DATABASE DESIGN AND IMPLEMENTATION ISSUES IN LIBRARY DOMAIN

ONUR İHSAN ARSUN

IŞIK UNIVERSITY
2006

DATABASE DESIGN AND IMPLEMENTATION ISSUES IN LIBRARY DOMAIN

A Thesis
Presented to the Institute of Science and Engineering
of
Işık University
In Partial Fulfillment of the Requirements for the Degree of
Master of Science
in
The Department of Computer Engineering

by
Onur İhsan Arsun

IŞIK UNIVERSITY
July 2006

Approval of the Institute of Science and Engineering

_____
Prof. Dr. Hüsnü Ata Erbay
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____
Prof. Dr. Selahattin Kuru
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____
Prof. Dr. Selahattin Kuru
Supervisor

Examining Commitee Members

Prof. Dr. Selahattin Kuru _____

Assist. Prof. Dr. Taner Eskil _____

Assist. Prof. Dr. Vedat Coşkun _____

ABSTRACT

DATABASE DESIGN AND IMPLEMENTATION ISSUES
IN LIBRARY DOMAIN

Onur İhsan Arsun

In this thesis, we study the design and implementation issues in library management systems. Depending heavily on data, library management systems should have effective means to access and store data. We propose a novel representation of the MARC21 format for relational databases, namely relational MARC. We also introduce methods for transaction processing in library management systems which are portioned database tables, precompiled SQL queries, connection pooling, and statement caching. We also discuss issues in development process and our approach with extreme programming (XP), object oriented analysis and design, and design patterns. All these approaches have been applied in a large web-based library management system, Library ON-LINE, and have been evaluated and experience is reported.

Keywords: Relational MARC, pre-compiled SQL queries, statement caching, connection pools, extreme programming, object oriented analysis and design.

ÖZET

DATABASE DESIGN AND IMPLEMENTATION ISSUES
IN LIBRARY DOMAIN

Onur İhsan Arsun

Bu çalışma ile, kütüphane yönetim sistemlerinde tasarım ve gerçeklenme hususları ele alınmıştır. Kütüphane yönetim sistemleri, büyük veriler içerdiklerinden, bu verilere etkin erişim ve depolama yönetimleri kullanmalıdırlar. MARC21 biçeminin ilişkisel bir sürümü olan, ilişkisel MARC gösterimini sunuyoruz. Ayrıca hareket işleme konusunda, bölümlenmiş veritabanı tabloları, önceden derlenmiş SQL sorguları, bağlantı havuzları ve sorgu önbellekleme metodları tartışılmıştır. Geliştirme sürecindeki hususlar da tartışılmış, bu hususlara aşırı programlama, nesne-yönelimli analiz ve tasarım ve tasarım örüntileri ile yaklaşımımız açıklanmıştır. Tüm bü öneriler, Library ON-LINE adlı web tabanlı kütüphane yönetim sisteminde gerçeklenmiş ve değerlendirilmiş, bu konudaki tecrübe aktarılmıştır

Anahtar Kelimeler: İlişkisel MARC, önceden derlenmiş SQL sorguları, sorgu önbellekleme, bağlantı havuzları, aşırı programlama, nesne-yönelimli analiz ve tasarım.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this M.Sc. thesis.

First and foremost, I am deeply indebted to my supervisor, Professor Selahattin Kuru, for his incredible support, guidance and encouragement, not only for this work, but throughout the last six years. Besides being an excellent supervisor, he is an inspirational leader. I have learned very much from his open-minded, motivating, and enthusiastic attitude towards me each and every day. I consider myself very fortunate to have met with such an exceptional person.

I would like to extend my gratitude to my colleagues Mustafa Yıldız, Gürol Erdoğan and Orhan Karahasan for creating such a critical and motivating atmosphere all the way through the development of Library On-Line and this thesis. Each and every one of them was there when I needed help, proving that interdependence is more valuable than independence.

I would like thank my friends, Sevkan Taşkan, Koray Derman, Cumhur Cecan, Tuncay Kamil Güçlü and Hakan Terzioğlu for all those years we spent together and supporting me through the good and the bad times and sharing memories we are always fond of telling.

I would also like to thank Nihan Üstünkol, for her unconditional support, love and understanding throughout this work and many years.

The last but definitely not the least, I would like to thank my family. They restlessly taught me good things that really matter in life. Their existence provided me a persistent inspiration for my journey of life. Without you, this would never become true.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Library Management Systems started using information system by the late 70s offering many opportunities to librarians. However, when it comes to developing library management systems, there are some essential issues to consider. In this thesis, we clearly introduce these issues and propose approaches for them.

Most important issues rise in catalog database storage and implementation. Library cataloging data can reach huge sizes in proportion to the number of materials in the library. This makes the data hard to manage. There are some methods for catalog data representation that can be use to store the catalog data as well. These are the MARC21 [1] and the MARC-XML [2]. In addition to those, we developed a relational model for representation and storage of catalog data.

The huge sizes of catalog data make transaction processing a vital issue. Accessing, modifying and displaying the catalog data require careful handling. We developed a set of methods to increase transaction processing performance. These are portioning catalog storage table, using pre-compiled SQL statements, utilizing connection pools, and using query caching mechanisms. We evaluate these approaches as well.

The other issues in developing and operating library management systems are in development process, catalog data conversion, integration with electronic online databases, and integration with other university information systems.

We will demonstrate how we deal with the issues in development process by using, eXtreme Programming [3], object-oriented analysis, design and programming, and effective utilization of design patterns.

Catalog data conversion is an important operational issue; proposals have been made to make this process automated and easier.

Integration is another important issue. As online electronic databases get larger and gain more users, integrating them with library management systems becomes more and more important. The site of research and development for this thesis is a university. Proposals have been laid out on how to integrate these systems with library management systems.

At the end of the thesis follows an evaluation of the approaches. The approaches have been implemented in an application, Library ON-LINE, and its evaluation is given in the chapter.

Finally, recommendations for future work and conclusion is given chapter 8.

CHAPTER 2. LIBRARY MANAGEMENT SYSTEMS

Library management systems use the means of information systems such as computers and remote accessing technologies to accomplish the well-defined library task like cataloging, circulation, serials management and document browsing. Library management systems help managing and sharing of all library resources in an optimum way.

Library Management Systems started in late 70s offering many opportunities to librarians helping them to improve their business practice and many benefits to patrons improving the quality of service supplied.

One of the most considerable features that come with Library Management Systems is the online catalog. Almost all of the modern systems allow access to the library catalog via means of special client tools or web browsers. Online public access catalogs facilitate searching of library catalog with computer aided interfaces both from inside and outside the library. Searches are conducted with many different criteria as opposed conventional sorted card-based catalog searches winning vast amounts time and effort.

Integrated Library Management Systems offer different functionalities like acquisition and circulation in a single box. Furthermore, these systems allow integration with other libraries' catalog databases. Also, integrated systems allow online databases to be tightly integrated with the local catalog.

Usage of Library Management Systems is inevitable for all kinds of libraries. The advances in digitized cataloging and online databases should be offered as parts of these systems. Especially for large libraries, these systems offer much more optimum usage of library resources both for the patron and library staff.

2.1. Modules of Library Management Systems

Modules of Library Management System are more or less identical in all systems in the market. The core modules are cataloging, circulation, acquisition, online catalog access, serials management, inter-library loans, and integration with electronic online databases. The helper modules include administration and reporting modules. We investigate these modules one by one.

2.1.1. Cataloging

Cataloging modules is responsible for managing catalog data for library materials. Every library material has a series of cataloging information associated with it. Some examples are title, subject headings, author, ISBN/ISSN, publisher, etc. With the introduction of information systems to the library domain, MARC (MAchine Readable Cataloging) has become the standard for cataloging library materials.

Usage of information systems exposes one of its most important benefits in material cataloging. The conventional cataloging process is all based on card based system. Selected cataloging data is written on catalog cards and are lexicographically based on call numbers in drawers. This kind of storage is unreliable, insecure and inconvenient. Utilization of information system helps librarians to better store the cataloging data. Operations done on the catalog are more stress-free as sorting, multiplication and searching the catalogs are much easier.

2.1.2. Circulation

Circulation module is responsible for check-in, check-out and renew processes. Library Management Systems hold the circulation data for patrons and stock items and manages the circulation according to the circulation rules as determined by the library management. This facilitates the process of keeping track of circulation in libraries. The traditional manner is to write down the patron name or identifier for every circulated item onto cards.

Using technologies such as barcodes, smart cards, RFIDs or automatic circulation machinery simplifies the circulation process even further.

2.1.3. Online Public Access Catalog

In traditional processes the patron should physically search the card catalog drawers sorted according to subject headings or material title. This is a tiresome process in many ways. Usage of information systems simplifies this process.

The library catalog is stored in a kind of database, and the database is open public access through many ways. One common way is terminal access to the catalog. Clients use terminal software to remotely access the catalog and conduct searches. Terminal services are generally text – based. The widespread protocol for this kind of access is the Z39.50 technology.

One other method for accessing the online library catalog is the Web OPAC. Web OPAC provides a web – based and graphical interface to the catalog using the web browsers on the client side. Web OPAC systems offer many tools for accurate search results, such as filtering, relevance ranking or fuzzy searches. Web OPACs also provides tools for remote searches on other libraries' catalogs.

Modern Web OPAC systems offer many services other than catalog searches to the patrons. These patron empowerment facilities are placing/canceling holds (reservation), self-view of patron records and online self-renewal.

2.1.4. Serials Management

Serials are periodical publications acquired by the library. Current Library Management Systems offer many tools to simplify managing of serials. The serials management tools should be tightly integrated with other modules, most importantly the online public access modules. The searches conducted on the catalog should also return matches from the serials.

Periodicals have slightly different cataloging issues when compared to other materials of the library like the books. The module should also be able to handle these kinds of varieties.

2.1.5. Inter-Library Loans

Modern Library Managements Systems offer tools for simplifying the process of inter-library loans (ILL). These can either automated tools, or un-automated tools. Most modern library management systems offer their patrons to search and reserve materials from other libraries.

The ILL module should also accept incoming requests from other libraries.

2.1.6. Integration with Online Electronic Library Databases

Electronic databases are becoming more and more common theses days. Libraries acquire licenses from electronic databases and offer it to their patrons.

Library Management Systems offer integration with those electronic databases. One common types of integration is that the catalog searches return results from electronic databases and offer hyperlinks to these resources online. Other type on integration is using proxies. Electronic databases are generally based on IP numbers. Library submits a number of IP addresses or IP blocks to the vendor, to grant access from these sources. This makes it impossible for patrons to access the electronic resources outside the library. Modern Library Management Systems act as proxies to these resources. Patrons logon to the local system and access the electronic databases through it.

2.1.7. Acquisition

Acquisition modules offer tools to manage library's material acquiring process. Acquisition module should be tightly integrated to the cataloging module. An unsuccessful integration may yield to duplicate acquisitions.

Acquisition tools should include tools for finance applications like budget management.

2.1.8. Helper Modules

Apart from the modules discussed above, there are many helper modules in modern Library Management Systems. These modules help administering or managing the library resources and improve service provided to the patrons.

System administration modules are designed for library directors. These modules offer tools to manage the systems behavior and help libraries easily tailor the system for their specific needs. Circulation rule management, catalog management, stock management, collection management, access management, and user management are common tools in these modules.

Reporting modules are designated towards library director and library management. Reports are used to measure the general library or librarian performance. It also offers statistics for services provided to the patrons. Common reports are circulation reports, cataloging reports, OPAC usage statistics, patron statistics and reports.

Circulation reports offer data for circulated items for specified time periods. Circulation reports are useful for monitoring materials circulated and patrons who circulated them.

Cataloging reports show newly cataloged materials or edited materials on a timely basis. These reports are useful for measuring the cataloging librarian performance and control what materials are added to the library catalog.

OPAC usage statistics give information on the quality of service provided by the OPAC interface. These statistics are also useful for tracking which materials are mostly searched for.

Patron statistics and reports are personal records of patrons. These show information like which materials had the patron circulated, when did she checked-in the material, on time or late. These are useful for monitoring the patrons' behavior, since there are library regulations based on patron behavior.

2.2. Issues in Database Design and Implementation in Library Domain

Database design and implementation is a crucial topic in library management systems. Here we discuss some issues in this topic.

Database Design: The database should be designed in such a way that it can handle vast amounts of catalog data belonging to the library materials. The design should also comply with the current cataloging standards such as MARC21 or MARC-XML.

Scalability: Library databases, especially cataloging tables may reach massive sizes as the material count in the library increases. The database should be able to scale to those sizes. One other important scalability issue is the number patrons getting service from the system. As the amount of the request made to the system increases, the system should be able handle the request efficiently and be able respond with the correct data.

Performance: As the catalog database size grows, performance of catalog searches through the interfaces get slower, and effectively returns more results. This can cause the whole catalog searching mechanism to be unusable from a a patron's

point of view. Performance increasing methods should be introduced as the database size grows.

Security: Keeping the library database secure is essential in many ways. Material cataloging information is the core of Library Management Systems and they are hard to produce and expensive to acquire. They are also critical to maintain the consistency of the whole catalog. Another critical information is the patron circulation records. They contain personal information for the patron and are important to keep a consistent and correct log of materials that are on loan to the outside of the library.

Transaction Processing: The results set returned by the system after a search is conducted may reach huge sizes. This returned data should be handled accordingly.

Data Conversion: The conversion of the legacy data to a newly installed system is very rigid and is a critical process. The critical data to be converted is the catalog and circulation data.

Catalog data conversion issue comes in many different ways. The library designated to use a new library management system might have no digital catalog data, meaning all of the data is on paper catalog cards. In this case, the catalog data might be produced or acquired from outside sources. If the library has some cataloging data in a digital format, this data should be converted algorithmically to the standards of the new system. If the legacy catalog data is in MARC21 or MARC-XML format, this process is more pain-free, else conversion software should be produced to handle the migration.

Orienting the library staff: Librarians, like many other professions, might resist the change to use a new library management system. Corporate policy towards using new technologies is a must.

CHAPTER 3. ISSUES IN CATALOG DATABASE DESIGN AND IMPLEMENTATION

As discussed in chapter 2, designing and implementing a library management system comes with issues to be solved in many areas. This chapter discusses the catalog storage design and implementation.

3.1. Catalog Storage

The cataloging information of a library material is a very well-defined description of the material itself. This description should syntactically and semantically comply with some standards. These standards are developed by the United States Library of Congress. Here we discuss these standards, namely MARC21 and MARC-XML, then present a novel representation which is more compatible with relational database management systems.

3.1.1. MARC21 (Machine Readable Cataloging)

MARC is the acronym for MAchine-Readable Cataloging. It defines a data format that emerged from a Library of Congress-led initiative that began thirty years ago. It provides the mechanism by which computers exchange, use, and interpret bibliographic information, and its data elements make up the foundation of most library catalogs used today. MARC became USMARC in the 1980s and MARC21 in the late 1990s [1]. MARC21 is a “harmonization” of both USMARC and CAN/MARC formats. From 1994-1997 the USMARC and CAN/MARC user communities worked to eliminate all remaining differences in their two already-similar formats. Compatibility had been a feature of the development processes for both formats for many years. In 1997 and early 1998, updates to the formats were issued that made the format specifications identical. MARC 21, a continuation of

both USMARC and CAN/MARC, publishes the formats in one edition under a new name. The Library of Congress serves as the official depository of United States publications and is a primary source of cataloging records for US and international publications. When the Library of Congress began to use computers in the 1960s, it devised the LC MARC format, a system of using brief numbers, letters, and symbols within the cataloging record itself to mark different types of information. The original LC MARC format evolved into MARC 21 and has become the standard used by most library computer programs. The MARC 21 bibliographic format, as well as all official MARC 21 documentation, is maintained by the Library of Congress. It is published as *MARC 21 Format for Bibliographic Data* [4].

"Machine-readable" means that one particular type of machine, a computer, can read and interpret the data in the cataloging record, while “Cataloging record" means a bibliographic record, or the information traditionally shown on a catalog card. The record includes (not necessarily in this order) [4]:

1. A description of the item: Librarians follow the rules in *Anglo-American Cataloguing Rules*, 2nd ed., 2002 revision to compose the bibliographic description of a library item. This "description" is shown in the paragraph sections of a card. It includes the title, statement of responsibility, edition, material specific details, publication information, physical description, series, notes, and standard numbers.
2. Main entry and added entries: AACR2 also contains rules for determining "access points" to the record (usually referred to as the "main entry" and "other added entries"), and the form these access points should take. Access points are the retrieval points in the library catalog where patrons should be able to look up the item.
3. Subject headings: The librarian uses the *Sears List of Subject Headings* (Sears), the *Library of Congress Subject Headings* (LCSH), or some other list of standard subject headings to select the subjects under which the item will be listed. Use of an approved list is important for consistency, to ensure that all items on a particular subject are found under the same heading and therefore in the same place in the catalog.

4. The classification or call number: The librarian uses a Dewey Decimal or Library of Congress classification schedule to select the call number for an item. The purpose of the call number is to place items on the same subject together on the same shelf in the library. Most items are sub-arranged alphabetically by author. The second part of the call number usually represents the author's name, facilitating this sub arrangement.

The information from a catalog card cannot simply be typed into a computer to produce an automated catalog. The computer needs a means of interpreting the information found on a cataloging record. The MARC record contains a guide to its data, before each piece of bibliographic information. The computer must have assistance if it is to read and interpret the bibliographic record.

If a bibliographic record has been marked correctly and saved in a computer data file, computer programs can then be written to punctuate and format the information correctly for printing a set of catalog cards, or for displaying the information on a computer screen. Programs can be written to search for and retrieve certain types of information within specific fields, and also to display lists of items meeting the search criteria.

Using the MARC standard prevents duplication of work and allows libraries to better share bibliographic resources. Choosing to use MARC enables libraries to acquire cataloging data that is predictable and reliable. If a library were to develop a "home-grown" system that did not use MARC records, it would not be taking advantage of an industry-wide standard whose primary purpose is to foster communication of information.

3.1.1.1. MARC21 Record Structural Elements

The data in a MARC21 record is organized into fields, each identified by a three-character tag. According to ANSI Z39.2 [5], the tag must consist of alphabetic or numeric ASCII graphic characters, i.e., decimal integers 0-9 or letters A-Z (uppercase or lowercase, but not both). The MARC 21 formats have used only

numeric tags.. The tag is stored in the directory entry for the field, not in the field itself. Variable fields are grouped into blocks according to the first character of the tag, which identifies the function of the data within a record, e.g., main entry, added entry, subject entry. The type of information in the field, e.g., personal name, corporate name, or title, is identified by the remainder of the tag. Table 3.1 shows a bibliographic record with textual signpost. Table 3.2 shows the record in MARC21 format.

Table 3.1. A bibliographic record with textual signposts

Signpost	**Data**
Main entry, personal name with a single surname: The name:	 Arnosky, Jim.
Title and Statement of responsibility area, pick up title for a title added entry, file under "Ra..." Title proper: Statement of responsibility:	 Raccoons and ripe corn / Jim Arnosky.
Edition area: Edition statement:	 1st ed.
Publication, distribution, etc., area: Place of publication: Name of publisher: Date of publication:	 New York : Lothrop, Lee & Shepard Books, c1987.
Physical description area: Pagination: Illustrative matter: Size:	 25 p. : col. ill. ; 26 cm.
Note area: Summary:	 Hungry raccoons feast at night in a field of ripe corn.
Subject added entries, from Library of Congress subject heading list for children: Topical subject:	 Hungry raccoons feast at night in a field of ripe corn.
Local call number:	599.74 ARN
Local price:	$15.00

Field: Each bibliographic record is divided logically into fields. There is a field for the author, a field for title information, and so on. These fields are subdivided into one or more "subfields." As previously noted, the textual names of the fields are

too lengthy to be reproduced within each MARC record. Instead they are represented by 3-digit tags.

Table 3.2. Bibliographic record in table 3.1 formatted in MARC21

Signpost	Data
100 1# $a	Arnosky, Jim.
245 10 $a	Raccoons and ripe corn /
$c	Jim Arnosky.
250 ## $a	1st ed.
260 ## $a	New York :
$b	Lothrop, Lee & Shepard Books,
$c	c1987.
300 ## $a	25 p. :
$b	col. ill. ;
$c	26 cm.
520 ## $a	Hungry raccoons feast at night in a field of ripe corn.
650 #1 $a	Raccoons.
900 ## $a	599.74 ARN
901 ## $a	8009
903 ## $a	$15.00

Tag: Each field is associated with a 3-digit number called a "tag." A tag identifies the field -- the kind of data -- that follows. Even though a printout or screen display may show the tag immediately followed by indicators (making it appear to be a 4- or 5-digit number), the tag is always the first 3 digits. Tag blocks used in MARC21 records are given in Table 3.3. Most frequently used tags and their definitions are given Table 3.4.

Table 3.3. Tag blocks used in MARC21

Tag	Definition
0XX	Control information, numbers, codes
1XX	Main entry
2XX	Titles, edition, imprint
3XX	Physical description, etc.
4XX	Series statements
5XX	Notes
6XX	Subject access fields
7XX	Name, etc. added entries or series; linking
8XX	Series added entries; holdings and locations
9XX	Reserved for local implementation

Table 3.4. Most frequently used MARC21 tags

Tag	Definition
010	Library of Congress Control Number (LCCN)
020	International Standard Book Number (ISBN)
100	Personal name main entry (author)
245	Title information
250	Edition
260	Publication information
300	Physical description
440	Series statement/added entry
520	Annotation or summary note
650	Topical subject heading
700	Personal name added entry

Indicators: Two character positions follow each tag (with the exception of Fields 001 through 009). One or both of these character positions may be used for indicators. In some fields, only the first or second position is used; in some fields, both are used; and in some fields, like the 020 and 300 fields, neither is used. When an indicator position is not used, that indicator is referred to as "undefined" and the position is left blank. It is the convention to represent a blank, or undefined, indicator position by the character "#". We take the 245 (Title) tag as example and describe what the indicators mean for this tag:

```
245 14   $a The emperor's new clothes /
         $c adapted from Hans Christian Andersen
            and illustrated by Janet Stevens.
```

A *first indicator* value of 1 in the title field indicates that there should be a separate title entry in the catalog. In the card catalog environment, this means that a title card should be printed for this item and an entry for "Title" added to the tracings. A first indicator value of 0 would mean that a title main entry is involved; the card would be printed with the traditional hanging indention, and no additional tracing for the title would be required (since it is the main entry).

Nonfiling character;: One of the more interesting indicators is the second indicator for the title field. It displays the number of characters at the beginning of the field (including spaces) to be disregarded by the computer in the sorting and filing process. For the title *The emperor's new clothes*, the second indicator is set to

"4" so that the first four characters (the "*T,*" the "*h,*" the "*e,*" and the space) will be skipped and the title will be filed under "*emperor's.*"

Subfield: Most fields contain several related pieces of data. Each type of data within the field is called a subfield, and each subfield is preceded by a subfield code. Fields 001 through 009 have no subfields.

For example, the field for a book's physical description (defined by the tag 300) includes a subfield for the extent (number of pages), a subfield for other physical details (illustration information), and a subfield for dimensions (centimeters):

```
300 ##   $a 675 p. : $b ill. ; $c 24 cm.
```

Subfield code: Subfield codes are one lowercase letter (occasionally a number) preceded by a delimiter. A delimiter is a character used to separate subfields. Each subfield code indicates what type of data follows it.

Delimiter: Different software programs use different characters to represent the delimiter on the screen or on printouts. Examples are an "at sign" (@), a dollar sign ($), an underline (_), or the graphic symbol "ǂ". In this publication the dollar sign ($) is used as the delimiter portion of the subfield code. In the example above, the subfield codes are $a for the extent, $b for other physical details, and $c for dimensions.

Preceding the main bibliographic record parts -- which are known to all librarians because of their presence on catalog cards -- the MARC record contains some less familiar information. Automated cataloging systems usually provide default data or prompt to help a cataloger input this information. These parts are the “leader”, the “directory”, and the “008” tag.

Leader: The leader is the first 24 characters of the record. Each position has an assigned meaning, but much of the information in the leader is for computer use.

MARC record creation and editing programs usually provide a window or prompts to assist the cataloger in filling in any leader data elements that require input.

Directory: MARC records are called "tagged" records. Before it becomes a tagged record, a MARC record (in what is called the MARC communications format), looks very different -- like one long run-on sentence. In the communications format, the fields are not preceded by tags. However, immediately following the leader is a block of data called a directory. This directory tells what tags are in the record and where they are placed (by a count of the characters to the position where each field begins). The directory is constructed (by computer) from the bibliographic record, based on the cataloging information, and, if any of the cataloging information is altered, can be reconstructed in the same way.

The 008 field: The 008 field is referred to as Fixed-Length Data Elements, or Fixed Field Codes. Its 40 characters contain important information, but in an abbreviated form. Although it is not yet used to its fullest in online catalog systems, this field can be used to identify and retrieve records matching specific criteria. For example, there is a code in this field to indicate whether a book is large-print, a code to identify the country of publication, a code to identify juvenile materials, a code to indicate the language of the text, and so on.

3.1.1.2. MARC21 Sample Records

After the definitions of the MARC21 structural elements, below is a bibliographical record presented in various formats [1].

MARC 21 communications format: The block of data below is what the programmer sees when he looks at the contents of a MARC file. The tags do not appear before the fields, but a directory to the data tells which tags should be used and where each field starts (in other words, where each tag belongs). Figure 3.1 shows an example MARC21 communications format file.

