2011

Metadata Management System for Healthcare Information Systems

Ketan Shripat Patil
Utah State University

Follow this and additional works at: http://digitalcommons.usu.edu/gradreports

Part of the Computer Sciences Commons

Recommended Citation
http://digitalcommons.usu.edu/gradreports/96

This Report is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Plan B and other Reports by an authorized administrator of DigitalCommons@USU. For more information, please contact dylan.burns@usu.edu.
Metadata Management System for Healthcare Information Systems

by

Ketan Patil

A report submitted in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE in
Computer Science

Approved:

_______________________
Dr. Stephen W. Clyde
Major Professor

_______________________
Dr. Curtis Dyreson
Committee Member

_______________________
Dr. Vladimir Kulyukin
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2011
ABSTRACT

Metadata Management System for Healthcare Information Systems

By
Ketan Patil, Master of Science
Utah State University, 2011

Major Professor: Dr. Stephen W. Clyde
Department: Computer Science

The Utah Department of Health (UDOH) uses multiple and diverse healthcare information systems for managing, maintaining, and sharing the health information. To keep track of the important details about these information systems such as the operational details, data semantics, data exchange standards, and personnel responsible for maintaining and managing it is a monumental task, with several limitations. This report describes the design and implementation of the Metadata Management System (MDM) that addresses the problem of documenting multiple information systems and exchange of data between them. The purpose of the MDM is to provide efficient and usable means of storing, accessing, and visualizing metadata of the information systems used by the UDOH in authenticated and authorized manner.
ACKNOWLEDGMENTS

I thank Dr. Stephen W. Clyde for his support and guidance throughout my course of study at USU and during the course of this project. I attribute my pursuing a Master’s degree to his encouragement and effort, and without him this project would not have been completed or written. Other than being a pool of knowledge, he has been a great source of inspiration to me.

I would like to thank Dr. Curtis Dyerson and Dr. Vladimir Kulyukin for their time and guidance for this project.

I also thank my family and friends whose moral support helped me in accomplishing this project.

Ketan Patil

(47 Pages)
# CONTENTS

ABSTRACT ........................................................................................................ iii
ACKNOWLEDGMENTS .................................................................................. iv
LIST OF FIGURES ......................................................................................... vii

ACRONYMS ...................................................................................................... viii

CHAPTERS

1. Introduction ...................................................................................................... 1

2. Analysis ........................................................................................................... 4
   2.1. Introduction .............................................................................................. 4
   2.2. Use-case Analysis ................................................................................... 4
   2.3. Reuse and Domain Analysis ................................................................... 7
   2.4. Functional Specification ......................................................................... 8
   2.5. Non-Functional Specification ................................................................ 9
       2.5.1 Usability ........................................................................................... 10
       2.5.2 Availability ....................................................................................... 11
       2.5.3 Quality of Service ........................................................................... 11
       2.5.4 Backup ............................................................................................. 11

3. ARCHITECTURE DESIGN ......................................................................... 12
   3.1. Introduction ............................................................................................. 12
   3.2. Architecture Design ............................................................................... 14
   3.3. Access-control Rules ............................................................................ 18
       3.3.1 Access Control Rules for Organization ........................................... 19
       3.3.2 Access Control Rules for Information System ................................. 19
       3.3.3 Access Control Rules for Message Structure .................................. 20
       3.3.4 Access Control rules for Person ....................................................... 20

4. USER INTERFACE ..................................................................................... 21

5. IMPLEMENTATION DETAILS ................................................................. 26
   5.1. Introduction ............................................................................................. 26
   5.2. Vitruvian DBObjects ............................................................................. 26
   5.3. Implementation Details and Challenges ............................................... 30
6. SOFTWARE TESTING..................................................................................................................32

6.1. Introduction................................................................................................................................32
6.2. Unit/Module Testing.......................................................................................................................32
  6.2.1 User Interface Module (MDM Website) Testing ................................................................. 33
  6.2.1 Metadata Information Web Services Module Testing ....................................................... 33
6.3. Validation Testing ..........................................................................................................................34
6.4. Stress Testing .............................................................................................................................. 35

7. CONCLUSION AND FUTURE WORK ....................................................................................... 36

REFERENCES ...................................................................................................................................... 38
LIST OF FIGURES

