12-2012

A Flexible Consent Management System for Master Person Indices

Aditya Pakalapati
Utah State University

A Flexible Consent Management System for Master Person Indices

by

Aditya Pakalapati

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

_____	_____
Stephen W. Clyde Major Professor	Curtis Dyreson Committee Member

_____
Vladimir Kulyukin
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2012

ABSTRACT

A Flexible Consent Management System for Master Person Indices

by

Aditya Pakalapati

Utah State University, 2012

Major Professor: Dr. Stephen Clyde
Department: Computer Science

In healthcare, a *Master Person Index* (MPI) is a system that integrates information of individual from multiple data sources. To ensure confidentiality, such systems, particularly in healthcare, need to respect individual and organizational constraints on the sharing of data. This report describes a reusable consent management system that enforces such constraints and how it has been tested in the context of the Utah Department of Health (UDOH) MPI for public health.

(65 pages)

ACKNOWLEDGMENTS

I am grateful to my major professor, Dr. Stephen Clyde, for his time and encouragement needed to complete this project. His recommendations and suggestions have been invaluable for both me and this project.

I would like to thank my committee members, Dr. Curtis Dyreson, Dr. Vladimir Kulyukin for their support and valuable suggestions. I would also like to thank Brian Smith and the MPI team. Their suggestions, support, and hard work have been invaluable.

Finally, I appreciate my family and friends, who made all things possible.

Aditya Pakalapati

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF XML

LIST OF XSD

CHAPTER 1

INTRODUCTION

A *Master Person Index* (MPI) is a software system that integrates information about individuals from a diverse set of data sources. Some MPI's also allow end users or other electronic systems, i.e., *data consumers*, to retrieve correlated data from one or more data sources for any person known to the MPI. An MPI in the healthcare domain called *Master Patient Index* and abbreviated also as MPI, deals specifically with patient information, including their demographics, health history, and clinical data.

The Utah Department of Health recently contracted with Utah State University to build an MPI for public health that integrates four initial data sources: Child Health Advanced Record Management (CHARM), birth certificates (VS-Birth), death certificates (VS-Death), and the statewide immunization registry (USIIS). This MPI, referred to as Utah's *Public-health Master Patient Index* or *phMPI*, will eventually integrate dozens of data sources.

A key requirement for the phMPI is to guarantee privacy, confidentiality, and security. Privacy is the patients' right to keep information about their secret [19]. In others words, an information system that ensures privacy is one that allows the patients to have a say in which data consumers can retrieve their data and what kinds of data those consumer can see. Confidentiality is the assurance that an information system will only disclose a patient's data to consumers according to that patient's consent [19]. If a patient does not explicitly grant or deny access to a certain kind of data, then the system must to follow organization, state, and federal policies relative to the sharing of that data.

Finally, security is the assurance that an information system will prevent unauthorized users to access data in the system [19].

To guarantee privacy, confidentiality, and security, an MPI must support strong user authentication and manage individual and policy-based consent rules. This report focuses on the later. Specifically, it describes the development of a general *Consent Management Facility* (CMF) and how it was tested in context of Utah's phMPI. Chapter 2 provides the necessary background for understand general consent-management issues and the phMPI.

The software development process for the CMF followed a modified Spiral Model [20], consisting of analysis, design, implementation, and testing activities. These activities were performed in successive cycles, with the early cycles focusing on analysis and design and with the later cycles giving more emphasis to implementation and testing. Chapters 3-6 present the end results of these activities, independent of the software development cycles. In other words, this report focuses on the final analysis, design, implementation, and testing artifacts, instead of the process that was used to development them. Chapter 7 summarizes the contribution of the CMF and suggests further enhancements.

CHAPTER 2

BACKGROUND

To understand working of the CMF, it is important to understand the fundamental concepts of a consent management system and overview of the phMPI. Section 2.1 and 2.2 provides overview of these two concepts.

2.1 Overview of consent management system

A consent management is a system, process, or set of policies for allowing individuals to determine what information they are permitting data consumers to access [17]. More specifically consent management in healthcare enables patients to affirm their participation in e-health initiatives, to establish consent directives, and to determine *who* will have access to their protected health information (PHI), for *what* purpose, and *under what* circumstances. Proper consent management also requires the establishment, management, and enforcement of organization and jurisdictional privacy policies [17].

The effective coordination of health care relies on communication of confidential information about patients between different care providers. Care providers must be able to give or withhold consent to those who wish to access their information. Organization consent rules are general restrictions that apply to patient information managed by the organization, unless overwritten from consent rule specified by a patient directly. For example, the UDOH can have a consent rule that denies access to its patient's phone information, except in emergency conditions. Individual consent rules allow patients to restrict data consumers from accessing their information. For example, a patient can have a consent rule to deny access to his/her birth information.

Some consent rules apply to specific groups of people, such as children under the age of 5 or adults in a witness protection program. New groups can be defined at any time and people can be placed in those groups or removed from those groups as needed. An MPI administrator can then define common consent rules for a whole group. This helps simplify the management of consent in certain situations.

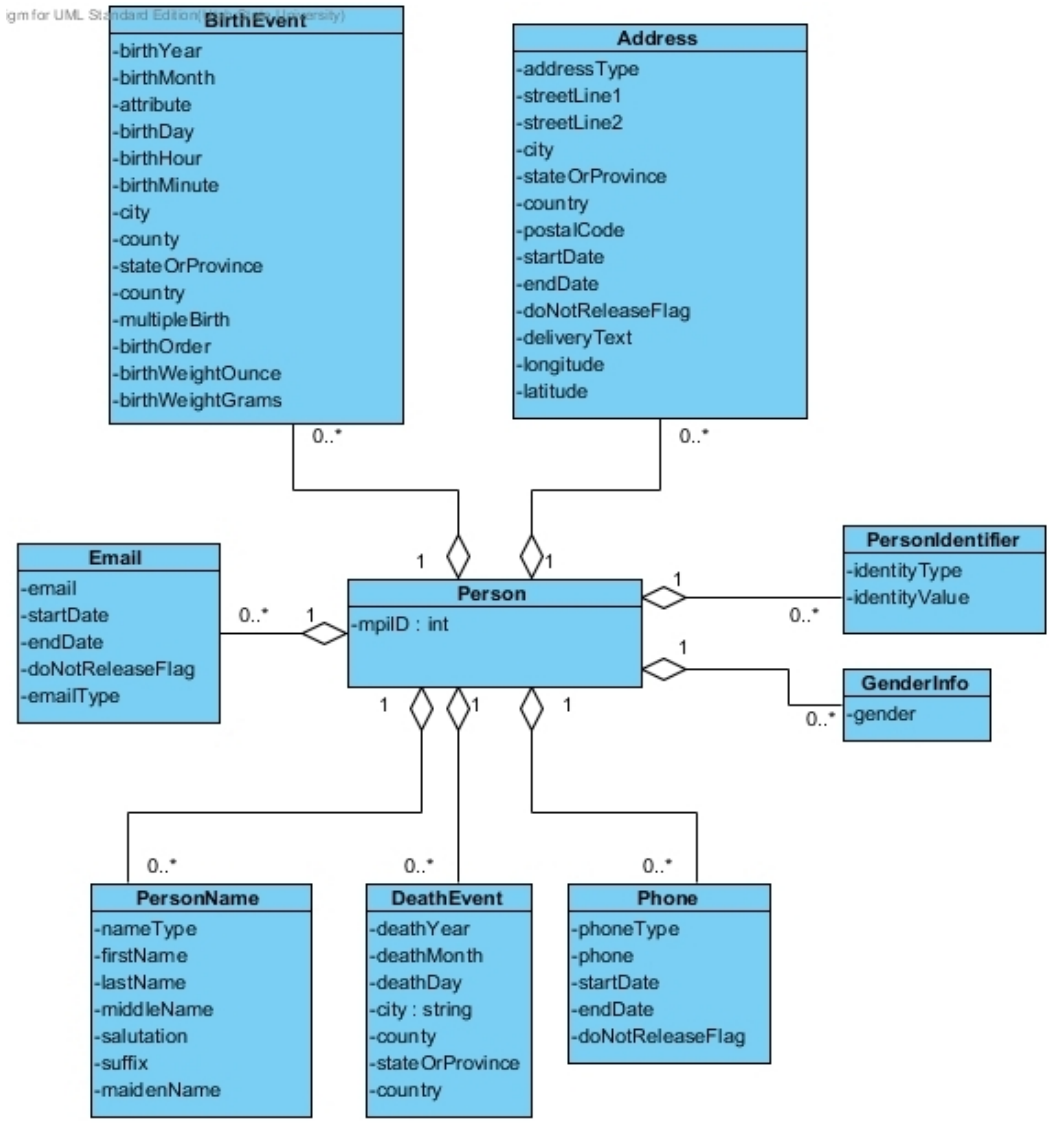
2.2 Overview of the phMPI

Utah's phMPI is a complex, distributed, service-oriented system that integrates data from a wide range of different sources, also called external systems. Currently, these systems are Vital Statistic's Birth Master Record, Vital Statistic's Death Record also called the EDEN, the *Utah Statewide Immunization Information System* (USIIS), and the *Child Health Advanced Record Management* (CHARM) system. An effort is currently underway to integrate In-Patient Data, Out-Patient Data and Emergency Care systems. There is also a future plan to add dozens of external systems later.

From a data-structure perspective, the primary concept is a "person" known to the phMPI. For documentation purposes, we refer to this abstraction as an *internal person*. Many records of various kinds linked to an internal person represent that person's identity, demographics, contact information, family relations, and more. These associated records are called *Data Chunks.* Figure 1 shows the logical data model for the internal person and a subset of the possible data-chunk types. For example, Address is a data chunk that consists of a person's street address, city, state or province, country, postal code, and miscellaneous information [11].

In general a *Data Chunk* is a cohesive set of values that came from an external data source or that was computed by the phMPI as the *best version of truth* [22]. Internal

person can have data chunks of the same type from different data source. For example,



CHARM and USIIS can contribute to a patient's name. It also possible for the phMPI to receive multiple data chunks of the same type from a single data source, particularly for contact information. For example, CHARM can contribute a child's Residence, Work, Mailing, and Mother's Mailing address information.

Each *Data Chunk* has meta-data that identifies it's source, the time it was stored in the database, last updated time, accuracy of the data, expiration date of the data, data

verification date, date the data was linked to a person and so on [22]. Figure 2 shows the relationship between data chunks and meta-data.

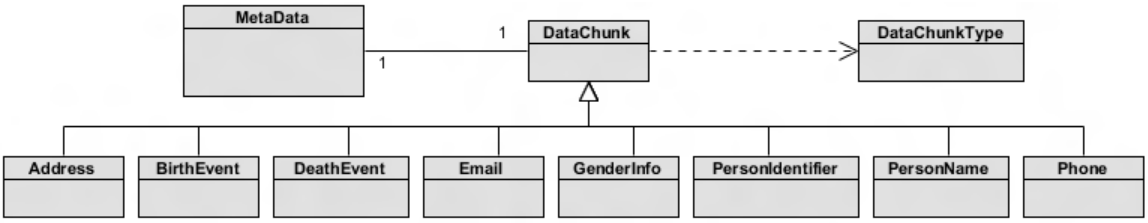


Figure 2. Relationship between data chunks and meta-data

In general, phMPI's database schema is similar to star schema, which a common approach for data warehouse. The star schema consists of one or more fact tables referencing any number of dimension tables. It is more effective for handling simpler queries [23]. All *Data Chunks* referencing a single person table makes phMPI's schema similar to star schema. The person table here is a fact table with *Data Chunks* being dimension tables.

The phMPI was built using *Vitruvian,* a development framework for building service-oriented distributed systems. Vitruvian enabled the phMPI to be designed and developed as a collection of services that can run in multiple processes on different host machines. Figure 3 shows the phMPI's architecture in terms of its major components, which are the MPI Application Programming Interface (API), Data source API, Request Queue, Workflow Manager, Publication Server, Web Service Publisher, and Consent Management Service (*CMS*). The Data Source API and CMS together form the CMF.

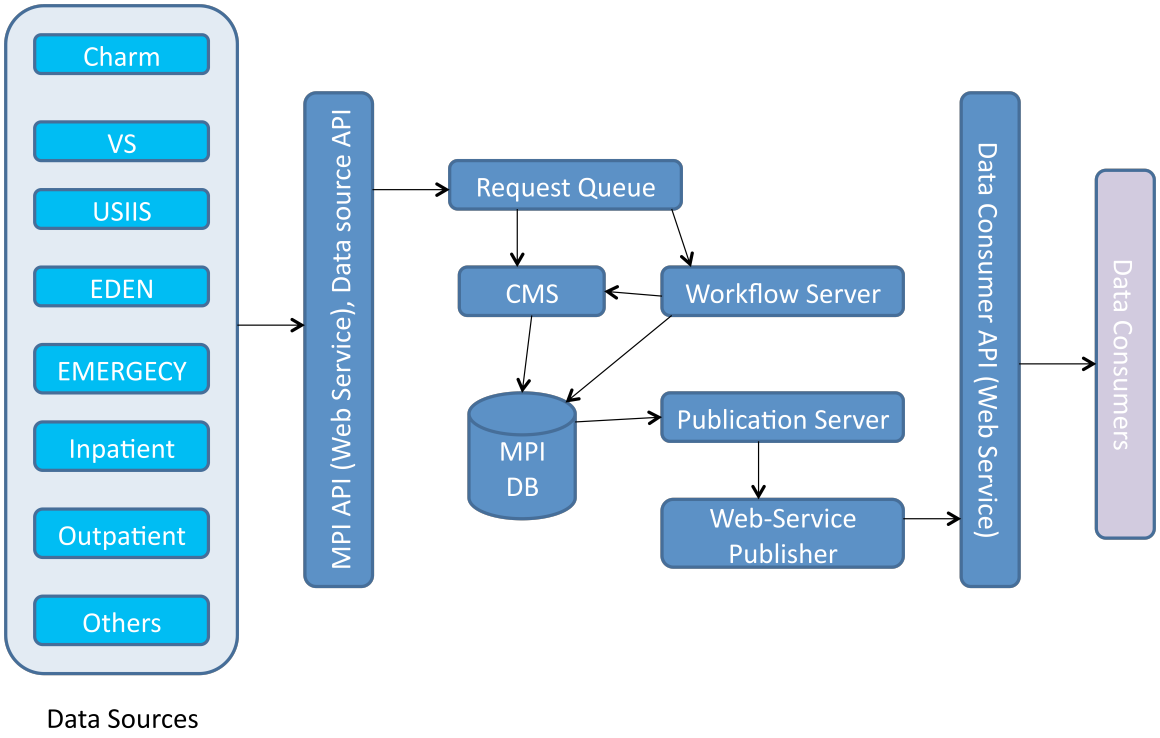


Figure 3. Overall Architecture of the phMPI [15]

External systems submit their requests to the phMPI via the MPI API (WebPortal). When the MPI API receives a request from an external system, it places it directly into the Request Queue. Besides the basic validation, it doesn't try to interpret the type or contents of the message. For scalability, security, and performance reasons, the MPI API and Request Queue run on different host machines.

