



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**ANALYSIS AND DESIGN OF A DISTRIBUTED SYSTEM
FOR MANAGEMENT AND DISTRIBUTION OF
NATURAL LANGUAGE ASSERTIONS**

by

Javier Palomo

September 2010

Thesis Co-Advisors:

Man-Tak Shing
Bret Michael

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2010	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE Analysis and Design of a Distributed System for Management and Distribution of Natural Language Assertions			5. FUNDING NUMBERS	
6. AUTHOR(S) Javier Palomo				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N.A.____.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) This research entails the design and development of an automated system that allows researchers working remotely to store, manage, and transfer assertion data to an external system run by the Intelligence Advanced Research Projects Activity. The research stems from the University of Maryland's involvement in the Social-Cultural Content in Language program which seeks to investigate methodologies, designs, and technologies that can contribute in the understanding of the social goals of persons or groups of people by demonstrating a relationship between these goals and their particular language use. In this research we interview the stakeholders to determine the software requirements of the system. After a careful analysis of the requirements we used the Unified Modeling Language notation to provide the reader a visual model of the software design. Finally, we develop a working prototype of the proposed system consisting of two Web services and a Web service client written in the Java programming language.				
14. SUBJECT TERMS SOA, Web services, Assertions, Knowledge Base			15. NUMBER OF PAGES 159	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**ANALYSIS AND DESIGN OF A DISTRIBUTED SYSTEM FOR MANAGEMENT AND
DISTRIBUTION OF NATURAL LANGUAGE ASSERTIONS**

Javier Palomo
Major, United States Marine Corps
B.A., California State University–Fullerton, 2000

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2010**

Author: Javier Palomo

Approved by: Man-Tak Shing
Thesis Co-Advisor

Bret Michael
Thesis Co-Advisor

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This research entails the design and development of an automated system that allows researchers working remotely to store, manage, and transfer assertion data to an external system run by the Intelligence Advanced Research Projects Activity. The research stems from the University of Maryland's involvement in the Social-Cultural Content in Language program, which seeks to investigate methodologies, designs, and technologies that can contribute in the understanding of the social goals of persons or groups of people by demonstrating a relationship between these goals and their particular language use.

In this research, we interview the stakeholders to determine the software requirements of the system. After a careful analysis of the requirements, we used the Unified Modeling Language notation to provide the reader a visual model of the software design. Finally, we develop a working prototype of the proposed system consisting of two Web services and a Web service client written in the Java programming language.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	THE PROBLEM.....	1
B.	THE SOLUTION.....	2
C.	ORGANIZATION	3
II.	BACKGROUND	5
A.	SOCIAL CONTENT IN LANGUAGE.....	5
1.	Introduction.....	5
2.	SCIL Architecture	6
3.	Assertions.....	8
a.	<i>Claim.....</i>	<i>8</i>
b.	<i>Evidence</i>	<i>8</i>
c.	<i>Support</i>	<i>8</i>
B.	SERVICE-ORIENTED ARCHITECTURE	9
1.	SOA Introduction.....	9
2.	Principles of SOA.....	11
a.	<i>Services Are Reusable.....</i>	<i>11</i>
b.	<i>Services Share a Formal Contract</i>	<i>11</i>
c.	<i>Services Are Loosely Coupled.....</i>	<i>11</i>
d.	<i>Services Abstract Away Underlying Logic</i>	<i>11</i>
e.	<i>Services Are Composable.....</i>	<i>12</i>
f.	<i>Services Are Autonomous.....</i>	<i>12</i>
g.	<i>Services Are Stateless.....</i>	<i>12</i>
h.	<i>Services Are Discoverable.....</i>	<i>13</i>
3.	Benefits of Using SOA	13
C.	WEB SERVICES	14
1.	RESTful Web Services	14
2.	SOAP-Based Web Services	15
3.	Web Service Components.....	15
a.	<i>Services.....</i>	<i>16</i>
b.	<i>Description</i>	<i>16</i>
c.	<i>XML Schema (XSD)</i>	<i>19</i>
d.	<i>Messaging.....</i>	<i>21</i>
D.	SUMMARY	23
III.	SYSTEM REQUIREMENTS SPECIFICATION (SRS)	25
A.	OVERVIEW	25
1.	Purpose.....	25
2.	System Perspective.....	25
3.	System Features and Domain Model.....	25
4.	Intended Audience	27
B.	FUNCTIONAL REQUIREMENTS.....	27
1.	Local User Stores Assertion.....	29