Figure 1. MDM user information retrieval ................................................................. 5
Figure 2. MDM user account management ............................................................... 6
Figure 3. MDM admin user management ................................................................. 7
Figure 4. MDM package diagram ............................................................................. 13
Figure 5. Class diagram of message structure ......................................................... 15
Figure 6. Design of object structure definition ....................................................... 16
Figure 7. Two-panel selector .................................................................................... 22
Figure 8. Card stack ................................................................................................. 22
Figure 9. Combination of three-panel selector and tree table .................................. 24
Figure 10. Relationships among DB, data entity, and UI .......................................... 29
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>ERD</td>
<td>Entity-Relationship Diagram</td>
</tr>
<tr>
<td>ERM</td>
<td>Entity-Relationship Model</td>
</tr>
<tr>
<td>HL7</td>
<td>Health level Seven</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>MDM</td>
<td>Metadata Management System</td>
</tr>
<tr>
<td>MVCC</td>
<td>Multiversion concurrency control</td>
</tr>
<tr>
<td>ORM</td>
<td>Object-Relation Mapping</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
A large organization often operates multiple information systems that support the various functions of that organization. For examples, a car manufacturing company will have an inventory system, assembly-line management system, order processing systems, and billing system. A government agency, such as the Utah Department of Health (UDOH), operates nearly 200 information systems that track everything from birth certifications, hearing screenings, to contagious diseases, cancer cases, to death records. Keeping track of the important details about the information system is a daunting task. These details include database scheme, data semantics, operational details, technical and business contracts, and much more.

In addition, information systems often need to exchange data with each other to optimize their business process, reducing errors in operational activities or for knowledge sharing. For example, an assembly-line management system uses information from an inventory system to optimize production rates. Child-Health Advanced Record Management System (CHARM) of the UDOH uses information from the HiTrack\textsuperscript{1}, USIIS\textsuperscript{2}, VS\textsuperscript{3}, BTOTS\textsuperscript{4}, NLIMS\textsuperscript{5}, and others, to provide the health care knowledge about the children with health, developmental and genetic conditions.

\textsuperscript{1} HiTrack is an information system for tracking hearing screening and their results.
\textsuperscript{2} USIIS – Utah State Immunization Information System – is a system for tracking vaccinations.
\textsuperscript{3} VS – Vital Statistics – is a system for tracking birth and death certificates.
\textsuperscript{4} BTOTS is an information system for planning and tracking early-intervention services.
\textsuperscript{5} LIMS is an information system for managing lab results, including newborn metabolic screenings.
The data exchange between the information systems is done by messages or file transfers, which follow various formats and standards. For example, an HL7\(^6\) version 2.5 messages uses a human-readable (ASCII), non-XML encoding syntax based on segments (lines) and one-character delimiters, whereas HL7 version 3.0 messages uses XML encoding. Furthermore, message segments between these standards have different meanings and semantics. Keeping track of which systems are exchanging data, what that data is, and how the exchanges occur is even more of a daunting task than just tracking the individual systems.

This report describes a system, called *Metadata Management System* (MDM) that addresses the problem of documenting multiple information systems and the exchange of information with each other. The MDM includes two interfaces: a web-based user interface and a web-services program interface. The user interface allows system and network administrators to access and update the metadata stored by the MDM. The program information allows information systems to update their own metadata directly, or query the meta-data about other systems or information exchanges. MDM restricts the capabilities of both end users and programs by the roles they play and their application domains.

As a proof of concept, the MDM is tested as a centralized repository for the metadata of the UDOH information systems and exchanges. In this environment, it allows the UDOH IT staff store, maintain, and visualize metadata in an information-rich website; and external information systems to retrieve and update metadata through its Web service interface. This metadata includes, technical attributes of the information systems, types of messages used between them

\(^{6}\) Health Level Seven (HL7) is an all-volunteer, non-profit organization involved in development of international healthcare informatics interoperability standards. It provides a framework (and related standards) for the exchange, integration, sharing, and retrieval of electronic health information.
for the exchange of data, the formats, the standards, and the semantics of the messages, and statistical information about them.

MDM is designed using modular software architecture. This architecture allows any of its modules to be upgraded or replaced independently as requirement and/or technology changes. It also incorporates Service Oriented Architecture (SOA) in its design. SOA allows software reusability, provides better abstraction and quicker development of the system [1]. See Chapter 3 for more details about the design of the MDM.

The metadata of healthcare information systems that are used by the UDOH is highly sensitive and only authenticated and authorized users should have access to it. The MDM uses industry compliant user authentication methods and organization-centric access control rules for the user data authorization. Managing the users and their data authorization is done using the MDM website. See Section 3.3 for more on access control rules.

To make the MDM user interface simpler, easy-to-use, and intuitive, its design take advantage of graphical user interface (GUI) design patterns to make it easier to navigation, to find and access information, update data, and to learn and remember [2]. See Chapter 4 for more details on MDM’s user interface design.

MDM’s implementation followed the design. See Chapter 5. However, to ensure that it satisfied to the functional and non-functional requirements, it underwent thorough unit testing, stress testing, and validation testing. Chapter 6 talks more on testing of the MDM.

Although MDM is specific to the healthcare information systems used by the UDOH, the design used in its development enables it to store and manages metadata of similar healthcare information systems outside UDOH. Chapter 7 contains some ideas for future work.
CHAPTER 2
ANALYSIS

2.1. Introduction

Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain [3]. The artifacts generated in this phase include Unified Modeling Language\(^7\) (UML) use-case diagrams, UML class diagrams, and functional and non-functional requirements document. The use-case diagrams help in better understanding the needs and expectations of the end users. The end user can be a real person that requires the system to achieve specific goals or an external system, interfacing for the data retrieval and manipulation. See Section 2.2 for more details on use-cases. Section 2.3 describes reuse and domain analysis carried out during analysis of the MDM. Reuse and domain analysis enables software reuse by systematic discovery and exploitation of commonality across related software system [4]. Functional requirements listed in Section 2.3 expand goals described by user-case diagrams with more detail statements and constraints about the systems features and the behavior.

