from what the project does (the requirements) to how the system will work. The design phase is often a set of iterations whereby the designer attempts to add more details to the design. The first iterations are typically concerned with identifying the major components of the system and presenting them in a high level model that graphically depicts how the components will be constructed and relate to one another. In subsequent iterations, the designer examines high level components and breaks them down into more concrete objective pieces. In the final iteration, the designer fills in the algorithmic details which will provide a roadmap for the implementation phase.

There are two popular design approaches used in practice. The first one is called the functional approach and this approach focuses on breaking the system into a set of functional modules; all elements of a module are related by interacting tasks, common data, or processes. The other approach is called the object oriented approach and this approach seeks to break the system into independent objects. Each object encapsulates its data and provides a public interface that all other objects must interact through.

The outcome of the design phase is a set of documents that detail the requirements for the code that is yet to be written. These transition documents are the key to ensuring that the customer's requirements will be met. Design documents serve as the base for implementation. In fact, an implementation should be a naïve translation of the design towards the syntax of a particular programming language. The design documents' relationship to requirements must be traceable. With the functional
approach, the traceability can be straight-forward. Each function has a specific purpose that typically addresses a single requirement in the SRS. With an object-oriented approach, however, an additional level of complexity is seen. Because object-oriented languages rely on the communications between various instances of classes, a single method may or may not fulfill a single requirement. Many classes and various communications between instances of those classes may be required to fulfill a single requirement.

The testing and integration phase begins once a portion of the project has been implemented. This phase includes two major tasks. One of them is to validate the product; i.e., to ensure that the product meets customer requirements. This is done by checking the outcome of the product against a set of expected values. The other task is to put together various pieces of source code to build up the final working product. Integration is one of the most difficult phases in software development. If the design is incorrect, the integration of pieces will not work because of discrepancies in interfaces, the realization of missing components, or problems in the translation of the design to source code. Correcting a mistake during the integration phase by merely adjusting the code is called patching. This process will lead to serious problems in traceability and maintenance. For integration to work correctly and smoothly, every document involved in the mistake should be traced and corrected. For example, if the problem lies in design, the issue will be traced back from integration to implementation and then back to design. If the design has to undergo major revisions, these changes will then have to be realized in the implementation of the source code which means repeating completed phases of the work. Worse yet, if a problem is traced back to a missing requirement or requirements in the specification, the inclusion of those requirements could require a rework of the architectural design which means starting over on one or more phases of the project.

The successful integration of various modules and components allows for the solution to be integrated into the client’s environment and the software project moves into the maintenance phase. This phase is concerned with resolving problems that
customers find with the final solution, often referred to as corrective maintenance. The incorporation of new functionalities that are just now being realized due to oversight or new corporate needs is also done during maintenance; this process is referred to as perfective maintenance.

Testing is one of the most important periods in software development. It is testing that ensures that the product meets customer requirements and each life cycle model includes one or more specific periods for testing. For example, one variation of the waterfall model indicates that testing must be performed at the end of the implementation and integration phase to discover and eliminate errors in individually coded modules and integrated subsystems [16]. Another version of the waterfall model specifies that unit testing and system testing are two specific phases that occur after implementation [11]. Other life cycles models such as the spiral model indicate testing to be an activity at the end of every phase so that the output of each phase is correct and traceable to previous phases [11]. No matter the life cycle models used, it is crucial that testing be incorporated to validate that the solution meets customer needs and errors in logic or implementation are uncovered and resolved.
3. The Development of IMSMS

IMSMS was developed using the waterfall model, which follows the life cycle phases in a strict linear sequence. This section describes the various phases of IMSMS development.

3.1 Requirements Gathering, Analysis, and Specification

Initially, the developer met with representatives from the Recreational Sports Department at UW-L (customers) several times to discuss the scope of the project, the customers’ needs and how the resulting software will help automate several tasks that are currently handled manually at the Recreational Sports Department. As with any other software development project, these initial meetings identified the major functionalities. However, a majority of these functionalities were incomplete or vague. The developer had to do an initial analysis to determine the boundaries and scope of the system to be developed. This initial analysis was conducted using a throw-away prototype. A significant additional requirement, namely “tracking sportsmanship points” was included after this analysis. The requirements were then written into a structured requirements document; this document followed the Institute of Electrical and Electronics Engineers (IEEE) Standards on Requirements Specification [4].