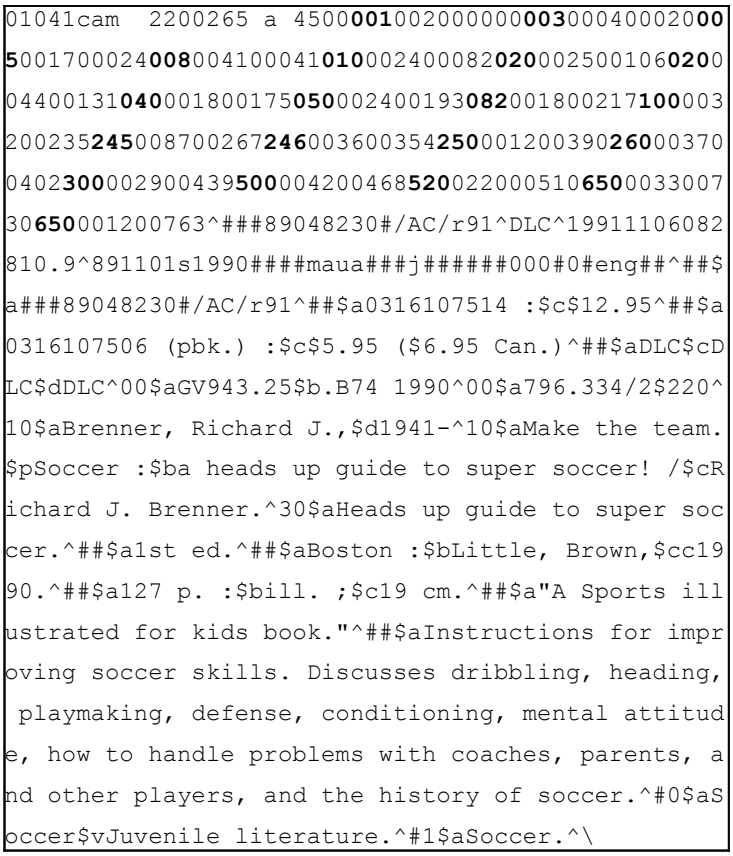


Figure 3.1. A sample record in MARC communication format

Usually, only the computer programmer and the computer come into contact with the record in MARC 21 communications format, but it is interesting to understand how the directory works.

The first 24 positions are the leader. In this example the leader fills approximately 1/3 of the first line and ends with "4500." Immediately following the leader, the directory begins. Tags have been underlined in this example. Each individual tag directory is 12 characters long. The first tag is 001. Following each tag, the next four positions show the length of the field. The data in the 001 field (control number) in this record is 20 characters long. The next 5 positions tell the starting point for this field within the data string that follows the directory. The 001 field begins at the 00000 position (the first position is position 0). The next tag is 003, which is 4 characters long and begins at the 20th position (the length of the previous position -- 20 -- added to its starting spot -- 00000 -- equals 20). The next

tag is 005. It is 17 characters long and begins at the 24th spot (4 + 20 = 24). The full meaning of the directory is given in Table 3.4.

Table 3.5. The meaning of the leading directory

Tag	Length	Starts at	Tag	Length	Starts at
001	0020	00000	100	0032	00235
003	0004	00020	245	0087	00267
005	0017	00024	246	0036	00354
008	0041	00041	250	0012	00390
010	0024	00082	260	0037	00402
020	0025	00106	300	0029	00439
020	0044	00131	500	0042	00468
040	0018	00175	520	0220	00510
050	0024	00193	650	0033	00730
082	0018	00217	650	0012	00763

Field terminators (displayed as a "^" character in this example) mark the end of the directory and the end of each field that follows. Notice that the sum of the 2nd and 3rd column in any row equals the number in the 3rd column in the next row. The starting point of one field plus its length equals the starting position of the next field.

This can be verified by counting the character positions within the data, remember that spaces count, as do the field terminators (^). (Two character positions are always reserved for indicators at the beginning of a field.) A record terminator (displayed as a "\" character in this example) ends each bibliographic record.

Tagged Display: If a librarian uploaded this record into a library automation system the data entry screen might look like Figure 3.2.

```
01041cam  2200265 a 4500
001 ###89048230
003 DLC
005 19911106082810.9
008 891101s1990    maua  j      001 0 eng
010 ## $a ###89048230
020 ## $a 0316107514 :
       $c $12.95
020 ## $a 0316107506 (pbk.) :
       $c $5.95 ($6.95 Can.)
040 ## $a DLC
       $c DLC
       $d DLC
050 00 $a GV943.25
       $b .B74 1990
082 00 $a 796.334/2
       $2 20
100 1# $a Brenner, Richard J.,
       $d 1941-
245 10 $a Make the team.
       $p Soccer :
       $b a heads up guide to super soccer! /
       $c Richard J. Brenner.
246 30 $a Heads up guide to super soccer
250 ## $a 1st ed.
260 ## $a Boston :
       $b Little, Brown,
       $c c1990.
300 ## $a 127 p. :
       $b ill. ;
       $c 19 cm.
500 ## $a "A Sports illustrated for kids book."
520 ## $a Instructions for improving soccer skills. Discusses
          dribbling, heading, playmaking, defense, conditioning,
          mental attitude, how to handle problems with coaches,
          parents, and other players, and the history of soccer.
650 #0 $a Soccer
       $v Juvenile literature.
650 #1 $a Soccer.
```

Figure 3.2. The record in figure 3.1 formatted in tagged display

3.1.2. MARC-XML

The Library of Congress' Network Development and MARC Standards Office is developing a framework for working with MARC data in a XML environment. This framework is intended to be flexible and extensible to allow users to work with MARC data in ways specific to their needs. The framework will contain many components such as schemas, style sheets, and software tools developed and maintained by the Library of Congress [2].

MARC XML could potentially be used as follows:

- for representing a complete MARC record in XML.
- as an extension schema to METS (Metadata Encoding and Transmission Standard) [6].
- to represent metadata for OAI harvesting.
- for original resource description in XML syntax.
- for metadata in XML that may be packaged with an electronic resource.

Some MARC XML advantages are:

- The schema supports all MARC encoded data regardless of format.
- The MARC XML framework is a component-oriented, extensible architecture allowing users to plug and play different software pieces to build custom solutions.

There are certain design considerations in MARC-XML:

Simple and Flexible MARC XML Schema: The core of the MARC XML framework is a simple XML schema which contains MARC data. This base schema output can be used where full MARC records are needed or act as a "bus" to enable MARC data records to go through further transformations such as toDublin Core and/or processes such as validation. The MARC XML schema will not need to be edited to reflect minor changes to MARC21. The schema retains the semantics of MARC.

All control fields, including the leader are treated as a data string. Fields are treated as elements with the tag as an attribute and indicators treated as attributes. Subfields are treated as sub elements with the subfield code as an attribute.

Lossless Conversion of MARC to XML: All of the essential data in a MARC record is converted and expressed in XML. MARC structural elements, such as the length of field and starting position of field data in directory entries are not needed in the XML record. Leader data positions not needed in the XML environment are retained as place holders or carried as blanks.

Roundtripability from XML back to MARC: As a consequence of the lossless conversion from MARC, information in a MARC XML record enables recreation of a MARC record without loss of data. A MARC record can also be created without data loss from MARC XML records.

Data Presentation: Once MARC data has been converted to XML, data presentation is possible by writing a XML stylesheet to select the MARC elements to be displayed and to apply the appropriate markup.

MARC Editing: Some single or batch updates such as adding, updating, or deleting a field to a MARC record can be accomplished with simple XML transformations

Data Conversion: Most data conversions can be written as XML transformations. For more complex transformations of the data, software tools which read MARC XML can be written.

Validation of MARC data: Validation with this schema is accomplished via a software tool. This software, external to the schema, will provide three possible levels of validation:

- Basic XML validation according to the MARC XML Schema
- Validation of MARC21 tagging (field and subfield)
- Validation of MARC record content, e.g., coded values, dates, and times.

Extensibility*:* By using XML as the structure for MARC records, users of the MARC in the XML framework can more easily write their own tools to consume, manipulate, and convert MARC data.

3.1.3. Relational Representation of MARC21

In designing a library management system, storage method of the catalog data is an issue. Above, we discussed possible representation of the catalog data. If we were to use one the above record formats, we would face the following obstacles:

- *Problems with MARC21:* The MARC21 record format is a fully textual format. If we store all the cataloging data in MARC21, the database size would tremendously large. This would make accessing the database extremely slow and programmatically hard. Read/write and update operations would require extreme indexing features, and the search operations would be sluggish.
- *Problems with MARC-XML*: Although MARC-XML offers many advantages and better standardization and exchange capability, it comes with problems as well. Accessing the record is relatively is, inserting, deleting or updating the records is subject to XML queries. Searching the catalog database is again programmatically easy. As the MARC-XML is still a text based representation, access is slow. XML Database products are not yet mature enough to handle large amounts of data.

Taking into account the above problems, a more efficient method should be used for catalog data storage. The method we choose is to use a relational database [7]. The advantages of using a relational database system are:

- Faster access to data.
- Easily queries for specific data.
- Built in systems for multiple requests and concurrent connections.
- Built in security and privilege systems.
- Application independence.
- Easier to program with.
- Expertise and several products.
- A relational database management system offers a better solution to hierarchical, part/whole and other relationships among bibliographic records by allowing explicit links from one record to another.

Although these significant advantages, there is an important overhead exposed when using relational representation. The data should be converted to MARC21 whenever there is a need to exchange data. Designing a proper relational scheme is

not easy either. Our database scheme is in table 3.5. Please note that all attributes of the table are marked as a composite primary key.

Table 3.6. The definition for relational representation of MARC data.

Attribute	Data Type	Other	Description
Materialid	integer	primary key	Identifies which material the data belongs to.
Tagid	Small integer	primary key	Identifies which tag the data belongs to.
Tag	char	primary key	Tag number of the field
Indicator1	char	primary key	First indicator of the field
Indicator2	char	primary key	Second indicator of the field
Subfield	char	primary key	Subfield information of the field
Field	varchar	primary key	MARC field.

A sample MARC record and its representation in Relational MARC are given in Figures 3.3 and 3.4.

```
001   1122240
005   19971106095945.2
008   940623s1992 tu a b 000 0 eng
020   __ |z 97576870801
050   00 |a DR739.C57 |b G85 1992
100   1_ |a Gülersoy, Çelik.
245   14 |a The Çerâgan palaces / |c Çelik Gülersoy.
260   __ |a Sultanahmet, Istanbul : |b Istanbul Kitapligi, |c 1992.
300   __ |a 208 p. : |b ill. (some col.) ; |c 22 X 31 cm.
500   __ |a "Publications of the "Istanbul Library."
504   __ |a Includes bibliographical references (p. 206-208).
610   20 |a Çiragan Sarayi (Istanbul, Turkey)
651   _0 |a Istanbul (Turkey) |x Buildings, structures, etc.
651   _0 |a Turkey |x History |y Tanzimat, 1839-1876.
651   _0 |a Turkey |x History |y 1878-1909.
651   _0 |a Turkey |x Court and courtiers.
```

Figure 3.3. A sample MARC record and its relational representation

20	0	020	#	#	z	97576870801 :
20	0	050	0	0	a	DR739.C57
20	0	050	0	0	b	G85 1992
20	0	100	1	#	a	Gülersoy, Çelik.
20	0	245	1	4	a	The Çerpağan palaces /
20	0	245	1	4	c	Çelik Gülersoy.
20	0	260	#	#	a	Sultanahmet, İstanbul :
20	0	260	#	#	b	İstanbul Kitaplığı,
20	0	260	#	#	c	1992.
20	0	300	#	#	a	208 p. :
20	0	300	#	#	b	ill. (some col.) ;
20	0	300	#	#	c	22 X 31 cm.
20	0	500	#	#	a	"Publications of the "Istanbul Library."
20	0	504	#	#	a	Includes bibliographical references (p. 206-208).
20	0	610	2	0	a	Çırağan Sarayı (Istanbul, Turkey)
20	0	651	#	0	a	Istanbul (Turkey)
20	0	651	#	0	x	Buildings, structures, etc.
20	1	001	#	#	#	1122240
20	1	005	#	#	#	19971106095945.2
20	1	008	#	#	#	940623s1992 tu a b 000 0 eng
20	1	651	#	0	a	Turkey
20	1	651	#	0	x	History
20	1	651	#	0	y	Tanzimat, 1839-1876.
20	2	651	#	0	a	Turkey
20	2	651	#	0	x	History
20	2	651	#	0	y	1878-1909.
20	3	651	#	0	a	Turkey
20	3	651	#	0	x	Court and courtiers.

Figure 3.4. Relational representation of the MARC data in figure 3.3

In our Library On-Line implementation we used the above relational MARC implementation. The evaluation of this representation is given in later chapters.

3.2. Catalog Exchange

The standardized MARC formats MARC21 and MARC-XML makes exchanging of catalog records a lot easier. There are two catalog access protocols, Z39.50 for MARC21 exchange and ZING for MARC-XML exchange.

3.2.1. Z39.50

The Z39.50 standard defines a client/server based service and protocol for Information Retrieval. It specifies procedures and formats for a client to search a database provided by a server, retrieve database records, and perform related

information retrieval functions. The protocol addresses communication between information retrieval applications at the client and server; it does not address interaction between the client and the end-user [8]. Figure 3.4 shows a simple flow on how Z39.50 works.

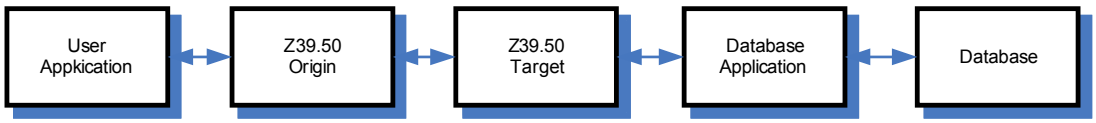


Figure 3.5. How Z39.50 works

Z39.50 is the formal designation of an international standard protocol that facilitates communication between a local software client and a remote information retrieval system. During the 1960s and early 1970s, there emerged a vision among library leaders in the United States to create a national bibliographic network, in order to share resources electronically, and especially cataloging records standardized in the MARC format. It is against this backdrop that Z39.50 was conceived.

The number, Z39.50, was assigned in 1988 by the National Information Standards Organization (NISO), a standards developer accredited by the American National Standards Institute (ANSI). Currently, development of the standard is the responsibility of a Z39.50 Implementer's Group (ZIG) composed of commercial database vendors, librarians and other interested parties. The latest iteration is Z39.50-1995 (Version 3). The official maintainer of the standard is the U.S. Library of Congress.

In this decade, as widespread adoption of the Internet opened possibilities for sharing other forms of networked information, Z39.50 expanded in scope to become a general IR (information retrieval) protocol. It increasingly serves diverse user communities, including museum professionals, those interested in working with geospatial records, government information, etc.

William Moen [9] offers the following description of Z39.50: “Z39.50 is a computer-to-computer communications protocol designed to support searching and retrieval of information -- full-text documents, bibliographic data, images, multimedia -- in a distributed network environment. Based on client/server architecture and operating over the Internet, the Z39.50 protocol is supporting an increasing number of applications. And like the dynamic network environment in which it is used, the standard is evolving to meet the changing needs of information creators, providers, and users.”

In consideration of the practical dimension, he observes that: “Z39.50 supports information retrieval in a distributed, client and server environment where a computer operating as a client submits a search request (i.e., a query) to another computer acting as an information server. Software on the server performs a search on one or more databases and creates a result set of records that meet the criteria of the search request. The server returns records from the result set to the client for processing. The power of Z39.50 is that it separates the user interface on the client side from the information servers, Search Engines, and databases. Z39.50 provides a consistent view of information from a wide variety of sources, and it offers client implementers the capability to integrate information from a range of databases and servers.”

Features and Services provided by Z39.50: In addition, it is important to note that Z39.50 provides for an articulated sequence of steps in a search session between two IR systems, known, respectively as the *origin* and the *target*, or what we commonly think of as the *client* and the *server*. The basic sequence of these interactions is as follows:

- Init: In this phase of the session, the two IR systems establish baseline parameters for intercommunication, such as determining the maximum file size that will be returned.

- Search: Here the query is formulated and sent to the target system.
- Present: Results are returned to the client and displayed to the user.

There are many services that can be facilitated by Z39.50. For example, complex Boolean searching, keyword searching, proximity searching, truncation or focusing on a particular field or subfield of a formatted record (e.g., MARC21 or GILS). It is vital to maintain a separation in one's thinking between Z39.50 proper and any particular IR system. Z39.50 by itself is a strictly a communications protocol. It can enable intercommunication between, say, a stand-alone software client and a remote database (or databases) returning the results in any number of formats. The target could be a library catalog (i.e., an OPAC), a database containing cultural heritage information in a museum setting, a collection of CD-ROM bibliographic citation databases, or even a collection of Web pages incorporating XML tags. What Z39.50 most essentially provides is a layer of abstraction between any two IR systems and a common language for interoperability. A Z39.50 query does not directly address the target database or databases. It talks to a Z39.50 server, which, in turn, translates the query, results and so forth, into the language of the Z39.50 protocol.

In addition, to what has already been mentioned, Z39.50 can provide for:

- Authentication: Searches can be restricted to authorized users only.
- Explain: This step provides the user a way of asking the target system to transmit human readable information about its particular capabilities.
- Definition of Record Formats: Records can be returned in MARC, GILS, SUTRS, etc.
- Index Browsing: Facilitates browsing within a controlled index of subject terms.

A full implementation of the Z39.50 protocol also permits the user to save a search session so that it can be run again at a later time, or to save a result set for later use. Result sets may be sorted. Databases may be updated, charges can be assessed and sessions can be closed. However, not every Z39.50 server will support all of these services.

Interoperability: A key issue in relation to Z39.50 is *interoperability*. There are challenges that must be overcome whenever one attempts to search a remote IR

System using something other than its native search interface. For example, when the user launches a search against a remote database using the term *author*, he or she may not be aware of how variable the result can be. Precision depends upon the manner in which the Z39.50 server has been mapped to different parts of the underlying record structure.

Purpose of Z39.50: It has been observed that Z39.50 lies at the very core of what libraries do; namely, when properly implemented, it affords the end user a powerful mechanism for search and retrieval. Above all, it provides a layer of abstraction, which isolates the search from any particular instantiation of, say, a database or other component of an Information Retrieval System. Instead of compelling the user to learn multiple search interfaces and query languages, Z39.50 offers simplicity. One now needs to know only a single search client, and sacrifices little in the way of power to retrieve and discover objects residing in the target system.

Strengths of the Protocol: Perhaps the primary strength of Z39.50 is that it enables the user to articulate a complex query statement just once, even if multiple systems are being searched, while masking the underlying complexity of the interaction between diverse information retrieval hardware and software systems. Working examples of such an implementation of Z39.50 would be the California Digital Library (www.cdlib.org) project or the Arts and Humanities Data Service (www.ahds.ac.uk), in the United Kingdom. These projects provide integrated, Web-based access to holdings records in databases at widely distributed sites

Among the many virtues of Z39.50 is that it;

- Permits multiple databases to be searched simultaneously by broadcasting a single search query to any number of targets.
- Is a standard accepted by a wide range of information providers/organizers
- Provides integrated access and access to distributed resources. In other words, a single search query can be launched against multiple databases, and the result set can be returned in a highly organized fashion, giving the appearance of uniformity to the end user.

- With Z39.50, in contrast to, say, an Internet Search Engine, the semantic intention of the user is preserved. Instead of returning every instance where a character string is matched, a Z39.50 client typically searches against structured data. In other words, it is possible to launch a broadcast search against a number of library catalogs where only results from the "author" field will be returned. The occurrence of non-relevant hits is thereby dramatically reduced.

Primary Limitations of Z39.50 and its Implementation by Database Vendors:

Additional limitations of Z39.50 interoperability include that;

- Libraries cannot exchange holdings and access information as yet. Although catalogs may well specify that an item has restrictions on access, is missing or damaged, this level of information would normally not be presented.
- Commercial database vendors may implement certain functions of Z39.50 and not others. The problem arises because the Z39.50 1995 (Version 3) standard only requires minimal conformance. It does not specify the details of an implementation. It is for this reason that various profiling efforts have developed.
- Suffers from many poor implementations. For example, some database vendors of library catalog software have been very vague in the way that attributes are mapped. For example, the implementers of the Vcuc project found that "[A] search using the Author attribute and author's name may return records with the author's name contained in a title, a conference name, subject, notes, etc. This type of imprecision in the records returned can be frustrating for the end user" [10].
- Costs and benefits, including the budget amount needed to implement Z39.50, are not well understood. In the case of library OPAC systems, some vendors regard Z39.50 as part of the basic functionality of their product, while others assess high fees for what they consider to be the addition of non-standard capabilities.

- Description of the Z39.50 model itself is sometimes encumbered by the unfamiliar language of the OSI Reference Model [11] (e.g. *origin* and *target* versus the currently familiar terms *client* and *server*).

What Z39.50 is* Not *Designed To Do:

- It is not meant to be a panacea for all information seeking needs. For many casual purposes, other means, such as an Internet Search Engine, may suffice. It is not meant to substitute for the kind of serendipitous browsing that can occur, for example, on the World Wide Web. It is a tool that offers precision and power to solve a particular kind of problem, and ultimately, may prove to be of greatest value to information professionals or to researchers whose information needs are highly specific.
- It should not be expected to substitute entirely for the local search interface. The main reason for this is that local systems may well deploy services that are not accessible through the Z39.50 implementation, either because the institution did not have the resources to do so or because it is technically infeasible.

Relationship of Z39.50 to Internet Search Engines: In Z39.50 and Internet Search Engines, we can see two contrasting models for networked information retrieval.

- The model for searching on the World Wide Web is centralized. Search Engines such as *Google* work by creating a vast central repository of indexed terms gathered by automated *harvesters* (also known as *spiders* or *crawlers*). Comparatively speaking, it is not too difficult to search a single server. The end user, however, is generally unaware of how these tools work and often has the illusion of distributed searching.
- Z39.50 operates according to a distributed model, with each, successive information retrieval system being searched in real time. It is both more powerful and more problematic to implement. By and large, information presently available through Search Engines is not structured; however, that could change in the future, as standardized metadata becomes more common on the Web. Relevance is determined through the Search Engine's

implementation of term weighting or collaborative filtering or some combination thereof.

In comparison to Web Search Engines, Z39.50 excels at handling structured data, such as formatted database records. It utilizes a standard set (or sets) of attributes which, in turn, are mapped to, say, MARC21 or GILS or any other standard way of structuring data.

The gradual emergence of metadata standards -- such as Dublin Core or XML tags -- could be mapped to Z39.50 Use attributes. If the use of standardized metadata should become widespread on the World Wide Web, the power of Z39.50 search clients will become readily apparent, since documents organized in this way are, essentially, structured data.

Virtual Union Catalog Projects: While Z39.50's promise is bright, significant obstacles remain. Three recent projects, attempts to create a virtual union catalog, illustrate the difficulties in achieving true interoperability between systems. In each case, what have been summarized are reports of projects that are still evolving, with the focus being just on the difficulties encountered.

- SILO (State of Iowa Libraries Online) (http://www.silo.lib.ia.us/bluang.html): This project seeks to provide integrated access to libraries in Iowa. An initial survey of Information professionals who are users of this system have indicated a range of problems, including: Inconsistency in the search results produced by different vendor implementations -- owing to the inconsistency in the way MARC tags are mapped to Z39.50 use attributes. Variation between search results obtained via the Z39.50 and those obtained from the vendor's proprietary interface. Difficulties in configuring clients. No item level detail. Lack of support for Boolean searching in some vendor products. Lack of documentation. Slow performance. Lack of functionality over native client. Affordability.
- CIC: (Committee on Institutional Cooperation): With headquarters in Champaign, Illinois, this entity comprises the academic consortium of the Big

Ten universities. A study made by the CIC group of its Z39.50 project found that "Some of the systems use the position attribute of *any position* in field to indicate a keyword search, while others ignore the position attribute completely. Some use a combination of Use and Structure attribute while others ignore the structure attribute completely and rely solely on use attributes to identify keyword searches as opposed to headings searches. When factoring in the effects of disparate handling of position and structure attributes, the *lowest common denominator space* is reduced." Moreover, the group found that "It is virtually impossible to send the exact same query to each of the CIC libraries and retrieve meaningful results from all of the servers."

Recommendations emerging from the study included: Encourage all vendors to offer support for a specific subset of the Z39.50 bib-1 Use Attributes, since vendors tended to vary in regard to which of the Z39.50 bib-1 use attributes they supported. Create and implement system-wide indexing policies. Verify support for complex Boolean searching. (Some users of the system apparently lacked confidence in the mechanism by which certain vendors provided support for Boolean searching.) Many vendors supported only the most basic Z39.50 functionality, and respondents hoped that vendors could be encouraged to continue their development in this area. Need for the "Explain" feature of Z39.50 was particularly apparent, since a good implementation of this service could ease the difficulties of configuring Z39.50 clients.

- vCuc (Virtual Canadian Union Catalog) : This project is ongoing among Canadian libraries who are using Z39.50 for searching distributed individual library catalogues and union catalogues. Major lessons learned from this initiative to date include that providing holdings and circulation information, often stored inconsistently in various OPAC records, requires a high level of coordination, both technical and administrative. The project has also encountered the problem of different database vendors mapping MARC record tags in different ways, thus raising the problem of interoperability.

It is noteworthy that most of these problems arise from problems in implementation rather than reflecting inherent limitations of the Z39.50 standard, which is not to say that they are not nonetheless genuine problems. Fortunately, for most of these issues, solutions are within sight.

Reference Profiles: Historically, Z39.50 has suffered because of uncertain or inconsistent implementation by commercial database vendors. The experiences of the virtual union catalog projects described in 2.9 illustrate how the utility of the standard has sometimes been hamstrung in the past. In an attempt to resolve these issues, different user communities are actively engaged in the development of profiles or reference standards against which vendor implementations can be measured. Significant current profiling efforts include the *Bath*, *CIMI*, *GEO* and *Z Texas* Profiles.

In the near future, as these efforts are consolidated, it will be possible for prospective purchasers of library or other information retrieval systems to specify conformance with one or more of these profiles as part of the original Request for Proposals (RFP), issued at the time when bids are being requested from vendors.

3.2.2. ZING

ZING (Z3950 International: Next Generation) covers a number of initiatives by Z39.50 implementers to make the intellectual/semantic content of Z39.50 more broadly available and to make Z39.50 more attractive to information providers, developers, vendors, and users, by lowering the barriers to implementation while preserving the existing intellectual contributions of Z39.50 that have accumulated over nearly 20 years [13]. Current ZING initiatives are SRW (including SRU), CQL, ZOOM, ez3950, and ZeeRex.

SRW/SRU (Search and Retrieve for the Web): Search and Retrieve Web Service (SRW) and Search and Retrieve URL Service (SRU) are Web Services-based protocols for querying databases and returning search results. SRW and SRU

requests and results are similar, their difference lies in the ways the queries and results are encapsulated and transmitted between client and server applications.