2.2. Use-case Analysis

As requirements are gathered, the software engineer (analyst) creates sets of scenarios that identify a thread of usage for the system to be constructed. These scenarios, often called *Use

---

\(^7\) Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. It includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.
Cases [5], provide descriptions depicting goals of end users. To create a use case, the analyst must first identify different types of users (or external systems) who use the system.

These users (or external system) actually play the role of actors. Defined more formally, an actor is an entity external to the system that communicates with it, having a desired goal. Keeping this in mind, the actors and scenarios captured during the use-case analysis of MDM are described below.

Figure 1 depicts a use case, wherein, an MDM user retrieves and updates the information through the user interface. This user must be authorized in order to view and update the information. The information viewed or updated is related to the information systems, message structures, object structures, and organizations.

Figure 1. MDM user information retrieval

Figure 2 describes a scenario related to management and retrieval of user’s password. Users are allowed retrieve their forgotten password for a particular username. The password will
be send to their registered email account, only after they are able to correctly answer the security question. Users are also allowed to change their password or security question, if and only if, they are able to successfully log into the system.

![Diagram of MDM User Account Management]

**Figure 2. MDM user account management**

Figure 3 describes a use-case scenario for an administrator. The MDM provides functionalities to the administrators to manage user accounts and their authorization. Using the MDM user interface, the administrator creates new user accounts or deletes existing ones. An administrator is allowed to successfully grant and revoke data-access rights using MDM.
2.3. Reuse and Domain Analysis

Systematic software reuse has led to the increased productivity, software quality, and economic benefits. To explore potential software reuse in the development of new software, reuse and domain analysis proves extremely beneficial. In reuse and domain analysis, a system is analyzed according to its individual business requirements and functionalities. This enables to identify modules and libraries that are reused latter for the development of the new software [6]. This analysis helps in reducing the software development time and also ensures there are fewer delivery defects, as these libraries are already tried and tested. Reuse and domain analysis was carried out in the initial phase of system analysis of the MDM, to ensure systematic software reuse. Using this analysis, we were able to discover certain classes that can be reused latter.

By using the reuse and domain analysis, the object structure definition was identified that can be reused latter. An object structure definition in context of the MDM stores the message structure information in hierarchical form. In object structure definition, the parent object is
composed of one or many child objects. Each object in this hierarchy represents a piece of information. For example, when object structure definition is storing information related to the HL7 version 2.5 messages, the parent object stores the information of the entire message, which in turn is composed of the objects storing information of the data segment of that message. Again, this data segment is composed of data fields. This object structure definition class can be reused to store the information about data schemes that follow the hierarchical structure. For example, it can be used to store the database schema, wherein, the parent object will store the table information and which in turn is composed of the child object that stores information regarding the table columns.

2.4. Functional Specification

In software engineering, a functional specification is a document that clearly and accurately describes the essential functional requirements of a software system. This document in turn is use to determine whether the software requirements have been met [7]. A functional specification does not define the inner workings of the proposed system; it does not include the specification of how the system will be implemented. Instead, it focuses on what various outside agents (people using the program, computer peripherals, or other computers, for example) might “observe” while interacting with the system. Following are the functional specifications for the MDM.

• Information Access – End Users
  
  o It should provide a graphical user interface to the end users through which they can access, analyze, and manipulate the information stored in the MDM.
o Only an authenticated and authorized person should be allowed to access information stored in the system.

o It should provide a feature for system administrators through which they can create new users and delete or deactivate existing users.

o It should provide functionality for administrators through which they can manage proper data authorization.

o The information exchange process should be done through a secured channel.

- Information Access – External System

  o It should provide an application programming interface through which external systems can request and update information stored in the MDM.

  o Only an authenticated and authorized client system should be allowed access to the information stored in the system.

  o The application programming interface exposing data should allow retrieval of information by external systems, regardless of their language of implementation or platform.

2.5. Non-Functional Specification

In software engineering, a non-functional specification is an artifact that specifies criteria that is used to judge the operation of a system, rather than specific behavior [8]. In general, functional requirements define what a system is supposed to do whereas non-functional requirements define how a system is supposed to be. The non-functional requirements include the constraints and qualities. Qualities are properties or characteristics of the system that its stakeholders care about and that hence will affect their degree of satisfaction with the system.
Constraints are not subject to negotiation and, unlike qualities, are (theoretically at any rate) off-limits during design trade-offs. Following represents the list of non-functional requirements for the MDM.

### 2.5.1. Usability

Usability is a qualitative attribute defined as to what extent a software is used by its users to achieve specific goals with effectiveness, efficiency, and satisfaction. Following are the usability criteria which the MDM should meet.