3.2 Design

A major challenge in the design of IMSMS was to choose the right technologies for the project. Since a web-based interface was included in the product, several web-based technologies were analyzed. Included in the list of such technologies are PHP, Java Servlets, Java Beans, Enterprise Java Beans, HTML, and ASP. Based on the
developer's experience in Java programming and also the availability of several free resources, Java Servlets with Java Beans were selected as the technology for this project. At the time IMSMS was designed, there was not sufficient information on web-based technologies, and hence the developer had to spend considerable time reading and analyzing literature support and prototyping using these technologies. Once research on the selected Java technologies was completed, an architectural design (Appendix C) was derived that supported the functionalities of the requirements document. The developer then iterated on the architectural design filling in algorithmic details until a detailed design was constructed that would be used straight away for the implementation.

3.3 Implementation, Testing, and Integration

The developer chose to do cycles of implementation followed by testing and integration by developing the modules derived in the architectural design based upon their importance to the system. For the first cycle, the developer created the administrative module and tested it with regard to the functionalities specified in the requirements document. Next the author developed the team captain module, and tested it upon completion. The team captain module was then integrated with the administrative module and tested to ensure that there were no conflicts between the two modules. Finally, the last two modules (participant and public) were developed, tested, and then integrated with the previous modules. Once all modules had been developed and integrated, a final set of tests were run to validate the integration of the various modules.
4. The IMSMS Application

This chapter presents the IMSMS application's general architectural design in Section 4.1 and a detailed architectural design in Section 4.2. The database design is presented in Section 4.3, and Section 4.4 discusses the graphical user interface and usability issues that help derive its construction.

4.1 General Architecture of IMSMS

The general architectural design of the IMSMS is presented in Figure 4.1. The IMSMS architecture is structured so a user interacts with a JSP (Java Server Page) by providing data for submission into the system, requesting data from the system, or invoking an action on the system. The JSP in turn communicates with an Apache Tomcat web container running on a server. The Apache Tomcat web container houses the Java Servlets that drive the IMSMS application, and these are contacted by the web container passing in the request object from the JSP. The Java Servlet then executes the requested action by retrieving, updating, or deleting data housed in an Oracle database.

![Figure 4.1. The general architecture of IMSMS](image)

The IMSMS application is constructed so that the system can be deployed as a two-tiered application where the Apache Tomcat web container and the Oracle database are located on one server, or as a three-tiered system where the Apache Tomcat web container and Oracle database are located on different servers.
4.2 User Classifications

The Recreational Sports Department at the University of Wisconsin-La Crosse requested that the system support four types of users.

The most general type of user is the public user. This user is assumed to be someone in the general public that is not part of the system. This user uses the system to view information about when and where contests are being held, and what teams are participating in currently open leagues. Public users may also request statistics about teams and players.

The second type of user is the participant. Participants register with IMSMS by storing their information in IMSMS. Specifically, a participant may be a player on a team or a player that is part of a free-agent listing hoping to join a team. Participants have the ability to create and update their personal profile and register a team in a league.

The third type of user is the team captain. A team captain is one that is not only a participant, but also has been named the captain of a team that is included in the IMSMS. Team captains have the ability to create and update teams by changing the name of the team or adding players to the team either by entering in an existing player's student ID or by selecting a player that is part of a free-agent waiting list for the league the team is competing in. Also, team captains can modify the profiles of the players on their team by updating the information in the profile (except for the student ID). In addition to these abilities, a team captain can send emails to all participants of teams for which they captain.

The final type of user is the administrator. An administrator has full control over the IMSMS. Administrators can create, update, and delete sports, seasons, divisions of play, leagues, participants, teams, contests, facilities and courts, as well as send email to sports, leagues, teams, or participants. Administrators can also print reports that present statistical information about teams, participants, schedules, and other relevant information to the administration of the department. Administrators also have the ability to archive data as it is deemed to be outdated.
4.3 Detailed Architecture of IMSMS

The users interact with the system through JSPs. Each JSP provides a single functionality for a user of the system. A set of JSPs therefore has been created for each type of user. Administrator JSPs and Java servlets are restricted to authorized users of the system only. This administrative security has been incorporated through the use of realms that the Apache Tomcat web container manages. The realm for this application has been configured to use the MD5 algorithm to prevent the passwords from being passed in plain text. Team captain and participant user security is based on the student ID that is presented during login. For future work on authentication, see Chapter 6 on continuing work.