Both protocols define three and only three basic "operations": explain, scan, searchRetrieve:

- explain. Explain operations are requests sent by clients as a way of learning about the server's database. At minimum, responses to explain operations return the location of the database, a description of what the database contains, and what features of the protocol the server supports.
- scan. Scan operations enumerate the terms found in the remote database's index. Clients send scan requests and servers return lists of terms. The process is akin to browsing a back-of-the-book index where a person looks up a term in a book index and "scans" the entries surrounding the term.
- searchRetrieve. - SearchRetrieve operations are the heart of the matter. They provide the means to query the remote database and return search results. Queries must be articulated using the Common Query Language. CQL queries range from simple freetext searches to complex Boolean operations with nested queries and proximity qualifications. Servers do not have to implement every aspect of CQL, but they have to know how to return diagnostic messages when something is requested but not supported. The results of searchRetrieve operations can be returned in any number of formats, as specified via explain operations. Examples might include structured but plain text streams or data marked up in XML vocabularies such as Dublin Core, MARCXML, MODS, etc.

The differences between SRW and SRU lie in the way operations are encapsulated and transmitted between client and server as well as how results are returned. SRW is essentially as SOAP-ful Web service. Operations are encapsulated by clients as SOAP requests and sent to the server. Likewise, responses by servers are encapsulated using SOAP and returned to clients.

On the other hand, SRU is essentially a REST-ful Web Service. Parameters are encoded as name/value pairs in the query string of a URL. As such operations sent

by SRU clients can only be transmitted via HTTP GET requests. The result of SRU requests are XML streams, the same streams returned via SRW requests sans the SOAP envelope.

SRW and SRU are "brother and sister" standardized protocols for accomplishing the task of querying databases and returning search results. If index providers were to expose their services via SRW and/or SRU, then access to these services would become more ubiquitous.

CQL (Common Query Language): CQL, the 'Common Query Language', is a formal language for representing queries to information retrieval systems such as web indexes, bibliographic catalogs and museum collection information. The CQL design objective is that queries be human readable and human writable, and that the language be intuitive while maintaining the expressiveness of more complex languages [14].

Traditionally, query languages have fallen into two camps: Powerful and expressive languages which are neither easily readable nor writable by non-experts (e.g. SQL, PQF, and XQuery), on one hand; one the other hand, simple and intuitive languages not powerful enough to express complex concepts (e.g. CCL or google's query language). CQL's goal is to combine simplicity and intuitiveness of expression with the richness of Z39.50's type-1 query. As any good text based interface, CQL is intended to 'do what you mean' for simple, every day queries, while allowing means to express complex concepts when necessary.

CQL was designed in conjunction with SRW and is the query language in which queries are expressed for both SRW and SRU, although it is not in any way limited to these two protocols. CQL is used in both the searchRetrieve and scan operations within SRW. In searchRetrieve, it is sent in the 'query' parameter, which may include any legal CQL. On the other hand, the 'scanClause' parameter in scan may only contain a single CQL searchClause, made up of index, relation (plus optional modifiers) and term. Both of these parameters are mandatory, and as such CQL forms a solid foundation on which to build the SRW protocol.

ZOOM (Z39.50 Object Orientation Model): The ZOOM initiative presents an abstract object-oriented API to a subset of the services specified by the Z39.50 standard, also known as the international standard ISO 23950 [19]. The API is:

- Abstract because we don't want to limit its use to a single implementation language.
- Object-oriented because the services lend themselves naturally to this widespread idiom.
- For a subset of the full Z39.50 services because at this stage simplicity is more important than completeness.

The ZOOM specifications, bindings for several languages, and many implementations, are available for free browsing and download.

ZeeRex: The ZeeRex, standing for Z39.50 Explain, Explained and Re-Engineered in XML, specifications are an attempt to solve two separate but related problems in the Z39.50 world. The first is that of finding Z39.50 resources - servers and their databases - for a client to use. The second is that once a resource is known, it's necessary to figure out exactly what it's capable of doing [16]. ZeeRex is not the first attempt at solving these problems; formerly there were Explain Classic and Explain Lite [8].

The problem of finding Z39.50 server and databases has traditionally been addressed simply by having poor, overworked humans building lists of servers. The difficulties with this approach are twofold:

- First, it's difficult to achieve good coverage of all available servers. There is no full list of how many Z39.50 servers are available for use.
- Second, maintaining such lists is awkward, and error-prone. It's possible to automate the process of pruning dead servers from a list, but not that of adding new servers, since there is no way to find out about them.

ZeeRex addresses these difficulties like this: Each ZeeRex record represents a single database that is available on a Z39.50 server somewhere. There may be

multiple databases on a single server, or course; in this case, there will be multiple ZeeRex records, each of them describing a single database.

When a Z39.50 client retrieves an ZeeRex record using the element set b (that is, the brief record), an F&N record is returned. It is very simple: an XML document with an <explain> element at the top level, a <serverInfo> element within it, and <host>, <port> and <database> elements within that, containing the hostname, IP port number and database name respectively of a Z39.50 database which may be on the same server as the ZeeRex record or a different one.

When a Z39.50 client retrieves an ZeeRex record using the element set f (that is, the full record), a full ZeeRex record is returned. In addition to the <serverInfo> section also found in F&N records, full records may also include the following sections:

- <databaseInfo>: contains human-readable information about the database: its title, a description, the address of a contact person, etc.
- <metaInfo>: information about the ZeeRex record itself: when it was created or last modified, when it was aggregated) if at all, etc.
- <indexInfo>: information about how to search in the database: which indexes exist and what combinations of attributes may be used to search against them, which indexes can be used for sorting, scan, etc.
- <recordInfo>: information about which record syntaxes the database can serve records in, and which element sets are supported.

ZeeRex records are found in ZeeRex databases. These are completely ordinary Z39.50 databases held on ordinary Z39.50 servers; the usual Z39.50 search and retrieval facilities may be used on them in the usual way.

ez39.50: ez39.50 stands for Simple Implementation of Z39.50 over SOAP using XER [17]. XER provides a mechanism to allow us to provide Z39.50 support over an alternative "internet" protocol without any additional amendments to the current ASN.1 standard. This would also allow any future amendments or additions to Z39.50 over this alternate protocol. At present the main contender for an XML

based client/server protocol is SOAP 1.1 [18]. A more versatile W3C recommendation is under development called XP [19].

CHAPTER 4. ISSUES IN TRANSACTION PROCESSING

A transaction is a unit of program execution that accesses and possibly updates various data items. A transaction is initiated by a user program. Transaction processing is an essential issue in designing and developing library management systems. This issue rises from the following:

Amount of Data Processed: As discussed on the previous chapters, library management system handle huge amounts stored data, especially in bibliographic catalog databases. In most of the cases the date transferred to the client may reach big sizes. This typically occurs in catalog searches. Search queries with fewer criterions tend to return hundreds of matches from the catalog database. There occurs an overhead in transferring the data from the server to the client.

Number of Connections: Generally, the library catalog searches are conducted online mainly from the Web OPAC interface in and outside the library location. The concurrent connections to the database may reach up to hundreds transactions per second. This slows down the system, sometimes making it unavailable.

As well as undertaking physical database design, we should have a clear idea of the intended use of the database by defining the queries and transactions that are expected to run on the database. For each query and we should specify the following:

1. The tables that will be accessed by the query.
2. The attributes on which any selection conditions for the query are specifed.
3. The attributes on which any join condition or conditions to link multiple tables or objects for the query are specified.
4. The attributes whose values will be retrieved by the query.

The attributes listed in items 2 and 3 are candidates for definition of access structures. For each update transaction or operation, we should specify the following:

1. The tables that will be updated.
2. The type of operation on each files (insert, update or delete).
3. The attributes on which selection conditions for a delete or update are specified.
4. The attributes whose values will be changed by an update operation.

Another issue is analyzing the expected frequency of invocation of queries and transactions. The frequency information, along with the attribute information collected on each query and transaction is used to compile a cumulative list of expected frequency of use for all queries and transactions. For our case we used the "80 – 20 rule" [20] which states that approximately 80 percent of the processing is accounted for by only 20 percent of the queries and transactions. Therefore, we did not collect exhaustive statistics on the queries and transactions.

Yet another issue is the design of indexes as the performance of queries largely depends on the choice of indexes.

For some of the optimizations we took advantage of the Java Database Connectivity (JDBC) [21] API of the Java platform. The JDBC API is a thin API, which wraps SQL and query responses in an object layer. The API is contained in package java.sql. Most JDBC drivers execute the following flow:

- Load a JDBC driver.
- Establish a connection to the database.
- Optionally interrogate the database for what capability subset it has.
- Optionally retrieve schema meta-information.
- Construct an SQL or callable statement object, and send the queries.
- Process query results; the transaction completes when the results have all been retrieved.
- Close the connection.

JDBC is based on ODBC from Microsoft [22]. There is a great deal of conformance between the two.

An important component of a JDBC implementation is the JDBC driver. The driver establishes the connection to the database, and implements any protocol required to move queries and data between the client and the server. From an API perspective, the driver does this by manufacturing an object that implements the java.sql.Connection interface for the client to apply JDBC calls against. From a protocol perspective, the Connection object created by the driver must either convert query transport requests into the DBMS native API calls, or marshal the requests into a stream and send them to a remote middleware component.

In addition the driver must therefore perform data type mapping between JDBC and database types, and interpret SQL escape sequences. The JDBC specifications permits data type extensions, so the driver can map vendor-specific database types to a vendor specific package of Java types.

Here, a number of solutions to are provided to solve the transaction of huge amounts of data from a relational database system to the client.

4.1. Portioning Catalog Tables

Using the Relational MARC representation strategy discussed in 3.1.3. access and storage of the bibliographic catalog database is simplified. As far as transaction processing speed is concerned, there is a very significant optimization that might be applied to this approach: Portioning the tables that hold catalog data with respect to the usage statistics of the individual MARC tags.

There are hundreds of distinct MARC tags that define distinct semantics. Storing all of the MARC tags for a specific record in a single table using the relational MARC approach described previously is a burdensome practice. Some records may take up to a hundred tuples for a single material. Studies show that only some certain tags are used in actual MARC records and only some of the tags are

actually valuable for catalog searches. Portioning catalog tables is based on this information. Table 4.1 shows search statistics for Library ON-LINE, a library management software.

Table 4.1. Most frequently used search criteria

Search Criteria	# of Searches / Month	Percentage (Approx.)
Title	7.899	34%
Author	7.712	33%
Subject	5.920	25%
ISBN/ISSN	1.056	4,5%
Call Number	397	1,7%
Total	22.984	100%

The other searches like publisher, language or abstract are very minimal. It is obvious that MARC tags representing the above search criteria may be placed in a separate table while the rest of the record is stored in another one. We can describe the procedure as follows:

1. Determine which tags are most frequently needed in catalog searches as catalog search is the most resource consuming function in the system.
2. Place the most frequently used tags (for our case we took 25% of the cumulative number of searches as a threshold) in a separate table.
3. For the rest of tags repeat steps 1 and 2.

The portioning of table is given in Table 4.2.

Table 4.2. Portioning MARC21 tags

Table 1	Table 2	Table 3
020: ISBN 100: Author 245: Title 250: Edition 600, 610, 650: Subject	Everything but table 1 and table 3	All 9XX tags

After the above procedure is applied in our case, we created three separate tables. One for the title, author, subject, ISBN/ISSN and call number fields, second table for the 9XX fields (local cataloging data which are virtually never used in public searches but still needed by the library staff), and third for the remaining tags.

The benefit of above approach is gaining on performance. Table 1 is only 27% of all records. The performance gain is obvious.

However, not all partition approaches wins performance. Programmatically, while displaying, searching or editing the data, there is a need for multiple SQL queries rather than one. This may slow down the system and take back what is gained from portioning. Also, this procedure reintroduces a partial functional dependency or a transitive dependency into the table.

4.2. Indexing of Catalog Tables

The attributes whose values are required in equality or range conditions and those that are keys or that participate in join conditions require access paths.

The performance of queries largely depends upon what indexes exist. On the other hand, during insert, update or delete operations, existence of indices adds to the overhead. This overhead must be justified in terms of the gain in efficiency by expediting queries and transactions. Design considerations for indexes might be categorized into the following categories:

1. *Whether to index an attribute:* The attribute must be a key, or there must be some query that uses that attribute either in a selection condition (equality or range of values) or in a join. One factor in favor of setting up many indexes is that some queries can be processed by just scanning the indexes without retrieving any data.
2. *What attribute or attributes to index on:* An index can be constructed on one or more attributes. If multiple attributes from the relation are involved together in several queries, a multi-attribute index is warranted. The ordering of attributes within a multi-attribute index must correspond to the queries.
3. *Whether to set up a clustered index:* At most one index per table can be primary or clustering index because this implies that the file be physically ordered on that attribute. If the attribute is a key, a primary index is created,

whereas a clustering index is created if the attribute is not a key. If a table requires several indexes, the decision about which should be a clustered index depends upon whether keeping the table ordered on that attribute is needed. Range queries benefit a great deal from clustering. If several attributes require range queries, relative benefits must be evaluated before deciding which attribute to cluster on. If a query is to be answered by doing an index search only (without retrieving data records), the corresponding index should not be clustered, since the main benefit of clustering is achieved when retrieving the records themselves.

Apart from designing the indexes, tuning them for performance is an important issue. After choosing the index with the above considerations, the choice may have to be revised for the following reasons:

- Certain queries may take too long to run for lack of an index.
- Certain indexes may not get utilized at all.
- Certain indexes may be causing excessive overhead because the index is on an attribute that undergoes frequent changes.

All of the above issues should be taken into consideration when deciding on the indexes. Remembering our relational MARC model Table 4.3 summarizes the index design.

Table 4.3. Relational MARC and indexes

Attribute	**Data Type**	**Other**	**Description**
Materialid	integer	primary key	Identifies which material the data belongs to.
Tagid	Small integer	primary key	Identifies which tag the data belongs to.
Tag	char	primary key	Tag number of the field
Indicator1	char	primary key	First indicator of the field
Indicator2	char	primary key	Second indicator of the field
Subfield	char	primary key	Subfield information of the field
Field	varchar	primary key INDEX	MARC field.

We see that only the *field* attribute is selected as an index. All other attributes are part of the composite primary key. This is because all these fields should be unique for a record. The field attribute is where all the information of specific tag is stored. For materialid, tagid, tag, indicator1, indicator2 and subfield attributes only equivalence selections are executed, but for field, equivalence and range selections are performed.

4.3. Data Sources and Connection Pools

In a multi-user, high volume database like library catalog databases, connection to the database from the user software may happen to be a overhead. Many application platforms support data sources and connection pools to the databases to overcome the overhead cost of connecting to the database for each and every query.

In our case we used the Java programming language and its Java 2 Enterprise Edition platform for developing enterprise applications.

Creating and maintaining database connections is an expensive operation in terms of performance. By allowing the J2EE application server to manage a data source's connection pool, we drastically alleviate the overhead cost of creating brand new connections and get the robustness provided by the application server's built-in JDBC service.

An object that implements the DataSource interface will typically be registered with a naming service based on the Java Naming and Directory (JNDI) API.

The DataSource interface is implemented by a driver vendor. There are three types of implementations:

- *Basic implementation:* Produces a standard Connection object
- *Connection pooling implementation:* Produces a Connection object that will automatically participate in connection pooling. This implementation works with a middle-tier connection pooling manager.

- *Distributed transaction implementation:* Produces a Connection object that may be used for distributed transactions and almost always participates in connection pooling. This implementation works with a middle-tier transaction manager and almost always with a connection pooling manager.

In our implementation we used the connection pooling implantation. On initial startup of the application, the system opens a pre-specified number of connections to the database. Whenever an application module needs to execute an SQL query, the module is granted an open connection from the pool. After the execution is completed, the application module returns the connection back to the pool.

If more connections are needed than the connection opened at the initialization, the connection pool manager creates more connections to be used.

One important issue with this approach is that, connections grabbed from the pool are not always returned back and idle connections cause the pool manager to create redundant connections. There two main reasons for that:

- Programmatically, the developer forgets to include the needed code for returning the acquired connection. This leaves the connection idle and marked as used.
- The Java platform does not guarantee that the connection is returned to the pool even if the connection release code is included. This is an issue with the garbage collection mechanism of the Java Virtual Machine. Connection release code is typically included in the finally block of the try catch statements. The Java VM does not guarantee that the finally block is executed during destruction.

Considering those issues, connection pools have idle connection time out mechanisms. If a connection is acquired by an application module, and is idle for a specified amount of time, pool manager automatically takes the connection back to the pool.

Using data sources and connection pools affect the performance of queries in a very positive way. However there are issues to be considered when implementing connection pools. The initial pool size and pool enlargement ratio should be configured properly. Table 4.4 gives an example configuration used in Library ON-LINE.

Table 4.4. Example connection pool configuration

Attribute	**Value**
Initial Pool Size	32
Pool Enlargement Increment	4
Maximum Pool Size	128
Connection Time Out	30000 milliseconds

Connection pools are good for performance. When using connection pools, the need for opening and closing a connection to the database is omitted. Acquiring an already-open connection is much faster than creating a new connection.

4.4. Pre-Compiled SQL Queries

DBMSs accept SQL queries from many clients concurrently and execute the queries as efficiently as possible against the data. Processing statements can be an expensive operation but databases are now written in such a way so that this overhead is minimized.

When a database receives a statement, the database engine first parses the statement and looks for syntax errors. Once the statement is parsed, the database needs to figure out the most efficient way to execute the statement. This can be computationally quite expensive. The database checks what indexes, if any, can help, or whether it should do a full read of all rows in a table. Databases use statistics on the data to figure out what is the best way. Once the query plan is created then it can be executed by the database engine.

It takes CPU power to do the access plan generation. Ideally, if we send the same statement to the database twice, then we'd like the database to reuse the access plan for the first statement. This uses less CPU than if it regenerated the plan a second time.

Things can get more complicated when we use a J2EE server. Normally, a prepared statement is associated with a single database connection. When the connection is closed, the preparedstatement is discarded. Normally, a fat client application would get a database connection and then hold it for its lifetime. It would also create all prepared statements eagerly or lazily. Eagerly means that they are all created at once when the application starts. Lazily means that they are created as they are used. An eager approach gives a delay when the application starts but once it starts then it performs optimally. A lazy approach gives a fast start but as the application runs, the prepared statements are created when they are first used by the application. This gives an uneven performance until all statements are prepared but the application eventually settles and runs as fast as the eager application. Which is best depends on whether you need a fast start or even performance.

The problem with a J2EE application is that it can't work like this. It only keeps a connection for the duration of the request. This means that it must create the prepared statements every time the request is executed. This is not as efficient as the fat client approach where the prepared statements are created once, rather than on every request. J2EE vendors have noticed this and designed connection pooling to avoid this performance disadvantage.

When the J2EE server gives your application a connection, it isn't giving you the actual connection; you're getting a wrapper. You can verify these by looking at the name of the class for the connection you are given. It won't be a database JDBC connection, it'll be a class created by your application server. Normally, if you called close on a connection then the jdbc driver closes the connection. We want the connection to be returned to the pool when close is called by a J2EE application. We do this by making a proxy jdbc connection class that looks like a real connection. It has a reference to the actual connection. When we invoke any method on the connection then the proxy forwards the call to the real connection. But, when we call methods such as close instead of calling close on the real connection, it simply returns the connection to the connection pool and then marks the proxy connection as invalid so that if it is used again by the application we'll get an exception.

Wrapping is very useful as it also helps J2EE application server implementers to add support for prepared statements in a sensible way. When an application calls Connection.prepareStatement, it is returned a PreparedStatement object by the driver. The application then keeps the handle while it has the connection and closes it before it closes the connection when the request finishes. However, after the connection is returned to the pool and later reused by the same, or another application,, then ideally, we want the same PreparedStatement to be returned to the application.

J2EE PreparedStatement Cache is implemented using a cache inside the J2EE server connection pool manager. The J2EE server keeps a list of prepared statements for each database connection in the pool. When an application calls prepareStatement on a connection, the application server checks if that statement was previously prepared. If it was, the PreparedStatement object will be in the cache and this will be returned to the application. If not, the call is passed to the jdbc driver and the query/preparedstatement object is added in that connections cache.

We need a cache per connection because that's the way jdbc drivers work. Any preparedstatements returned are specific to that connection.

If we want to take advantage of this cache, the same rules apply as before. We need to use parameterized queries so that they will match ones already prepared in the cache. Most application servers will allow you to tune the size of this prepared statement cache.

To summarize, most relational databases handle a JDBC / SQL query in four steps:

1. Parse the incoming SQL query
2. Compile the SQL query
3. Plan/optimize the data acquisition path
4. Execute the optimized query / acquire and return data

A Statement will always proceed through the four steps above for each SQL query sent to the database. A PreparedStatement pre-executes steps (1) - (3) in the execution process above. Thus, when creating a PreparedStatement some pre-optimization is performed immediately. The effect is to lessen the load on the database engine at execution time.

While evaluating, we have seen that prepared statements affect performance most obviously in for loops. When used with caching mechanism as described in the coming sections, the performance is up to 70% better off.

4.5. Caching Query Result Sets

Implicitly, JDBC drivers can perform caching of various kinds to speed up queries. Some kinds are:

- Row pre-fetching: When an application module retrieves a result set, the rows are not actually transferred until you request them one by one. Some drivers perform row pre-fetching in anticipation that you will ask for the next row before the application actually does it.
- Row caching: This is related to row pre-fetching; the driver retrieves a block of rows at a time.
- Connection caching: A driver may maintain a s et of reusable database connections for a given user; clients that connect as this user transparently reuse this persistent connection, thereby avoiding database connection time and reducing the overall server-side footprint per client.
- Schema Caching: Drivers have all the job of translating SQL from generic SQL to vendor's specific SQL. To do this, they often need to consult the database schema. A driver may perform schema caching to speed this up.

We implemented all of three caching mechanism. For evaluation, we have executed 10 distinct select queries, 6 of them being complex join queries. For an average, after the first execution of the query, the second execution is up to %300 times faster. Evaluation is given in more detail in chapter 8.

CHAPTER 5. ISSUES IN DEVELOPMENT PROCESS, CATALOG DATA CONVERSION, INTEROPERABILITY AND INTEGRATION

In this chapter, we discuss the issues in development process employed along with catalog data conversion, interoperability and system integration.

5.1. Development Process

When we provide a service or a product, whether it be developing software, or writing a report we follow a sequence of steps to accomplish a set of tasks. These tasks are usually performed in the same order each time. Thus, we can think of a set of ordered tasks as a process: a series of steps involving activities, constraints and resources that produce an intended output of some kind. Any process has the following characteristics:

- The process prescribes all of the major process activities.
- The process uses resources, subject to a set of constraints (such as a schedule), and produces intermediate and final products.
- The process may be composed of sub-processes that are linked in some way. The process may be defined as a hierarchy of processes, organized so that each sub-process has its own process model.
- Each process activity has entry and exit criteria, so that we know when the activity begins and ends.
- The activities are organized in a sequence, so that it is clear when one activity is performed relative to the other activities.
- Every process has a set of guiding principles that explain the goals of each activity.
- Constraints or controls may apply to an activity, resource, or product.

So why is a software process important? The concern for quality has been on the increase inmost industrial sectors. In addition, awareness of the central importance of the production processes has been increasing. Process are important because industry cares about their inherent qualities, such as performance across different projects and productivity, with the aim of improving time to market and reducing production costs.

The relationship between processes and the quality of the products hold especially in software production, because of the intrinsic nature of software. If an explicit process is in place, software development proceeds in a predictable and orderly fashion, reducing the chance of introducing faults into the product and providing a means for controlling the quality of what is being developed.

In developing Library ON-LINE, we followed the Extreme Programming software development model [3] [23] [24]. The design and development of the software has costed only 3 man/months and 90% of the work was done by only one person. Considering that the project size is over 60.000 lines of hard coded Java code, the project time is very short. This advantage comes from the method XP answers the changing requirements. The used development processes and methods are the key to this short project size. Here we take a look at them.

5.1.1. Extreme Programming

Extreme Programming has some very obvious advantages when compared to the traditional approaches.

We discussed the definition and characteristics of a process above. When the process involves building a software product, it is called a software lifecycle, because it describes the life of a software product from its conception to its implementation, delivery, use and maintenance. OD-STD-2167A/498, the current prevailing standard guiding software development, has been interpreted as mandating as specific process for use on all military acquisitions. This process is represented by the "Waterfall Model", which serves as the conceptual guideline for almost all Air

Force and NASA software development. The waterfall model was first describer by Kelley [25]. The waterfall model is an activity-centered life cycle model that prescribes a sequential execution of a subset of the development processes and management processes. The requirement activities are all completed before the systems design activity starts. The goal is to never turn back once an activity is completed. The key feature of his model is the constant verification activity that ensures that each development activity does not introduce unwanted or deletes mandatory requirements. Many of the phases require successful completion of a government review process. Critics of the "Waterfall" Model, in fact, find that the model is geared to recognize documents as a measure of progress rather than actual results.

The nine major activities described in 2167A/498 are as follows: [26]

1. Systems Concept/System Requirements Analysis
2. Software Requirements Analysis
3. Software Parametric Cost Estimating
4. Preliminary Design
5. Detailed Design
6. Coding and Computer Software unit (CSU) Testing
7. Computer Software Component (CSC) Integration and Testing
8. Computer Software Configuration Item (CSCI) Testing
9. System Integration and Operational Testing

As a response to rapid-change in requirements and the increase of failures in software development projects due to the incapability of traditional software engineering approaches like the waterfall model above to this change, new agile development methodologies are introduced. Extreme Programming (XP) is one of those methodologies. It is described as “a lightweight discipline of software development, which is designed for use with small teams who need to develop software quickly in an environment of rapidly-changing requirements, based on principles of simplicity, communication, feedback, and courage.” Most significant and major difference of XP from traditional software engineering methodologies is; XP focuses on the product whereas process is focused by many of the other software development methodologies. XP is based on rapid application development and it is

informal and verbal. Change in requirements is not considered as a problem. XP is based on some practices that are easy to follow but not strict and formal rules. XP Practices are grouped in four; *Planning Practices*, *Designing Practices*, *Coding Practices* and *Testing Practices* [3,23].

XP improves a software project in four essential ways; communication, simplicity, feedback, and courage. XP programmers communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested. With this foundation XP programmers are able to courageously respond to changing requirements and technology.