- **Learnability:** The MDM website should be easy to learn for new users, such that they can effortlessly accomplish basic tasks in their very first use.
- **Efficiency:** After subsequent use and familiarity with the website, MDM users should easily perform a given task with less expenditure of time and efforts.
- **Memorability:** MDM users should be able to establish proficiency with the system, even after not using it for a long duration of time.
- **Errors:** MDM users should be able to perform a specific task with no or minimal number of errors. Furthermore, if an error has been made by the user, it should facilitate its restoration to a prior stable state.
- **Satisfaction:** MDM users should find the interface pleasant and intuitive.
2.5.2. **Availability**

Availability is defined as the degree to which a system, subsystem, or piece of equipment is in a specified operable state for the proposed users. In high availability applications, a metric known as nines, corresponding to the number of nines following the decimal point, is used (for example .99999 means the system should be available 99.999% of the time). The MDM should be available to its users .9999, i.e. 99.99%. When there is a need for maintenance or an upgrade of the MDM, all the services should run on alternative servers.

2.5.3. **Quality of Service**

Quality-of-service requirements in software engineering define the performance of the system in terms of response throughput, response time, transit delay, latency, etc. The MDM should be able to perform with optimized quality of service.

2.5.4. **Backup**

In information technology, backing up refers to making copies of data, so that these additional copies are used for data restoration in event of data loss. The MDM must have a backup database server that periodically backs up all the data from the production database. Thus, if the production database server crashes, backup data is used to restore it.
CHAPTER 3
ARCHITECTURE DESIGN

3.1. Introduction

The architecture of a program or computing system is the structure of the system, comprising software components, the externally visible properties of those components, and the relationships among them [9]. The term also refers to documentation of a system’s “software architecture.” Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design, and allows reuse of design components and patterns between projects.

The MDM design uses three-tier architecture and has the following modules: user interface module, Web service module, security module, and database access module. Figure 4 depicts the package diagram of the MDM.

The user interface package contains all the GUI classes and has forms for the information retrieval and manipulation. It also consist forms for the access rights management of the data and for the user management.

The web-service module contains classes that provide programmable application logic accessible via standard web protocols. This module describes an interface in a machine-processable format (specifically, Web Services Description Language, known by the acronym WSDL). This module enables different applications from different sources to communicate with the MDM for data exchange.
The security module contains classes that are responsible for user authentication and authorization. It implements access control rules used in the MDM for data authorization. More on access control rules is described in section 3.3.

The database access module contains the DBObjects and DBLists corresponding to the tables in the database. It is a mapping of a relational database to the object model and is generated by Vitruvian DBObjects. This module provides simplified access to data stored in an entity-relational database, without exposing underlying database detail. Refer section 5.2 for more information on Vitruvian DBObjects.
3.2. **Architecture Design**

The architecture of a software system is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution [10]. Software architecture is commonly organized in views, which are analogous to the different types of blueprints made in civil engineering. It provides a representation of a set of system components and relationships among them. This view or representation in context of object-oriented methodologies is called “object-oriented design.”

Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis. It maps the model produced in the analysis phase onto implementation classes and interfaces. The result is a model of the solution domain that gives a detailed description of how the system is to be built [11].

The MDM provides statistical as well as structural information about messages used between the Healthcare Information Systems (HIS) for exchange of the data. Figure 5 shows a hierarchical class diagram of the message structure. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system’s classes, their attributes, operations or methods, and the relationships between the classes [12].
A message is an instrument that is used by the information systems to communicate the data. The format of messages varies from simple delimiter-separated text file to a complex XML message.

![Message Structure Diagram]

**Figure 5. Class diagram of message structure**

Each message has an exchange reason that specifies the need for this communication. To store metadata information of messages, multiple attributes of them to capture of them. These message attributes includes its type, version, size, average count, average size, frequency, frequency units, and description.

This metadata information will be used by system analysts, data administrators, and investigators for various purposes. One of the important applications of metadata information stored in the MDM is detecting data loss or corruption. For example, consider an HIS receives 100–120 messages per day with an average message size of 20 MB. But on one particular day, the message count drastically drops to 30 messages per day or its average message size increases
to 35 MB. In this situation, the metadata information of the messages will be used to detect discrepancies between expected communication pattern and the actual.

In the component design of the MDM, well-known design patterns were identified and implemented. A design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. In the context of the MDM design, the message structure is stored as an object structure definition. Object structure definition stores the hierarchical structure and related information of the message. Figure 6 shows class diagram of object structure definition utilizing composite design pattern.

**Figure 6. Design of object structure definition**

A composite design pattern composes objects into tree structures to represent part–whole hierarchies [13]. It lets clients treat individual objects and compositions of objects uniformly. As
data transmitted and received by HIS can be a simple flat file or complex XML, it is highly desirable to design an object structure that accommodates most kinds of message-structure information. For example, metadata of a message that uses an XML document as a way to transmit data can be easily stored in the MDM in a hierarchical format using object structure definition. Here, the head node of an XML document is stored as a top-level object, which in turn is composed with the subsequent node. This design is also efficient to store schema of various versions of HL7 messages. HL7 provides a framework (and related standards) for the exchange, integration, sharing, and retrieval of electronic health information.