The detailed architecture of the IMSMS has been broken into five general modules as shown in Figure 4.2. The administrator, participant, and team captain modules are concerned with the functionalities associated with the corresponding type of users of that module and contain Java Servlets that are utilized to support those functionalities. The shared module includes functionalities that are shared among the above three modules. Specifically it contains functionalities for free agent lists, as well as participant and team captain authentication. Finally, the utils module includes utilities that support system functionalities. Included in the utils module are functionalities to manage a connection pool used for database connections.

![Figure 4.2. Modules of IMSMS](image)

The detailed design for the administrator module is presented in Figure 4.3. This module is comprised of several Java servlets, diagrammed as rectangles titled with "servlet" in the figure. Each Java Servlet provides the services requested through a subset of JSPs. For example, when working with the sports in the system, an
administrator interacts with one of the following JSPs: createsport.jsp, updatesport.jsp, or deletesport.jsp. Each of these JSPs contacts the administration sport servlet that provides the respective functionality. The sport servlet then interacts with the Oracle database performing the requested task. Figure 4.4 depicts the interactions between the JSPs, Java servlet, and the Oracle database for administrative sports functionalities. Also, because each Java Servlet works with a specific entity in the system, each Java servlet is associated with an entity in the Oracle database. Therefore, each Java servlet maps to a separate table in the Oracle database with a corresponding name.

Because JSPs are compiled into Java servlets by Apache Tomcat, JSPs have the ability to use the Java servlets that are part of the system before the corresponding HTML code is generated for a HTTP Request. Because of this, each Java servlet in the IMSMS can create a Java bean, which is an object that contains one attribute for each field in the table corresponding to the Java servlet that instantiates the bean. The Java bean provides accessor and mutator methods for each attribute in the bean. Java beans can be embedded and used by JSPs when generating the HTML that is passed to the user upon request. In the IMSMS, Java beans are used for the filling of combo boxes, text boxes, and other controls with data that currently exists in the IMSMS. For example, when a user attempts to delete a sport, a combo box lists the sports that are currently part of the system and can be deleted. The loading of this combo box is through a small JavaScript located in the “deletesport.jsp” page that utilizes Java beans returned from a call to the Administrative Sport Servlet.
administrator interacts with one of the following JSPs: createsport.jsp, updatesport.jsp, or deletesport.jsp. Each of these JSPs contacts the administration sport servlet that provides the respective functionality. The sport servlet then interacts with the Oracle database performing the requested task. Figure 4.4 depicts the interactions between the JSPs, Java servlet, and the Oracle database for administrative sports functionalities. Also, because each Java Servlet works with a specific entity in the system, each Java servlet is associated with an entity in the Oracle database. Therefore, each Java servlet maps to a separate table in the Oracle database with a corresponding name.

Because JSPs are compiled into Java servlets by Apache Tomcat, JSPs have the ability to use the Java servlets that are part of the system before the corresponding HTML code is generated for a HTTP Request. Because of this, each Java servlet in the IMSMS can create a Java bean, which is an object that contains one attribute for each field in the table corresponding to the Java servlet that instantiates the bean. The Java bean provides accessor and mutator methods for each attribute in the bean. Java beans can be embedded and used by JSPs when generating the HTML that is passed to the user upon request. In the IMSMS, Java beans are used for the filling of combo boxes, text boxes, and other controls with data that currently exists in the IMSMS. For example, when a user attempts to delete a sport, a combo box lists the sports that are currently part of the system and can be deleted. The loading of this combo box is through a small JavaScript located in the “deletesport.jsp” page that utilizes Java beans returned from a call to the Administrative Sport Servlet.
Figure 4.3. Detailed architecture of the administrative module
Figure 4.4. Interaction map for sports administration

The administrative side of the system has the following Java Servlets to provide the administrative functionalities:

- Archive Servlet: The Archive Servlet provides archival functionalities. When archiving, an administrative user can specify a date and archive all data before that date to a storage medium of their choice (hard disk, CD, DVD, etc.).
- Athlete Servlet: The Athlete Servlet allows an administrator to create, update, or delete athletes in the system. Information located in an athlete's profile
include: student ID, name, address, city, state, zip, phone number, email address, year in school (freshman, sophomore, junior, senior, graduate, faculty, and staff), sex, and residential status indicating living on campus or off campus.

- Contest Servlet: The Contest Servlet allows for the manual creation, updating, and deletion of contests for a specific league.