Workflow Manager tries to access a request from the Request Queue and selects a workflow based on the request's type, message format, and its originating source. The manager will then process the request by following the workflow selected. A workflow is a sequence of actions that can lead to changes in the phMPI's data.

Together the Publication Server, Web Service Publisher, and Data Consumer API form the *Publication-Subscription Subsystem* (PSS). When a change occurs on the phMPI's data, the PSS will determine if it matches the definition of any publication

channels, and for each one that it matches, it triggers a publication to be sent to all the subscribers of that channel [21].

External systems submit consent rule requests to the phMPI via the Data Source API. A consent rule is a declaration signed by a patient to allow or deny information to all/some of data consumers. The Data Source API is designed to support multiple communication protocols such as TCP, UDP, RMI, or Web Services, and multiple message formats such as XML and *eXtensible Access Control Markup Language* (XACML). XACML is used for attribute based access control systems and role-based access control systems. However, Data Source API currently supports only TCP communication. It uses the CMS to insert, lookup, update, or delete consent rules, and the Workflow Manager uses the CMS to apply consent rules before information is sent back to the requesting data source.

CHAPTER 3

SYSTEM ANALYSIS AND REQUIREMENTS

3.1 Overview

Software systems can be difficult to understand, and once a particular design is in place, it can be difficult to change if it is not built properly. Software systems and the problems that they solve can be abstract and hard to visualize. A visual modeling language, like *Unified Modeling Language* (UML), can help software engineering visualize and reason about the problems that a system needs to solution and then semantically worked out a design for the system [18]. MPI team chose UML to model use cases for the system, classes, relationship between classes, and interactions between objects of those classes, because it is easy to remodel a solution in case of change in requirements or better understanding of the problem.

From an analysis perspective, UML can describe a system (and the problems it addresses) in terms of concepts that related directly the environment that the system must live. For the CMF, this means that describing what the phMPI needs from the CMF, objects (or classes of objects) that can fulfill those needs, and how those object interact. Section 3.2 outlines the needs of the phMPI in terms of UML use cases. Section 3.3 describes the CMF's object structure using UML class diagrams and Section 3.4 summarizes an analysis of key object interactions using UML sequence diagrams. The functional requirements in Section 3.5 provide details about the use cases and the system structure and Section 3.5 describes the non-functional

3.2 phMPI's needs for the CMF

Use-case diagrams are used to describe all of the available functionality of the system at a very higher level. Use-case diagrams, which are part of UML, are the starting point for UML based software development projects. They describe the interaction of any person or external device with the system under design. Use-case diagrams capture the primary actors and their goals.

There are three main actors that use the CMF. Figure 4 shows the primary actors of the system. Wokflow Manager and External System are software components that are described in Section 2.2. MPI Administrator is a user who manages organization and group consent rules.

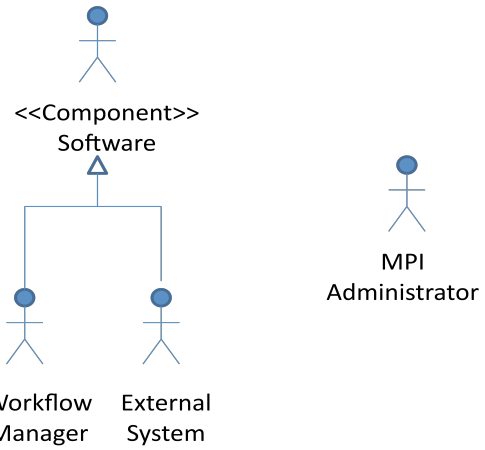


Figure 4. Primary actors of Consent Management Facility

3.2.1 Use-Cases/Goals for External System

Figure 5 describes the use-case goals for an external system. The stick figure in it represents an external system and the bubbles describe the actions performed by external systems. Addition of consent rules is possible with/without consent rule document in the message payload. CMF should support both Simple XML and XACML message formats.

3.2.2 Use Cases/Goals for Workflow Manager

The Workflow Manager uses the CMF to access patient-specific consent information and applies applicable and valid consent rules during patient information lookup operations. Figure 6 shows the use-case diagram of a workflow manager.

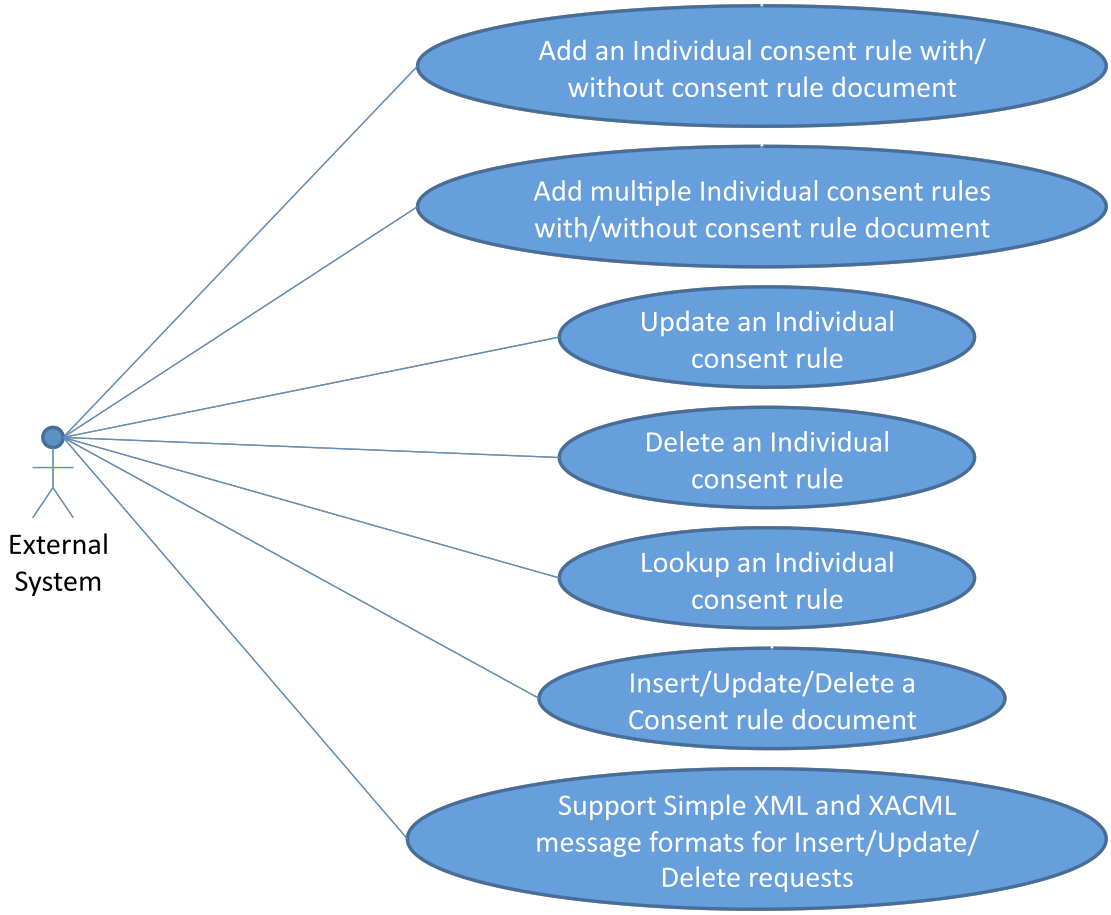


Figure 5. Use-cases for External System.

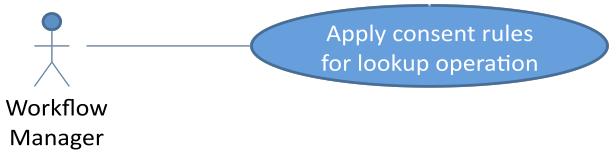


Figure 6. Use-case for workflow manager

3.2.3 Use Cases/Goals for MPI Administrator

The administrator of the phMPI has privileges to manage organization and group consent rules. MPI Administrator can create, delete, update and lookup organization and group consent rules. Figure 7 shows the use-case diagram of MPI Administrator.

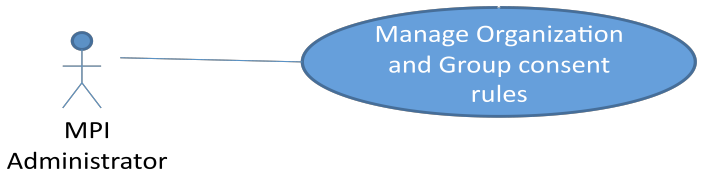


Figure 7. Use-case for MPI Administrator

3.3 Analysis of System's Object Structure

For an analysis perspective, a UML class diagram is a visual description of types of objects that exist in a system and the relationships that exist among them, in terms that relate directly to the systems environment or the "real world". They can help developers solidify the understanding of a system's components and structure, and thus set the stage for a more informed design [3].

UML class diagrams are the most widely used diagrams in modeling systems in the object-oriented software development paradigm. A class diagram can contain packages, classes, interfaces, associations, generalizations, and other kinds modeling components for describing structural ideas. Class diagrams are not just for visualizing and documenting structure models, but also for constructing an executable system with forward, reverse, and round-trip engineering [4]. Figure 8 shows the class diagram of the CMF. Consent Management Service is the main class responsible for creating, updating, and deleting consent rules. It is also responsible for searching consent rules and

applies them on patient information during lookup operation. Operation Type enumeration is used to identify either an insert/update/delete operation on the consent rule request.

A consent rule can have a scanned copy of a paper document signed by the patient. The Consent rule or document enumeration is used in differentiating the action to be performed either on a consent rule or a document associated with the consent rule. The Consent rules count helps in identifying the consent rule count in a request by data sources. The Consent rules count enumeration is used to switch to a transaction mode for saving multiple consent rules or consent rule documents. Similarly, it is also used in Updating or Deleting consent rules and consent rule documents. An XML Parser Strategy is an abstract class that defines the common interfaces that are to be implemented for parsing an incoming request, which can be either a Simple XML or a XACML. An XACML Parser is a concrete implementation of the XML Parser Strategy class, which is responsible for parsing an incoming request in XACML format. Similarly, a Simple XML Parser is another concrete implementation of the XML Parser Strategy class, which is responsible for parsing an incoming request in simple xml.

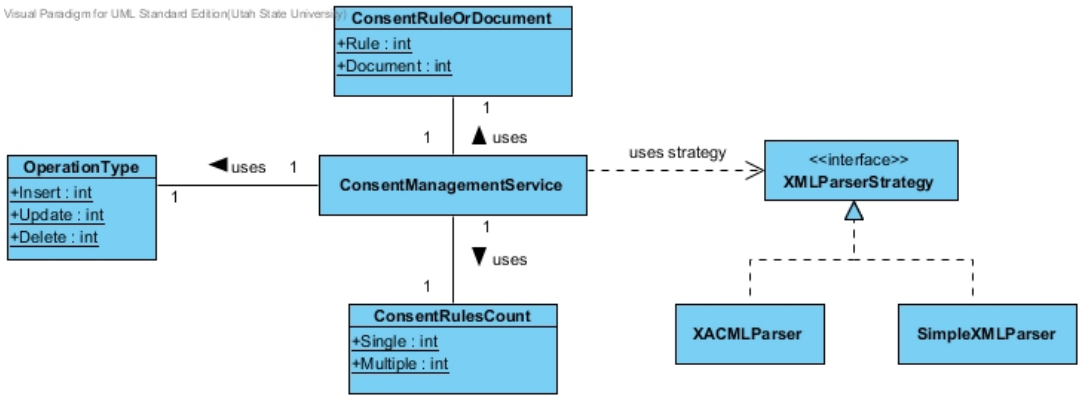


Figure 8. Class diagram of the CMF

3.4 Key Interactions

Sequence diagrams show how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespan of objects relative to those messages. Figure 9 shows the key interactions between the data source and the CMS. The external source or data source sends a consent management request to the consent management portal. The consent management portal in turn will send the consent management request to CMS with the help of request queue. A detailed description of interactions is described in Appendix C.