The major distinction between XP and other approaches is that; XP focuses on the product whereas traditional SE approaches focus on the process. The idea is to develop hundred percent working software products no matter how you develop it. The process is defined with practices, which are fruitful and easy to follow. Other major distinction is; changes in the requirements are welcomed at any stage in an XP project but change in the requirements is controlled and avoided in other SE methodologies.

McCormick compares the two approaches as given in Table 2.1. [27]

Table 5.1. Traditional software engineering approaches vs. XP

Software Engineering	**Extreme Programming**
Avoid change in coding phase	Code change is no problem
Specifications must be formal and written	Specifications may be informal and verbal
Design must be completed before coding	Code is the design
Communication is a problem for programmers	Programmers can communicate well
Change control should be enforced for requirements	Informal requirements suffice
Rapid Application Development is avoided	Rapid Application Development is favoured

As seen from the table, XP is informal and verbal, is based on rapid application development, and change is not considered a problem, whereas software engineering is formal and written, and change is controlled and avoided. Software engineering attempts to complete design before starting coding whereas XP does not devote much effort to design, and code itself is considered to be the design. In short, XP focuses on programming and on directly producing the code.

XP proves to be very useful and productive especially in cases where requirements change frequently and where architectural issues are already solved either because of the nature of the application or because of the tools and the platforms chosen. While developing Library ON-LINE, the development team worked closely with the library staff. The library of installation has never used a library management system before and was much unrelated and estranged to the usage of computer systems. This made the requirements change often. These substantial changes in the requirements were handled with the usage of XP practices which lead to time – saving conditions under many situations.

Apart from the core XP practices we used some practices introduced in [28]. These are:

- *Issue-Based Programming* is a designing practice where issues determine priorities. At any time instance during the software development project there are many issues to be resolved and many issues already resolved. At the early stages of development, most of the issues identified are related to requirements rather than design or implementation, whereas at later stages most issues are related to implementation. Still there are implementation-related issues at early stages and requirements-related issues at later stages. An issue table is maintained throughout the software development project and issues are identified as *closed* when they are resolved. A closed issue may become *open* later. The prioritization of issues is done regarding the requests of the customer and by negotiation between the customer and the development team.

- *Comment-first coding is* a coding practice. It breaks the coding of a single module into two phases: coding the semantics, and coding the syntax. When coding a module, first the algorithm of that code is written to the editor as comment lines in natural language. The level of detail should be such that the architecture of the algorithm should be understood at a glance and that converting each comment line to the valid syntax of the language used is easy for all the programmers in the team. After the first phase is completed, the output is a formatted text, which is easy to read and easy to convert to programming language's syntax. In the second phase, the programmer starts to code each construct following an *outside-in* approach. The reason why we call outside-in but not top-down is that the level of granularity is more or less the same at the end of the two phases.
- *JIT collective code ownership* is an extension of collective code ownership, which is an existing coding practice. On an XP project, any pair of programmers can improve any code at any time. This is called collective code ownership. This means each programmer is aware of everything in the project and each programmer is responsible of all parts of the code. This kind of a practice leads to efficiency in some cases such as when a programmer is waiting for a piece of code to be written by another programmer that is responsible of that code. On the other hand, the overhead of being aware of everything for each single programmer may be very high in many cases. To overcome this problem, XP teams follow a common coding standard, so that all the code looks as if it is written by a single programmer.

5.1.2. Object – Oriented Programming

While developing Library ON-LINE, the development team used an object – oriented programming language and had considerable benefits.

Object – oriented method is a software design method that models the characteristics of abstract or real objects using classes and objects. In procedural programming languages, programming tends to be action – oriented, and the unit of

programming is the function. In object – oriented programming languages the unit of programming is the class from which objects are eventually instantiated. There are many definitions of an object, such as found in [28]: "An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable". This is a "classical languages" definition, as defined in [29] where "classes play a central role in the object model", since they do not in prototyping/delegation languages. "The term object was first formally applied in the Simula language, and objects typically existed in Simula programs to simulate some aspect of reality" [28]. Other definitions referenced by Booch include Smith and Tockey: "an object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain." and [30]: "anything with a crisply defined boundary" (in context, this is "outside the computer domain". A more conventional definition appears on pg 54). Booch goes on to describe these definitions in depth. [31] defines: "An "object" is anything to which a concept applies", and "A concept is an idea or notion we share that applies to certain objects in our awareness". [32] defines: "We define an object as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand." [33] defines: "An object is an abstraction of a set of real-world things such that:

- All of the real-world things in the set - the instances - have the same characteristics
- All instances are subject to and conform to the same rules"

And on identifying objects: "What are the *things* in this problem? Most of the things are likely to fall into the following five categories: Tangible things, Roles, Incidents, Interactions, and Specifications." [28] covers "Identifying Key Abstractions" for objects and classes based on an understanding of the problem domain and [34] provides a novel approach to identifying objects through use-cases (scenarios), leading to a use-case driven design. Use cases have become very important and popular today, providing an easy way to identify objects and methods by their use in satisfying system requirements and uses. Uses cases occur throughout the development lifecycle.

The basic idea behind an object is that of *simulation*. Most programs are written with very little reference to the real world objects the program is designed to work with; in object oriented methodology, a program should be written to simulate the states and activities of real world objects. This means that apart from looking at data structures when modeling an object, we must also look at *methods* associated with that object, in other words, functions that modify the objects attributes. A *method* is an operation which can modify an objects behavior. In other words, it is something that will change an object by manipulating its variables.

A class is a blueprint for an object. What this basically means is that we provide a blueprint, or an outline of an object. This blueprint is valid whether we have one or one thousand such objects. A class does not represent an object; it represents all the information a typical object should have as well as all the methods it should have. A class can be considered to be an extremely extended TYPE declaration in the C programming language, since not only are variables held but methods too.

OO Programming method has a very distinguishing feature: Inheritance and sub-classes. A subclass is a class definition which derives functionality from another class definition. Inheritance provides a natural classification for kinds of objects and allows for the commonality of objects to be explicitly taken advantage of in modeling and constructing object systems. Natural means we use concepts, classification, and generalization to understand and deal with the complexities of the real world. See the example below using computers.

Inheritance is a relationship between classes where one class is the parent (base/superclass/ancestor/etc.) class of another. Inheritance provides programming by extension (as opposed to programming by reinvention [35] and can be used as an is-a-kind-of (or is-a) relationship or for differential programming. Multiple Inheritance occurs when a class inherits from more than one parent/superclass. For example, a person is a mammal and an intellectual_entity, and a document may be an editable_item and a kind of literature.

So why user inheritance?: Inheritance is a natural way to model the world or a domain of discourse, and so provides a natural model for OOA and OOD (and even OOP). This is common in the AI domain, where semantic nets use inheritance to understand the world by using classes and concepts for generalization and categorization, by reducing the real-world's inherent complexity.

Inheritance also provides for code and structural reuse. In the above Computer class diagram, all routines and structure available in class Computer are available to all subclasses throughout the diagram. All attributes available in Personal computers are also available to all of its subclasses. This kind of reuse takes advantage of the is-a-kind-of relationship. Class libraries also allow reuse between applications, potentially allowing order-of-magnitude increases in productivity and reductions in defect rates (program errors), as library classes have already been tested and further use provides further testing providing even greater reliability.

With differential programming, a class does not have to be modified if it is close to what's required; a derived class can be created to specialize it. This avoids code redundancy, since code would have to be copied and modified otherwise.

Polymorphism is often explicitly available in many OO languages (such as C++, CLOS, Eiffel, etc.) based on inheritance when type and class are bound together (typing based on subclassing, or subclass polymorphism), since only an object which is a member of (inherits from) a class is polymorphically assignment compatible with (can be used in place of) instances or references of that class. Such assignment can result in the loss of an object's dynamic type in favor of a static type (or even loss of an object's representation to that of the static class, as in C++ slicing). Maintaining the dynamic type of objects can be provided (and preferred); however, C++ provides both sliced and non- sliced replacement in a statically typed environment.

OO Programming with all of the above features increases code reusability. Library ON-LINE received great benefit from this feature. Most of the business logic was encapsulated and reused throughout the code. The same business modules were

use in the web tier and in the desktop tier. Using the advantages of inheritance, the data model was mapped to an object model, forming an data access layer and an abstraction model. All of the data access was done through this layer, giving a DBMS vendor independent code.

5.1.3. Software Design Patterns

With usage of software design patterns, a framework was constructed during the development of the library management system Library ON-LINE. In this section we take a closer look on these patterns and how they affected the development of the library management system. Most of these patterns can be found in [36].

Data Access Object

Access to data varies depending on the source of the data. Access to persistent storage, such as to a database, varies greatly depending on the type of storage (relational databases, object-oriented databases, flat files, and so forth) and the vendor implementation. For library management software applications, persistent storage is implemented with different mechanisms, and there are marked differences in the APIs used to access these different persistent storage mechanisms. Other applications may need to access data that resides on separate systems. For example, the data may reside in mainframe systems, Lightweight Directory Access Protocol (LDAP) repositories, and so forth. Another example is where data is provided by services through external systems such as business-to-business (B2B) integration systems, credit card bureau service, and so forth. For example, local MARC21 catalog information is stored in a database but external sources like the Library of Congress catalog service or the OCLS, the cataloging data is in textual form. Instead of converting this textual data for each access, we use a data access object and form an intermediate layer.

The DAO shown in Figure 5.1, implements the access mechanism required to work with the data source. The data source could be a persistent store like an RDBMS, an external service like a B2B exchange, a repository like an LDAP

database, or a business service accessed via CORBA Internet Inter-ORB Protocol (IIOP) or low-level sockets. The business component that relies on the DAO uses the simpler interface exposed by the DAO for its clients. The DAO completely hides the data source implementation details from its clients. Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes, this pattern allows the DAO to adapt to different storage schemes without affecting its clients or business components. Essentially, the DAO acts as an adapter between the component and the data source.

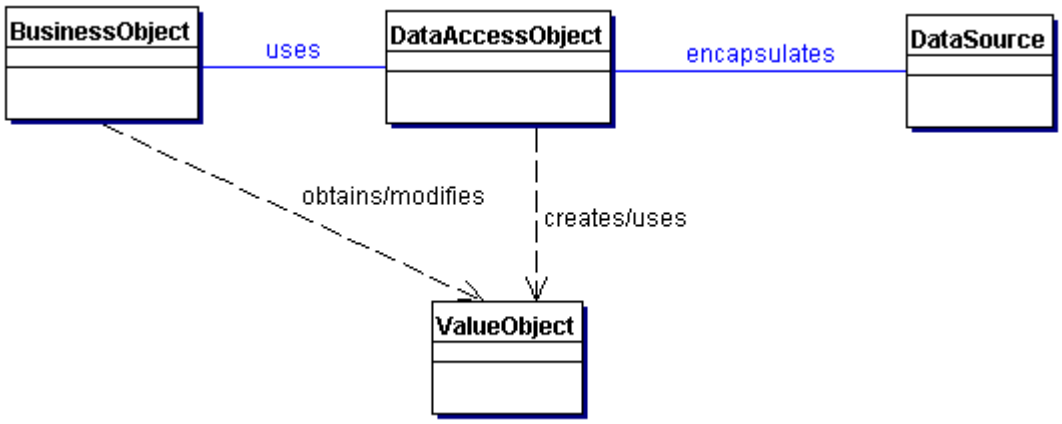


Figure 5.1 Data access object design pattern

Service Activator

Enterprise beans and other business services need a way to be activated asynchronously. When a client needs to access an enterprise bean, it first looks up the bean's home object. The client requests the Enterprise JavaBeans (EJB) component's home to provide a remote reference to the required enterprise bean. The client then invokes business method calls on the remote reference to access the enterprise bean services. All these method calls, such as lookup and remote method calls are synchronous. The client has to wait until these methods return.

Another factor to consider is the life cycle of an enterprise bean. The EJB specification permits the container to passivate an enterprise bean to secondary storage. As a result, the EJB container has no mechanism by which it can provide a process-like service to keep an enterprise bean constantly in an activated and ready state. Because the client must interact with the enterprise bean using the bean's remote interface, even if the bean is in an activated state in the container, the client still needs to obtain its remote interface via the lookup process and still interacts with the bean in a synchronous manner. The solution is to use a Service Activator to receive asynchronous client requests and messages. On receiving a message, the Service Activator locates and invokes the necessary business methods on the business service components to fulfill the request asynchronously.

The Service Activator visualized in Figure 5.2, is a JMS Listener and delegation service that requires implementing the JMS message listener-making it a JMS listener object that can listen to JMS messages. The Service Activator can be implemented as a standalone service. Clients act as the message generator, generating events based on their activity.

Any client that needs to asynchronously invoke a business service, such as an enterprise bean, may create and send a message to the Service Activator. The Service Activator receives the message and parses it to interpret the client request. Once the client's request is parsed or unmarshalled, the Service Activator identifies and locates the necessary business service component and invokes business methods to complete processing of the client's request asynchronously.

The Service Activator may optionally send an acknowledgement to the client after successfully completing the request processing. The Service Activator may also notify the client or other services on failure events if it fails to complete the asynchronous request processing.

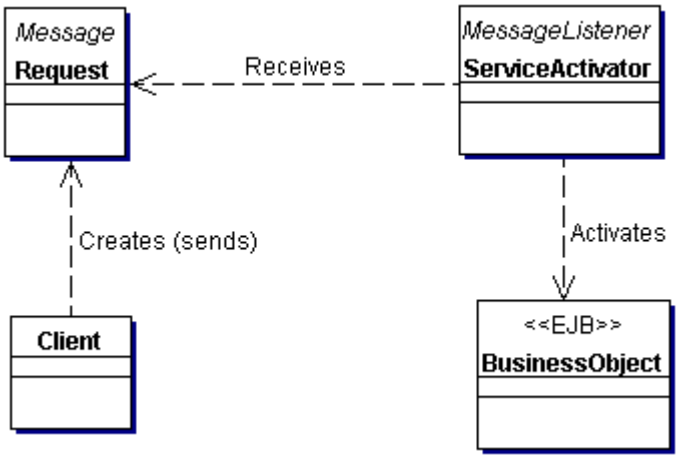


Figure 5.2 Service activator design pattern

Transfer Object

Application clients like the desktop cataloging module need to exchange data with enterprise beans. Java 2 Platform, Enterprise Edition (J2EE) applications implement server-side business components as session beans and entity beans. Some methods exposed by the business components return data to the client. Often, the client invokes a business object's get methods multiple times until it obtains all the attribute values.

Session beans represent the business services and are not shared between users. A session bean provides coarse-grained service methods when implemented per the Session Facade pattern. Entity beans, on the other hand, are multiuser, transactional objects representing persistent data. An entity bean exposes the values of attributes by providing an accessor method (also referred to as a *getter* or *get method*) for each attribute it wishes to expose. Every method call made to the business service object, be it an entity bean or a session bean, is potentially remote. Thus, in an Enterprise JavaBeans (EJB) application such remote invocations use the network layer regardless of the proximity of the client to the bean, creating a network overhead. Enterprise bean method calls may permeate the network layers of the system even if

the client and the EJB container holding the entity bean are both running in the same JVM, OS, or physical machine. Some vendors may implement mechanisms to reduce this overhead by using a more direct access approach and bypassing the network.

As the usage of these remote methods increases, application performance can significantly degrade. Therefore, using multiple calls to get methods that return single attribute values is inefficient for obtaining data values from an enterprise bean.

The solution is to use a Transfer Object to encapsulate the business data. A class diagram is given in Figure 5.3. A single method call is used to send and retrieve the Transfer Object. When the client requests the enterprise bean for the business data, the enterprise bean can construct the Transfer Object, populate it with its attribute values, and pass it by value to the client. Clients usually require more than one value from an enterprise bean. To reduce the number of remote calls and to avoid the associated overhead, it is best to use Transfer Objects to transport the data from the enterprise bean to its client. When an enterprise bean uses a Transfer Object, the client makes a single remote method invocation to the enterprise bean to request the Transfer Object instead of numerous remote method calls to get individual attribute values. The enterprise bean then constructs a new Transfer Object instance, copies values into the object and returns it to the client. The client receives the Transfer Object and can then invoke accessor (or getter) methods on the Transfer Object to get the individual attribute values from the Transfer Object. Or, the implementation of the Transfer Object may be such that it makes all attributes public. Because the Transfer Object is passed by value to the client, all calls to the Transfer Object instance are local calls instead of remote method invocations.

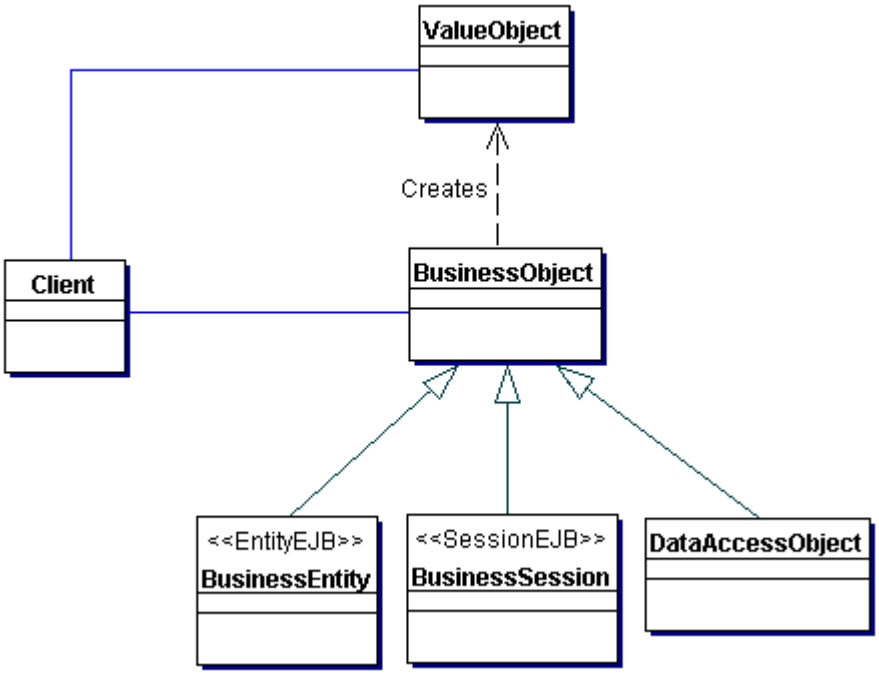


Figure 5.3 Transfer object design pattern

Composite Entity

Entity beans are not intended to represent every persistent object in the object model. Entity beans are better suited for coarse-grained persistent business objects. Entity beans represent distributed persistent business objects. Whether developing or migrating an application to the J2EE platform, object granularity is very important when deciding what to implement as an entity bean. Entity beans should represent coarse-grained business objects, such as those that provide complex behavior beyond simply getting and setting field values. These coarse-grained objects typically have dependent objects. A dependent object is an object that has no real domain meaning when not associated with its coarse-grained parent.

A recurring problem is the direct mapping of the object model to an Enterprise JavaBeans (EJB) model (specifically entity beans). This creates a relationship between the entity bean objects without consideration of coarse-grained versus fine-grained (or dependent) objects. Determining what to make coarse-grained versus fine-grained is typically difficult and can best be done via modeling relationships in Unified Modeling Language (UML) models.

The solution to this problem is to use a Composite Entity shown in Figure 5.4, to model, represent, and manage a set of interrelated persistent objects rather than representing them as individual fine-grained entity beans. A Composite Entity bean represents a graph of objects.

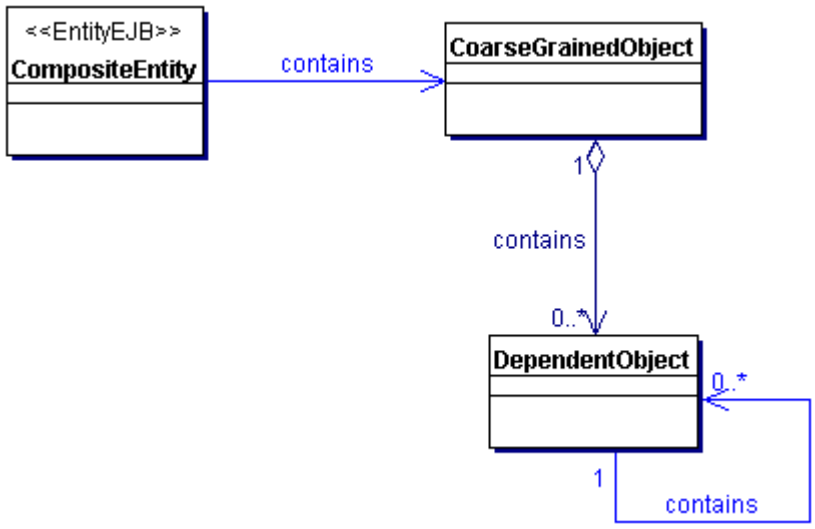


Figure 5.4 Composite entity design pattern

Value List Handler

The client requires a list of items from the service for presentation. The number of items in the list is unknown and can be quite large in many instances. Most Java 2 Platform, Enterprise Edition (J2EE) applications have a search and query requirement to search and list certain data. In some cases, such a search and query operation could yield results that can be quite large. It is impractical to return the full result set when the client's requirements are to traverse the results, rather than process the complete set. Typically, a client uses the results of a query for read-only purposes, such as displaying the result list. Often, the client views only the first few matching records, and then may discard the remaining records and attempt a new query. The search activity often does not involve an immediate transaction on the matching objects. The practice of getting a list of values represented in entity beans by calling an ejbFind() method, which returns a collection of remote objects, and then calling each entity bean to get the value, is very network expensive and is considered a bad practice.

There are consequences associated with using Enterprise JavaBeans (EJB) finder methods that result in large results sets. Every container implementation has a certain amount of finder method overhead for creating a collection of EJBObject references. Finder method behavior performance varies, depending on a vendor's container implementation. According to the EJB specification, a container may invoke ejbActivate() methods on entities found by a finder method. At a minimum, a finder method returns the primary keys of the matching entities, which the container returns to the client as a collection of EJBObject references. This behavior applies for all container implementations. Some container implementations may introduce additional finder method overhead by associating the entity bean instances to these EJBObject instances to give the client access to those entity beans. However, this is a poor use of resources if the client is not interested in accessing the bean or invoking its methods. This overhead can significantly impede application performance if the application includes queries that produce many matching results.

The solution is to use a Value List Handler shown in Figure 5.5 to control the search, cache the results, and provides the results to the client in a result set whose size and traversal meets the client's requirements.

This pattern creates a ValueListHandler to control query execution functionality and results caching. The ValueListHandler directly accesses a DAO that can execute the required query. The ValueListHandler stores the results obtained from the DAO as a collection of Transfer Objects. The client requests the ValueListHandler to provide the query results as needed. The ValueListHandler implements an Iterator pattern to provide the solution.

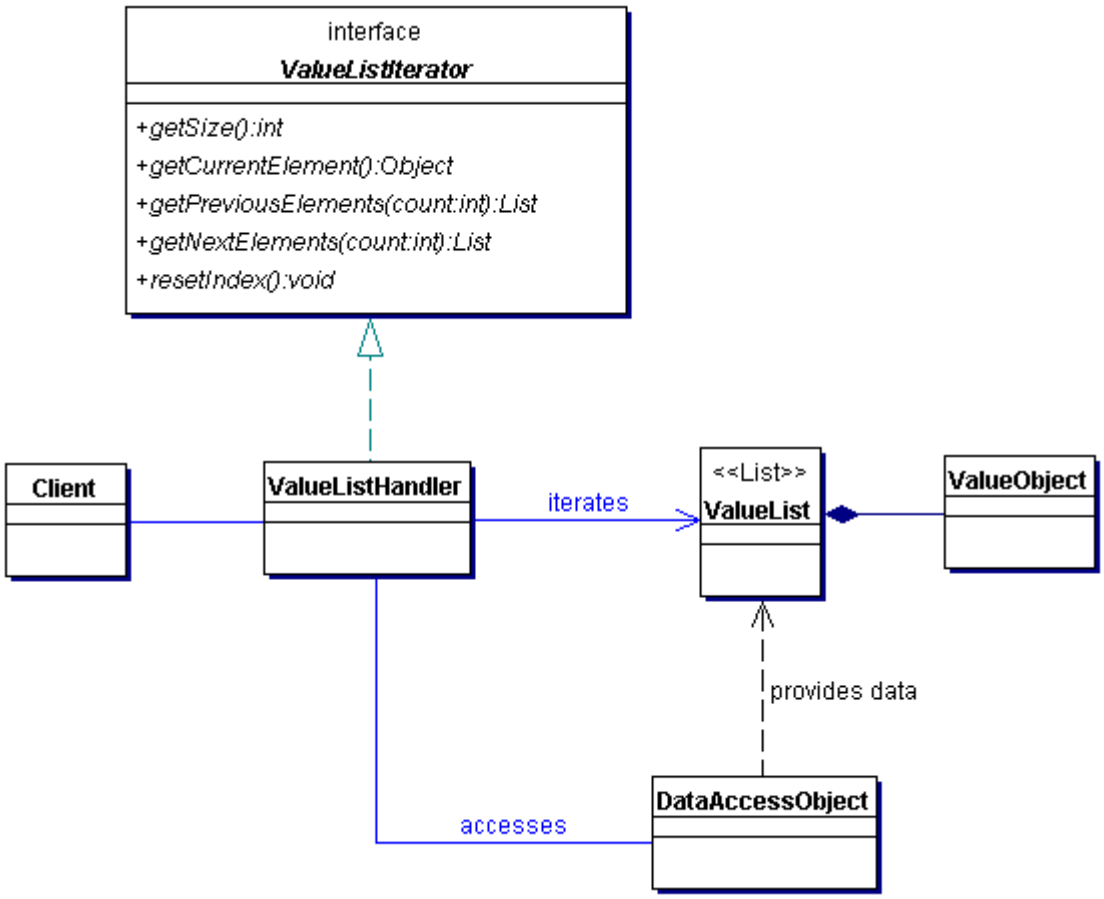


Figure 5.5 Value list handler design pattern

Business Delegate

A multi-tiered, distributed system requires remote method invocations to send and receive data across tiers. Clients are exposed to the complexity of dealing with distributed components. Presentation-tier components interact directly with business services. This direct interaction exposes the underlying implementation details of the business service application program interface (API) to the presentation tier. As a result, the presentation-tier components are vulnerable to changes in the implementation of the business services: When the implementation of the business services change, the exposed implementation code in the presentation tier must change too.