- Contest Auto Schedule Servlet: The Contest Auto Schedule Servlet provides the functionality for auto-scheduling contest for a league after the registration period has closed and no contests have been scheduled. The contest auto scheduling algorithm is round-robin scheduling and applies the polygon method [3].

- Court Servlet: The Court Servlet allows for the creation, updating, and deletion of courts within a facility. Administrators can specify the type of sports that each court supports.

- Division Servlet: The Division Servlet allows for the creation, updating, and deletion of divisions in the system. Example divisions would be: Men’s, Women’s, and Co-Recreational.

- Email Servlet: The Email Servlet provides email functionalities.

- Facility Servlet: The Facility Servlet allows for the creation, updating, and deletion of facilities in the system.

- IMSMS Element Servlet: The IMSMS Element Servlet is an abstract class that contains common actions of several servlets in the administrative side. For example, all servlets have a doPost method that is called by the Apache Tomcat web container. This method resides in the IMSMS Element Servlet since most of all the administrative servlets have a common functionality and algorithm for this method.

- League Servlet: The League Servlet allows for the creation, updating, and deletion of leagues in the system. It also provides functionalities for retrieving objects associated with leagues: contests, teams, participants, etc.
• Report Servlet: The Report Servlet allows for the creation of statistical or schedule reports for administrative uses. Examples include team rosters and male/female ratios for leagues, seasons, teams, or the system as a whole.

• Season Servlet: The Season Servlet provides functionalities for the creation, updating, and deletion of seasons. Seasons could be Fall, Winter, Spring, and Summer or could be based upon semesters of the school year.

• Sport Servlet: The Sport Servlet allows for the creation, updating, and deletion of sports in the system. Example sports are Volleyball, Hockey, Basketball, and Badminton.

• Team Roster Servlet: The Team Roster Servlet works with the rosters of teams. Specifically it allows for adding and removing participants from team rosters, as well as general data retrieval for team rosters.

• Team Servlet: The Team Servlet provides functionality for creating, updating, and deletion of teams in the system.

• Team Wait List Servlet: The Team Wait List Servlet provides the ability for teams that attempt to register for a league that is full, to be wait listed and possibly joined to the league by administration. This could occur if teams withdraw from the league, do not pay their dues, or if the administration decides to add more teams to a league beyond the league capacity.

4.4 Database Design of IMSMS

The database for the IMSMS application was developed based on the data for athletes and additional information furnished by the Recreational Sports Department. Several additional entities were created by the developer in order to facilitate the services for other entities. Figure 4.5 shows the database design for the IMSMS. In this design, each entity is denoted using a rectangular box with the entity title. Below each title is a list of attributes that define the entity (right hand column) and modifiers for each field; PK represents the primary key(s), FK represents the foreign key(s), U identifies that the attribute’s values must be unique.
During the database design, a critical decision was made how the Oracle database would be used by the IMSMS. This decision determined what type of locking, optimistic or pessimistic, would be used by the IMSMS application. With pessimistic locking, the database would be locked whenever a read occurred on the database with the intent of updating or deleting the information. With this solution however, the database could have one or more records locked for an extended period of time that would significantly affect the performance of the system. With optimistic locking, the
The IMSMS database would not be locked for read commands with the intent of updating the data after processing. Instead, it was assumed that the data would not change between the time it was read and updated or read and deleted. The optimistic locking approach avoids data corruption by sending users error messages whenever an improper race condition occurs. The IMSMS database uses the optimistic locking approach.

4.5 Graphical User Interface of IMSMS

The IMSMS contains a web-based graphical user interface. This interface is comprised of several JSPs through which the user interacts. JSPs are essentially HTML pages that can incorporate JavaScript. A major difference however between HTML pages and JSPs is that JSPs are compiled and converted to Java Servlets.

Several issues were researched and decisions made about how to handle the user interactions with the JSPs. For example, the JSPs were created to minimize user errors. Therefore, whenever possible, the use of combo boxes, calendar controls, and other preloaded form objects are used so that the users of the system are not required to manually type in data. This approach alleviates many of the problems that could arise from invalid or improperly formatted input. For example, the create league JSP depicted in Figure 4.6 is constructed in such a way that a user never manually types any data since data is selected from combo boxes and calendar controls.