Design of the MDM incorporates service-oriented architecture. A service-oriented architecture (SOA) is a set of principles and methodologies for designing and developing software in the form of interoperable services [1]. These services are essential to the MDM as a means of providing data interfaces to external systems for exchange of data.

Web services are defined as a standalone unit of functionality available only via a formally defined interface. These interfaces are published using universal description discovery and integration (UDDI) specification, which defines a way to publish and discover information about Web services. SOA allows development of simple but effective systems, lowers the cost of development, increases system integration, and encourages software reusability.

To design the MDM system, modular three-tier architecture is employed. In software engineering, three-tier architecture is a client–server architecture in which the presentation, the application processing, and the data management are logically separate processes. Tree-tier architecture allows the MDM to be database and platform independent. The benefit of utilizing
three-tier architecture is that it allows any of its tiers (also called layers) to be upgraded or replaced independently as requirements or technology change.

3.3. Access-control Rules

Access-control rules used in the data access of the MDM revolve completely around organization. Users have access to the data of an organization if they are authorized by the data administrator. These access-control rules can be explained using Datalog, which is a query-and-rule language for deductive databases that is based on logic programming [14]. Given below is the terminology used in defining access-control rules.

Abbreviations

- Organization is denoted as O.
- Information system is denoted as I.
- Message structure is denoted as M.
- Person is denoted as P.
- Access right is denoted as R.
- Any anonymous object is denoted as X.

Object Determinations

- Org(X) means object X is an instance of organization.
- Inf(X) means object X is an instance of information system.
- Mess(X) means object X is an instance of message structure.
- Per(X) means object X is an instance of person.

Predicates

- OrgOwns(O, X) is a predicate that evaluates to be true if there is a simple or transitive relationship between organization object O and an object X.
• HasAccess(P, O, R) is a predicate that evaluates to be true when P has access to O with access right R.

• HasRelation(X) is a predicate that evaluates to be true if there is any relational link between X and some other object.

• CanRead(P, X) is a predicate that evaluates to be true if the right hand side of the expression is true, which basically determines whether P can read object X.

• CanUpdate(P, X) is a predicate that evaluates to be true if the right hand side of the expression is true, which basically determines whether P can update object X.

• CanDelete(P, X) is a predicate that evaluates to be true if the right hand side of the expression is true, which basically determines whether P can delete object X.

Following are the data-access rules used in the MDM, which are explained using Datalog logic.

3.3.1. Access Control Rules for Organization

These rules specify access control for data related to organizations. Based upon the evaluation of the predicate, a user may or may not have access to the organization’s information.

CanRead(P, X):- Org(X) ^ HasAccess(P, X, ‘Read’)

CanUpdate(P, X):- Org(X) ^ HasAccess(P, X, ‘Update’)

CanDelete(P, X):- Org(X) ^ HasAccess(P, X, ‘Delete’) ^ ~ HasRelation(X)

3.3.2. Access Control Rules for Information System

These rules specify access control for data related to information systems. A user has access to data only if the information systems are related to an organization and, in turn, the user has access to that organization.
CanRead(P, X) :- Inf(X) ^ OrgOwns(O, X) ^ HasAccess(P, O, ‘Read’)
CanUpdate(P, X) :- Inf(X) ^ OrgOwns(O, X) ^ HasAccess(P, O, ‘Update’)
CanDelete(P, X) :- Inf(X) ^ OrgOwns(O, X) ^ HasAccess(P, O, ‘Delete’) ^ ~ HasRelation(X)

3.3.3. Access Control Rules for Message Structure

These rules specify access control for data related to message structures. A user has access to data only if the message structures are related to an organization and, in turn, the user has access to that organization.

CanRead(P, X) :- Mess(X) ^ OrgOwns(O, X) ^ HasAccess(P, O, ‘Read’)
CanUpdate(P, X) :- Mess(X) ^ OrgOwns(O, X) ^ HasAccess(P, O, ‘Update’)
CanDelete(P, X) :- Mess(X) ^ OrgOwns(O, X) ^ HasAccess(P, O, ‘Delete’) ^ ~ HasRelation(X)

3.3.4. Access Control rules for Person

These rules specify access control for data related to personal information. A user has access to data only if the person is related to an organization and, in turn, the user has access to that organization.

CanRead(P, X) :- Per(X) ^ BelongsTo(X, O) ^ HasAccess(P, O, ‘Read’)
CanUpdate(P, X) :- Per(X) ^ OrgOwns(O, X) ^ HasAccess(P, O, ‘Update’)
CanDelete(P, X) :- Per(X) ^ OrgOwns(O, X) ^ HasAccess(P, O, ‘Delete’) ^ ~ HasRelation(X)
CHAPTER 4
USER INTERFACE

Designing the visual composition and temporal behavior of the graphical user interface (GUI) is an important part of software application programming, particularly in the area of human–computer interaction. Its goal is to enhance the efficiency for the underlying logical design of a stored program. Various key elements should be taken into consideration while designing a GUI, such as organizing the content, showing navigations, displaying page layout, and maintaining consistency. The MDM website is designed keeping these considerations in the mind.