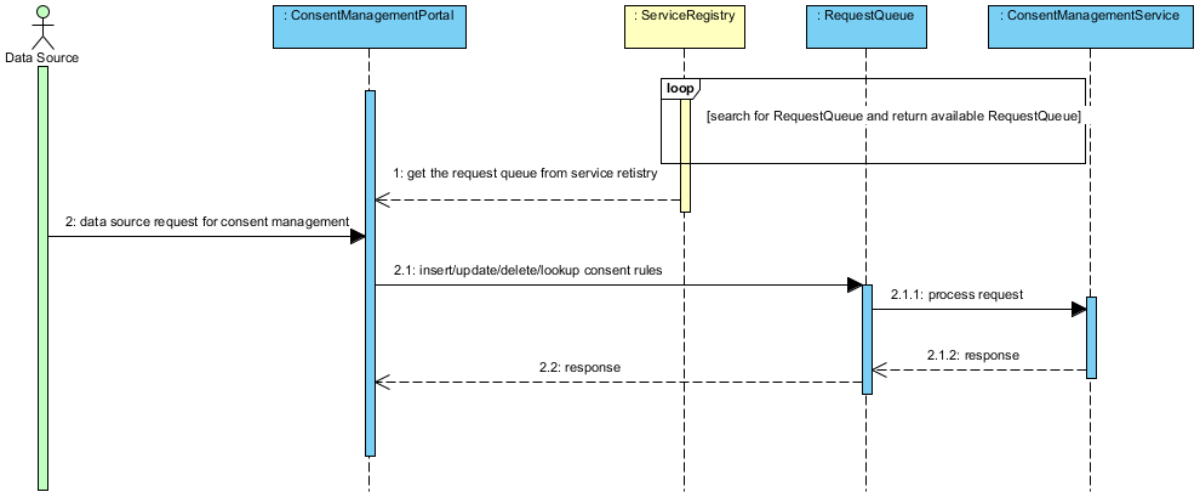


Figure 9. Key interaction diagram of data source with the CMS

Sequence diagram of work flow manager with CMS is shown in Figure 10. Detailed explanation of actual interaction between the work flow manager and MCS is described in Section 2.2.

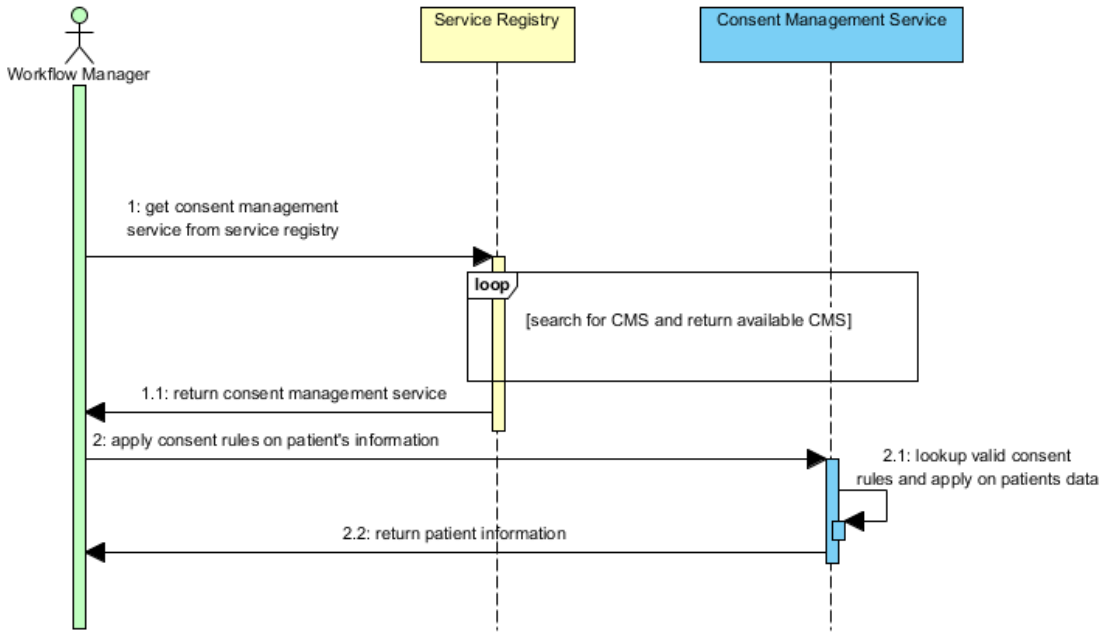


Figure 10. Key interaction of Workflow manager with CMS

3.5 Functional Requirements

In software engineering, a functional requirement defines a function of a software system or its component. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish [24]. A detailed analysis of the phMPI goals for the CMF, object structures, and interaction resulted in the following requirements for the CMF.

3.5.1 MPI Administrator Requirements

The consent management system must allow the MPI administrator to setup and maintain general data sharing rules and restrictions.

- Data sharing can be granted or restricted based on data-chunk type.
- Data sharing can be granted or restricted based on data source.

3.5.2 External System or Data Source Actions

The consent management system must allow data sources to record consent information for individual patients.

- The consent information may allow or restrict data sharing between data sources and consumers.
- The consent information may give permissions to use personal data in research study.

3.5.3 Workflow Manager Requirements

- The consent management service must provide a service for the workflow manager to access patient-specific consent information.

3.6 Non-functional Requirements

The CMF is a new module developed for the existing phMPI. It needs to be integrated to the phMPI. For this reason, the CMF must comply with the development standards defined for phMPI. Functional requirements are supported by non-functional requirements, which impose constraints on the design or implementation[24]. Following are the non-functional requirements for the CMF.

3.6.1 Operating System

The consent management system should be developed on windows based operating system.

3.6.2 Languages and Platform

- Programming language shall be C# (C-Sharp).
- The system shall use Vitruvian framework for distribution.

3.6.3 Software Testing

- The system shall have comprehensive unit tests for each class.
- Integration testing is essential to ensure functionality, performance and reliability requirements placed on major design items.

3.6.4 Documentation

Design documents and a report about the system shall be given to assist the end users in understanding the design and functionalities of consent management system.

3.6.5 Logging

- The system shall support logging at all layers.
- The logging levels shall be configurable at run time

CHAPTER 4

ARCHITECTURAL DESIGN

4.1 System Architecture

The CMF's design had to be flexible so it could support the addition of new message formats in the future. The system is designed in such a way that the maintenance, extension and testing will be easier. The design consideration included good modularization and low dependency among modules making the CMF reusable in other MPIs.

The packages and reusable components used in developing the CMF are described in Section 4.2. It explains the class diagrams and various dependencies between packages used in CMF. Section 4.3 illustrates the necessity of database design and the data tables used by the data layer.

4.2 CMF system design

Vitruvian framework is used to build the system. It made distribution and logging easier and also takes care of the connection problems. It provides an ORM tool to generate a subsystem of classes for providing a persistence layer. This persistent layer follows an object-gateway pattern [13].

An abstract class diagram of the CMF is shown in Figure 11. The Consent Management Service class implements an IService interface. It contains all the methods required for inserting, updating, deleting, and lookup of consent rules. The class also contains the methods required for applying consent rules on person data for a lookup

request. The package *Vitruvian* contains classes and methods for distributing objects. It also contains classes and methods for logging. The package *DataLayer* is a persistence layer implemented using Vitruvian DbObjects, which are similar to Hibernate [12]. DbObjects follow the Object-gateway pattern [13]. All the classes in the DataLayer inherit and implement the Vitruvian DBObject interface (Vitruvian needs this). Classes in Data Layer are automatically generated using the Vitruvian object generation tool. Each class maps to a database table and properties of the class maps to columns of the table.

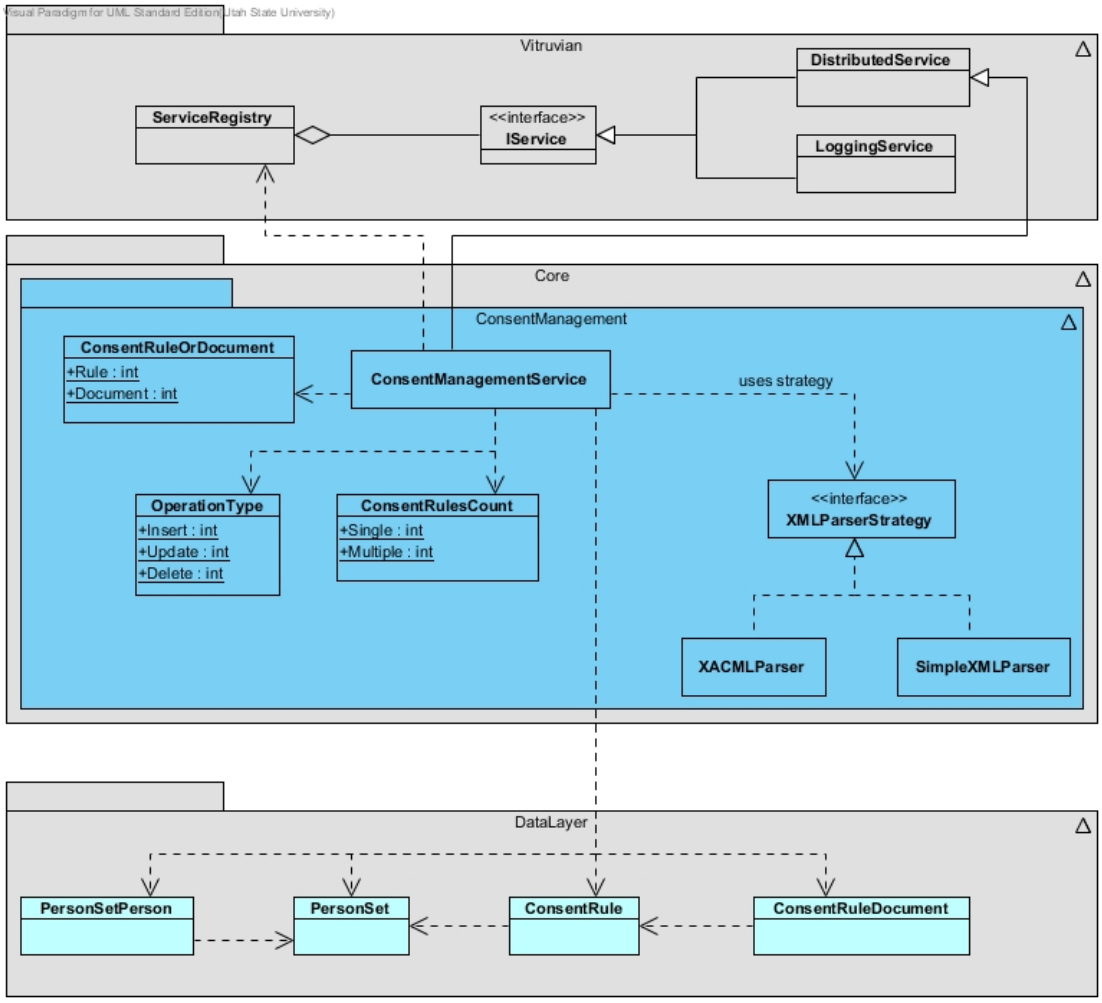


Figure 11. Class diagram of Consent Management Service with data layer

Figure 12 shows the class diagram of the CMS and its interaction with the web portal and workflow services. The Web Portal API uses the Request Queue service to send an incoming consent rule request to the Consent Management Service class present in the Core.ConsentManagement package.

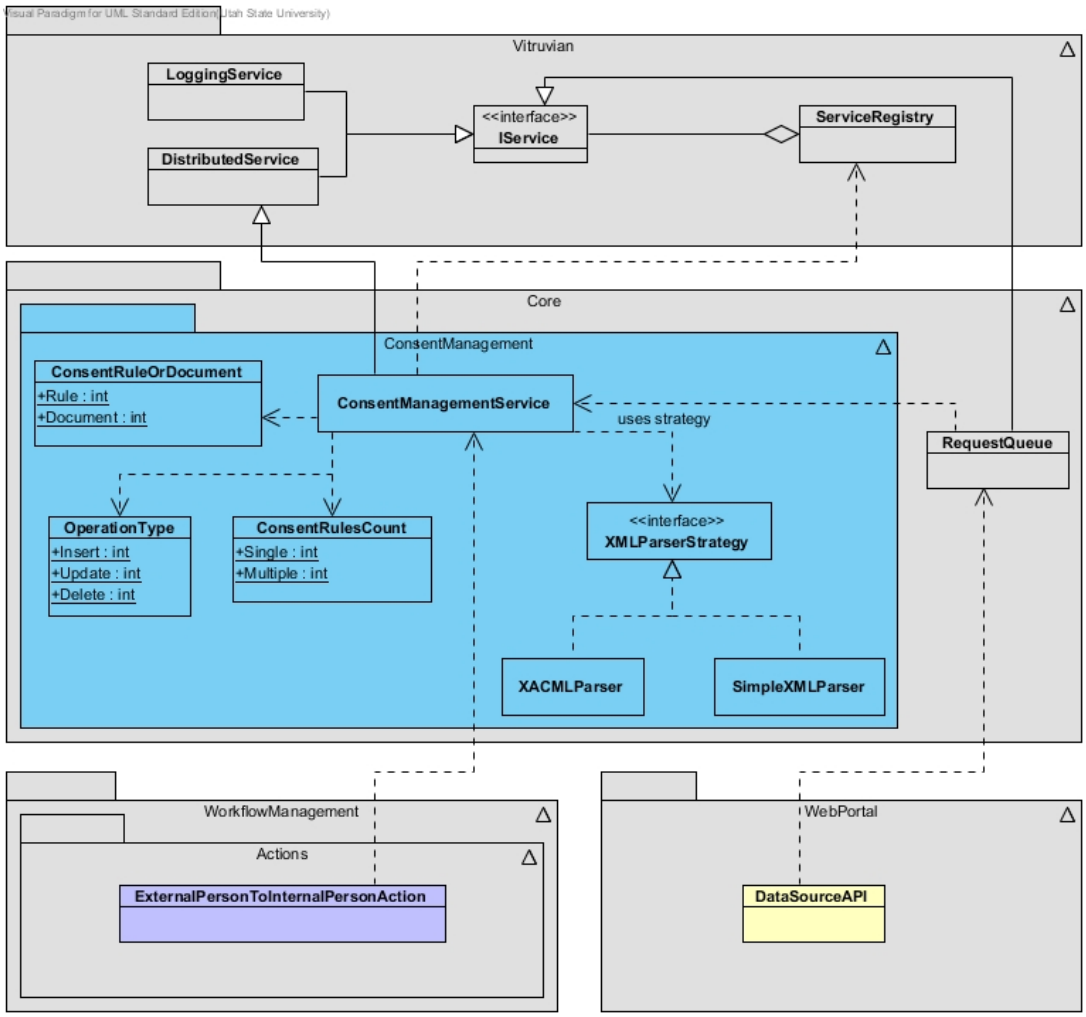


Figure 12. Class Diagram of Consent Management Service and web portal

The External Person to Internal Person Actions class of the workflow service uses the Consent Management Service class to apply the consent rules and remove the required person's data before returning the patient information to external systems.