For each JSP, custom graphics were created to support navigation between JSPs. These were developed as “png” files using Macromedia Fireworks, optimized to remove colors that were not being used, and then exported as “gif” files. During the optimization process, it was noted that most of the “rollover” buttons that were created for navigation were approximately 29 Kb in size. After the optimization process was completed, all but one button was 1 Kb in size. The optimization of all graphics helps to improve the performance of downloading files across a network.

In addition to optimizing the graphics, the IMSMS Element Servlet gets the “Accept-Encoding” header from the HTTP Request object that is passed into the doPost() method. If the header contains a value then the user's browser supports compression and IMSMS compresses all returned web content to further reduce the
downloaded size of files and network traffic. This increases the download speed and helps to further reduce network traffic.

![Create League JSP](image)

Figure 4.6. Create league JSP

In addition to optimizing the graphics, the IMSMS Element Servlet gets the "Accept-Encoding" header from the HTTP Request object that is passed into the doPost() method. If the header contains a value then the user’s browser supports compression and IMSMS compresses all returned web content to further reduce the downloaded size of files and network traffic. This increases the download speed and helps to further reduce network traffic.

### 4.6 Deploying IMSMS

As stated earlier, the IMSMS application runs in an Apache Tomcat web container residing on a server. The web container needs to be configured to handle several
aspects of the system. First, the server’s configuration file, “server.xml”, needs to include the IMSMS context. This context specifies the path to the application based upon the root of the web container. Also included in the context are the log file location and the log file name. Additionally, the context provides the database driver name, the database connection URL, and the database account (user ID and password) that has been created to allow Apache Tomcat to connect and authenticate.

Outside of the configuration file, the IMSMS contains two files that need to be configured in order to establish connections to the database (outside of Realms) and to an SMTP server. These files are located in the IMSMS/WEB-INF folder. The first file is the “dbconfig” file that specifies the database that the system will use, the IMSMS account (user ID and password) to connect to, and the minimum and maximum number of connections to the database that the system can establish. The minimum and maximum limits were set so that IMSMS could not hold and utilize all the licensed connections to a specified Oracle database if that were not desired. This allows other applications that may also utilize Oracle to have free connections for use when the IMSMS has a light load or is sitting idle. The second file (located in the same directory) is the “smtpconfig” file that includes an entry for the SMTP server to use when performing email services.

Both of these files are used by the IMSMS during the initialization of the application. The “web.xml” file for the IMSMS application has been configured to initialize IMSMS by creating the connection pool and compile and load an instance of each Java Servlet immediately after the Apache Tomcat web container is started.

4.7 Design Decisions for IMSMS

What follows is a list of major design decisions that were made to support the architectural, database, and GUI designs.

- All data that the user enters will be stored in upper case. This is because it would be possible otherwise to enter names in multiple formats (lowercase or mixed cases).
• There is a class corresponding to each table in the database with the same name. This decision provides a logical association between classes and tables.

• The deletion of a primary object (facility, for example) will delete all secondary objects associated with it (courts, for example, that were part of the facility).

• All primary objects have a Java Servlet associated with them. This means that each table in the database has a single Java Servlet that manages it (i.e., if there is a league table in the database, there will be a league Java Servlet that handles inserts, updates, and deletes in the league table).

• The system will use client-side validation for content. This is to reduce the load on the server. Note that this requires clients to have JavaScript turned on to allow the JavaScript validation to occur. The system could be changed in the future to allow server side validation, but that could strain the server and reduce performance.

• The GUI was built to minimize the amount of validation by the application code. Specifically:
  o Field lengths will be handled by the text fields that are used for entry. The text field will not accept more characters than the max field length.
  o Dates can only be selected from a calendar (no manual entry of dates) so that the format of the date does not need to be validated. The only validation required is to check whether the date is in the future or past based on the requested operation.
  o When searching or selecting elements that exists in the system, combo boxes will be used to present valid selections.

• All emails sent by administrative users will utilize a specific administrative email account for the “from” address.

• All emails from team captain users will show the team captain’s email address in the “from” address.
• If there is a problem with a recipient’s email address, and there is more than one recipient of the email, the invalid email address(es) will not be emailed, the valid email address(es) will be emailed, and a message will indicate the invalid email address(es).

• Sport Names, Division Names, Season Names, Team Names, Participant IDs, and Facility Names are unique in the system.

• Court Names are not unique in the system. Court Name and Facility Name together will be unique in the system. For example, in “Memorial Gym Court #1”, “Memorial Gym” is the Facility Name and “Court #1” is the Court Name.