The MDM users largely deal with lists of objects, such as information systems, messages, and organizations. To efficiently deal with the list objects, two-panel selector is utilized, where the panels are placed side by side on the interface. The left panel shows a set of items that the users can select, and the right panel shows the information of the selected item. This pattern is used in design for following reasons. First, it reduces physical effort, ensuring the user’s eyes do not have to travel a long distance; second, it reduces the visual cognitive load and memory burden. Figure 7 shows implementation of two-panel selector.

It is highly desirable for a user interface to present only few entry points that are task oriented and descriptive. Further, every page should have a small section showing a consistent set of links or buttons that takes the user to key sections of the website. The card stack design pattern ensures global navigation and provides clear points of entry. This pattern is implemented using tab control. Tab control is a UI element in which sections of content are put onto separate
panels and are stacked up such that only one is visible at a time. This element helps to organize information into easily digestible chunks. These chunks provide instant gratification, easy exploration, and quick movement through the website. Figure 8 shows implementation of card stack pattern.

Figure 7. Two-panel selector

Figure 8. Card stack
The primary purposes of the MDM website are information visualization and manipulation. It is highly desirable from the user point of view to group the UI controls that provide similar functionality. Group buttons, which are small cluster of buttons with related actions, were used. They help make the UI self-describing and instantly communicate availability of those actions.

A combination of different design patterns is required to provide specific functionality. Figure 9 depicts a combination of a three-panel selector and a tree table pattern, used in the design of the MDM interface, which displays the object structure information. The top left panel shows hierarchical data using the tree view; the top right shows children, if any, of the node selected; and the bottom panel shows information of the element selected in the tree view. The tree view is designed using customized image button controls and a panel. It provides efficient and intuitive traversal through various nodes of itself.

The MDM website provides consistent layout and style between pages. Consistency throughout the MDM website makes it easier for the MDM users to navigate the site. Additionally, MDM users do not get a feeling that they have left the website and landed on another website, when the main elements remain the same. Various color schemes are use in user interface design of MDM website. It uses light and dark contrast for body of the pages. Essentially, the whole body of the web page has a lighter contrast as compare to the parts of the web page that displays information. The light and dark contrast scheme helps to effectively show the work areas of the website.
The MDM website provides easy and global navigation between pages. It has a consistent tab menu, through that the user can navigate to the key sections of the website. It allows the MDM users to easily determine in which section they are. The MDM website uses error summary if user is adding inconsistent or invalid data. For example, if in the field of email the user enters an invalid email format or in the phone number field enters an invalid phone number format, then MDM website will give a popup with the list of errors, the description and the field names that error belongs to. Error summary allows user to quickly understand the causes of the errors.
Information retrieval and manipulation is performed in the MDM website by fewer clicks. This helps the MDM user to easily retrieve and manipulate information. The placement of the various web controls for showing and updating information follows a standard accepted pattern. Such as a list of items are always displayed on the left side of the page and their related information on right.
CHAPTER 5
IMPLEMENTATION DETAILS

5.1. Introduction

MDM is implemented using C#.Net 2005 and Postgre SQL 8.3. The database layer is modeled using object-relational mapping (ORM) with the help of Vitruvian DBObjects (for more information on Vitruvian DBObjects see Section 5.2). The MDM website is built using ASP.NET, which is a Web application framework. To make MDM website interactive and consistent, CSS and JavaScript are used. The MDM web service module is built using ASP.NET XML Web services.

5.2. Vitruvian DBObjects

Vitruvian DBObjects is a part of the Vitruvian Distribution Framework [15]. DBObjects are similar to object-relational mapping (ORM). ORM is a programming technique for converting data from a relational model to object model. This creates, in effect, a “virtual object database” that is used from within the programming language [16].

The relational model is represented and maintained in an object model via DBObjects and is embedded into attributes of the object model. A table maps to the class and a column maps to the properties in the class corresponding to the table. Following shows the mapping of a CommunicationPoint database table shown in Table 1 to a CommunicationPoint class by using Vitruvian DBOject Wizard.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Serial, primary key</td>
</tr>
<tr>
<td>name</td>
<td>Character varying (64)</td>
</tr>
<tr>
<td>Type</td>
<td>Character varying (64)</td>
</tr>
<tr>
<td>description</td>
<td>Character varying (4096)</td>
</tr>
<tr>
<td>Lastchanged</td>
<td>timestamp</td>
</tr>
</tbody>
</table>