Figure 13 shows the class diagram of the CMS. The use of enumerations is explained in Section 4.4 and the public methods available for data consumers are explained in Appendix D. The XML Parser Strategy interface has three abstract methods which are to be implemented by the concrete classes. The abstract methods are:

- Parse Consent Rule: This method takes an Xml Node object as parameter, parses the incoming consent rule and creates a Consent Rule object, which is either used to insert new consent rule or update an existing consent rule or delete the incoming consent rule. This method expects the incoming Xml Node to be valid.
- Validate Xml Document: This method takes an Xml Document and Rules Count enumerator as input. The Xml Document is an xml document object constructed from the incoming consent rule request. XML schema definition used to validate xml depends on the number of consent rules present in the consent rule request
- Validation Call Back: This is an event handler for xml document validation. It is fired in case of an error in validating the xml.

Implementation of the tools to support the goals of MPI Administrator is not cost effective. A general database management GUI can satisfy all the goals for the MPI administrator.

4.3 Database Design

Database design can be thought of as the logical design of the base data structures used to store the data. While designing the database the below steps were carried out

- Relationships between the different data elements were determined

- Created a concrete structure of the data on the basis of the defined relationships

Figure 14 shows an entity-relationship diagram that describes the data tables used for the CMF.

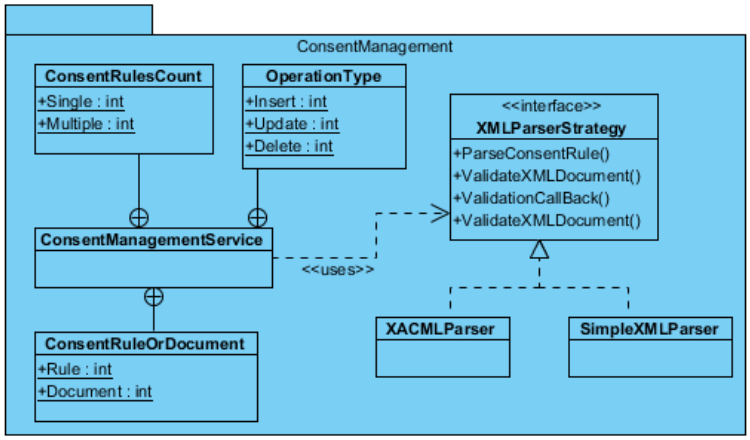


Figure 13. Class diagram of Consent Management Service

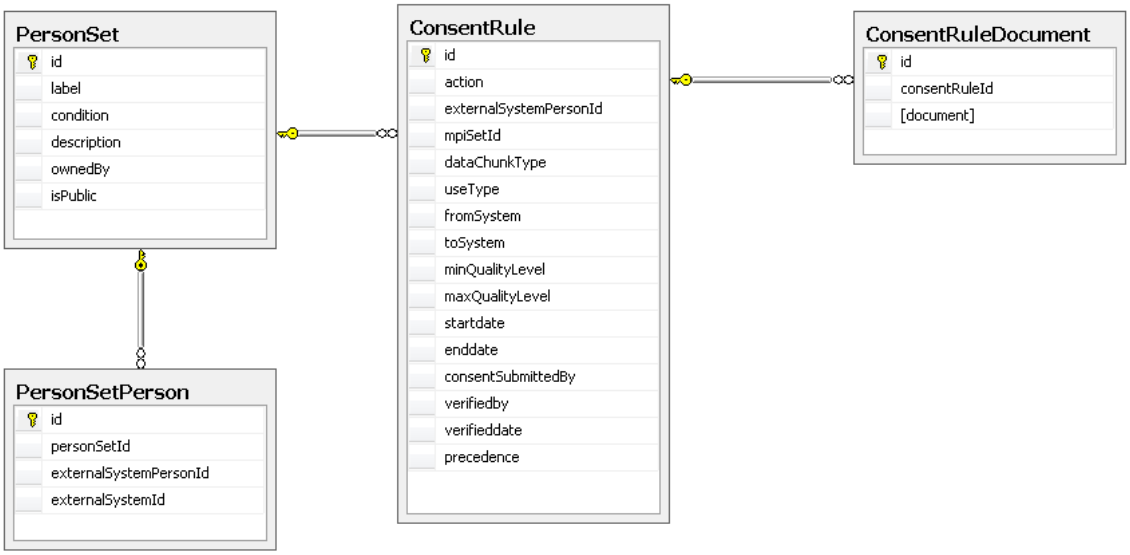


Figure 14. Entity-relation Diagram for data layer of the CMF

The main data table for the CMF is the Consent Rule, which contains the consent rule information (a detailed explanation of the Consent Rule class is given in chapter 4). The Person Set table is used to store information of group-based (known as MPI Set) consent rules. The Person Set Person class has entries of external persons belonging to the MPI Set. Each consent rule can have a consent rule document signed by the individual or organization. The document is stored in the system as a scanned grayscale PDF document. Initial design had this document present in the Consent Rule table, but frequent access of the Consent Rule table made the lookup operations time consuming. The decision to store the consent rule document in a separate table was taken to optimize the lookup operation on the Consent Rule table. It is also safer to store the document in the data base against storing it in a file location on the server.

CHAPTER 5

IMPLEMENTATION

5.1 Overview

Several key implementation challenges had to be addressed during the development of the CMF, such as ensuring that the application of consent rules did not unduly slow down MPI's processing of lookup requests. Section 5.2 describes a physical data structure that helps solve this problem. Representation of consent rules within CMF system and external to CMF system is different. Section 5.3 explains the solution designed to this problem. Section 5.5 describes another challenge in optimizing the performance of XACML validation.

5.2 Physical data structure

Section 4.2 described the database design of the CMF. Table 1 explains the contents of the ConsentRule table. It also describes the data type of each column and constraints applied before a new rule is saved. The external systems uniquely identify a patient by by externalSystemPersonId. The main reason for storing the externalSystemPersonId is that it remains constant for a patient even when an update, merge, and split operations are performed on patient's data.

Saving consent rule document along with the rule in the ConsentRule table made the lookup of consent rules during information retrieval operation time consuming. To improve the performance of the lookup process, the rule document is stored in a separate data table called ConsentRuleDocument. This normalization process helped in reducing the unnecessary lookup of consent rule document (varies in size between hundreds of kilo

bytes to a couple of mega bytes) each time a consent rule is used. Table 2 describes the content of the ConsentRuleDocument table in detail. Indices have been created on frequently queried columns like externalSystemPersonId and mpiSetId to improve the performance of the system.

Table 1. Detailed explanation of Consent Rule table

Column Name	Column Type	Values/Notes
id	int	Primary key
action	char(1)	A = Allow D = Deny
externalSystemPersonId	varchar(32)	null means "ANY", otherwise it's a specific external system Id for a person
mpiSetId	int	set to which an mpiId belongs
dataChunkType	varchar(512)	null means any data-chunk types. Otherwise, a coma-separated list of data-chunk types
useType	char(1)	N = Normal, C = Conditional, E = Emergency
fromSystem	varchar(16)	null means any data source. Otherwise, a string that identifies the external system
toSystem	varchar(16)	null means any data source. Otherwise, a string that identifies the external system (system or user group with an organization)
minQualityLevel	float	null means any. Otherwise, a real that specifies a minimum quality score.
maxQualityLevel	float	null means any. Otherwise, a real that specifies a maximum quality score.
startDate	Timestamp	null means no start date. Otherwise, a date/time on which this rule begins
endDate	Timestamp	null means no start date. Otherwise, a date/time on which this rule expires
consentSubmittedBy	varchar(16)	string that identified an external system which submitted the consent
verifiedBy	varchar(32)	The person who verified the consent rule
verifiedDate	DateTime	The date on which the consent rule was verified
precedence	int	Precedence of the rule for applying in case all other fields of the rule match with another rule.

Table 2. Detailed explanation of Consent Rule Document table

Column Name	Column Type	Values/Notes
id	int	Primary key
consentRuleId	int	Foreign key to ConsentRule table
document	nvarchar(max)	Utf-8 encoded string format of a pdf document.

The three types of ConsentRules are differentiated by either null or non-null values present in the externalSystemPersonId and the mpiSetId fields of the ConsentRule table.

- Organization – level (null externalSystemPersonId and null mpiSetId)
- MPI Set – level (null externalSystemPersonId)
- Individual – level (non-null externalSystemPersonId)

Examples for the three types of consent rules are given below. All the consent rules described here are examples, not actual consent rules.

5.2.1 Organizational Consent Rule

Table 3 describes an organizational consent rule to deny address information submitted by UU of any patient when requested by IHC from 1/1/2012 to 12/1/2012.

Table 3. Organizational Consent Rule example

Column Name	Column Value
id	1
action	D
externalSystemPersonId	NULL
mpiSetId	NULL

dataChunkType	ADDRESS
useType	N
fromSystem	IHC
toSystem	UU
minQualityLevel	NULL
maxQualityLevel	NULL
startDate	1/1/2012
endDate	12/1/2012
consentSubmittedBy	IHC
verifiedBy	NULL
verifiedDate	NULL
precedence	2

5.2.2 MPI Set Rule

Table 4 shows an MPI Set consent rule to deny address information submitted by UU of patients belonging to set Id 3 when requested by IHC from 1/1/2012 to 12/1/2012.

Table 4. MPI Set Consent Rule example

Column Name	Column Value
id	1
action	D
externalSystemPersonId	NULL
mpiSetId	3
dataChunkType	ADDRESS
useType	N
fromSystem	IHC
toSystem	UU
minQualityLevel	NULL
maxQualityLevel	NULL
startDate	1/1/2012
endDate	12/1/2012
consentSubmittedBy	IHC
verifiedBy	NULL
verifiedDate	NULL
precedence	8

5.2.3 Individual level Consent Rule

Table 5 describes an organizational consent rule to deny address information submitted by UDOH of patient with id 2010 042512 when requested by UU from 1/1/2012 to 12/1/2012.

Table 5. Individual Consent Rule example

Column Name	Column Value
id	1
action	D
externalSystemPersonId	2010 042512
mpiSetId	NULL
dataChunkType	ADDRESS
useType	N
fromSystem	UDOH-VS
toSystem	UU
minQualityLevel	NULL
maxQualityLevel	NULL
startDate	1/1/2012
endDate	12/1/2012
consentSubmittedBy	UDOH-VS
verifiedBy	NULL
verifiedDate	NULL
precedence	3

5.3 Format of consent rules

The CMF provides consent management services to external systems as well as MPI administrators. For this purpose two formats of consent rules had to be supported. MPI administrators, who are internal to the CMF, have direct access to the data layer (Figure 8) and the DBObjects of the data layer help in managing the set based and organizational rules efficiently.

A standard format for the requests or operations coming from or going out to external sources had to be defined. The CMF provides SimpleXML and XACML formats to help external sources to communicate with the CMF. The XML schema definitions for the two supported rule formats are explained in Appendix A. Section 5.5 describes the problem induced by using XACML and its solution in detail.

5.4 Returning Person Data Based on Consent

An algorithm was designed to determine the consent when multiple rules apply to the same type of data. Let us consider a query (Normal, Conditional, or Emergency) involving some patient *p* and data consumer *dc*, the consent rules that pertinent to the query include

- Organization rules that have *dc* as a ToDataSource or ToDataSource being null
- Individual rules that have *p.idValue (external system's unique id)* equal to consent rule's externalSystemPersonId
- MPI set rules by the dc

The above rules should not have expired and are in affect (started) and where the uses of rules are either unspecified or matches the use of the query. The algorithm is as follows.

For each data chunk

1. Select just those rules that apply
   1.1. Select just those rules pertinent to the data chunk type.
   1.2. Then, select just those rules whose FromDataSource is null or the FromDataSource is the same as the data chunk's data source.
   1.3. The MinQualityLevel is either null or greater than the data source's quality level.
   1.4. The MaxQualityLevel is either null or less than the data source's quality level.
2. Order the applicable rules as follows: individual first (externalSystemPersonId not null), then MPI set rules and finally organizational.
   2.1. Within each set of rules, order consent rules based on number of null fields (FromSystem, ToSystem and DataChunkType) in ascending order.
   2.2. In case there are same number of null fields then
      2.2.1. Give first preference to consent rule whose DataChunkType is not null, then to the rule whose FromSystem is not null and finally to the rule with ToSystem not null. If all the above three conditions are same, order the consent rule with highest precedence first. In case the precedence also matches, give preference to the first saved consent rule.
3. Step through each rule with a "D" or "A" action is found.
4. If "A" is found then include the data chunk in the resulting view of person

Algorithm for applying consent rules

The algorithm to sort applicable consent rules is explained with examples. For simplicity externalSystemPersonId is abbreviated as espId in the following sub-sections.

5.4.1 Order rules by number of null values in ascending order

Let us consider the following Individual rules with distinct null values as shown in the Table 6

Table 6. Sample Individual consent rules before sorting

id	action	espId	mpiSetId	dataChunk Type	use Type	from System	to System
1	D	1234	NULL	NULL	N	NULL	NULL
2	P	1234	NULL	Address	N	NULL	IHC
3	D	1234	NULL	Address	N	NULL	NULL
4	D	1234	NULL	Address	N	UDOH-VS	IHC

Upon applying the sort criteria the sort order of the rules are shown in the Table 7.

Table 7. Sample individual consent rules with distinct null values after sorting

id	action	espId	mpiSetId	dataChunk Type	use Type	from System	to System	No. of Nulls
4	D	1234	NULL	Address	N	UDOH-VS	IHC	0
2	P	1234	NULL	Address	N	NULL	IHC	1
3	D	1234	NULL	Address	N	NULL	NULL	2
1	D	1234	NULL	NULL	N	NULL	NULL	3

5.4.2 Null value column count in a consent rule is one.

Let us consider the following Individual rules shown in Table 8, which have one null value among dataChunk, fromSystem and toSystem values.