Additionally, there may be a detrimental impact on network performance because presentation-tier components that use the business service API make too many invocations over the network. This happens when presentation-tier components use the underlying API directly, with no client-side caching mechanism or aggregating service.

The solution is to use a Business Delegate shown in Figure 5.6, to reduce coupling between presentation-tier clients and business services. The Business Delegate hides the underlying implementation details of the business service, such as lookup and access details of the EJB architecture. The Business Delegate acts as a client-side business abstraction; it provides an abstraction for, and thus hides, the implementation of the business services. Using a Business Delegate reduces the coupling between presentation-tier clients and the system's business services. Depending on the implementation strategy, the Business Delegate may shield clients from possible volatility in the implementation of the business service API. Potentially, this reduces the number of changes that must be made to the presentation-tier client code when the business service API or its underlying implementation changes.

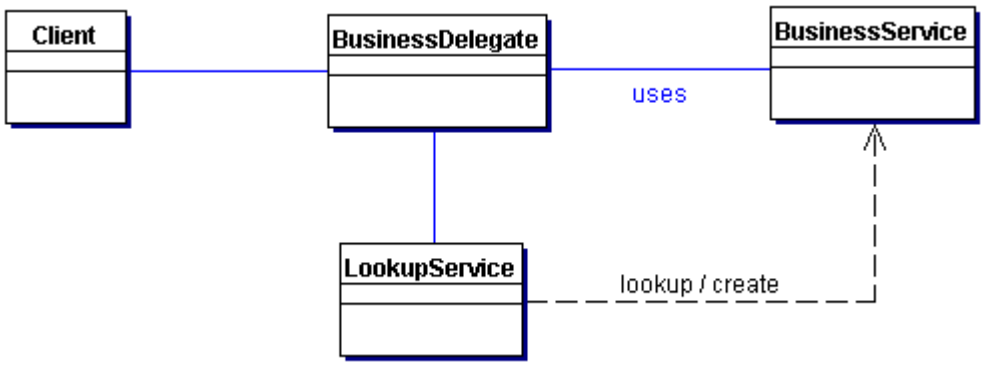


Figure 5.6 Business delegate design pattern

View Helper

The system creates presentation content, which requires processing of dynamic business data. Presentation tier changes occur often and are difficult to develop and maintain when business data access logic and presentation formatting logic are interwoven. This makes the system less flexible, less reusable, and generally less resilient to change.

The solution is to use a view that contains formatting code, shown in Figure 5.7, delegating its processing responsibilities to its helper classes, implemented as JavaBeans or custom tags. Helpers also store the view's intermediate data model and serve as business data adapters.

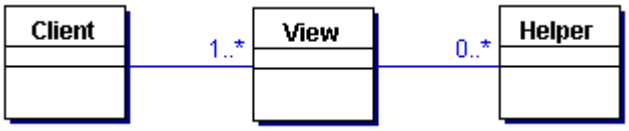


Figure 5.7 View helper design pattern

Front Controller

The presentation-tier request handling mechanism must control and coordinate processing of each user across multiple requests. Such control mechanisms may be managed in either a centralized or decentralized manner. The system requires a centralized access point for presentation-tier request handling to support the integration of system services, content retrieval, view management, and navigation. When the user accesses the view directly without going through a centralized mechanism, two problems may occur:

- Each view is required to provide its own system services, often resulting in duplicate code.
- View navigation is left to the views. This may result in commingled view content and view navigation.

Additionally, distributed control is more difficult to maintain, since changes will often need to be made in numerous places.

The solution is to use a controller as the initial point of contact for handling a request. The controller manages the handling of the request, including invoking security services such as authentication and authorization, delegating business processing, managing the choice of an appropriate view, handling errors, and managing the selection of content creation strategies.

The controller provides a centralized entry point that controls and manages Web request handling. By centralizing decision points and controls, the controller also helps reduce the amount of Java code, called *scriptlets,* embedded in the JavaServer Pages (JSP) page.

Centralizing control in the controller and reducing business logic in the view promotes code reuse across requests. It is a preferable approach to the alternative-embedding code in multiple views-because that approach may lead to a more error-prone, reuse-by-copy- and-paste environment.

Typically, a controller coordinates with a dispatcher component. Dispatchers are responsible for view management and navigation. Thus, a dispatcher chooses the

next view for the user and vectors control to the resource. Dispatchers may be encapsulated within the controller directly or can be extracted into a separate component.

While the Front Controller pattern suggests centralizing the handling of all requests, it does not limit the number of handlers in the system, as does a Singleton. An application may use multiple controllers in a system, each mapping to a set of distinct services.

5.2 Catalog Data Conversion

Catalog data conversion is an essential issue in developing library management systems. The catalog data tends to reach huge size even for small libraries. The object of installation library may have the catalog records in different forms:

- There are no digital data in any form: The library might have used a card based system to catalog materials and had never used a computer system to store these records. This was the case we had with Library ON-LINE.
- There are digital data but none in MARC format: Some libraries use computer software to catalog their materials, but this software might not necessarily be designed specially for catalog storage. This means that the catalog information does exist but usually in an ad hoc form.
- Catalog data exists in MARC format: This is the simplest case for data conversion. It is usually easy to acquire this catalog data and use it in a new system.

We will look at the above cases one by one and develop processes for data conversion from the legacy system to a new system.

The hardest and the toughest case is that there are no catalog data in any digital form. The cataloging data is hard to produce from scratch and requires a deep knowledge of MARC21 format. It will be a common case that there are not enough librarians with an adequate knowledge of the MARC format to produce all of the

material's cataloging information considering that there are tens of thousands of materials even in a small library. We had a similar case in Library ON-LINE. There are two ways of dealing with this problem: There are companies specialized in library catalog data conversion, subject to fees. These can be used or processes can be developed to produce these catalog records locally and mechanistically. We produced and followed a data conversion process like this one below:

1. Create a digital list of all materials in the library by their ISBN or ISSN number and number of copies, volumes, and parts. This is relatively a hassle free operation.
2. Use the created ISBN list to grab the matching catalog record from Z39.50 server publicly available. In this part, a catalog acquiring software should be developed. The software should search the public Z39.50 servers for records, grab the record, convert it into a local form and store it, passing to the next record. One question here is which catalog server to search: The Library of Congress catalogs and some portion of the OCLC catalog is publicly available without a charge, and they are good sources of catalog records for the cataloging quality is very high. Also there are many catalog server available for a fee, the most important being the OCLC. Most libraries have OCLC subscriptions that can be used. However, it should be noted that the OCLC is not a cheap service, and the total amount might reach considerable amount for large number of materials. This approach has another problem: Catalog records for local books (like Turkish in our case) are hard to find and might have low cataloging quality. The cataloging source should be selected with care and studied deeply to attain a certain level of cataloging quality. Yet another problem is that some materials have no ISBN or ISSN numbers associated with them. In this case, this step should probably be skipped, because other information associated with a material does not generally represent an unique value for a given material. One might need at least three (title,

author, edition) distinct attribute to search for, and yet this might not lead to the correct catalog record.

In this case and in our Library ON-LINE instance where we used the relation MARC model introduces in former chapters, we need converter software to convert the textual MARC21 data to the relational form. The pseudo code for this software module is given in the figure 5.7.

```
Increment_materialid();
while (!end_of_marc21) {
       parse_location_from_leader(pos);
       while (x < parse_length_from_leader(pos)) {
              str = get_char_from_pos(pos);
              db_insert(str);
              rs = remember_pos();
              if (rs == str) {
              tagid +=;
              }
              db_insert(tagid);
       }
       pos = next_pos();
}
```

Figure 5.8. Pseudo code for MARC21 to Relational MARC conversion

3. Produce a list of not found ISBN or ISSN value. The automated search in remote Z39.50 servers would possibly produce a not-found list as not all of the materials are available in catalogs. This problem occurs especially in local materials. If there is no more import facilities to use, these materials should be cataloged manually or their cataloging information should be purchased.

The other case of existence of cataloging data is that the library might have used cataloging software that is compatible with MARC21 format. This case introduces the need for conversion software. This software should read from the application software's database, convert the data to MARC21 and store it in the local database. One important problem is that the catalog data may not be fully compatible with MARC21 data. For some fields the attributes of the old database may not exactly match with the MARC21 format. The solution we have introduced is to use the minimum set of needed data for conversion and leave out the remaining part.

After this process, librarians go through the catalog records and check the records both syntactically and semantically.

The last common case is that the library might already have cataloging data in MARC21 format. This is obviously the simplest case. However, there still may be a need to transfer the data across platforms. This again might be done using a software module.

All of the above cases share a common problem of localization. MARC21 records in the remote servers and in the legacy application does not generally have local language character support. The conversion to the local character is burdensome process and it unsurprisingly slow down the total data conversion process. In our experience, we try to find pattern to match the corrupted character to the right one. These patterns are easily found. For example, in our records the Turkish character “i” appeared corrupted as “+”. The difficulty is that the plus sign may appear uncorrupted in some of the fields and an automated process might corrupt the uncorrupted character. The idea is to correct as many characters as possible automatically and leave some of them for manual correction.

5.3 Integration with Online Databases

Integration with other electronic resources and other online catalog databases is an essential feature. In this integration, the library management software should act as a proxy between the Web OPAC interface and the remote database. This can be seen as a resource linking scheme. This resource linking can be done in two ways, linking from resource to resource and linking initiatives.

Linking from resource to resource integration comes in two distinct ways, dynamic and contextual linking. Dynamic linking means that a link from a record in the catalog or other electronic resource (such as a fulltext database) is not a static URL, but uses logic that dynamically generates the URL. Instead of using a static link, the link is generated by the known search syntax of the resource. The key advantage to dynamic linking is that the URL is maintained in one central place, and

needs to be updated only once rather than maintaining it in every record that has that particular link. It also means that a link can be formed with specific information that will be different depending on the user's search or metadata from the record. With contextual linking, only certain resources are relevant to certain records. For instance, an aggregator that supplies the full text of a particular journal will be offered as a link only if the patron is looking at that particular journal record in the Web OPAC. In the same manner, a particular citation in an A&I database will offer a link only to a full-text resource if that resource provides the full text. The circumstances under which these related resources are offered are defined and maintained by the library. The relationships established behave like an automated reference interview that suggests related resources based on the patron's search or based on metadata from the record. For example, a subject search under "neoplasms" may result in a list of related resources in the health sciences area such as CINAHL, Medline, a select number of medical library catalogs, as well as library-selected, high-quality medical Web sites. Another search in the Web OPAC resulting in The Poisonwood Bible by Barbara Kingsolver will result in a different list of related resources that might include the book jacket image, the Book Review Index, Amazon.com, and Contemporary Authors. These links are not created by static links in the record. Rather, the links are dynamically generated based on library-defined relationships. Linking from resources other than the Web OPAC is also a component of the integration scheme. For instance, a link can be used to take a user from a citation database to the full-text or library holding of that article. A user starts a search in Sociological Abstracts, finds a citation that is relevant, and clicks on the link. Based on an analysis of the information from the citation, the integration scheme offers a number of related links. One is a direct link to Ebscohost's full-text version of the article, and another is a link to the library catalog holding indicating the availability of the print version of the journal. The integration scheme should provides the ability to suggest related resources from a particular record in the Web OPAC, from the staff modules, or from other electronic resources such as a full-text index or an A&I resource. The list of resources is dynamically generated and resources are offered only if specific criteria are met based on elements from the metadata or the user's search.

Linking between initiatives is another way of integration. In order for linking to work across resources, there must be agreement between the various vendors on how this information should be transferred. Initiatives such as the OpenURL [37], the DOI, and CrossRef (http://www.crossref.org/) help to define how and what information should be transferred from resource to resource. The following is brief explanation of how the integration scheme relates to these linking initiatives. OpenURL is an emerging standard for transporting information within a URL to a ‘”resolution server” that can accept the URL syntax and provide context-sensitive services based on the information in the URL. The metadata in the URL describes the resource that is being requested. OpenURL is a standardized way to pass information in the URL between these different resources. Some common elements that are passed are ISSN/ISBN, title, volume/issue number, author, and date. When this OpenURL is passed to the resource, the resource can deliver the appropriate information--e.g., a link from Journal of Clinical Psychology in the WebOPAC will deliver information pointing to the full text of this journal in one of the library’s licensed full-text collections. The OpenURL is currently under consideration by NISO (National Information Standards Organization) as a new standard. Other standards that are often mentioned in relation to linking functionality are the DOI (Digital Object Identifier) and CrossRef. The DOI is a persistent identifier of intellectual property and grew out of the content/publisher arena so that publishers and content owners could have control over the digital content. DOI does not define how the linking is done, but the DOI is an identifier for the object that is being accessed. DOI can work in conjunction with the OpenURL so that a DOI can be passed in a OpenURL. CrossRef is a collaborative reference linking service that uses the DOI as the primary identifier for linking between citation and full-text article at the appropriate publisher site. CrossRef allows a researcher to click on a reference citation in a journal and immediately access the cited article. Library On-Line can work with the OpenURL, DOI, and CrossRef. It can pass OpenURLs to any resource and, if that resource is OpenURL-aware, that resource can provide context-sensitive information appropriate to the user. It can also pass a DOI to any resource when appropriate, whether it is transported in an OpenURL or by some other transport mechanism such as CrossRef. The integration scheme’s strength is that it works with

any resource that has a predictable URL formation. This provides maximum flexibility as these new standards and initiatives are being developed.

5.4. Integration with Other University Information Systems

In location of implementation of our library management system, there are other university information systems available. These are Campus ON-LINE; student registration and information system, Course ON-LINE; course web-pages management system, Campus ON-SMS; SMS based information distribution system, Pay ON-LINE; electronic payment system and Access ON-LINE, electronic access control system. Library ON-LINE, in this sense, has conceptual and physical connections with these systems. In this section, information about the integration is delivered.

The integration is done in three different ways: Integration at the data layer, and integration at the business layer. These will be described in detail.

5.4.1. University Information Systems Involved

Campus ON-LINE

Campus ON-LINE is the student information and online registration system of IŞIK University. Students use this system when first registering to the university and when registering to courses at the beginning of each semester. Instructors use the system to see class lists during the semester, submit letter grades at the end of each semester and advise online to students during the registration periods. There are also some functions defined for some administrative units.

There are more that 2000 users of the system, which are, grouped into more that 10 different user groups. As development technology, ASP is used with MS SQL Server 2000 as database management system. Components are developed in Visual Basic 6.0.

Course ON-LINE

Course ON-LINE is a course home page management system in use in IŞIK University. The system tightly integrated to Campus ON-LINE. Courses opened for a specific academic term have their course home pages automatically created by Course ON-LINE. These web page include class-lists, assignments, course descriptions, source resources, and communication tools like the forum or instant messaging tools.

The development is done in Microsoft .NET platform using Microsoft SQL Server 200 as the DBMS.

Campus ON-SMS

Campus ON-SMS is an SMS based information distribution service. The service currently, delivers course final notes or announcements from the university to the students who are registered to the system.

Campus ON-SMS is based on the Microsoft SQL Server 2000 and transact-sql language.

Pay ON-LINE

Pay ON-LINE is an electronic payment system. The system is in use in IŞIK University campuses. The system allows payment in campuses to be done through an electronic proximity card and holds the student information in this card. The system has reports for the accounting department and other departments of interest.

PAY ON-LINE is developed in Microsoft .NET platform and uses Microsoft SQL Server as a DBMS.

5.4.2. Integration at Data Layer

The information system above all share a common user database. The users of the system are students, instructors (part and full time), alumni, administrative staff,

and sub-contractors of the university that access to the university. There are also guest users in the system. The user database is therefore designed to be common as shown in figure 5.8.

The common user database is responsible only from storing the student (or administrative) id numbers, passwords, and names. Other types of information are left out, because each system needs distinct data. For example, Campus ON-LINE needs to distinguish between the grades of the students, while Library ON-LINE needs to know whether he/she is undergraduate or graduate. Storing of user types are also left to the individual systems, as the level of granularity changes from one system to other.

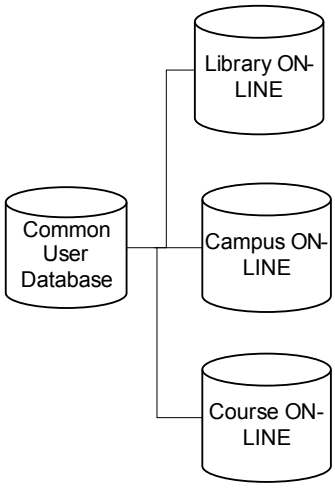


Figure 5.9 Integration of databases

This common database scheme has the following advantages:

- Redundant data is kept minimal. When there is not a common database, all system should keep track of a common set of information on its own

separately. This means that the same information would exist in many databases.

- Simplifies record updates. Keeping the same data across multiple databases delivers the problem of updating this data. For example, when student number changes or a student graduates, the related data on all of these databases should be updated. Some trigger mechanisms might be deployed, but this is problematic as well, as these trigger programs at the database level, makes the database design more complex than it should be and makes the system platform dependent as these triggers should be re-coded if the database system should change. Likewise, every time a user changes a password, the change should be propagated though all databases
- Enables some common software modules to be used by all systems. A good example is the common authentication system which is described in detail in the next chapter. With the module all the authentication is done through a common application, which is a web service.
- Reduces the need for storage. Minimizing the existence of redundant data reduces the need for physical storage and helps increase the overall performance.
- Simplifies operational processes. This is an important issue when the systems are in interest of various university departments. For example, undergraduate students' information is under control of the student registrar department, while the student's tuition information is under control of the accounting department. The common database model allows the processes to be separated. A change in student information is done only through a single database, and does not need to propagate.
- Reduces complexity of the database. Eliminating redundant data reduces the database complexity and dependencies across the databases.

There is only one table in the common database, which is the user table. The definition is given in figure 5.9.

user	
PK	**userid**
	password

Figure 5.10. The common user table.

The userid attribute is the student id or the administrative id of the user. Password attribute is the SHA1 hash of the password. It is a fixed 40 character column. The SHA1 hash of a string is a 40 character hexadecimal number.

When a message of any length < 2^64 bits is input, the SHA-1 produces a 160-bit output called a message digest. The message digest can then, for example, be input to a signature algorithm which generates or verifies the signature for the message. Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature. Any change to the message in transit will, with very high

Probability, result in a different message digest, and the signature will fail to verify. The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.

The common database as seen above has a very simple design but is sufficient and cost effective for the university information system.

5.4.3. Integration at the Business Layer

The university information systems used does not share many functionality except the system operation which are handled by the development platforms. One common function implemented is the authentication.

All of the university information systems share a common user base, and as described above a common database is designed for this user base. For this user database, a common authentication software module is implemented and used across all systems.

The problem that arises when implementing a common software module is that all of the systems in the university information system are developed in different platforms. Some are in Microsoft Active Server Pages (Visual Basic 6.0), some are in Microsoft .NET, and the Library ON-LINE is developed in Java platform. Although there are many ways of sharing common modules across these platforms, most of them are inefficient and costly. Some methods are listed below:

- Using a C module: Windows native Dynamic Link Libraries written in C are accessible by both Visual Basic 6.0 and the .NET platform. Java can also access these components through the JNI (Java Native Interface) . The JNI allows Java code that runs within a Java Virtual Machine (VM) to operate with applications and libraries written in other languages, such as C, C++, and assembly. In addition, the *Invocation API* allows you to embed the Java Virtual Machine into your native applications. Using this method has many drawbacks. First the performance is limited. The authentication module is busy software, as it handles the entire authentication from all the university information systems. The other problem is coding and accessing these modules are programmatically hard to implement.
- Using Middleware Architecture: Middleware architectures like the CORBA can be used, but again implementing these architectures are hard.
- Using a XML web service: The most appropriate solution is using this method. Java, .NET and the Visual Basic 6.0 has methods or libraries that simplify accessing XML web services.

From the above choices, we took taking XML web services into account. A software module was implemented complying with the web services standard in the Java programming language. The web service uses SOAP [19] over HTTP to

communicate with the clients. The clients in this case are the application software like the Library ON-LINE, Campus ON-LINE, Course ON-LINE, and Pay ON-LINE. These applications software send the username / password pair to the web service in XML. The web service calculates the SHA1 hash of the password and authenticates it against the pair stored in the user database. The web service responds with 'true' or 'false' boolean value. A discussion of the used technologies are briefly explained below:

SHA1

As seen above the passwords of the users are stored as their SHA1 message digest. The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard in 1993; a revised version was issued in 1995 and is generally referred to as SHA1. The SHA1 is based on the MD4 algorithm and its design closely models MD4. The SHA1 algorithm takes as input a message with maximum length of less than 2^{64} bits and produces as output a 160-bit message digest. The input is processed in 512-bit blocks The SHA1 hash is a constant 40 digit long hexadecimal number.

There are other hash algorithms available like the MD5, which generates a 32 digit hexadecimal number. Because both are derived from MD4, SHA1 and MD5 are quite similar to each other. Accordingly their strengths and other characteristics should be similar. A comparison is given for them in [26]:

- *Security against brute-force attacks:* The most obvious and most important difference is that the SHA1 digest is 32 bits longer than the MD5 digest. Using a brute-force technique, the difficulty of producing any message having a given message digest is on the order of 2^{128} operations for MD5, and 2^{160} for SHA1. Again using a brute-force technique, the difficulty of producing tow messages having the same massage digest is on the order of 2^{64} operations for MD5, and 2^{80} for SHA1. Thus, SHA1 is considerably stronger against brute-force attacks.
- *Security against cryptanalysis:* MD5 is vulnerable to cryptanalytic attacks discovered since its design. SHA1 appears not to be vulnerable to such

attacks. However, little is publicly known about the design criteria for SHA1, so its strength is more difficult to judge than would otherwise be the case.

- *Speed:* Because both algorithms rely heavily on addition modulo 2^{32}, both do well on a 32-bit architecture. SHA1 involves more steps (80 versus 64) and most process a 160-bit buffer compared to MD5's 128-bit buffer. Thus, SHA1, should execute more slowly than MD5 on the same hardware.
- *Simplicity and compactness:* Both algorithms are simple to describe and simple to implement and do not require large programs or substation tables.
- *Little-endian versus big-endian architecture:* MD% uses a little-endian scheme for interpreting a message sequence of 32-bit words, whereas SHA1 uses a big-endian scheme (this is probably the NSA designers of SHA used a Sun Sparc based machine for the prototype implementation). There appears to be no significant advantage to either approach.

XML

The Extensible Markup Language (XML) was originally envisioned as a language for defining new document formats for the World Wide Web. XML is derived from the Standard Generalized Markup Language (SGML), and can be considered to be a meta-language: a language for defining markup languages. SGML and XML are text-based formats that provide mechanisms for describing document structures using markup tags. Web developers may notice some similarity between HTML and XML, which is due to the fact that they are both derived from SGML. As the use of XML has grown, it is now generally accepted that XML is not only useful for describing new document formats for the Web but is also suitable for describing structured data.

Besides being able to represent both structured and semi-structured data, XML has a number of characteristics that have caused it to be widely adopted as a data representation format. XML is extensible, platform-independent, and supports internationalization by being fully Unicode compliant. The fact that XML is a text-

based format means that when the need arises, one can read and edit XML documents using standard text-editing tools.

XML's extensibility manifests itself in a number of ways. First of all, unlike HTML it does not have a fixed vocabulary. Instead, one can define vocabularies specific to particular applications or industries using XML. Secondly, applications that process or consume XML formats are more resistant to changes in the structure of the XML being provided to them than applications that use other formats, as long as such changes are additive. For instance, an application that depends on processing a <Customer> element with a customer-id attribute typically would not break if another attribute, such as last-purchase-date, was added to the <Customer> element. Such flexibility is uncommon in other data formats and is a significant benefit of using XML.

XML is not tied to any programming language, operating system or software vendor. In fact, it is fairly straightforward to produce or consume XML using a variety of programming languages. Platform independence makes XML very useful as a means for achieving interoperability between different programming platforms and operating systems.

Below (figure 5.10) is a sample XML document that represents a customer order for a music store. A point of note is how the document easily represents both the rigidly structured data that describes information about compact discs as well as the semi-structured data containing special instructions and comments about a specific customer.

Web Services

The web services platform is a combination of various technical standards like XML, UDDI, WSDL, and SOAP [18].

It is important for application development to allow Internet communication between programs.

Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic. A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

The Web Services Description Language (WSDL) is an XML message format for describing the network services offered by the server. You use WSDL to create a file that identifies the services provided by the server and the set of operations within each service that the server supports. For each of the operations, the WSDL file also describes the format that the client must follow in requesting an operation.