Table 1 CommunicationPoint Table
`CommunicationPoint` class generated by Vitruvian DBOBJECT Wizard.

```csharp
public partial class CommunicationPoint : DBOBJECT
{
    [Column("id")]
    private Key id = new Vitruvian.Data.DBObjects.SequenceKey("communicationpoint_id_seq");
    [Column("name")]
    private object name;
    [Column("type")]
    private object type;
    [Column("description")]
    private object description;
    [Column("lastchanged")]
    private object lastChanged;

    /// <summary>
    /// Column = id
    /// This column is a primary key.
    /// </summary>
    [Column("id")]
    [PrimaryKey]
    [Browsable(false)]
    public Int32? Id
    {
        get
        {
            if (this.id.Value == null)
                return null;
            else
                return (Int32)this.id.Value;
        }
        set
        {
            if (!IsEqual(this.id.Value, value))
            {
                this.id.Value = value;
                SetModified(typeof(CommunicationPoint));
            }
        }
    }

    /// <summary>
    /// Column = name
    /// </summary>
    [Column("name")]
    public String Name
    {
        get { return this.name as String; }
        set
        {
            if (!IsEqual(this.name, value))
            {
                this.name = value;
                SetModified(typeof(CommunicationPoint));
            }
        }
    }

    /// <summary>
```
Data transfers from the database to a data entity (see Figure 10) in software, and then is possibly displayed in a user interface. The end user may need to edit the data, which is then transferred to the data entity and eventually stored in the DB. Important features of DBOObjects include the following:
1. It creates objects, allows navigation between related objects, and easily makes objects persistent.

2. It gives opportunities for automatic object loading as needed (lazy loading) rather than explicit loading.

3. It eliminates the need for writing direct SQL for basic CRUD and query operations.

![Figure 10. Relationships among DB, data entity, and UI](image)

DBObjects maintain the relationships between tables. DBObjects provide the following methods:

1. **Load():** Load the data into the DBObject or DBList. Data is filtered before loading.

2. **Reload():** Load the new set of data from the database.

3. **Save():** Save the DBObject to the database.

4. **Delete():** Delete the DBObject from the database.

5. **ResetValues():** Reset the values (i.e., all properties) of a DBObject.

6. **RelationalSave():** Save the DBObject and the children tables of the current DBObjects.

7. **RelationalDelete():** Delete the DBObject and the children tables of the current DBObjects.

A DBObject also keeps track of its current state, which is one of the following:
1. **New**: DBObject is created but not saved into the database.

2. **Synced**: DBObject is in synchronization with the database.

3. **Modified**: DBObject is modified and is not the same as in the database, i.e., one of the properties has been modified and not saved in the database.

4. **Deleted**: DBObject is deleted.

5. **Detached**: DBObject is marked for deletion but still exists in the database.

DBObjects have a data wizard that automatically generates the entire table into classes, and all columns into the properties of a given class. It also generates relationships between the tables. It is intelligent enough to distinguish between one-to-one and one-to-many relationships.

While generating DBObjects, the users are allowed to customize class names, their properties, and relationships. The users are also allowed to choose the strategy to generate the key for IDs.

### 5.3. Implementation Details and Challenges

This section describes the implementation details of the MDM, challenges faced, and solution to those problems.

The implementation of object structure definition was a challenge. To implement it, Customized three-panel selector was needed to be designed and coded. The key issue was how to display the hierarchical data in an interactive tree. This tree needed to be easy to traverse and also provide the functionality to delete or add a node in it. TreeView control of ASP.NET provides the functionality of displaying the hierarchical data, but it is not intuitive. So, new TreeView is implemented using a combination of the ASP.NET image controls, panels, JavaScript, and CSS, which is easy to traverse and manipulate.
The implementation of the Web services that would provide consistent data access to any system independent of programming language or platform was a key goal. Again, it should also allow transformation of the data into the objects and vice-versa. This problem was solved using Vitruvian Serialization Library.

Designing a custom two-panel selector that will allow the users to view and update information without excessive navigation was a goal. To achieve this goal, combination of list view, to show a list of items, as well as a details view to show its related information. Here, the DetailsView control of the ASP.NET is customized to contain a DropDownList control. The key advantage of this customized control is that it allows viewing and updating information in the same page with fewer clicks.
CHAPTER 6
SOFTWARE TESTING

6.1. Introduction

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation. Software testing also provides an objective, independent view of the software to allow the system users to appreciate and understand the risks of software implementation [17]. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs. Software testing is also stated as the process of validating and verifying that a software program/application/product:

1. Meets the business and technical requirements that guided its design and development
2. Works as expected

To test the MDM’s performance and proper functioning, unit testing, stress testing, and validation testing were performed. The next section discusses these three individual testing methods employed in software testing of MDM in detail.

6.2. Unit/Module Testing

Unit testing focuses a verification effort on the smallest unit of software the software component or module [7]. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module.

MDM has two important modules:

1. User interface module (MDM Website)
2. Metadata information Web services module

Both of these modules are tested individually and details of their testing are described below.