Table 8. Sample Individual Consent rules with single null value

id	action	espId	mpiSetId	dataChunk Type	use Type	from System	to System
1	D	1234	NULL	Address	N	NULL	IHC
2	P	1234	NULL	Address	N	UDOH-VS	NULL
3	D	1234	NULL	NULL	N	UDOH-VS	IHC

Upon applying the sorting rule algorithm, the consent rules are shown in Table 9

Table 9. Sample Individual Consent Rules with single null value after sorting

id	action	espId	mpiSetId	dataChunk Type	use Type	from System	to System	No. of Nulls
2	P	1234	NULL	Address	N	UDOH-VS	NULL	1
1	D	1234	NULL	Address	N	NULL	IHC	1
3	D	1234	NULL	NULL	N	UDOH-VS	IHC	1

5.4.3 Null value column count is two

Let us consider the following Individual rules shown in Table 10 having two null values among dataChunk, fromSystem and toSystem values.

Table 10. Individual consent rules with two null values

id	action	espId	mpiSetId	dataChunk Type	use Type	from System	to System
1	D	1234	NULL	NULL	N	UDOH-VS	NULL
2	P	1234	NULL	NULL	N	NULL	IHC
3	D	1234	NULL	Address	N	NULL	NULL

Upon applying the sorting rule algorithm, the consent rules are shown in Table 11.

Table 11. Sample Individual consent rules with two null values after sorting

id	action	espId	mpiSetId	dataChunk Type	use Type	from System	to System	No. of Nulls
3	D	1234	NULL	Address	N	NULL	NULL	2
1	D	1234	NULL	NULL	N	UDOH-VS	NULL	2
2	P	1234	NULL	NULL	N	NULL	IHC	2

In all the above cases, if there are two rules that satisfy all the conditions, then the rule with the highest precedence number will be applied first. In case the rules are tied with the same precedence number, then the rule with least Id is considered first. Examples of individual consent rules can be found in Appendix B.

5.5 Optimization for validation of XACML

The original XML schema definition [16] for validating XACML slowed down the process of validating an incoming XACML rule due to many recursive constraints. It took 6-8 seconds on an average to validate a XACML request. The CMF does not use all the attributes provided by the XACML to identify a consent rule, so to improve the

performance of validating an incoming XACML request, the XML schema definition was trimmed down to complete validating a request in couple of tens of milliseconds. The updated XML schema definition is as below.

XSD 1. XSD FOR XACML

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  xmlns:xacml="http://www.mpi.org/XACML"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.mpi.org/XACML"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="PolicySet" type="xacml:PolicySetType"/>
  <xs:complexType name="PolicySetType">
    <xs:sequence>
      <xs:element ref="xacml:Target"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="xacml:Policy"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="VersionType">
    <xs:restriction base="xs:string">
      <xs:pattern value="(\d+\.)*\d+"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="Policy" type="xacml:PolicyType"/>
  <xs:complexType name="PolicyType">
    <xs:sequence>
      <xs:element ref="xacml:Target"/>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="xacml:Rule"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
    <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
    <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
  </xs:complexType>
  <xs:element name="Rule" type="xacml:RuleType"/>
  <xs:complexType name="RuleType">
    <xs:sequence>
      <xs:element ref="xacml:Target" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="RuleId" type="xs:string" use="required"/>
    <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>
  </xs:complexType>
  <xs:simpleType name="EffectType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Permit"/>
      <xs:enumeration value="Deny"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="Target" type="xacml:TargetType"/>
  <xs:complexType name="TargetType">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xacml:AnyOf"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="AnyOf" type="xacml:AnyOfType"/>
```

```
  <xs:complexType name="AnyOfType">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element ref="xacml:AllOf"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="AllOf" type="xacml:AllOfType"/>
  <xs:complexType name="AllOfType">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element ref="xacml:Match"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Match" type="xacml:MatchType"/>
  <xs:complexType name="MatchType">
    <xs:sequence>
      <xs:element ref="xacml:AttributeValue"/>
      <xs:element ref="xacml:AttributeDesignator"/>
    </xs:sequence>
    <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
  </xs:complexType>
  <xs:element name="AttributeDesignator" type="xacml:AttributeDesignatorType"/>
  <xs:complexType name="AttributeDesignatorType">
    <xs:attribute name="MustBePresent" type="xs:anyURI" use="required"/>
    <xs:attribute name="Category" type="xs:anyURI" use="required"/>
    <xs:attribute name="AttributeId" type="xacml:AttributeIdType" use="required"/>
    <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
  </xs:complexType>
  <xs:simpleType name="AttributeIdType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="ExternalSystemPersonId"/>
      <xs:enumeration value="DataChunkType"/>
      <xs:enumeration value="UseType"/>
      <xs:enumeration value="FromSystem"/>
      <xs:enumeration value="ToSystem"/>
      <xs:enumeration value="MinQualityLevel"/>
      <xs:enumeration value="MaxQualityLevel"/>
      <xs:enumeration value="StartDate"/>
      <xs:enumeration value="EndDate"/>
      <xs:enumeration value="VerifiedDate"/>
      <xs:enumeration value="Precedence"/>
      <xs:enumeration value="Document"/>
      <xs:enumeration value="DocumentId"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="AttributeValue" type="xacml:AttributeValueType"/>
  <xs:complexType name="AttributeValueType" mixed="true">
    <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
  </xs:complexType>
</xs:schema>
```

5.6 Ensuring that the implementation is reusable

As long as MPI uniquely identifies a patient with a unique identifier, the CMF can be reused for consent management. It differentiates between different kinds of tasks performed by various components, making it easier to support reusability of components and also maintain loose coupling between the layers. Implementation of management of consent rules is explained in detail in Appendix C.

CHAPTER 6

SOFTWARE TESTING

6.1. Introduction

Software testing is a key component of the software development life-cycle. Software testing is a very broad term that includes a wide spectrum of different activities from the testing of a small piece of code (unit testing), to the customer validation of a large information system (acceptance testing), to the monitoring of an at run-time network-centric service-oriented application [6].

Software testing is a process of improving quality and validating and verifying that a software product: meets the requirements, works as expected, and is implemented with the same characteristics. Verification is have we built the software right? (*Does it match the specification?*) And validation is Have we built the right software? (*Is this what the customer wants?*)

"The test cases should aim at objectives, such as exposing deviations from user's requirements, assessing the conformance to a standard specification, evaluating robustness to stressful load conditions to malicious inputs, measuring given attributes, such as performance or usability, estimating the operational reliability, etc." [7].

Unit and integration test suits are written to validate the functionality of the CMF. These test suites ensure that any changes made to the existing code will not break the existing CMF functionality.

6.2. Unit Testing

The main purpose of unit testing is to take the smallest piece of testable software in the application, isolate it from the rest of the code, and determine if it behaves exactly as we expect. Each unit is tested separately before we integrate them together. Unit testing is essential because it is proven that a large percentage of defects are identified during the unit test's usage. If unit testing is done properly, later testing phases will be more successful [8].

I have written extensive test cases for each method of the Consent Management Service with the help of Vitruvian framework. All the methods were tested to verify the expected results. .

6.3. Integration Testing

During integration testing individual software modules are combined and tested as a group. This testing comes after unit testing and before system testing. In integration testing, two or more units that have already been tested and are combined into a component where the interfaces among these units are tested [8]. Integration testing identifies problems that occur when units are combined. The idea is to test combinations of units and eventually expand the process to test the modules with those of other groups. Eventually all the modules making up a process are tested together [9]. Unit tests among different modules have been executed to test the module interations. These were easily integrated into integration testing with the help of Vitruvian framework.

CHAPTER 7

SUMMARY

The CMF restricts the data retrieval from the MPI by any given user to grant access to that information that can be shared with that user based on all the consent rules that apply. It also allows external systems to add, update, or delete individual consent rules. It provides a service for the phMPI administrators to add, update, or delete organizational and group-based consent rules. It can be easily reused to implement Consent Management for any Mater Patient Index as long as a patient can be uniquely identified by a single identifier.

The CMF manages individual consent rules by web services. Future work could focus on designing and developing an administrative graphical user interface to manage organizational and group-based consent rules. The system should make sure that the consent rules are not physically deleted for audit purposes.

While working on this project, I had an opportunity to participate in all the phases of the software development process, which includes planning, implementation, testing, documenting, deploying, and maintaining a complex software application. This experience improved my skills in software engineering, data base design, and object-oriented programming. It also improved my style of coding to develop modules with low coupling and high cohesion.

REFERENCES

1. System Analysis, newton.uor.edu/courses/sysanades/pdf/anaintro.pdf
2. Eriksson, Hans-Eric; Penker, Magnus; Lyons, Brian; Fado, David. (2004). UML 2 Toolkit. Indianapolis, Indiana: Wiley Publishing Inc.
3. Dustin, E. (2002). Effective Software Testing. Boston, MA: Pearson Education.
4. http://www.visual-paradigm.com/product/vpuml/provides/structuralmodeling.jsp
5. Eriksson, Hans-Eric; Penker, Magnus; Lyons, Brian; Fado, David. (2004). UML 2 Toolkit. Indianapolis, Indiana: Wiley Publishing Inc
6. Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. IEEE Computer Society
7. Dustin, E. (2002). Effective Software Testing. Boston, MA: Pearson Education.
8. Unit Testing, Microsoft, http://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx
9. Integration Testing, Microsoft, http://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx
10. Li, E. Y. (1990). Software Testing In A System Development. Journal of Systems Management, 23-31.
11. MPI Requirements Document
12. Hibernate http://www.hibernate.org/
13. Martin Fowler Enterprise pattern Object Gateway http://martinfowler.com/eaaCatalog/rowDataGateway.html
14. A Brief overview to XACML https://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html

15. Clyde, S. Presentation Slides, Public Health Master Patient Index Project. Utah Department of Health, Mar 2012.
16. XACML 3.0 schema definition http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd
17. Wikipedia reference, http://en.wikipedia.org/wiki/Consent_management
18. Overview on why to use UML? http://www.cragsystems.co.uk/why_use_uml.htm
19. Dale G. O'Brien, William A. Yasnoff (1999). Privacy, confidentiality, and security in information systems of state health agencies: American Journal of Preventive Medicine, Volume 16, Issue 4
20. Roger Pressman, Software Engineering – a practitioner's approach
21. Abhishek A, phMPI Publication-Subscription Subsystem (PSS)
22. Sarvesh Jain, Automated Resolution Service For Public-Health Master Person Index.
23. DWH Schemas, http://www.dwhworld.com/dwh-schemas/
24. R. T. Yeh (1982) "Requirements Analysis- A Management Perspective," Proc. COMPSAC '82, Nov. 1982

25.

Appendix A

Consent Rule XML Formats

As discussed in Chapter 1, Data Source API is designed to support multiple message formats. Both XML and XACML message formats are supported by the Data Source API. The CMS is designed to add support for additional message formats at ease. This is done by adding a new class that implements the XMLParserStrategy interface. This chapter discusses the implemented message formats (Simple XML and XACML) in detail.

Simple XML

Below is the xsd to validate for simple xml format of consent rule to be passed as parameter for data source API.

XSD 2. XSD TO VALIDATE CONSENT RULE IN SIMPLE XML FORMAT

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.mpi.org/simpleXML"
       xmlns="http://www.mpi.org/simpleXML">
  <xsd:element name="ConsentRule">
    <xsd:complexType>
<xsd:sequence>
    <xsd:element ref="Id" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="Action" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="ExternalSystemPersonId" minOccurs="0"    maxOccurs="1"/>
<xsd:element ref="DataChunkType" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="UseType" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="FromSystem" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="ToSystem" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="MinQualityLevel" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="MaxQualityLevel" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="StartDate" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="EndDate" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="VerifiedBy" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="VerifiedDate" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="Precedence" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="ConsentRuleDocument" minOccurs="0" maxOccurs="unbounded"
        type="ConsentRuleDocumentType" />
    </xsd:sequence>
    </xsd:complexType>
```

```
  </xsd:element>
  <xsd:element name="Id" type="xsd:integer"/>
    <xsd:element name="Action">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="A"/>
                <xsd:enumeration value="D"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="ExternalSystemPersonId">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="DataChunkType" nillable="true">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="UseType">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="N"/>
                <xsd:enumeration value="C"/>
                <xsd:enumeration value="E"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="FromSystem">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="ToSystem">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="MinQualityLevel" type="xsd:double"/>
    <xsd:element name="MaxQualityLevel" type="xsd:double"/>
    <xsd:element name="StartDate" type="xsd:dateTime"/>
    <xsd:element name="EndDate" type="xsd:dateTime"/>
    <xsd:element name="VerifiedBy">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="VerifiedDate" type="xsd:dateTime"/>
    <xsd:element name="Precedence" type="xsd:integer"/>
    <xsd:complexType name="ConsentRuleDocumentType">
        <xsd:sequence>
      <xsd:element minOccurs="0" maxOccurs="1" name="DocumentId" type="xsd:string" />
      <xsd:element minOccurs="0" maxOccurs="1" name="Document" type="xsd:base64Binary"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Below is the xsd to validate for simple xml format of consent rule list which is passed as parameter for web portal api.

XSD 3. XSD TO VALIDATE CONSENT RULE LIST IN SIMPLE XML FORMAT

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema attributeFormDefault="unqualified"
            elementFormDefault="qualified"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.mpi.org/simpleXML"
            xmlns="http://www.mpi.org/simpleXML">
  <xsd:include schemaLocation="ConsentRule-SimpleXML.xsd"/>
  <xsd:element name="ConsentRules">
    <xsd:complexType>
      <xsd:sequence>
      <xsd:element minOccurs="1" maxOccurs="unbounded" ref="ConsentRule" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Insert Request

Below is an example for inserting Simple XML individual consent rule with document attached in the same xml.

XML 1. SAMPLE SIMPLE XML FOR INSERTING AN INDIVIDUAL CONSENT RULE