• The “web.xml file” is configured to load the IMSMS Java Servlets at startup to prevent errors in accessing the database before initialization of the connection pool. This also has the benefit of pre-compiling all Java Servlets before they are accessed.

• The IMSMS will not auto-schedule contests until after the registration period has closed and has validated that no contests have been created for a specific league.

• An administrative user cannot modify or delete a contest after the league has been completed.

• An Administrative user cannot send emails to leagues that have completed.
5. Limitations

The IMSMS was initially conceived as a general application that could be utilized by both the Recreational Sports Department at the University of Wisconsin-La Crosse and other institutions or sports leagues to manage their sports and facilities. However, as the project progressed and changes were made based upon the business rules of the Recreational Sports Department, the system design moved towards a customized system that only supports the needs of the Recreational Sports Department. Because of this, there are several limitations with regard to use in a general environment.

One limitation of the system revolves around the automated contest scheduling for a specific league. The algorithm for auto scheduling supports the philosophy of the Recreational Sports Department whereby a league is created for a specific day of the week and time of day. Therefore, the auto scheduling algorithm does not vary the time of day or day of the week when scheduling the contests. To be used in a more generic setting, this may be more desirable or required.

Another limitation with the automated contest scheduling is the determination of appropriate facility. Currently the product retrieves all facilities that are available during that date and time, and supports the sport of the league. This does not take into account the number of facilities or the number of sports that each facility/court supports. This may lead to problems if a facility/court F supports two sports, say sport A and sport B, where sport A is supported in many facilities but sport B is only supported in the current facility. If an administrator schedules a league of sport A first and the facility/court F is selected by the algorithm, and if the administrator attempts to schedule a league of sport B, it is possible that no facility will be available because no other facility/court supports the sport.
When looking at the team registration by a team captain, the system does not require that the team captain supply a team roster for the team. This allows a team without players to reserve a spot in a league. A solution to this would be to force the team captain to enter the student ID of each student that will be on the team. This, however, could cause a problem if the athletes never provide their information through the participant side of the system. This is further complicated by the fact that participants cannot add themselves directly to a team. For a team captain to add a participant, the participant’s information must have been entered into the system previously, or the team captain must enter the information and then add the individual.

The issue of authenticity with regard to participants and team captains exists in the system. This could however, be alleviated by allowing IMSMS to validate information with the university’s database. This validation could be handled during participant registration. Furthermore, if a tie is successfully added, there is no need for participants to provide the currently required information as the participant’s profile could be directly retrieved from the student registration system and the university database.

Another limitation exists in the system with respect to the tracking of team and player information versus participation. The Recreational Sports Department has a rule that a participant cannot play on two teams in a single league; this rule is easily enforceable. However, the department also wants the ability to keep entire teams together so that teams can be tracked for performance data such as the number of league championships won, the number of leagues a team has participated in, the number of different sports a team has participated in, and other statistics. At present, a player of a team may participate in a basketball league, but may decide to not participate with this same team in a volleyball league. The participant may wish to participate on another team for the volleyball league. The issue is to either create many instances of a single team with different rosters for different leagues, or to keep a single instance of the team and then not include enforcement rules when adding
players to the team and teams to leagues. Based on the conversations with the Recreational Sports Department, the system currently does not enforce these participation rules, but could, at the same time, also include invalid data for a participant of a team.

Because the system is transmitting data over the Internet, security is another primary concern. The system does not encrypt data that is transmitted and hence it is possible to intercept plain text information that is being sent or view cached copies of visited JSPs on a public computer. However, the data being transmitted was not considered to be a threat to any individual by the Recreational Sports Department and therefore the system was not built for the inclusion of encryption or password usage. The Recreational Sports Department’s argument for this decision was that they currently require the team captain to obtain all participant information for each member of their team and therefore that information was already known to other athletes and therefore not confidential. In addition, the “TALON” system used by the university requires that users not only enter their student IDs but also a private password and hence someone that intercepted or viewed that information would not have access to the individual’s records.

Limitations exist for students with disabilities. For example, the system was not designed or tested for users with visual impairments or other disabilities.