6.2.1. **User Interface Module (MDM Website) Testing**

The first stage of testing a website is to ensure valid HTML (or XHTML) has been used. The developer’s own browser may ignore certain errors, but there is a significant risk that markup errors will result in display problems in some browser or other. Thus, the MDM website is tested to confirm that it has valid HTML on every page. Searching the MDM website for broken links is another important step in testing. Manual testing is performed to check all the submit buttons and URL redirection for valid links. For example, on the information system page every link was clicked, and a check was performed to ascertain that it redirects to a valid page. Furthermore, every page was tested to ensure each displays valid information. This testing was done manually by loading each and every page on the website then cross referencing the information on the webpage to the data in the database. Since MDM users require a valid pair of username and password to get access to the website, testing user authentication was an important step. Various invalid combinations of username and password were entered during testing to make sure only valid users could get access to the website.

6.2.2. **Metadata Information Web Services Module Testing**

The Web services module is an essential part of the MDM that implements services for sharing data. For testing this web services module, a separate piece of software called the MDM Web Service Tester was developed. This software consumes the web services hosted on the server and then communicates with it to gain access to the data by providing valid username and
password. Each and every web services were tested through this software to make sure valid and expected data was consumed.

Furthermore, testing was perform to make sure return type data, which the web service sends back, was readable and understood by the external systems. Various combinations of parameters were used to invoke every single web service, and then the return data was tested for its validity. Again, the user access and authorization was tested using the same testing approach.

MDM web services were also tested manually through web browser, thus checking that it creates a valid XML representation of the data. This is a very essential step; as if it is unable to create valid XML then cross platform communication will be not achieved.

6.3. Validation Testing

Validation testing ensures that software functions in a manner that is reasonably expected by the customer [7]. Reasonable expectations are defined in the Software Requirements Specification, which is a document that describes all user-visible attributes of the software. In a nutshell, validation testing ensures that the developer constructs the right product. Software validation is achieved through a series of black-box tests that demonstrate conformity with requirements. Here, each and every page of the MDM website as well as every web services was tested for its functionality. For example, a web page that presents the object structure definition of the message was analyzed through a functional point of view. Such analysis verifies that the web page displays the right message structure and child-to-parent relations while a user browses through the information displayed on the page. Does the web service ensure it only provides data to an authenticated and authorized user of the data? Thus, we made sure that all functionalities spelled out in Software Requirements Specification document were implemented and the system is processing in the expected manner.
6.4. Stress Testing

Stress tests are designed to confront programs with abnormal situations. Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume [7]. For example,

1. Special tests are designed that generate ten interrupts per second, when one or two is the average rate.

2. Input data rates are increased by an order of magnitude to determine how input functions will respond.

3. Test cases that require maximum memory or other resources are executed.

Stress testing was extensively carried out during testing of the MDM. Incremental stress testing was followed by increasing the volume and complexity of the data to test the system. Three classes of data were used to test the system, which was Basic Test data, Extensive Test Data and Bad Test Data. The system was executed while fetching this three classes of data and then tested to make sure the system does not crashes or operates in an unexpected manner. Thus using Stress Testing technique, it was ensure that the MDM is stable and processing properly.
Presently the MDM is efficiently able to perform the data manipulation task through its website and also data dissemination through its Web services. Using both of these modules, an MDM user has access to the metadata stored in MDM. External software systems are enable to access the MDM by using its Web services, thus giving a great degree of interoperability to the system.

Various testing methodology such as unit testing, validation testing, and stress testing were adopted to make sure that the MDM is meeting all functional and non-functional requirements, is stable, and operating in expected manner.

While working on this project, I was responsible for the analysis, design, development, and testing of the MDM. The resulting product is software designed to be flexible to change and easy to maintain. Further, the product fuses streamlined and efficient database design with object-oriented programming. The lines of code needed to implement various modules of the MDM are as follows.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Lines of Code (Thousand)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDM website</td>
<td>15 - 16</td>
</tr>
<tr>
<td>MDM web services</td>
<td>2-3</td>
</tr>
<tr>
<td>Access control</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2 Lines of Code Metric

As the database access layer, which unifies the communication between the MDM and database, was implemented using Vitruvian DBOjects, it reduced the developmental effort while building the MDM.

To make the MDM more effective, there are various functionalities that may be added to it, with no or minimal effect on the design of the system. Presently, all the data entry is done
using the MDM website, but it could be extended to build automatic parsers to extract meaningful data about a message’s structure. For example, an SQL parser could be added to the system, through which an SQL script depicting the structure of the message could be parsed to retrieve meaningful information.

Currently, the MDM does not support any analytical functionality. One could build an analytics system that would monitor the communication between two or more information systems against the statistical metadata stored in the MDM. So, whenever there were abnormalities in this communication (for example, the weekly frequency count of a message drops suddenly), this analytics system will notify the system administrator of this abnormality and provide related information ratifying the abnormality.

We have developed a system that will help system UDOH IT staff to efficiently manage and maintain metadata of the UDOH healthcare information systems. It will enable them to better understand data communications in healthcare domain. Further, it will be highly beneficial in detecting data loss or abnormalities in medical data exchange.
REFERENCES