Universal Description, Discovery and Integration (UDDI) is an industry specification for publishing and locating information about Web services. It defines an information framework that enables you to describe and classify your organization, its services, and the technical details about the interfaces of the Web services you expose. The framework also enables you to consistently discover services, or interfaces of a particular type, classification, or function. UDDI also defines a set of Application Programming Interfaces (APIs) that can be used by applications and services to interact with UDDI data directly. For example, you can develop services that publish and update their UDDI data automatically, react dynamically to service availability, or automatically discover interface details for other services with which they interact.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?xml-stylesheet href="orders.xsl"?>

<order id="ord123456">
  <customer id="cust0921">
    <first-name>John</first-name>
    <last-name>Doe</last-name>
    <address>
      <street>State Street</street>
      <city>Boston</city>
      <state>MA</state>
      <zip>98052</zip>
    </address>
  </customer>
  <items>
    <compact-disc>
      <price>16.95</price>
      <artist>Nelly</artist>
      <title>Nellyville</title>
    </compact-disc>
    <compact-disc>
      <price>17.55</price>
       <artist>Baby D</artist>
       <title>Lil Chopper Toy</title>
    </compact-disc>
  </items>

  <!--  Always go the extra mile for the customer →
  <special-instructions xmlns:html=”http://www.w3.org/1999/xhtml/”>
    <html:p>If customer is not available at the address then attempt
      leave package at one of the following locations listed in order of
      which should be attempted first
    <html:ol>
      <html:li>Next Door</html:li>
      <html:li>Front Desk</html:li>
      <html:li>On Doorstep</html:li>
    </html:ol>
    <html:b>Note</html:b> Remember to leave a note detailing where
      to pick up the package.
    </html:p>
  </special-instructions>
</order>
```

Figure 5.11. A sample XML document.

CHAPTER 6. AN EXAMPLE APPLICATION: LIBRARY ON-LINE

6.1. Modules and Users

In the previous chapters, a conceptual overview has been given on modern library management systems. Following the guidelines given, an example application has been developed, namely Library ON-LINE.

Library ON-LINE, has 6 modules. These are:

- Cataloging
- Circulation
- Online Public Access (WebOPAC)
- Management & Reporting
- Helper Modules

In the next sections, these modules will be described.

6.1.1. Cataloging Module

The cataloging module is used to catalog library materials in the library management system. The cataloging clerks of the library can access this module. They can create or import cataloging reports, or modify existing ones. The cataloging module is the heart of a library management system.

The cataloging module consists of two separate modules, a web-based module generally for quick modification of records and a desktop module for creating new records.

Figure 6.1 shows the login dialog of the desktop cataloging module. Access is controled according to the access control lists.

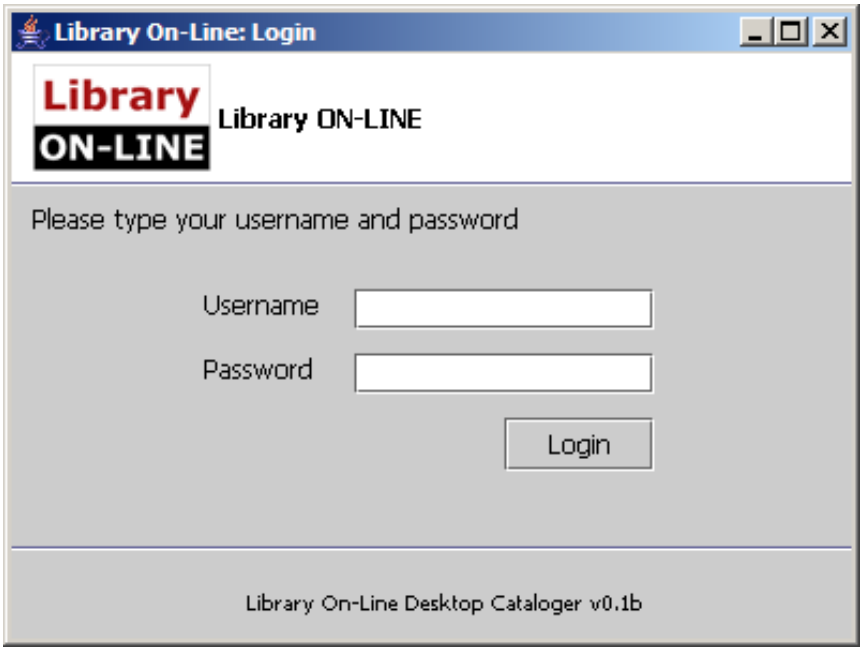


Figure 6.1. Cataloging desktop module login dialog

The cataloging module has an advanced MARC21 editor as shown in figure 6.1, 6.2, 6.3, and 6.4, with syntax checking and online help on MARC21 tags. Control tags of the record may be created using a set of drop-down graphical elements without the need for creating them manually. The module also has an import facility shown in figure 6.5. The import facility can search many online electronic catalog databases and acquire the MARC21 cataloging record directly into system. This facility is very helpful for the cataloging clerk. Creating MARC21 records from scratch is a burdensome effort, and generally contains many syntactical and semantical mistakes that will lead to inconsistent catalogs. This is especially bad whilst catalog searches. Incorrect records may result in an existing material of the library to be totally inaccessible. Using pre-edited and controlled records is much better in this way.

The desktop module also facilitates assigning of call number to the library materials. All call numbers can be controlled with the module including number of

copies, volumes or parts. Also, the location of the material, in the sense of library branches are defined or modified from this module.

The cataloging desktop module converts the given MARC21 text record into the Relational MARC format described in previous chapters.

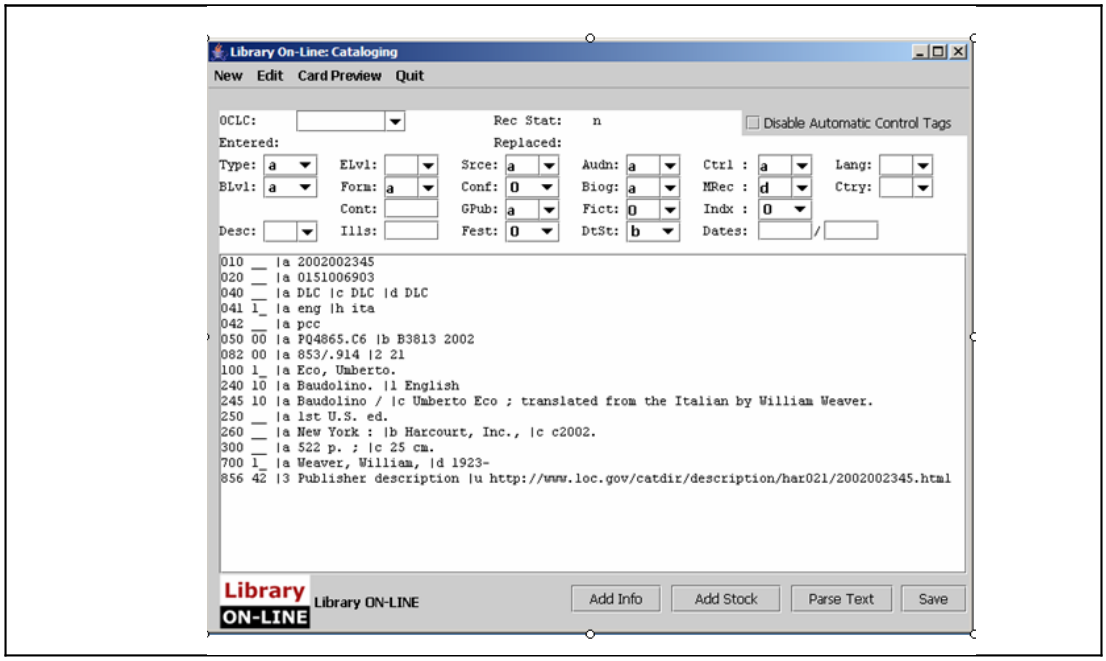


Figure 6.2. Cataloging module main window

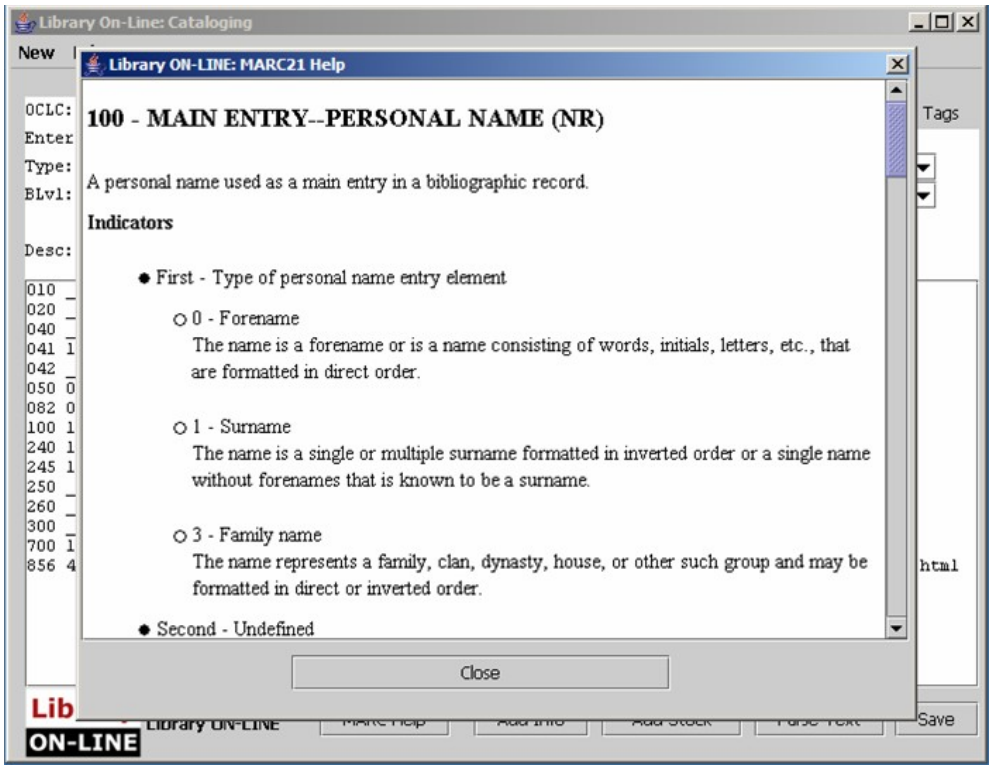


Figure 6.3. Online help facility of the desktop module

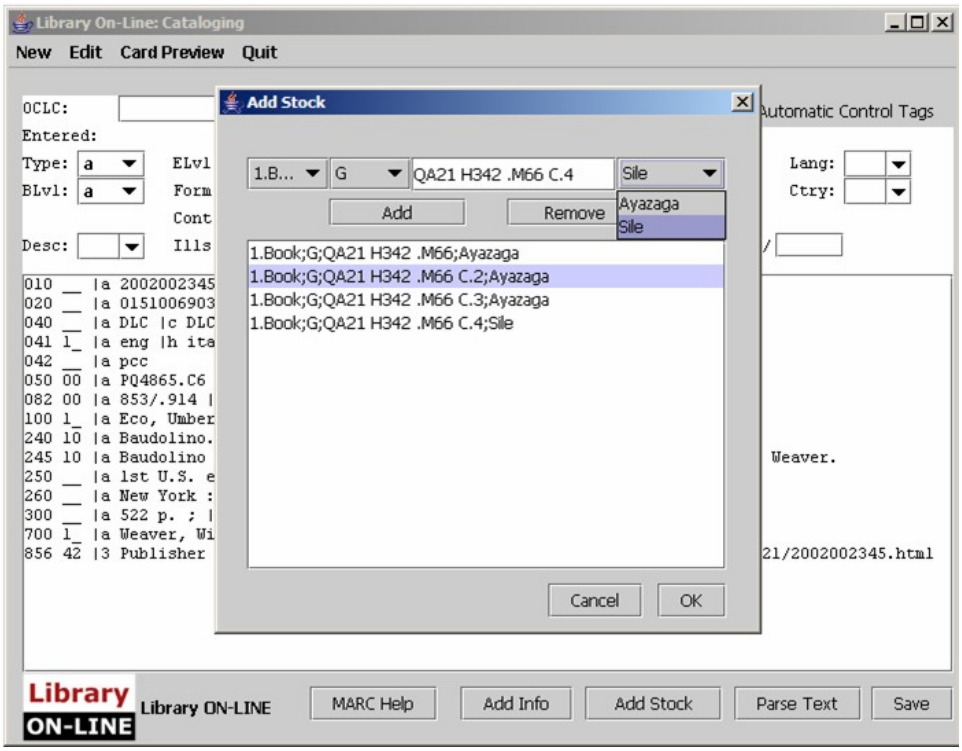


Figure 6.4. Call number and stock control dialog

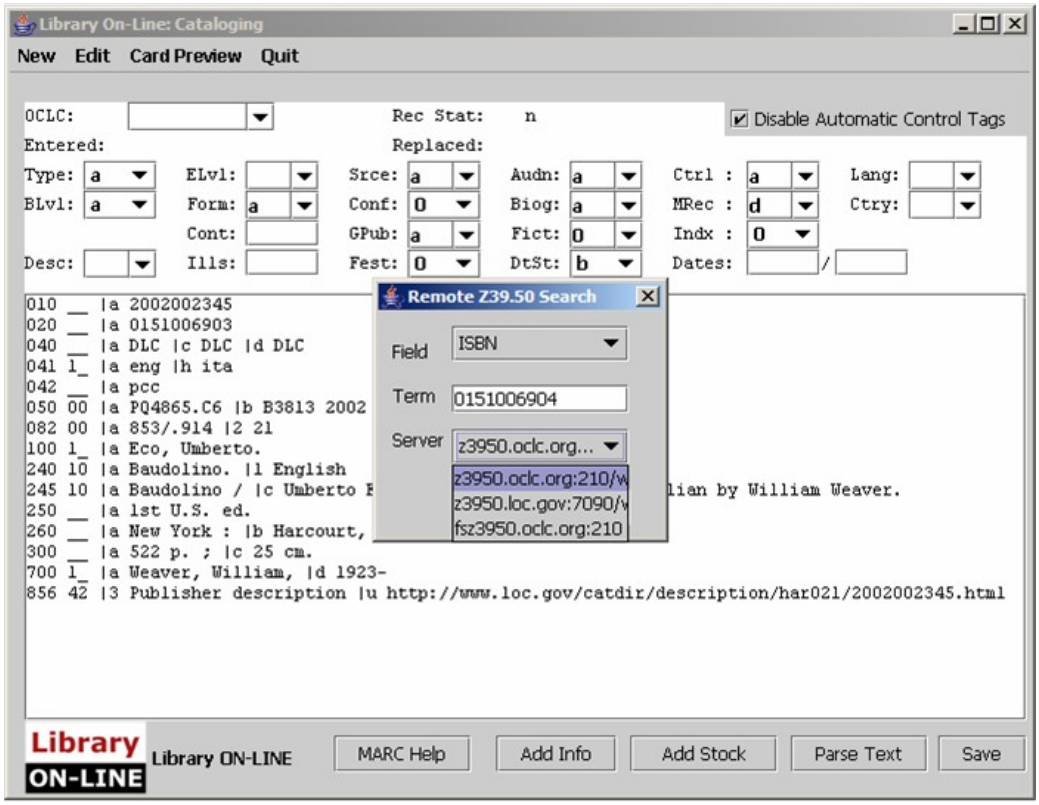


Figure 6.5. Import via remote search facility

6.1.2. Circulation Module

The circulation module is responsible from circulation library materials and keeping track of the circulation process. The interface is designed to be simple to lead to a quick process as shown in figure 6.6. The circulation desk is a busy unit of the library and checking in or out the materials in a fast and efficient way is essential.

When checking out a material, the library circulation clerk only enters the student or staff id of the patron and the call number. The system calculates the due date according to the patron type, material type and a pre-specified list of circulation rules. From the circulation module, the circulation clerk can view a patron's record of circulation as shown in figure 6.7. According the specified rules, a patron can not checkout a material if he/she has an overdue material associated with. Also there are a certain number of allowed number circulation items for a given time. The system also checks for these rules before checking out a material.

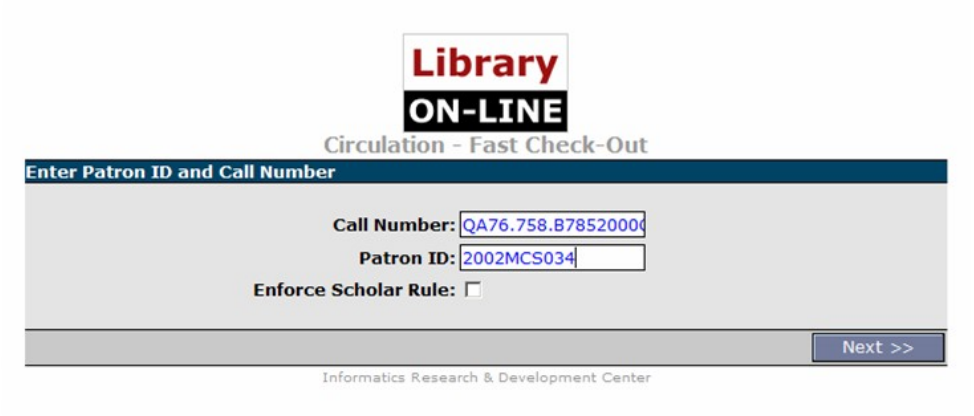


Figure 6.6. Checkout screen of the circulation module

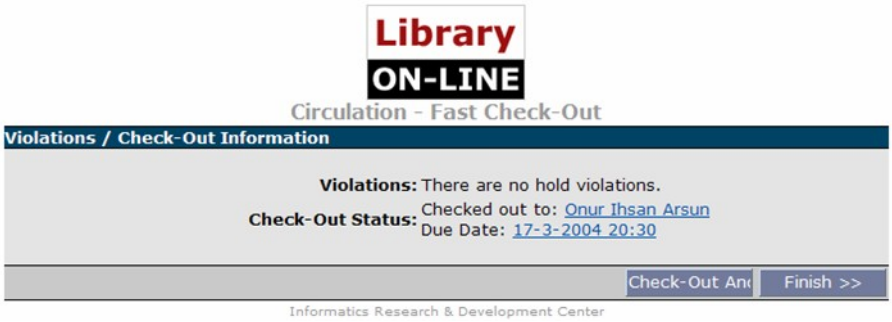


Figure 6.7. Checkout screen displays after the checkout process

The renew and check in screens are likewise as shown figure 6.8. A patron with an overdue material cannot renew a material unless the patron checks in the overdue material. The check in and renew dialogs also calculates the fines to be imposed according to the pre specified fine lists.

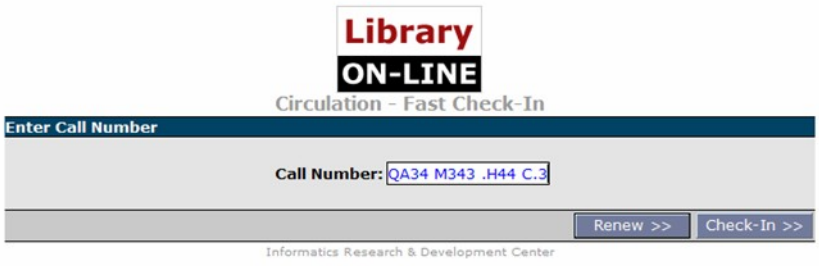


Figure 6.8. Check out screen

6.1.3. WebOPAC

The WebOPAC module is the system's one module that is open to public. The WebOPAC integrates much functionality into one module.

The patron can search the library catalog from the web via this module. The search facility has many advanced features. The search can be conducted trough combination many search criteria and can be sorted in many different ways. There is also a basic search facility (figure 6.9) for limited number criteria where a patron can reach a material faster.

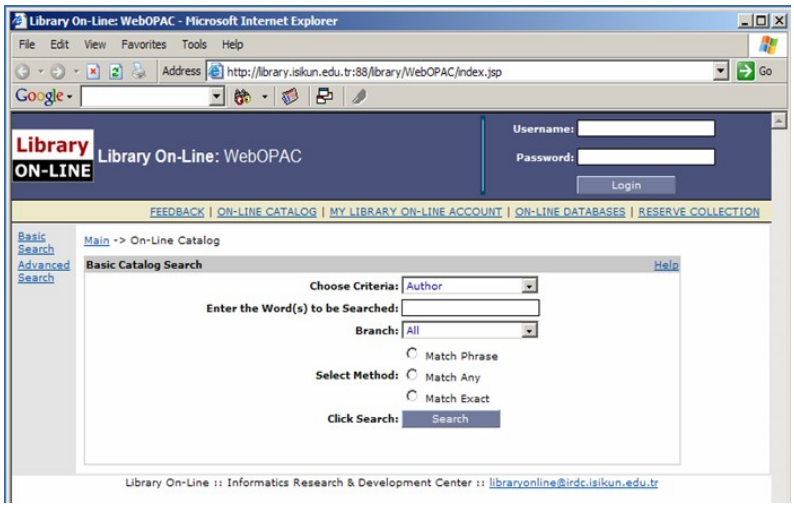


Figure 6.9. Basic search screen

The conducted searches give a detailed description of the selected material (figure 6.10). This screen also facilitates the reservation of materials. Patron can issue reservation to a certain number materials through the WebOPAC for a limited amount of time as shown in figure 6.11.

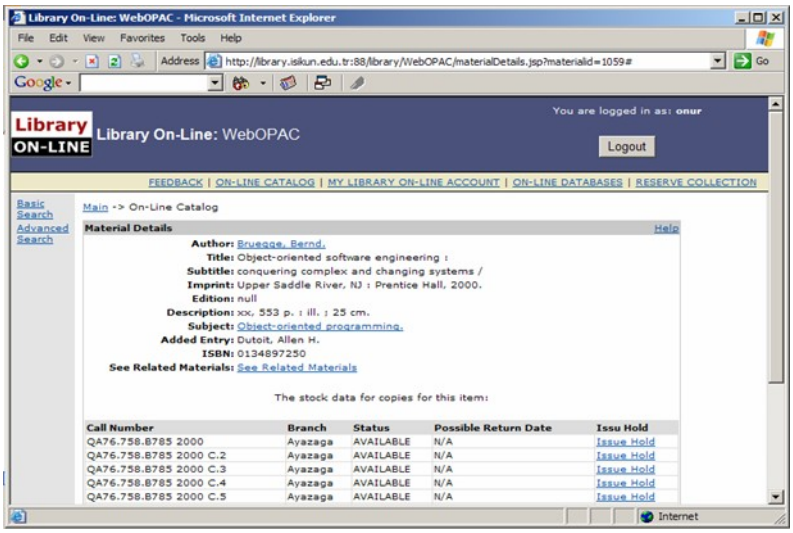


Figure 6.10. Material details screen

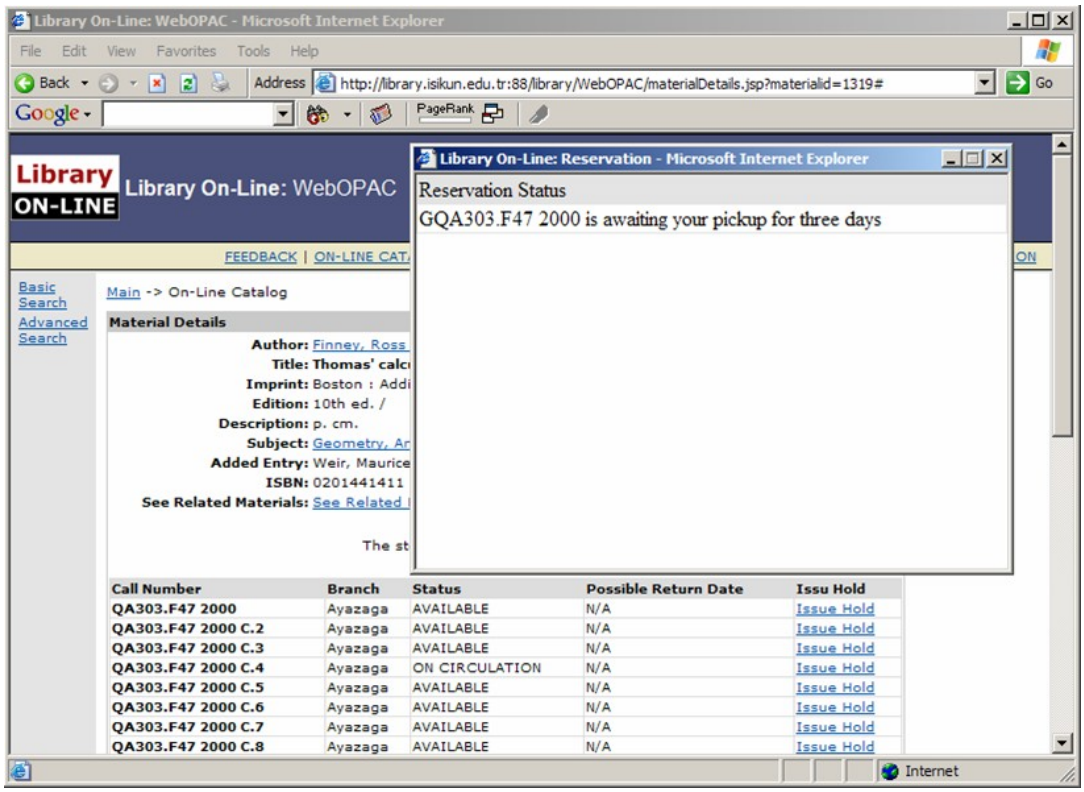


Figure 6.11. Reservation screen

The WebOPAC also includes a personalized screen for every patron as shown in figure 6.12. From this screen the patron can keep track of checkout materials and see overdue warnings. Other warnings like the recall notice are also displayed on this screen. A list of previously checked out materials can also be displayed.

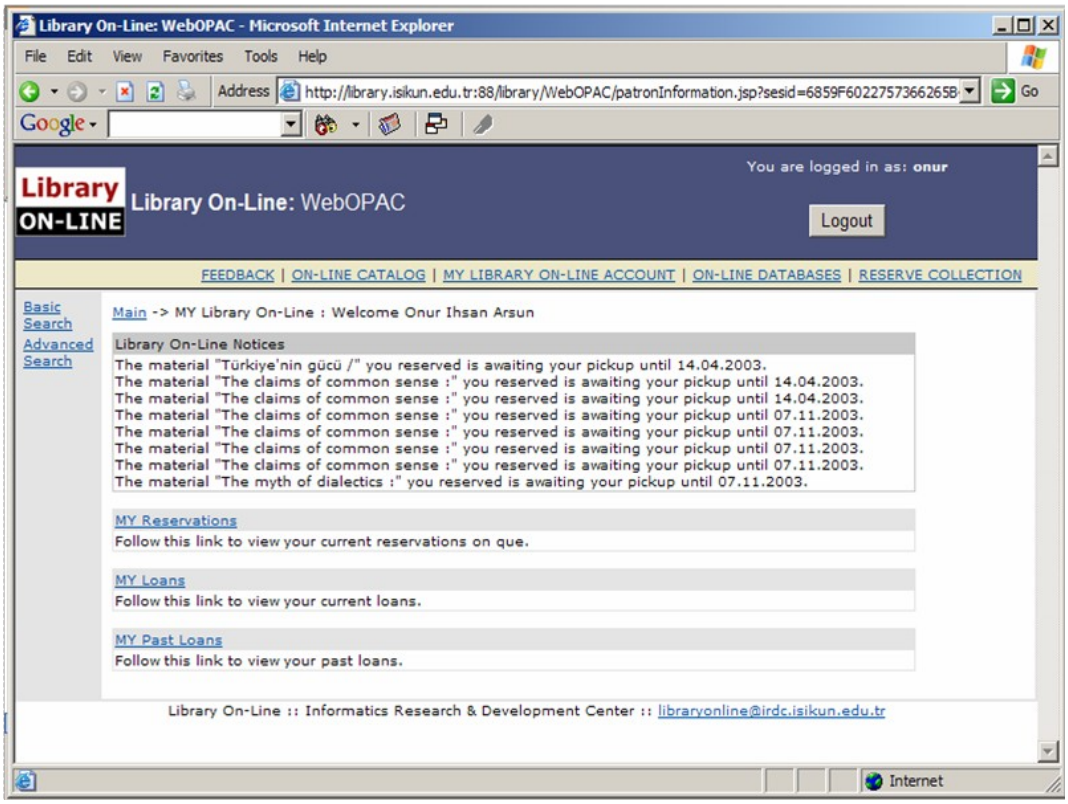


Figure 6.12. Personal patron information screen

6.1.4. Management & Reporting

The management and reporting module targets the library manager (figure 6.13). This module gives the user an overall control of the system. All system activates can be monitored via this module. The system configuration items are also modified from the module.

The catalog management sub module offers tools to monitor the catalog status. All kinds of modifications of the catalog can be carried out from this module. There is also a simple MARC21 editor for quick catalog modifications. The material details screen includes the status of all copies of the material, whether they are checkout or reserves, etc. Pre defined hold types can be issued on the materials, or recalls can be processed form the module.

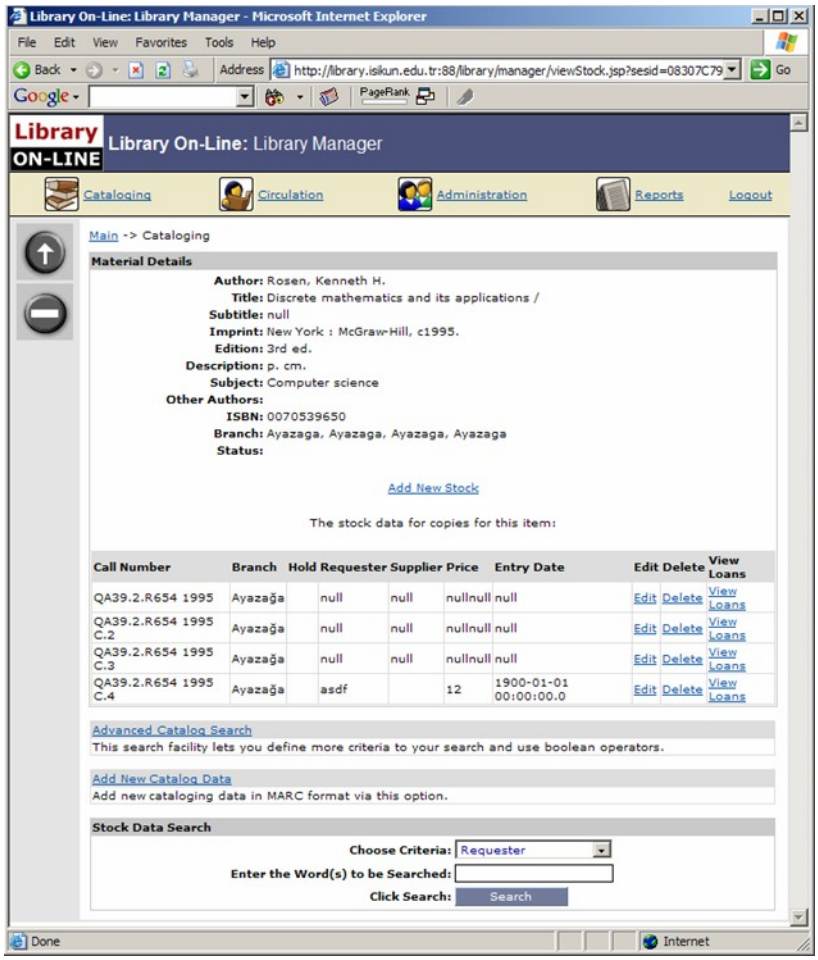


Figure 6.13. Material details screen

The patrons information and activities can be controlled or viewed from the circulation management sub module (figure 6.14). Patrons can be searched according to a lot of different criteria and their current and former circulation statistics and data can be monitored. Holds or bans to the patrons can be issued from this sub module.

In this sub-module there are many cross references to the catalog management sub module. All information is hyperlinked to the related screens. This cross referencing easies the monitoring process for the library manager.

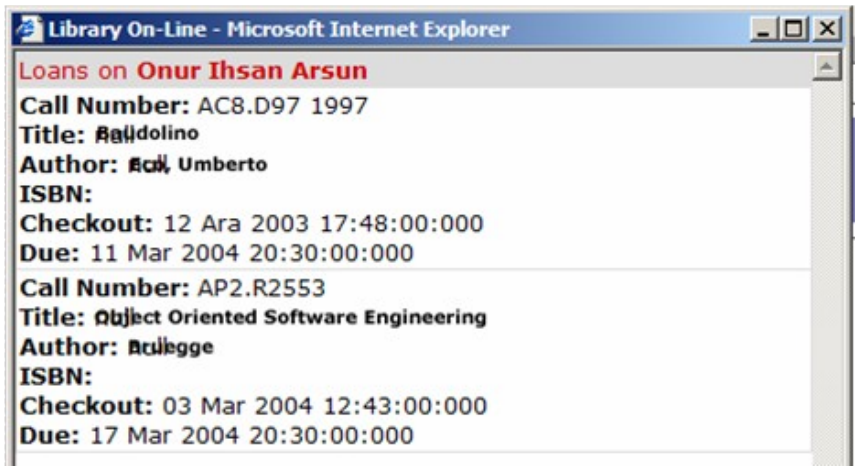


Figure 6.14. Patron circulation information screen

The other sub module is the system behavior management as shown in figure 6.15. The following operations can be carried out from the sub module:

- User Type Management: As has been explained before, the university information systems including Library ON-LINE share a common user database. However, user type management is left to individual modules. From the user types management screen, the library manager can define valid user types for the system. The user type's definitions are especially important for circulation module. The circulation module calculates the due date of a material during check out with regard to the user type information of the patron.
- Circulation Rules (figure 6.16): The loan rules screen is an effective and scalable feature of the system. The circulation rules can be defined according to many criteria like the user type, material type or specific information like the scholar rules. The circulation rules screen covers all criteria for implementing a particular rule.
- Opening & Closing Hours: The opening & closing hours, and working day of the library are important in circulation, especially in reserve collection materials. The due date of these materials are calculated according to the working hours for the library.
- Hold Management: Holds are important information of library catalogs. Hold types can be defined from this screen. Also, the

actions to be taken in the existence of a hold is also given from this screen. For example, if a recall hold is issued the steps to be taken can be defined as follows: Issue the hold, if the material is in circulation, send an e-mail to the patron and display a message on personalized part of the WebOPAC. The hold management screen is a sophisticated behavior modeling module.

- Collection Management: Collections are also important in circulations. The collection which a material belongs to is also taken into account in calculating the due date of materials to be checked out.

There are other minor tools that the system management sub module provides, like the messaging management, material type, etc.

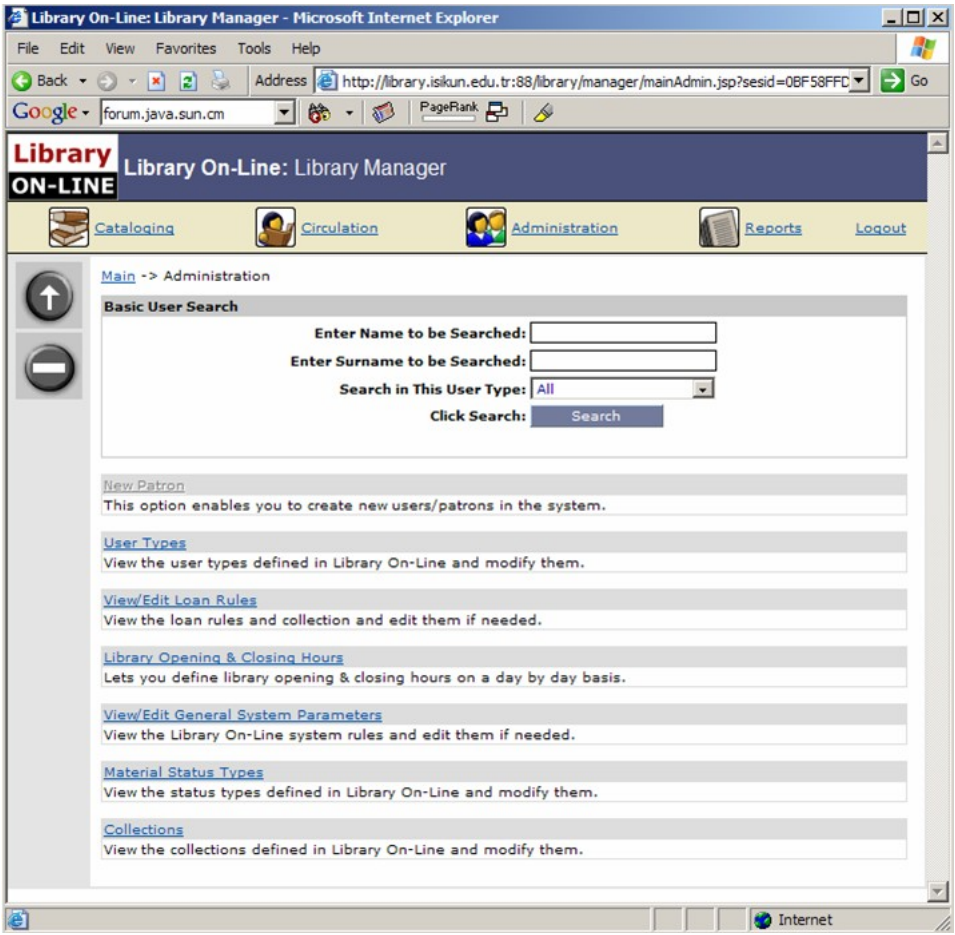


Figure 6.15. System administration sub module

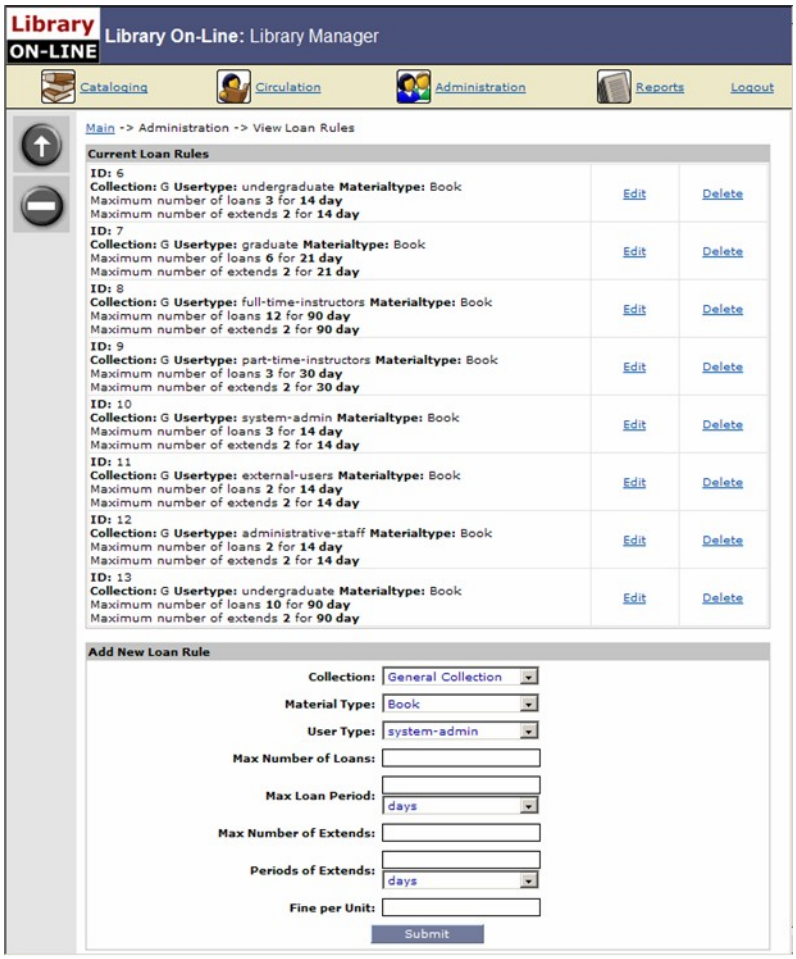


Figure 6.16. Circulation rule management screen

The system administration module includes many pre-defined reports a generic report generator. Some reports are given in the following figures 6.17 and 6.18.

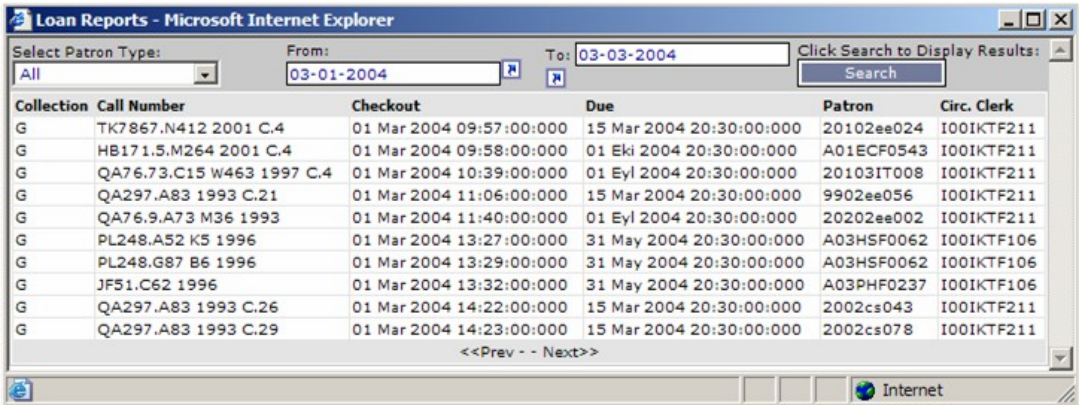


Figure 6.17. Circulation report

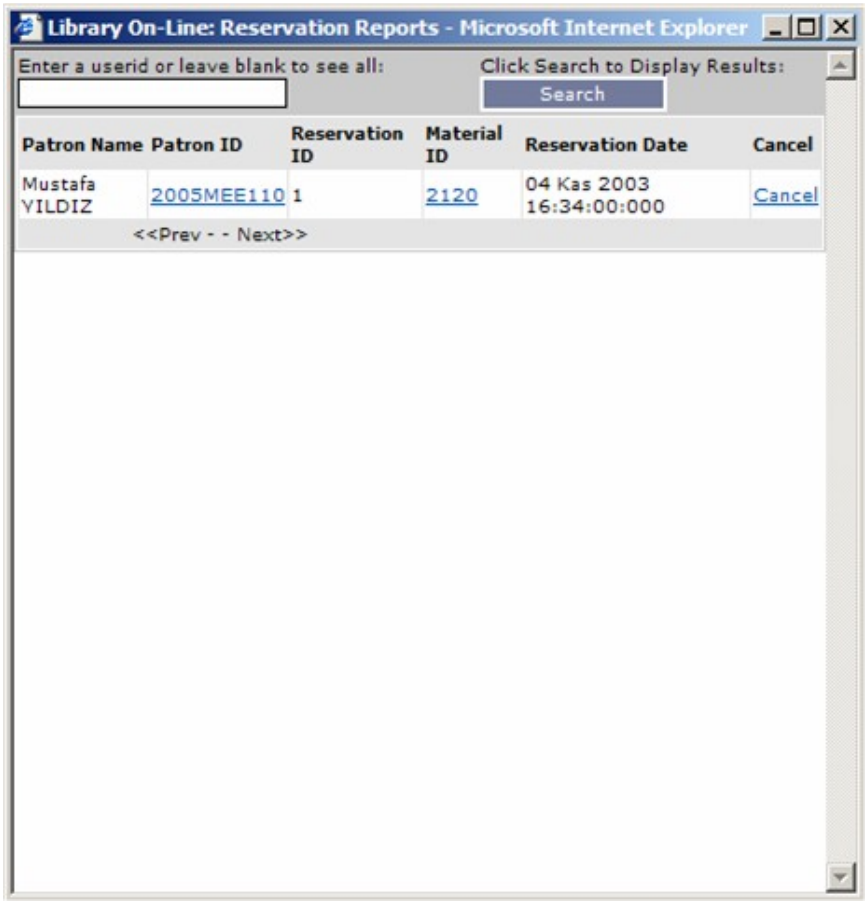


Figure 6.18. Reservation report

6.1.5. Helper Modules

The Library On-LINE library management system includes many helper modules varying in sizes and mostly transparent to the end user. These are the electronic mailing system, z39.50 server, z39.50 client, circulation service, etc.

The helper modules specifies the facilities of the system in general and helps the system to run smoothly.

For example, the circulation service, checks for the integrity of the circulation data. It verifies the circulation tables against the catalog and stock table to reduce redundancy

6.2. Technical Overview

6.2.1. Architecture

Library ON-LINE is developed completely in the Java programming language. It implements a 3-tier architecture, namely, the data (persistence) layer, business layer and the presentation layer as shown in figure 6.19.

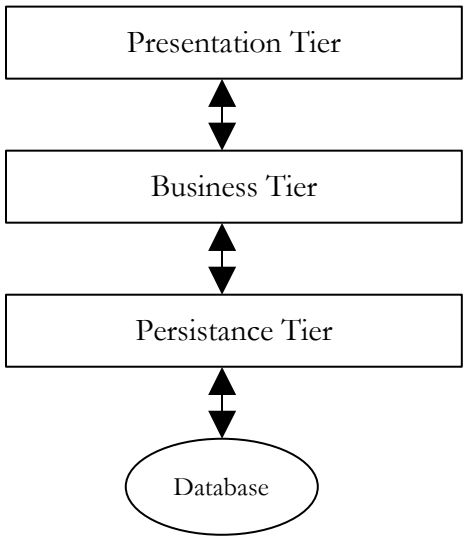


Figure 6.19. Architecture diagram of Library ON-LINE

The data or the persistence layer consists of object mappings of the database entities. All the tables and attributes are mapped to objects, and the business layer never need to directly access the database. For the model, JDO (http://java.sun.com/products/jdo/index.jsp) model developed by Sun Microsystems. Using a persistence layer significantly reduces the maintenance time of the code.

The business layer implants all the functionality of the system. All processes, tools, and facilities are implanted on this layer. The business layer never actually accesses the database. It uses the persistence layer for data management.

The presentation layer is what the end user sees. The presentation layer in Library ON-LINE has two different platforms. A web based interface and a desktop interface. The presentation tiers are actually thin clients and they do not implement any functionality. All the job is transferred to the business layer.

The advantages of using a 3-tier approach is given below:

- Clear separation of user-interface-control and data presentation from application-logic. Through this separation more clients are able to have access to a wide variety of server applications. The two main advantages for client-applications are clear: quicker development through the reuse of pre-built business-logic components and a shorter test phase, because the server-components have already been tested.
- Re-definition of the storage strategy won't influence the clients. RDBMS' offer a certain independence from storage details for the clients. However, cases like changing table attributes make it necessary to adapt the client's application. In the future, even radical changes, like let's say switching form an RDBMS to an OODBS, won't influence the client. In well designed systems, the client still accesses data over a stable and well designed interface which encapsulates all the storage details.
- Business-objects and data storage should be brought as close together as possible, ideally they should be together physically on the same server. This way - especially with complex accesses - network load is eliminated. The client only receives the results of a calculation - through the business-object, of course.
- In contrast to the 2-tier model, where only data is accessible to the public, business-objects can place applications-logic or "services" on the net. As an example, an inventory number has a "test-digit", and the calculation of that digit can be made available on the server.

- As a rule servers are "trusted" systems. Their authorization is simpler than that of thousands of "untrusted" client-PCs. Data protection and security is simpler to obtain. Therefore it makes sense to run critical business processes, that work with security sensitive data, on the server.
- Dynamic load balancing: if bottlenecks in terms of performance occur, the server process can be moved to other servers at runtime.
- Change management: of course it's easy - and faster - to exchange a component on the server than to furnish numerous PCs with new program versions. To come back to our VAT example: it is quite easy to run the new version of a tax-object in such a way that the clients automatically work with the version from the exact date that it has to be run. It is, however, compulsory that interfaces remain stable and that old client versions are still compatible. In addition such components require a high standard of quality control. This is because low quality components can, at worst, endanger the functions of a whole set of client applications. At best, they will still irritate the systems operator.
- It is relatively simple to use wrapping techniques in 3-tier architecture. As implementation changes are transparent from the viewpoint of the object's client, a forward strategy can be developed to replace legacy system smoothly. First, define the object's interface. However, the functionality is not newly implemented but reused from an existing host application. That is, a request from a client is forwarded to a legacy system and processed and answered there. In a later phase, the old application can be replaced by a modern solution. If it is possible to leave the business object's interfaces unchanged, the client application remains unaffected. A requirement for wrapping is, however, that a procedure interface in the old application remains existent. It isn't possible for a business object to emulate a terminal. It is also important for the project planner to be aware that the implementation of wrapping objects can be very complex.

Library ON-LINE currently runs on Sun Java System Application server and uses Microsoft SQL Server 2000 as a RDBMS.

The system does not use any third party module, except the JDBC driver.

6.2.2. Library ON-LINE Metrics

Some metrics about Library ON-LINE is given in below:

- Business modules code lines: 42,433 LOC.
- Visual interfaces code lines: 74,462 LOC.
- Desktop Module code lines: 11,336 LOC.

- Searching and displaying the material table (480,000+ tuples) for a "LIKE" query, returning 56 rows for 10 requests/second: 1,287 secs.

- Searching and displaying the material table (480,000+ tuples) for a "LIKE" query, returning <10 rows for 10 requests/second: 0,89 secs.

- Searching and displaying the material table (480.000+ tuples) for a "=" query, returning <10 rows: 0,39 secs.

- Compiling SQL queries for first use after deployment: ~0,13 secs/query (average).

- Compiling Web Interfaces for first use after deployment: ~2,88 secs/page (average).

- Physical Memory Usage (all business module loaded) with 10 requests/sec: 14 Mb.

- Physical Memory Usage for Desktop Module: 2,6 Mb.

- Load sharing between Database Server and Application server during searching and displaying the material table (480,000+ tuples) for a "LIKE" query, returning 56 rows for 10 requests/second: 83% (DB Server) – 17% (App. Server). (Appr.).

- Total number of tuples in the catalog tables: 830.290

CHAPTER 7. EVALUATION OF APPROACHES

This chapter introduces an evaluation of approaches discussed through the previous chapters.

7.1. Using Portioned Tables

Approach: Relocate less used MARC21 tags to separate table, leaving the most used few in one table.

Implementation: In Library ON-LINE, we created three separate tables. The tags in each table are given in Table 7.1.

Table 7.1. Portioning MARC21 tags

Table 1	Table 2	Table 3
020: ISBN 100: Author 245: Title 250: Edition 600, 610, 650: Subject	Everything but table 1 and table 3	All 9XX tags

Evaluation: Gain on performance is the most important outcome expected. A sample table is taken, which is a large subset of the original table we are using in Library ON-LINE. This table stores all the MARC tags in the Relation MARC Representation described in the previous chapters. Table 7.2 gives statistics for the table.

Table 7.2. MARC storage table statistics

Attribute	Value
Total Number of Rows	469.337
Number of Rows Storing 9XX tags	237.390
Number of Rows Storing 020, 100, 245, 600, 610 and 650 tags	64.330

As seen in the table, Table 1 which is the most used table has only 64.330 rows. Instead of using a table of 469.337 rows, this is much faster. Table 7.3 shows the query execution times of each table for a SELECT query that returns all rows in tables.

Table 7.3. Query execution times

Table	Duration (secs)
Original Table	7.2
Table 1	2.7
Table 2	3.4
Table 3	3.1

This table summarizes all. There is an obvious performance gain. Table 1 is the most frequently used table for catalog operations, since it stores most frequently needed tags.

The drawback of this method is that it creates dependencies among tables. Whenever catalog information for a material is to updated, delete, or a new record is to be inserted, changes should be made for all tables. This creates complexities in programmatically implementing these operations.

7.2. Using Data Sources

Approach: Connection pools and data source should be used n order to avoid connection opening and closing overhead for each SQL query.

Implementation: Connection pools in Library ON-LINE are implemented through java ConnectionPool and DataSource classes. The application server handles the creation and management of the connection pool, given a configuration. The connection pools are reached using Java Naming and Directory Interface (JNDI). A sample code is given in figure 7.1 below.