Another limitation is the use of the default concurrency mechanisms of the connection and statement objects. These objects have default optimistic locking mechanisms defined in the Oracle driver where locking occurs at the record level to maximize database concurrency. Each insert, update, and delete statement that is sent to the database for execution is then committed upon successful completion of the statement. There are occasions, such as working with waiting lists and leagues, where two statements need to be executed in order to complete a transaction. In these cases, an improvement would be to lock the associated records through a begin transaction statement (pessimistic locking), followed by the sequence of statements to perform the transaction, and then followed by a commit or rollback statement. Currently, if a
failure occurs in the middle of a transaction, the IMSMS system automatically
reverses the changes made by the previous statement(s) to return the database to its
previous state. By incorporating transaction statements, it would be possible to
incorporate a higher lock granularity to utilize the DBMS transaction processing;
albeit, a reduction in concurrency could occur in the system.
6. Continuing Work

At present, no validation process exists for intramural participant registration against university records. A module could be designed and integrated into the system that would automatically verify eligibility based on student registration system records. When an intramural participant attempted to register, the system would interface with the university system to validate the user as a valid student, faculty, or staff member. With this approach, the user would not need to enter IMSMS registration information separately.

In its current configuration, the system requires a database administrator to grant or revoke rights and privileges for access to the administrative functionalities through the inclusion of user IDs, passwords, and user roles that the Apache Tomcat Realm authenticates against. The development and integration of administrative management tools that would allow for the insertion and removal of administrator users of the system would be beneficial for administrative purposes.

Another portion of the system that could be addressed is the SQL statements used in the solution. Currently, all SQL statements are created on the fly with a call to a connection object’s “getStatement()” method. It could be advantageous to use prepared statements for each entity in the database. This would allow the system to have a precompiled statement on the database and simply send the necessary parameters to improve processing speed on the database server.

Lastly, some functionalities listed in the requirements document are currently not completed. The bulleted list below denotes requirements not included in the IMSMS application.

- A public user should be able to view contest schedules and results, a listing of teams, and team rosters for a league.
• Reports that include statistical data including:
  o Number of active sports in a season.
  o Number of contests each team has participated in (date based).
  o League contest schedules and results printed by day, week, or season.
  o View participant information with respect to leagues:
    ▪ The number of teams the participant was part of.
    ▪ The number of leagues the participant participated in.
    ▪ The number of championship teams the participant was part of.
  o A listing of team champions based on:
    ▪ Leagues that have completed.
    ▪ Division.
    ▪ Team name.

• Score sheets are available in the system. However they do not print out the team name or the participant names. The JSP page for each score sheet needs to be updated to include this information for a specific contest.

• The system should develop tournament brackets for a league after the completion of league competition; the department has currently requested that this be performed manually.
7. Conclusion

The IMSMS application has been developed to meet the needs that were specified by the client in the requirements phase and developed using sound object-oriented principles. The system currently supports the following functionalities accessible through a web interface.

Administrative functionalities:

- Creation, modification, and deletion of sports.
- Creation, modification, and deletion of seasons.
- Creation, modification, and deletion of divisions.
- Creation, modification, and deletion of leagues.
- Creation, modification, and deletion of contests.
- Auto-schedule contests for a league.
- Creation, modification, and deletion of facilities.
- Creation, modification, and deletion of courts.
- Creation, modification, and deletion of teams.
- Creation, modification, and deletion of athletes.
- Ability to add teams to leagues from team waiting lists.
- Ability to add participants to teams.
- Generation of a report to present teams and their associated roster for specified leagues.
- Generation of a report to present athletes and their associated information.
- Generation of email by league, sport, division, athlete, and team.
- Data archival to reduce the size of the database and to remove unwanted or outdated data.
- Security to prevent unauthorized use of administrator functionalities.
Team Captain Functionalities:
• Create their participant profile.
• Log into the system using their student ID.
• Update their participant profile.
• Have the system recognize a user as a team captain through their login and present team captain functionalities.
• Online team registration for league participation.
• Join the team waiting lists when league registration is unsuccessful because a league is full.
• Create a team.
• Update a team by changing the team name.
• Add participants to a team by student ID or through the free agent lists.
• Send an email to team participants.

Participant functionalities:
• Create and update their participant profile.
• Log into the system using their student ID.
• Join a free agent list.