```
<ConsentRule>
  <Action>D</Action>
  <ExternalSystemPersonId>2000 1235</ExternalSystemPersonId>
  <DataChunkType>Address, PersonName</DataChunkType>
  <UseType>C</UseType>
  <ToSystem>UDOH-VS</ToSystem>
  <MinQualityLevel>2.3</MinQualityLevel>
  <MaxQualityLevel>4.5</MaxQualityLevel>
  <StartDate>2012-10-10T00:00:00</StartDate>
  <EndDate>2014-10-10T23:59:59</EndDate>
  <VerifiedDate>2012-10-02T11:23:32</VerifiedDate>
  <Precedence>2</Precedence>
  <ConsentRuleDocument>
    <Document>UTF 8 encoded string</Document>
  </ConsentRuleDocument>
</ConsentRule>
```

Below is an example of Simple XML multiple individual consent rules within the same xml.

XML 2. SAMPLE SIMPLE XML FOR INSERTING INDIVIDUAL CONSENT RULES

```
<ConsentRules>
  <ConsentRule>
    <Action>D</Action>
    <ExternalSystemPersonId>100</ExternalSystemPersonId>
    <DataChunkType>Address</DataChunkType>
    <UseType>N</UseType>
  </ConsentRule>
  <ConsentRule>
    <Action>A</Action>
    <ExternalSystemPersonId>102</ExternalSystemPersonId>
    <DataChunkType>GenderInfo</DataChunkType>
    <UseType>N</UseType>
  </ConsentRule>
  <ConsentRule>
    <Action>D</Action>
    <ExternalSystemPersonId>104</ExternalSystemPersonId>
    <DataChunkType>PersonRace</DataChunkType>
    <UseType>N</UseType>
  </ConsentRule>
  <ConsentRule>
    <Action>A</Action>
    <ExternalSystemPersonId>106</ExternalSystemPersonId>
    <UseType>N</UseType>
    <FromSystem>UU</FromSystem>
  </ConsentRule>
  <ConsentRule>
    <Action>D</Action>
    <ExternalSystemPersonId>100</ExternalSystemPersonId>
    <DataChunkType>Address</DataChunkType>
    <UseType>N</UseType>
    <FromSystem>UU</FromSystem>
  </ConsentRule>
  <ConsentRule>
    <Action>D</Action>
    <ExternalSystemPersonId>108</ExternalSystemPersonId>
    <DataChunkType>Address, GenderInfo</DataChunkType>
    <UseType>N</UseType>
  </ConsentRule>
</ConsentRules>
```

The reply for an insert operation would be a simple xml containing success tag or error tag with error message.

A success message reply would be:

XML 3. SUCCESS MESSAGE

```
<Response>
  <Success/>
</Response>
```

A failed insert request would get a response

XML 4. RESPONSE IN CASE OF AN ERROR

```
<Response>
    <Error>Invalid Date format.</Error>
</Response>
```

Lookup Request

A sample consent rule lookup request would be:

XML 5. SAMPLE SIMPLE XML FOR LOOKUP CONSENT RULES

```
<ConsentRule>
  <ExternalSystemPersonId>104</ExternalSystemPersonId>
</ConsentRule>
```

For a consent rule lookup, the data consumer has to send only the externalSystemPersonId in the request. All the consent rules associated with the person is returned back to the data consumer without documents attached in the reply. For the above lookup request, the reply would be something like:

XML 6. SAMPLE SIMPLE XML FOR LOOKUP CONSENT RULES

```
<ConsentRules>
    <ConsentRule>
            <Id>4</Id>
            <Action>D</Action>
            <ExternalSystemPersonId>104</ExternalSystemPersonId>
            <DataChunkType>PersonRace</DataChunkType>
            <UseType>N</UseType>
            <ConsentRuleDocument>
                    <DocumentId>4 4</DocumentId>
                    </ConsentRuleDocument>
                    <ConsentRuleDocument>
                    <DocumentId>5 4</DocumentId>
            </ConsentRuleDocument>
    </ConsentRule>
</ConsentRules>
```

The reply consists of a single consent rule with reference to two documents. The first document is with document Id “4 4,” which is a concatenation of the primary key of

the ConsentRuleDocument and primary key of the ConsentRule separated by a space. This concatenation of the consent rule id with the consent rule document id helps in identifying the correct document for a correct consent rule.

To look up a consent rule document, the data consumer has to send an XML that looks like sample XML 7 and the response would look like sample XML 8.

XML 7. SAMPLE LOOKUP CONSENT RULE DOCUMENT IN SIMPLE XML

```
<ConsentRule>
    <ConsentRuleDocument>
            <DocumentId>4 4</DocumentId>
    </ConsentRuleDocument>
</ConsentRule>
```

XML 8. RESPONSE FOR CONSENT RULE DOCUMENT LOOKUP IN SIMPLE XML

```
<ConsentRule>
<ConsentRuleDocument>
    <DocumentId>4 4</DocumentId>
<Document> UTF-8 encoded string</Document>
</ConsentRuleDocument>
</ConsentRule>
```

Update Request

To update a consent rule, the request should contain the entire consent rule information with the id of the consent rule that is sent during the lookup operation. An example of an update request is below in XML 9.

XML 9. SAMPLE UPDATE CONSENT RULE REQUEST IN SIMPLE XML

```
<ConsentRule>
  <Id>1</Id>
  <Action>D</Action>
  <ExternalSystemPersonId>2000 1235</ExternalSystemPersonId>
  <DataChunkType>Address</DataChunkType>
  <UseType>C</UseType>
  <ToSystem>UDOH-VS</ToSystem>
  <MinQualityLevel>5.3</MinQualityLevel>
  <MaxQualityLevel>4.5</MaxQualityLevel>
  <StartDate>2012-10-10T00:00:00.0000000</StartDate>
  <EndDate>2014-10-10T00:00:00.0000000</EndDate>
  <VerifiedDate>2012-10-02T11:23:32.0000000</VerifiedDate>
  <Precedence>2</Precedence>
</ConsentRule>
```

The <Id>1</Id> is the id that is send to data consumer for lookup operation. The response for the update operation is same as that of insert request. To update multiple consent rules the request should be encoded in <ConsentRules> tag.

Delete Request

An example request for deleting a consent rule is shown in XML 10.

XML 10. SAMPLE DELETE CONSENT RULE REQUEST IN SIMPLE XML

```
<ConsentRule>
    <Id>2</Id>
</ConsentRule>
```

To delete multiple consent rules, the consent rule delete requests should be enclosed in a <ConsentRules> tag. The reply for a delete request is the same as that of an insert request.

XACML

XACML stands for eXtensible Access Control Markup Language. “It is an OASIS (Organization for the Advancement of Structured Information Standards) standard that describes both a policy language and an access control decision request/response language” (both written in XAML) [14].

The phCMS only accepts XACML 3.0 [16] format for consent rules. Below is a sample XACML consent rule request. Each consent rule to be inserted/updated/deleted should be placed inside a <Rule> tag. All the attributes required for the operations are to be placed inside a <Match> tag of the xacml. Each attribute goes into a different match tag. The attributes required to populate, lookup, or delete the consent documents that are to be provided in the subsequent <AllOf> tag. The value of any given attribute of the consent

rule should be enclosed by `<AttributeValue>` and the attribute to which the value corresponds should be mentioned in the `AttributeId` of `<AttributeDesignator>`. The values to be populated for the insert/lookup/update/delete request are the same as that of Simple XML. The return values/types are also the same as that of Simple XML. In the case for any operation where the PolicyId, RuleId, and Effect are not required, default values have to be provided. A sample XACML to insert a consent rule is shown in sample XML 11.

XML 11. SAMPLE INSERT CONSENT RULE REQUEST IN XACML

```
<PolicySet>
  <Target/>
  <Policy PolicyId="00"
      Version="1.0"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Target/>
    <Rule RuleId="101" Effect="Deny">
      <Target>
        <AnyOf>
          <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">2000
1235</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="ExternalSystemPersonId"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Address,
PersonName</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="DataChunkType"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">N</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="UseType"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">UDOH-
VS</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="FromSystem"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </Match>
```

```
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">IHC</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="ToSystem"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">2.3</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="MinQualityLevel"
                DataType="http://www.w3.org/2001/XMLSchema#double"/>
            </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">9.3</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="MaxQualityLevel"
                DataType="http://www.w3.org/2001/XMLSchema#double"/>
            </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">2012-10-
10</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="StartDate"
                DataType="http://www.w3.org/2001/XMLSchema#dateTime"/>
            </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">2012-10-
10</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="EndDate"
                DataType="http://www.w3.org/2001/XMLSchema#dateTime"/>
            </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">2</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="Precedence"
                DataType="http://www.w3.org/2001/XMLSchema#integer"/>
            </Match>
          </AllOf>
          <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">UTF-8 encoded
string</AttributeValue>
              <AttributeDesignator
                MustBePresent="true"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:resource"
                AttributeId="Document"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </Match>
          </AllOf>
        </AnyOf>
      </Target>
    </Rule>
```