```
if (this.connection == null || this.connection.isClosed()) {

                InitialContext ic = new InitialContext();
                Context envContext = (Context) ic.lookup ("java:comp/env");
                DataSource dataSource = (DataSource) envContext.lookup ("jdbc/jdbc-
library");
                this.connection = dataSource.getConnection ();
            }
```

Figure 7.1. Data source code fragment

Evaluation: Connection pools are good for just one thing. They eliminate the overhead of opening or closing connections to the database for each query. But they must be configured properly. An example configuration is given in table 7.3.

Table 7.4. Example connection pool configuration

Attribute	**Value**
Initial Pool Size	32
Pool Enlargement De/Increment	4
Maximum Pool Size	128
Connection Time Out	30000 milliseconds

There are problems implementing connection pools. These rise from idle, open connections. There are two reasons for that: Either the programmer forgets to include code fragment to release acquired connection back to the connection pool, or even if the code for that is included, the platform may not return the connection back. This results in idle connections, and because of them new connections are created in the connection pool, resulting in many idle connections to the database. This takes from performance. The configuration in Table 7.3 is an optimum one. Table 7.4 shows the gained times while using connection pools.

Table 7.5. Connection pooling performance gain

Opening Connection to the Database per Query	0,18 seconds
Closing Connection to the Database After Query	0,07 seconds

The table above shows that by using connection pooling we gain 0,25 seconds per query. That’s a huge amount of time considering concurrent users sending queries to the database each.

7.3. Using Pre-Compiled SQL Queries

Approach: Pre-compile the SQL queries so that we gain on performance on every query.

Implementation: The Java programming language presents a special class that implements pre-compiled SQL queries approach, namely PreparedStatement class. During the initialization of the application, the PreparedStatement queries are compiled. In Library ON-LINE, we have used this Java class. A sample code fragment is shown in Figure 7.2.