This product will assist the Recreational Sports Department at the University of Wisconsin-La Crosse by streamlining the current intramural administrative practices. Specifically, it will remove much of the face-to-face contact that is required with the captains of teams that participate in the system by providing a team captain and participant side to the system. This will allow the team captains and participants of the system the ability to create and update personal information that is maintained by the department. Also, it will allow the administrators to configure a set of leagues in the system, auto-schedule contests, and simply record the results of those contests so that administrators and the public can track the teams during the course of league play. Finally, the product addresses the needs of the department as the IMSMS was custom built.
Bibliography


Appendix A: Sample IMSMS Screen Shots

IMSMS Home Page
Welcome to the Participant Login Page

You are currently viewing the login page for the system. Before you can access restricted functionality you must first login. Please provide your student ID and then click submit. If you select Home, you will return to the Intramural Sport Management Home Page.

Please enter your student ID: ____________________________

Submit  |  Home

If you are a new participant and would like to register your information in the Intramural Sport Management System go to the CREATE NEW ACCOUNT page.
You are currently viewing the home page for the system. From here, you can navigate to the different administrative functionalities of the system. To go to a particular functionality, select the functionality from the menu on the left. When you click on a functionality, you will be taken to a page that supports that functionality.
Welcome to the Sports Administrative Page

You are currently viewing the Sports Administrative Page for the system. From here, you can create, update, and delete the sports that are part of the system. Simply select the action you wish to perform from the menu on the left side of the page. If you select 'Help', you will be taken back to the Administrative Home Page.
Welcome to the Create Sport Page.

From this page you can create sports that will become part of the Behavioral Sport Management System. To create a sport, simply enter the name of the sport and click submit.

Enter the sport name:  

[Submit]
Welcome to the Modify Sport Page

From this page you can modify a sport in the Intramural Sport Management System. Simply select the name of the sport in the list box below, update the sport name and then click submit.

Select the sport to modify:

Enter the new name for the sport:

Select a Sport

Soccer
Basketball
Flag Football
Field Hockey
Badminton
Water Polo
Ultimate Frisbee
Tennis
Welcome to the Delete Sport Page.

From this page you can delete a sport from the Intramural Sport Management System. Simply select the name of the sport in the list box below and click submit.

Select the sport to delete: [Sport Name]

[Clear/Reset] [Submit]
Welcome to the Add Team to League Page

You are currently viewing the Add Team to League Administrative Page for the system. From here, you can add teams to an league in the Incremental Sport Management System. Simply select the league you wish to add a team to, select the team from the watching list, and click submit. If you select None, you will be taken back to the Administrative Home Page.

Select League:
- Incremental League

Select Teams from Waiting List:
- None
- Submit

IMSMS Add Team to League Page
Welcome to the Modify Team Page

From this page you can modify a team in the Intramural Sport Management System. To modify a team select the name of the team from the list box. You can then update the team name and/or the team captain. To save your updates click Submit. If you wish to add players, click the Add Players button.

Updating Team:  
New Team name:  
New Team captain's student id:  

IMSMS Modify Team Page
Welcome to the Add Player To Team Page

From this page you can add a player to a team in the Intramural Sport Management System. Simply select the name of the team in the list box below and then select a player and click submit. If the player does not exist, you first need to create the player and then return to add the player to the team. If you select either, you will be taken back to the Administrative Home Page.

Select the team:

Select the player to add:

One / Two
Welcome to the Modify Athlete Page

From this page you can update athlete information in the system. The current information for the athlete is presented below. Update the necessary fields and click submit. If you click the homepage button you will be taken back to the administrative main page.

<table>
<thead>
<tr>
<th>Select An Athlete</th>
<th>First Name</th>
<th>Last Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Direct Address</th>
<th>City</th>
<th>Zip Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>123 W Ave</td>
<td>NY</td>
<td>12345</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phone Number</th>
<th>Email Address</th>
<th>On Campus</th>
<th>Off Campus</th>
</tr>
</thead>
<tbody>
<tr>
<td>555-123-4567</td>
<td><a href="mailto:blake@abc.com">blake@abc.com</a></td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year in School</th>
<th>Gender</th>
<th>Other Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Senior</td>
<td>Male</td>
<td></td>
</tr>
</tbody>
</table>

"Clear Form" [Submit]
Welcome to the Create Facility Page

From this page you can create facilities that will become part of the Automated Sport Management System. To create a facility, simply enter the name of the facility and click submit. If you select name, you will be taken back to the Administrative Home Page.

Enter the facility name:

Submit Form | Create
Appendix B: Sample IMSMS Web Interaction Maps

IMSMS Administrative Main Web Map

IMSMS League Administration Web Map
IMSMS Email Administration Web Map
Appendix C: Sample IMSMS Class Diagrams

IMSMS Administrative Package Servlet to Bean Class Diagram