```
  </Policy>
</PolicySet>
```

Appendix B

Sample Individual Consents Rules

External systems or Data Consumers can insert, update, or delete only Individual Consent Rules. This section gives individual consent rule examples for various situations.

The main properties of the consent rule, which defines the applicability of the rule for any data chunk, are action, dataChunkType, useType, from and to System, Quality levels, start date, and end date. The use of these properties are explained in the following eight scenarios.

Scenario 1: Hide an entire person for all data consumers, for normal usage effective 1/1/2012 is shown in Table 12.

Table 12. Sample Individual Consent Rule to hide all person information for all data consumers, for normal usage effective from 1/1/2012

Column Name	Column Value
id	1
action	D
externalSystemPersonId	1234
mpiSetId	NULL
dataChunkType	NULL
useType	N
fromSystem	NULL
toSystem	NULL
minQualityLevel	NULL
maxQualityLevel	NULL
startDate	1/1/2012
endDate	NULL
consentSubmittedBy	UDOH-VS
verifiedBy	NULL
verifiedDate	NULL

precedence	4

Scenario 2: Hide data of a specific type for all data consumers, for normal usage up to 9/8/2013 is shown in Table 13.

Table 13. Sample Individual Consent Rule to hide specific type of data for all data consumers, for normal usage till 9/8/2012

Column Name	Column Value
id	4
action	D
externalSystemPersonId	1012
mpiSetId	NULL
dataChunkType	PERSONRACE
useType	N
fromSystem	NULL
toSystem	NULL
minQualityLevel	NULL
maxQualityLevel	NULL
startDate	NULL
endDate	09/08/2013
consentSubmittedBy	IHC
verifiedBy	NULL
verifiedDate	NULL
precedence	7

Scenario 3: Hide data from a specific source, for normal usage is shown in Table 14.

Table 14. Sample Individual Consent Rule to hide data from a specific data source, for normal usage belonging to a specific patient

Column Name	Column Value
id	5
action	D
externalSystemPersonId	1752
mpiSetId	NULL
dataChunkType	NULL
useType	N

fromSystem	UDOH-VS
toSystem	NULL
minQualityLevel	NULL
maxQualityLevel	NULL
startDate	NULL
endDate	NULL
consentSubmittedBy	UDOH-VS
verifiedBy	NULL
verifiedDate	NULL
precedence	6

Scenario 4: Prohibit a specific data consumer from seeing any data about a patient, for normal usage is shown in Table 15.

Table 15. Sample Individual Consent Rule to prohibit a specific data consumer to lookup any data belonging to a patient for normal usage

Column Name	Column Value
id	8
action	D
externalSystemPersonId	1030
mpiSetId	NULL
dataChunkType	NULL
useType	N
fromSystem	NULL
toSystem	UNIH
minQualityLevel	NULL
maxQualityLevel	NULL
startDate	NULL
endDate	NULL
consentSubmittedBy	UNIH
verifiedBy	NULL
verifiedDate	NULL
precedence	1

Scenario 5: Prohibit low-quality data sharing for a specific patient, for normal usage is shown in Table 16.

Table 16.Sample Individual Consent Rule to deny information up to quality level factor 3.5 relating to a particular patient for normal use

Column Name	Column Value
id	20
action	D
externalSystemPersonId	1228
mpiSetId	NULL
dataChunkType	NULL
useType	N
fromSystem	NULL
toSystem	NULL
minQualityLevel	NULL
maxQualityLevel	3.5
startDate	NULL
endDate	NULL
consentSubmittedBy	IHC
verifiedBy	NULL
verifiedDate	NULL
precedence	9

Scenario 6: Prohibit data sharing except for emergency use for a specific patient is shown in Table 17.

Table 17. Sample Individual Consent Rule to sharing data except for emergency use of a particular patient for normal use

Column Name	Column Value
id	11
action	A
externalSystemPersonId	1362
mpiSetId	NULL
dataChunkType	NULL
useType	E
fromSystem	NULL
toSystem	NULL
minQualityLevel	NULL

maxQualityLevel	NULL
startDate	NULL
endDate	NULL
consentSubmittedBy	UDOH-VS
verifiedBy	NULL
verifiedDate	NULL
precedence	9

Scenario 7: Prohibit data sharing for a specific date range, for normal usage is shown in Table 18.

Table 18. Sample Individual Consent Rule to deny information up to quality level factor 3.5 relating to a particular patient for normal use

Column Name	Column Value
id	1
action	D
externalSystemPersonId	1321
mpiSetId	NULL
dataChunkType	NULL
useType	N
fromSystem	NULL
toSystem	NULL
minQualityLevel	NULL
maxQualityLevel	NULL
startDate	1/1/2012
endDate	12/31/2012
consentSubmittedBy	IHC
verifiedBy	NULL
verifiedDate	NULL
precedence	3

Scenario 8: Prohibit data sharing for date range to a Data Consumer, for normal usage is shown in Table 19.

Table 19. Sample Individual Consent Rule to deny all information related to a patient when requested by a specific data consumer for a specific date range for normal usage

Column Name	Column Value
id	15
action	D
externalSystemPersonId	1521
mpiSetId	NULL
dataChunkType	NULL
useType	N
fromSystem	NULL
toSystem	UPDB
minQualityLevel	NULL
maxQualityLevel	NULL
startDate	1/1/2012
endDate	12/31/2012
consentSubmittedBy	UPDB
verifiedBy	NULL
verifiedDate	NULL
precedence	1

Appendix C

Implementation of *Create Read Update Delete* (CRUD) operations on ConsentRules

As mentioned before, insert, update, lookup, or delete operations on consent rules or consent rule documents are sent to the phMPI by data sources using the Data Source API. The Data Source API uses the methods provided in the consent management service to perform the requests and then return back the response. The responses for various operations are discussed in detail in Chapter 5. Extensive testing has been done to verify the successful implementation of the methods. A system test is conducted with dummy data, and a regression test is performed from time to time when a new component is integrated into the system.

Add Single Consent Rule

Figure 15 shows the sequence diagram for implementing the addition of a single consent rule. The data source sends an add request to the ConsentManagement Portal using the Data Source API. The AddConsentRule method gets the request queue from the ServiceRegistry and calls the AddConsentRule with the parameters passed by the data source. The request queue uses the consent management service to perform the addition of a new consent rule. A success message is returned upon the successful addition of the consent rules. In case of a failure, an error message with the appropriate error is returned back. The success and error message formats returned for all the actions are discussed in Appendix B.

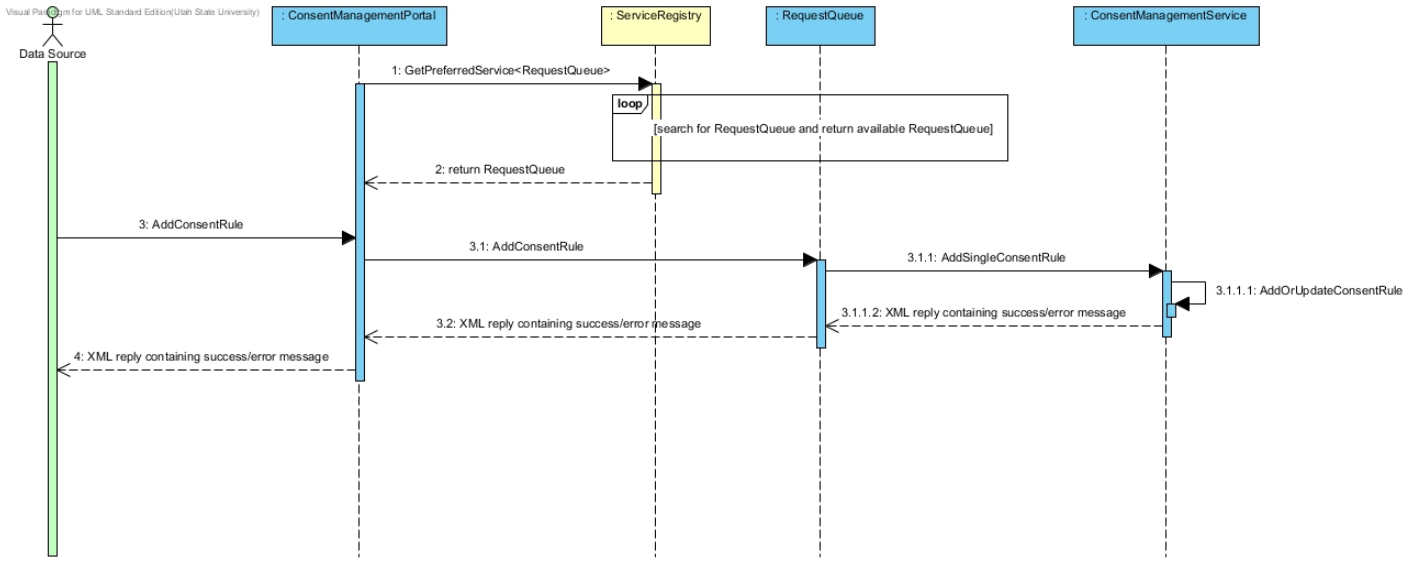


Figure 15. Sequence diagram for adding a single consent rule

Add Multiple Consent Rules

Figure 16 shows the sequence diagram for implementing an addition of a multiple consent rules. The data source sends an add request to the ConsentManagement Portal using the Data Source API. The parameter passed consists of either a Simple XML or a XACML containing multiple consent rules in the same string parameter. The AddMultipleConsentRules method gets the request queue from the ServiceRegistry and calls the AddMultipleConsentRules with the parameters passed by the data source. The request queue uses the consent management service to perform the addition of multiple consent rules. Upon the successful addition of all the consent rules, a success message is returned. Either all or none of the consent rules are saved during this action. The transaction fails even if the service fails to add a single consent rule from the given set of rules.