```
PreparedStatement updateSales = con.prepareStatement(
            "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ? ");
updateSales.setInt(1, 75);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate():
```

Figure 7.2. Java PreparedStatement sample code fragment

Evaluation: As can be seen, it is important to set the parameter of the query via the setXXX method of the PreparedStatement class, otherwise the there is no advantage for the PreparedStatement class. Using parameterized queries with prepared statements reduces the load on the database by allowing it to reuse access plans that were already prepared. This cache is database-wide so if you can arrange for all your applications to use similar parameterized SQL, you will improve the efficiency of this caching scheme as an application can take advantage of prepared statements used by another application. This is an advantage of an application server because logic that accesses the database should be centralized in a data access layer

When using PrepatedStataments, there are a few things to consider. Things can get more complicated when we use a J2EE server. Normally, a prepared statement is associated with a single database connection. When the connection is closed, the preparedstatement is discarded. Normally, a fat client application would get a database connection and then hold it for its lifetime. It would also create all prepared

statements eagerly or lazily. Eagerly means that they are all created at once when the application starts. Lazily means that they are created as they are used. An eager approach gives a delay when the application starts but once it starts then it performs optimally. A lazy approach gives a fast start but as the application runs, the prepared statements are created when they are first used by the application. This gives an uneven performance until all statements are prepared but the application eventually settles and runs as fast as the eager application. Which is best depends on whether you need a fast start or even performance.

The problem with a J2EE application is that it can't work like this. It only keeps a connection for the duration of the request. This means that it must create the prepared statements every time the request is executed. This is not as efficient as the fat client approach where the prepared statements are created once, rather than on every request. J2EE vendors have noticed this and designed connection pooling to avoid this performance disadvantage.

When the J2EE server gives your application a connection, it isn't giving you the actual connection; you're getting a wrapper. You can verify this by looking at the name of the class for the connection you are given. It won't be a database JDBC connection, it'll be a class created by your application server. Normally, if you called close on a connection then the jdbc driver closes the connection. We want the connection to be returned to the pool when close is called by a J2EE application. We do this by making a proxy jdbc connection class that looks like a real connection. It has a reference to the actual connection. When we invoke any method on the connection then the proxy forwards the call to the real connection. But, when we call methods such as close instead of calling close on the real connection, it simply returns the connection to the connection pool and then marks the proxy connection as invalid so that if it is used again by the application we'll get an exception.

Wrapping is very useful as it also helps J2EE application server implementers to add support for prepared statements in a sensible way. When an application calls Connection.prepareStatement, it is returned a PreparedStatement object by the driver. The application then keeps the handle while it has the connection and closes it before

it closes the connection when the request finishes. However, after the connection is returned to the pool and later reused by the same, or another application, then ideally, we want the same PreparedStatement to be returned to the application.

Finally, the correct use of prepared statements also lets you take advantage of the prepared statement cache in the application server. This improves the performance of your application as the application can reduce the number of calls to the JDBC driver by reusing a previous prepared statement call. This makes it competitive with fat clients efficiency-wise and removes the disadvantage of not being able to keep a dedicated connection.

7.4. Using Statement Caches

Approach: Make extensive use of available SQL statement caching mechanisms.

Implementation: Almost all database management systems provide query caching mechanism by default. In Library ON-LINE, we used Microsoft SQL Server 2000, which has query caching mechanism. Also, we took advantage of statement caching facilities of Java PreparedStatements.

Evaluation: Query caching mechanisms provided by the DBMS, are absolutely offer better performance. SQL Server is configure to use a pool of memory of the server and it will allocate the majority of this memory pool to hold both data pages that have been read and the compiled execution plans for all Transact-SQL statements. It is this dynamic pool of memory that is being referred to and the data cache and procedure cache, keep in mind that in versions of SQL Server before SQL Server 7.0 the data cache and procedure cache were two separate pools of memory and could be controlled separately, today one pool of memory is used both for data and execution plans.

SQL Server will manage the objects in its cache in a few main ways: freeing up buffers or aging execution plans. A buffer is a page in memory that is the same size

as a data or index page and is used to hold one page of data from the database. The buffer pool is managed by a process called the lazywriter, this lazywriter uses a clock algorithm to sweep through the buffer pool and free up any clean buffers to keep a supply of buffers empty for the next set of data pages. As the lazywriter visits each buffer it will determine whether that buffer has been referenced since the last lazywriter sweep, it does this by examining a reference count value in the buffer header, the reference count is adjusted up by 1 each time a statement references that buffer. If the reference count is not 0, the buffer will stay in the pool, but its reference count will be adjusted downward for the next sweep. To make this downward adjustment the lazywriter will divide the reference counter in the buffer page header by 4 and discard the remainder. When the reference counter goes to 0, the dirty page indicator is checked and if the page is dirty(modifications have been made to the data since the data page was placed in memory), a write is scheduled to write the modifications to disk. The lazywriter will also sweep the buffer pool when the number of pages on the free list falls below a minimum value, this value is computed as a percentage of the overall buffer pool size but is always between 128KB and 4MB. SQL Server will adjust this computed size based on the load on the system and the number of buffer stalls occurring. A buffer stall is when a process needs a buffer to hold data but none are available. This process will be go to sleep until the lazywriter can free some buffers. If the number of stalls increased to more than a few a second then SQL Server will adjust the computed size of the free list upward, the computed size will be adjusted downward if the load is light and very few buffer stalls are occurring.

SQL Server will also manage the cache by aging execution plans. Execution plans used to just mean the execution plans compiled for stored procedures, but with SQL Server 2000 these execution plans can also refer to ad-hoc SQL statement plans, an ad-hoc SQL statement is basically any statement that is not a stored procedure, an autoparameterizex query, a sp_executesql statement or a statement prepared and executed with the ODBC/OLE DB SQLPrepare/SQLExecute or ICommandPrepare commands. Once an execution plan is compiled the plan will be saved to the cache along with a cost factor that is determined by the cost of actually creating the plan, this value will be set to 0 if the statement was an ad-hoc statement

and to the actual cost if the plan is not for an ad-hoc statement, the cost is largely the I/O needed to compile the plan. A 0 cost factor value means that the plan can be immediately dropped from the cache. SQL Server's lazywriter will sweep the cache and deallocate the execution plan if the memory manager requires memory and all available memory is currently in use, if the cost factor value is 0 and if the object is not currently referenced by a connection. Execution plans, even ad-hoc plans, can stay in memory until SQL Server is shut down if another process determines it can use the compiled plan and the plan is constantly being reused. Ad-hoc plans will have their cost factor value increased by 1 each time it is reused, the highest ad-hoc cost factor value can go is its actual cost to compile. Non ad-hoc plans will the cost factor value set back to their original compile cost values.

We also use the statement caching mechanism provided by the Java PreparedStatatment class. All statements (not just prepared statements) are usually cached by the Database Server. The reason Prepared Statements execute quicker is because the SQL is the same regardless of any query parameters. So using normal statements like in the queries below

```
select * from my_table where name='a'

select * from my_table where name='b'
```

will be parsed and cached separately, where as using a prepared statement of the form like below

```
select * from my_table where name=?
```

will only be cached once, and the second time the query is executed (perhaps using a different bind parameter) the overhead of parsing the sql is not required.

This only provides a performance gain if;

- The prepared statement is used frequently,

- The time taken to parse the query is a significant percentage of the total query execution time,
- A large number of values are used for the bound variable. (e.g. if 'a' and 'b' were the only valid values for name in the example above then both statements would still be cached and re-used with the same performance gain).

The other advantage of using prepared statements (as mentioned above) is that prepared statements don't requiring escaping of data dynamically added to queries, where as data dynamically added to statements should *always* be escaped to ensure against problems with accidental and/or unauthorized database access.

However, there is also a dis-advantage with Prepared Statements with some databases (e.g. Oracle) in that the Cost Based Optimizer cannot effectively optimize queries that contain bound variables (because the optimizer only runs on the SQL statement), which means you can actually introduce a performance loss by unilaterally switching to Prepared Statements.

J2EE PreparedStatement Cache is implemented using a cache inside the J2EE server connection pool manager. The J2EE server keeps a list of prepared statements for each database connection in the pool. When an application calls prepareStatement on a connection, the application server checks if that statement was previously prepared. If it was, the PreparedStatement object will be in the cache and this will be returned to the application. If not, the call is passed to the JDBC driver and the query/preparedstatement object is added in that connections cache.

We need a cache per connection because that's the way JDBC drivers work. Any preparedstatements returned are specific to that connection.

If we want to take advantage of this cache, the same rules apply as before. We need to use parameterized queries so that they will match ones already prepared in the cache. Most application servers will allow you to tune the size of this prepared statement cache.

CHAPTER 8. CONCLUSION AND RECOMMENDATIONS FOR FURTHER WORK

In this thesis, issues catalog design and implementation, transaction processing, development process, catalog data conversion, interoperability, and integration for library management systems have been discussed.

We employed a new set of approaches to these issues in relational MARC representation, pre-compiled SQL queries, caching mechanisms, use of eXtreme Programming, object oriented design, analysis, and programming and design patterns.

We utilized these approaches in our library management system called Library ON-LINE. Library ON-LINE is now being used in Işık University, serving more than 2000 students and staff. Evaluations have been made on these approaches in chapter 8.

To go further with our work, we plan to do in-depth evaluations on the usage of our proposals and define new ontology for describing cataloging information. We also need to develop means to further increase the performance of library management systems, or generally application software that relies on heavily stored data. Some further evaluations should be done about the proposed methods, especially on database indexing and usage of development processes.

REFERENCES

[1] Furrie, B., "Understanding MARC Bibliographic: Machine-Readable Cataloging," *Catalog Distribution Service, Library of Congress,* 6th ed. Washington, DC, 2000

[2] Sally, H. M., "MARC Data in an SGML Structure," *MARC Standards Office, Library of Congress*, Washington DC, October 9, 1996

[3] S. Hayes, "An Introduction to extreme Programming," *Second International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Sardinia, 2001

[4] Library of Congress. Processing Department, "MARC 21 Format for Bibliographic Data," *Catalog Distribution Service, Library of Congress,* 2nd ed., Washington, DC, 1999

[5] National Information Standards Organization Staff, "Information Interchange Format, Z39.2-1994," *NISO Technical Document Number: ANSI/NISO Z39.2-1994,* Bethesda, MD, 1994

[6] Tingle, B., "METS Schema Documentation," http://ark.cdlib.org/mets/schema_documentation/, 2002

[7] Crawford, R. G, Becker, H. S., Ogilvie, J. E., "A Relational Bibliographic Database", *The Canadian Journal of Information Science*, Vol. 9, pp. 21-28, June 1984,.

[8] National Information Standards Organization Staff, "Information Retrieval (Z39.50): Application Service Definition and Protocol Specification," *NISO Technical Document Number: ANSI/NISO Z39.50-2003,* Bethesda, MD, 2003

[9] Moen, W., "The ANSI/NISO Z39.50 Protocol: Information Retrieval in the Information Infrastructure," pp. 40-44, Bethesda, MD, 2003

[10] Lunau, C., "Issues Related to the use of Z39.50 to Emulate a Centralized Union Catalogue", *Prepared for the ARL Access Committee. National Library of Canada,* pp. 39-56, April, 1997

[11] Taeyeon, K., Noh, B., "Internetworking between OSI and TCP/IP network managements with security features," *Proceedings of the 1995 International Conference on Network Protocols,* pp. 278, 1995

[12] Blue Angel Technologies, Inc., “An evaluation of Z39.50 within the SILO Project,”. Chester, PA, 2001

[13] Z39.50 International Maintenance Agency Staff, “ZING: z39.50 Next Generation,” *Library Of Congress Catalog Distribution Service*, Washington, DC, 2002

[14] Taylor, M., “A Gentle Introduction to CQL,” http://zing.z3950.org/cql/intro.html, 2003

[15] Taylor, M., “ZOOM: The Z39.50 Object Model,” http://zoom.z3950.org/index.html, 2001

[16] Taylor, M., “An Overview of ZeeRex,” http://explain.z3950.org/overview/index.html, 2002

[17] Dovey, M. J., “Simple Implementation of Z39.50 over SOAP using XER,” *An Introduction to Jafer*, Oxford University, UK, 2001

[18] Curbera F., et al. “Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI,” *IEEE Internet Computing,* pp. 86-93, March, 2002.

[19] Mitra N., “SOAP Version 1.2 Part 0: Primer”, *W3C Recommendation,* 2003

[20] Pareto, V., “Manual of Practical Economy,” 1927

[21] Altendorf E., Hohman M., Zabicki R., “Using J2EE on a Large, Web-Based Project”, *IEEE Software*, pp.81-89, March 2002

[22] Torp, K, Jensen, C. S., Snodgrass, R. T., “Stratum Approaches to Temporal DBMS Implementation,” *Proceedings of International Database Engineering and Applications Symposium,* pp. 4, July, 1998

[23] W.C. Wake, The XP series: Extreme Programming Explored, Addison-Wesley, NJ, 2002

[24] S. Hayes, “An Introduction to extreme Programming,” *Second International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Sardinia, 2001

[25] Kelley, C., “The Waterfall Model,” http://www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm, 2000.

[26] Royse, W.W., “Managing the development of large software systems”, *in Tutorial: Software Engineering Project Management, IEEE Computer Society*, Washington, DC, pp. 118-127, 1970.

[26] M. Kircher, Jain P., Corsaro A., "XP + AOP = Better Software?," *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Sardinia, 2002

[27] McCormick M., "Programming Extremism," *Communications of the ACM*, Volume 44, No. 6, 2001

[28] Booch, G., "Object-Oriented Analysis And Design With Applications", 2nd Ed. Benjamin Cummings. ISBN 0-8053-5340-2. pp.83, 1998

[29] Coplien, O. J., "Advanced C++ Programming Styles and Idioms". Addison Wesley pp280, 1997

[30] Cox, Brad J., "Object-Oriented Programming, An Evolutionary Approach," Addison Wesley, NJ, 2001

[31] Martin J., Odell, J. J., "Object-Oriented Analysis and Design," Prentice-Hall, Englewood Cliffs, NJ. Pp241, 1998

[32] Rumbaugh J., et al., "Object-Oriented Modeling and Design". Prentice Hall, NJ, 1997

[33] Shlaer S., Mellor S. J., "Object-Oriented Systems Analysis: Modeling the World in Data," Pp14, 1996

[34] Jacobson I., et al. "Object-Oriented Software Engineering - A Use Case Driven Approach," ACM Press/Addison Wesley, 1994

[35] LaLonde, W. R., Pugh, J. R., "Inside Smalltalk: Volume 1,", Prentice Hall, NJ, 1994

[36] Alur, D., "Core J2EE Patterns: Best Practices and Design Strategies, Second Edition," Prentice Hall, NJ, 2003

[37] Sompel, H. V., Bergmark, D., "Distributed Registry for OpenURL Metadata Schemas with an OAI-PMH Conformant Central Repository," *Proceedings of 2002 International Conference on Parallel Processing Workshops (ICPPW'02),* pp.469, 2002

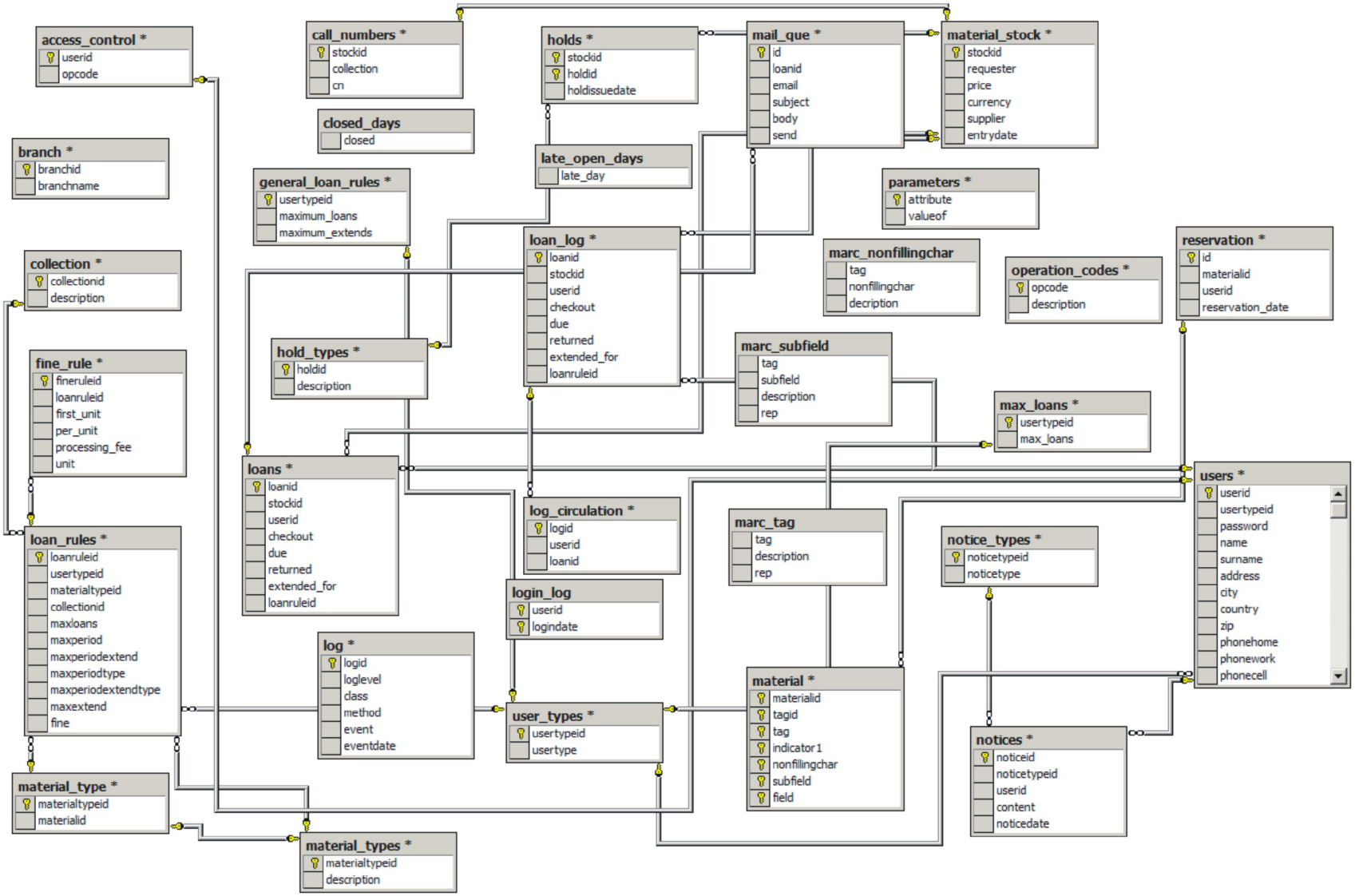
[38] Eastlake, D., Jones, P., "US Secure Hash Algorithm 1 (SHA1)," *Netwoek Working Group, RFC: 3174*, 2001

[39] Stallings, W., "Cryptography and Network Security," pp. 362, 2001

APPENDIX A. LIBRARY ON-LINE DATABASE DIAGRAM

In this appendix, the database diagram of Library ON-LINE is given. The database is implemented in Microsoft SQL Server 2000. All of the tables and relationships are shown in the diagram. Primary keys are also marked.

The relationship representation also displays the level of participation in the relationship for each participating table.


access_control *
userid
opcode
call_numbers *
stockid
collection
cn
holds *
stockid
holdid
holdissuedate
mail_que *
id
loanid
email
subject
body
send
material_stock *
stockid
requester
price
currency
supplier
entrydate
branch *
branchid
branchname
closed_days
closed
late_open_days
late_day
general_loan_rules *
usertypeid
maximum_loans
maximum_extends
parameters *
attribute
valueof
loan_log *
loanid
stockid
userid
checkout
due
returned
extended_for
loanruleid
marc_nonfillingchar
tag
nonfillingchar
decription
operation_codes *
opcode
description
reservation *
id
materialid
userid
reservation_date
collection *
collectionid
description
hold_types *
holdid
description
marc_subfield
tag
subfield
description
rep
max_loans *
usertypeid
max_loans
fine_rule *
fineruleid
loanruleid
first_unit
per_unit
processing_fee
unit
loans *
loanid
stockid
userid
checkout
due
returned
extended_for
loanruleid
log_circulation *
logid
userid
loanid
marc_tag
tag
description
rep
notice_types *
noticetypeid
noticetype
users *
userid
usertypeid
password
name
surname
address
city
country
zip
phonehome
phonework
phonecell
loan_rules *
loanruleid
usertypeid
materialtypeid
collectionid
maxloans
maxperiod
maxperiodextend
maxperiodtype
maxperiodextendtype
maxextend
fine
login_log
userid
logindate
log *
logid
loglevel
class
method
event
eventdate
user_types *
usertypeid
usertype
material *
materialid
tagid
tag
indicator1
nonfillingchar
subfield
field
notices *
noticeid
noticetypeid
userid
content
noticedate
material_type *
materialtypeid
materialid
material_types *
materialtypeid
description

APPENDIX B. USER'S MANUAL FOR LIBRARY ON-LINE

B.1. Introduction

Library ON-LINE is a web based library management software developed by the Informatics Research & Development Center at Işık University. This version of Library ON-LINE includes the following modules:

- Cataloging.
- Circulation.
- Web OPAC.
- User Management.
- Reports.
- Administration.

With the cataloging module, the whole catalog is stored in a database. Addition to the catalog may be either inserted manually or acquired automatically from remote catalogs like the Library of Congress catalog or the OCLC catalog.

The circulation module enables the library to check-in, check-out and renew materials according to previously defined flexible rules. Due date calculation is done automatically by the software.

Web OPAC module enables library patrons to conduct searches in the catalog via their web browsers. Web OPAC also includes a "MY Library ON-LINE" sub-module where the patron can keep track of his/her current and past loans make on-line renewals and issue reservations on materials.

Library ON-LINE is fully integrated to the Campus ON-LINE system. All users are able to login to Library ON-LINE with their existing accounts. The user management module automatically assigns loaning rights to users according their user types.

The reports module is a powerful reporting tool that enables library managers to make custom reports.

Administration module enables library managers to view and manage all properties of the system.

This manual only covers the Web OPAC module.

The software is fully developed in pure Java without using any third-party modules. It has a modern three-tier architecture.

B.2. Access to Library ON-LINE

Library ON-LINE can be accessed from the following web address: http://library.isikun.edu.tr.

You need to have the following versions of Internet browsers to use Library ON-LINE effectively:

- Microsoft Internet Explorer 5.0 or higher.
- Mozilla Firefox 1 or higher.
- Netscape Navigator 4.75 or higher.
- Opera 6.0 or higher.

Please make sure that you have your Campus ON-LINE username and password if you would like to use the “MY Library ON-LINE” features of the systems.

B.3. Web OPAC Module

Entering the http://library.isikun.edu.tr address takes you directly to the Web OPAC pages. Let's take a look at the functions of this module.

Please note that the online catalog does not contain all of the books or other non-book materials available in the library. The process of transferring the catalog to our system is still underway for this huge stock that contains thousands of materials.

- *Basic catalog searches:* From the main screen you can conduct a basic catalog search without logging in to the system. You may choose from the following criteria; Author, Title, ISBN, Subject or Keyword. You can also limit your search results to a specific library branch. The words you enter in the second input box will be searched in the appropriate criteria filed you have chosen. You may select the Keyword option to look for the search words in all of the fields of the catalog information of the materials. You may use Turkish characters if needed. There is also a contextual help available. Please note that, you should enter at least 4 letters to conduct a search. There are also methods available for your search. The 'Match Exact' method will return the result that exactly matches the word(s) entered. The 'Match Any' will return entries that matches either one of the words you have given. The 'Match Phrase' method will return the entries that contain the phrase entered.

- *Advanced Search:* To the left of the screen you will see the advanced search link. This will lead you to a page where you can input more parameters to limit the results of your search. You can enter as many parameters as you wish. At the bottom of the parameter input boxes you may select a logical operator "AND" or "OR". This logical operator will be placed in between the parameters.

- *Search results:* Both the basic and the advanced search dialogs will lead you to the same search results interface. The search results page will display ten matching records per page. Below the records your will see the navigation bar. This navigation bar contains the "Previous" link, takes you to the previous ten records, "Next" link, takes you one ten records forward, and the page numbers where you can use to jump to specified page.

  The record rows give you brief information about the entry. You will notice that some fields lead to cross – searches in the catalog. For example clicking on the author field automatically displays other materials by that particular author. You may follow the "Details" link to see more details about the material and its stock information. If you are logged in you may directly issue holds to materials.

- *Details:* The details screen shows you detailed catalog information on the selected material. You will also notice the "See Related Materials" link. This will search the catalog sharing the similar subjects or classification with the selected material. At the bottom you will see the copies, volumes or supplements for this material that is available in the library stock. You will also see the corresponding branch and the stock item's current status. This status message tells you if the stock item is available for circulation or not. If the stock item is on circulation, Library ON-LINE will display the possible return date of the material. Please note that this return date is only an estimation based on the current loan rules.

- *Issuing holds on materials*: You should be logged in to Library ON-LINE to use this feature. If you click the "Issue Hold" link in the search results page or the material details page, a new pop-up window will appear giving you information about your request. The possible messages are:

  - This material is reserved for you until <date>: This shows a successful reservation. You will have three days to pick up your reserved book. If you fail to do that, your reservation will be

cancelled. You will see this information in the “Library ON-LINE Notices” box in “MY Library ON-LINE” page.

- This material is reserved for you, you are in slot <slot>: If other users issued a hold before you, you will be lined up in the reservation queue.
- You have already issued a reservation for this material: This means that prior to your action, you have already a hold issued for the material. You may see your reservations in the “MY Reservations” box in the “MY Library ON-LINE” page.

- *MY Library ON-LINE Page:* From this page you will be able to see all your information related to Library ON-LINE. This page contains several boxes:

    - Library ON-LINE Notices: You will see messages from the system in this box: The message types are; overdue material warning, hold warning and recall warning. Recall warning will appear if you have a material that has been recalled by the library management. In this case you should return the material at once.
    - MY Reservations: You will se your pending reservations from this box. Please note that, the reservation that is waiting our pick up will not be displayed here but in the Library ON-LINE notices box.
    - MY Loans: You will see your current loans in this box. Overdue materials will be displayed separately for your convenience.
    - MY Past Loans: You may view your previous loans from this box. Please note that this may not include all your loans as loan data is backed-up and garbage collected in a regularly.