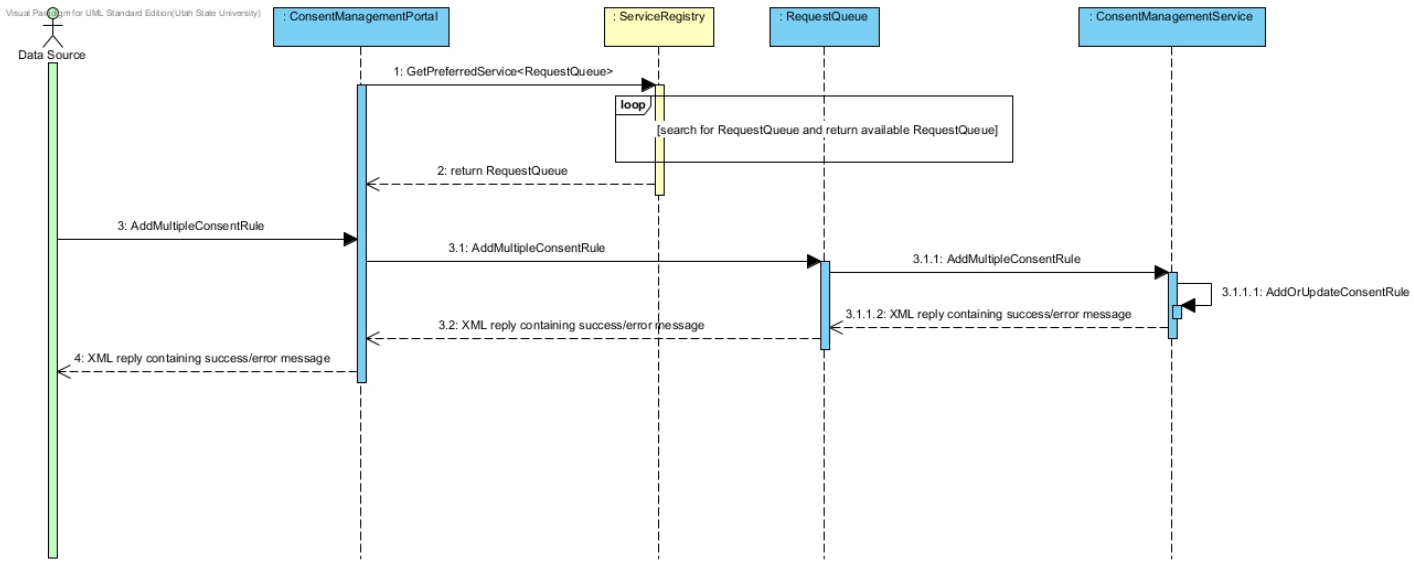


Figure 16. Sequence diagram for adding multiple consent rules

Lookup Consent Rules

The externalSystemPersonId is used to identify the consent rules applicable for an individual. The reason being that upon insert/update/merge/split operations performed on a person's data in MPI, there are chances that the MPI ID used to identify an individual changes. Only the externalSystemPersonId does not change irrespective of the operations performed on an individual's data. Figure 17 shows the sequence diagram for lookup of consent rules for an individual. The External Systems (data sources) send their externalSystemPersonId with which they uniquely identify their patients/individuals. Depending on the id, all the Individual, Organizational, and MPI Set consent rules are returned back in either Simple XML or XACML format. The return value contains the document id in the form of `<DocumentId>5 4</DocumentId>`. Here the numeric `5` is the Consent Rule ID and `4` is the document id of the linked document.

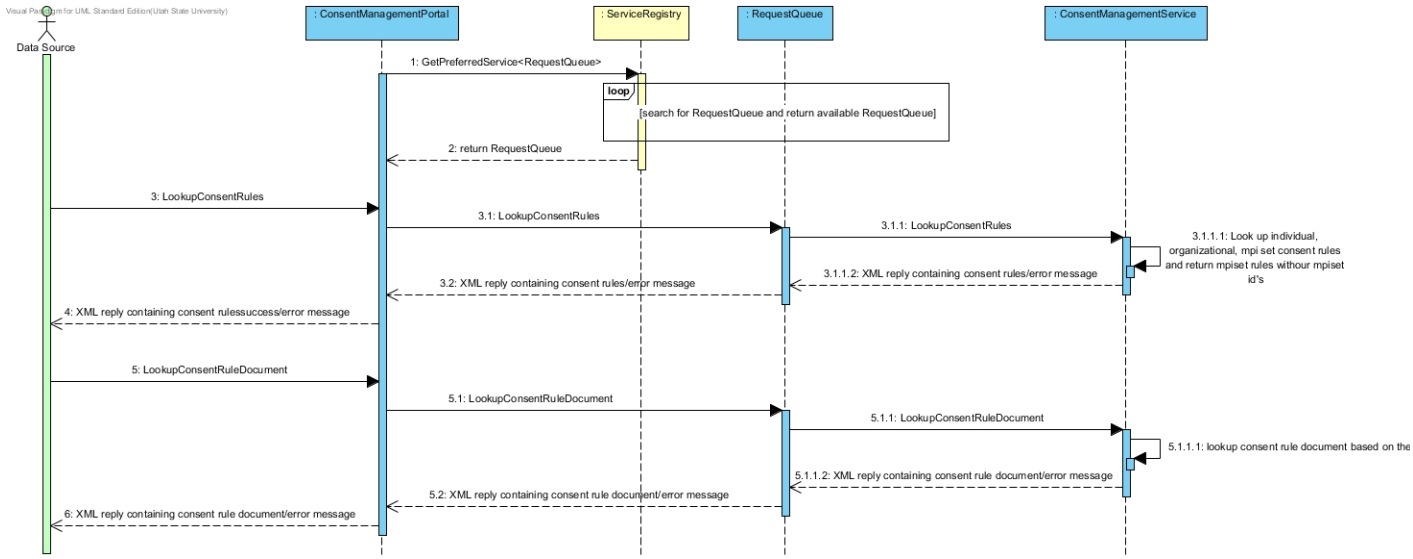


Figure 17. Sequence diagram for consent rule lookup

To lookup a consent rule document, the data source has to send back the `<DocumentId>5 4</DocumentId>` value. The lookup for the consent rule document operation then returns the requested consent rule document(s) in the requested format. The steps to looking up the consent rule documents are shown in the second action of Figure 17.

Update Consent Rule

To update a consent rule, the data source has to know the consent rule id, and a lookup operation for the consent rules should be performed to get the required consent rule id. Figure 18 shows the sequence diagram for updating a consent rule. The ConsentManagementService uses the same method “AddorUpdateConsentRule” used in addition to the consent rules to update an existing consent rule. The return message is explained in detail in Appendix B.

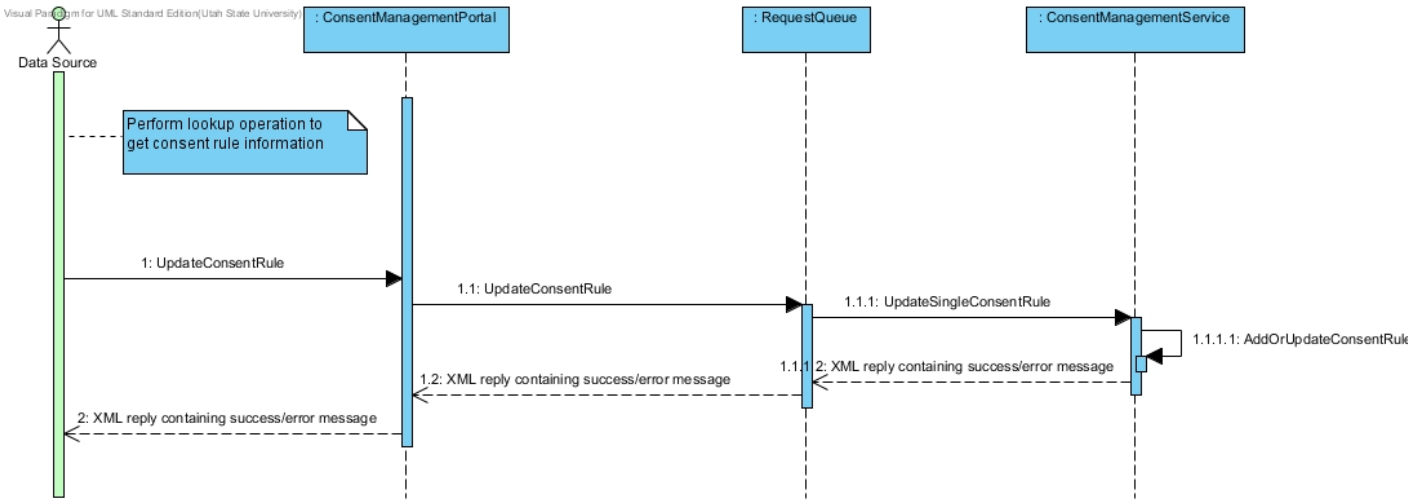


Figure 18. Sequence diagram for updating a single consent rule

Update Multiple Consent Rules

Updating multiple consent rules is done in transaction. Either all or none of the consent rules are updated. Figure 19 shows the sequence diagram for updating multiple consent rules. As described earlier the data source should first lookup for consent rules. The Data source then sends the consent rules along with their consent rule ids to update. Upon completion of the update request, the Data Source API returns either a success or failure message. The same AddorUpdateConsentRule method is called internally in the ConsentManagementService to update multiple consent rules.

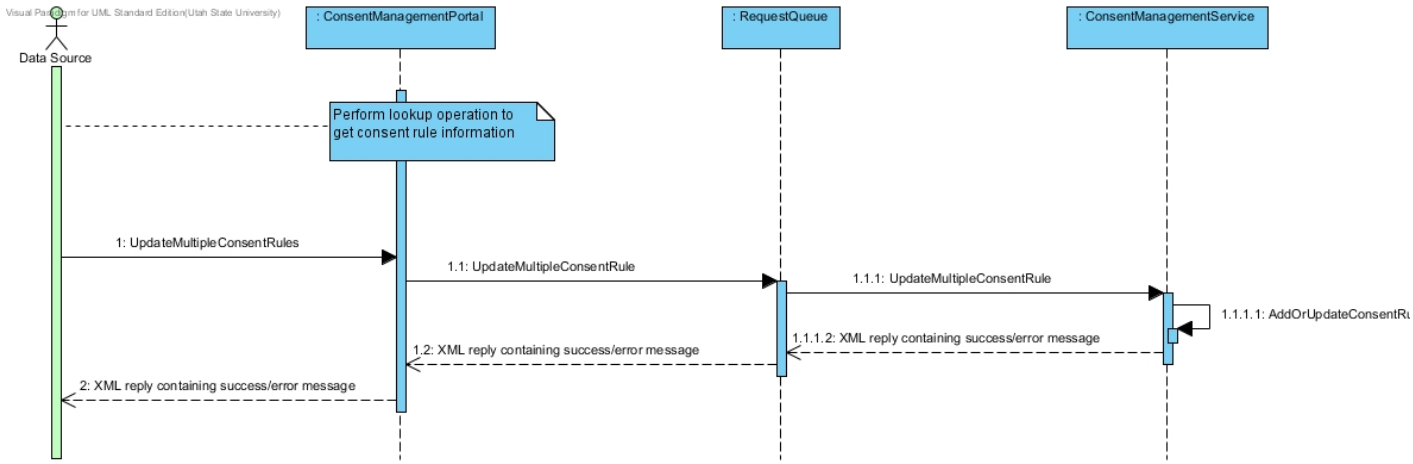


Figure 19. Sequence diagram for updating multiple consent rules

Update Consent Rule Document(s) belonging to an individual.

Updating a consent rule document(s) signed by an individual is done by calling the same method UpdateCosentRuleDocument of the ConsentManagementPortal. Either all or none of the documents are updated in case multiple documents are sent in an update request. Figure 20 explains the sequence of steps to be followed in updating the consent rule documents belonging to an individual. To update a consent rule document, first the document id and the consent rule to which it belongs should be known. Lookup operation returns a set of consent rule documents linked to a consent rule.

Delete Consent Rule

The first step in deleting a consent rule is to lookup for a consent rule. Figure 21 explains the steps in the deletion of the consent rule by a data source. When a consent rule is deleted, the consent rule document(s) related to the consent rule is also deleted.

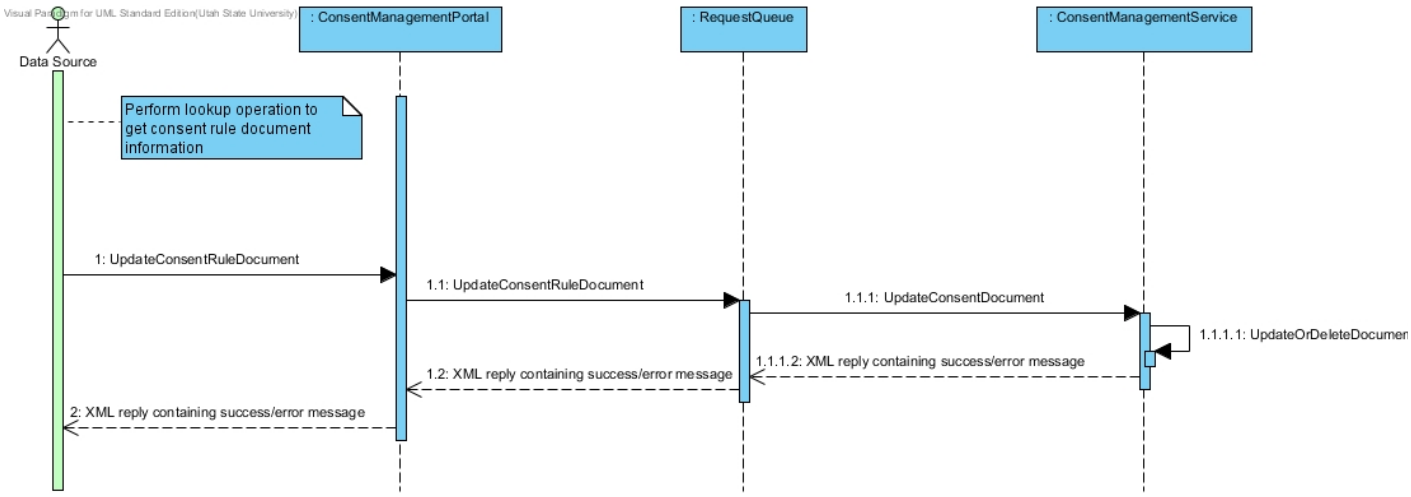


Figure 20. Sequence diagram for updating consent rule document(s)

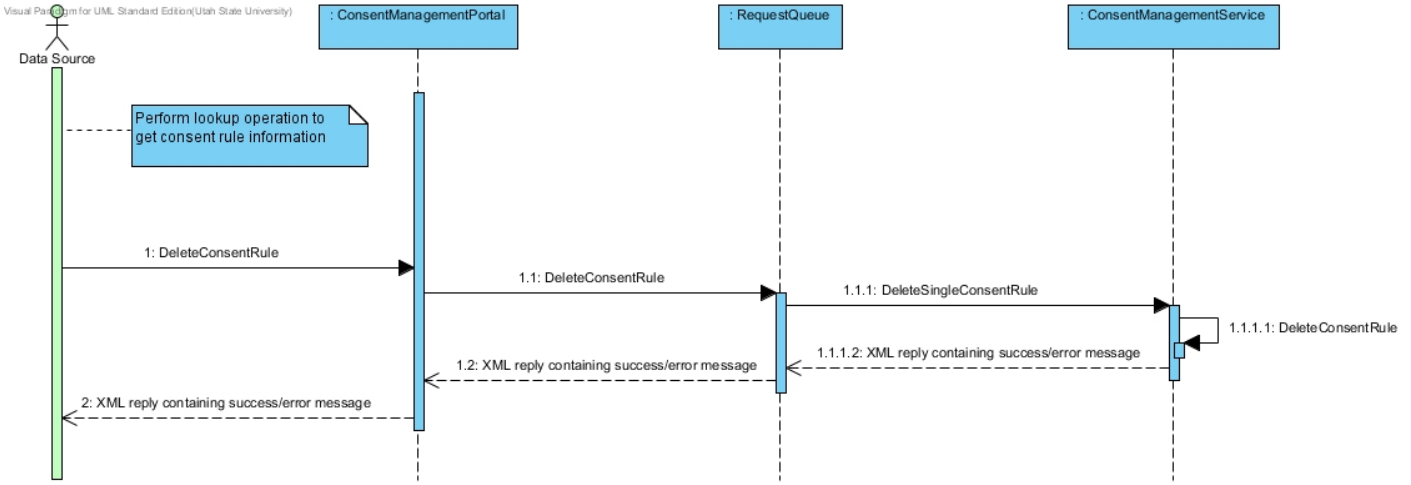
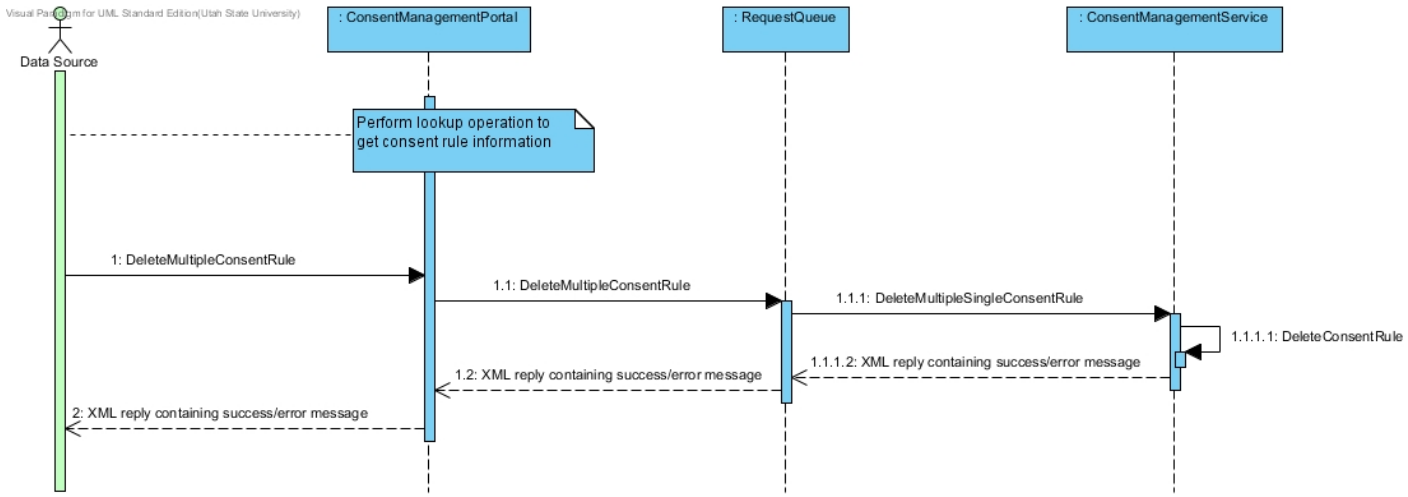


Figure 21. Sequence diagram for deleting a consent rule

Delete Multiple Consent Rules

Similar to updating multiple consent rules, deleting multiple consent rules will delete either all of the requested consent rules or none in case of an error during the operation. The initial lookup has to be performed to get a list of the consent rules that are to be deleted. The consent rule documents linked to the consent rules are automatically deleted. Figure 22 shows the sequence diagram for deleting multiple consent rules. The

same internal method DeleteConsentRule is called for deleting single as well as multiple consent rules.



Delete Consent Rule Document

A single method is provided in the Data Source API to delete single/multiple consent rule documents related to an individual or patient. Figure 23 explains the steps in deleting consent rule document(s). The data source has to look up for consent rules to get the consent rule document(s) associated with the consent rule.

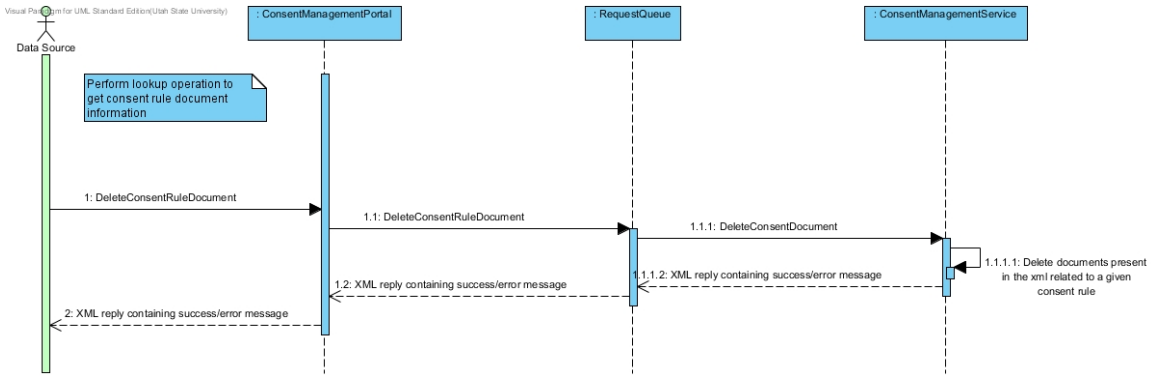


Figure 23. Sequence diagram for deleting consent rule document

Appendix D

Data Source API

Data sources can talk to phMPI only through the web portal. The web portal of the phMPI has Portal, DataConsumerPublishAPI, and ConsentManagementPortal. The portal acts as an API for performing insert, update, lookup, merge and split operations on patient information. The DataConsumerPublishAPI defines the methods that data consumers should implement to receive updates on patient information. This section describes the web methods present in the ConsentManagementPortal of a web portal.

There are three parameters for any operation in a consent management service. The first parameter is the request, which has the consent rule information in either SimpleXML or XACML format. The second parameter is the data source that is requesting the operation. This value is stored in the consent rule while saving a new consent rule request. The third parameter defines that type of consent rule request. Its value is either “SimpleXML” or “XACML.” The formats for sending the consent rules and the return values for each operation are explained in detail in Chapter 5. The API allows the consent rule requests to be in XACML 3.0 [16] format only.

Table 20. ConsentManagementPortal web methods

Method Name	Return Type	Parameters(type string)
AddConsentRule	string	consentRuleRequest dataSource consentRuleType
AddMultipleConsentRules	string	consentRuleRequest dataSource consentRuleType
LookupConsentRules	string	consentRuleRequest dataSource consentRuleType
UpdateConsentRule	string	consentRuleRequest dataSource consentRuleType
UpdateMultipleConsentRules	string	consentRuleRequest dataSource consentRuleType
UpdateConsentRuleDocument	string	consentRuleRequest dataSource consentRuleType
DeleteConsentRule	string	consentRuleRequest dataSource consentRuleType
DeleteMultipleConsentRules	string	consentRuleRequest dataSource consentRuleType
DeleteConsentRuleDocument	string	consentRuleRequest dataSource consentRuleType