2.	<i>Local User Transfers Assertion to Knowledge Base</i>	31
3.	<i>Local User Replaces Assertion in the External System</i>	33
4.	<i>Local User Edits Assertion in Local Databas</i>	36
5.	<i>External User Queries ExplanationGetWS</i>	39
6.	<i>External User Queries StatusReportWS</i>	41
7.	<i>Administrator Sets the Capability Status</i>	42
8.	<i>Administrator Modifies a User Account</i>	44
C.	OTHER FUNCTIONAL REQUIREMENTS	48
1.	<i>System Access</i>	48
2.	<i>Database</i>	49
3.	<i>Local System Functionality</i>	50
4.	<i>Services and Clients</i>	51
5.	<i>User Interface (UI)</i>	52
D.	NONFUNCTIONAL REQUIREMENTS	53
1.	<i>Usability</i>	53
2.	<i>Reliability</i>	54
3.	<i>Portability</i>	54
4.	<i>Supportability</i>	54
E.	SUMMARY	55
IV.	SYSTEM DESIGN	57
A.	DATABASE TIER	57
1.	ER Diagram	59
2.	Relational Database Schema	60
a.	<i>Mapping of Regular Entity Types</i>	60
b.	<i>Mapping of Weak Entity Types</i>	61
c.	<i>Mapping of Binary 1:1 Relationship Types</i>	61
d.	<i>Mapping of Binary 1:N Relationship Types</i>	62
e.	<i>Mapping of Binary M:N Relationship Types</i>	62
f.	<i>Mapping of Multivalued Attributes</i>	63
B.	BUSINESS LOGIC TIER	64
1.	XML Schema	64
a.	<i>StatusReportWS</i>	65
b.	<i>ExplanationGetWS</i>	66
c.	<i>KbUpdate Client</i>	70
2.	Class Diagrams	73
a.	<i>StatusReport</i>	74
b.	<i>ExplanationGet</i>	77
c.	<i>KbUpdate</i>	81
C.	WEB TIER	82
1.	UMD Services	83
2.	UMD Client	85
D.	PRESENTATION TIER	86
1.	UI Web Pages	87
a.	<i>Sign In</i>	88
b.	<i>Main Menu</i>	89

c.	<i>Store Assertion</i>	90
d.	<i>Transfer Assertion</i>	92
e.	<i>Edit Assertion Data</i>	94
e.	<i>Set Capability Status</i>	96
f.	<i>User Accounts</i>	97
2.	UI Class Diagrams.	99
E.	SUMMARY	106
V.	PROTOTYPE	107
A.	MYSQL DATABASE	107
1.	Status	108
2.	AssertionData	108
B.	UMD OPERATIONS	110
1.	StatusReport	111
a.	<i>Setting the Status</i>	111
b.	<i>The Web Service</i>	113
2.	KbUpdate client	114
a.	<i>Storing the assertion</i>	114
b.	<i>Updating the External Database</i>	116
3.	ExplanationGetWS	119
b.	<i>The Web Service</i>	119
C.	USER FEEDBACK	119
D.	SUMMARY	120
VI.	CONCLUSION	121
A.	SYNOPSIS	121
B.	FUTURE WORK	122
	APPENDIX. XML SCHEMA	125
	LIST OF REFERENCES	135
	INITIAL DISTRIBUTION LIST	137

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	SCIL architecture	7
Figure 2.	SOA architecture.....	10
Figure 3.	REST Web service request / response (From Spies, 2008)	14
Figure 4.	SOAP Web service request / response (From Spies, 2008).....	15
Figure 5.	WSDL, abstract interface description	18
Figure 6.	WSDL, concrete implementation description.....	19
Figure 7.	XSD I	20
Figure 8.	XSD II.....	21
Figure 9.	SOAP message.....	22
Figure 10.	Domain model.....	27
Figure 11.	Use case diagram	28
Figure 12.	Store assertion SSD.....	30
Figure 13.	User transfers assertion SSD.....	33
Figure 14.	User replaces assertion SSD	36
Figure 15.	Local user edits assertion SSD.....	38
Figure 16.	External user queries ExplanationGetWS SSD	40
Figure 17.	External user queries StatusReportWS SSD.....	42
Figure 18.	Administrator sets statuses SSD	44
Figure 19.	Modify user account (create) SSD.....	47
Figure 20.	Modify user account (edit & delete) SSD.....	48
Figure 21.	Four tiers of development	57
Figure 22.	Database ER diagram.....	60
Figure 23.	Mapping of regular entity types.....	61
Figure 24.	Mapping of weak entity types.....	61
Figure 25.	Mapping of binary 1:1 relationship types	62
Figure 26.	Mapping of binary 1:N relationship types	62
Figure 27.	Mapping of binary M:N relationship types.....	63
Figure 28.	Mapping of multivalued attributes.....	64
Figure 29.	XML schema StatusReportResponseMsgPart	65
Figure 30.	XML schema ServiceState and ServiceStatusReport	66
Figure 31.	XML schema ExplanationGetResponseMsgPart.....	67
Figure 32.	XML schema KbAssertion	68
Figure 33.	XML schema DataSource	68
Figure 34.	XML schema KbClaim	69
Figure 35.	XML schema KbEvidence.....	69
Figure 36.	XML schema KbSupport	70
Figure 37.	XML schema KbUpdateRequestMsgPart.....	71
Figure 38.	XML schema AssertionAddBundle	71
Figure 39.	XML schema AssertionDeleteBundle	72
Figure 40.	XML schema AssertionReplaceBundle.....	73
Figure 41.	StatusReport class diagram part I	75
Figure 42.	StatusReport class diagram part II	76

Figure 43.	ExplanationGet class diagram part I	77
Figure 44.	ExplanationGet class diagram part II.....	78
Figure 45.	ExplanationGet class diagram part III	79
Figure 46.	ExplanationGet class diagram part IV	80
Figure 47.	KbUpdate class diagram part I.....	81
Figure 48.	KbUpdate class diagram part II	82
Figure 49.	UMD services WSDL part I	83
Figure 50.	UMD services WSDL part II	84
Figure 51.	UMD services WSDL part III.....	84
Figure 52.	UMD services WSDL part IV.....	85
Figure 53.	UMD client WSDL	86
Figure 54.	Web architecture (From Booch, 2001)	87
Figure 55.	UI workflow.....	88
Figure 56.	Sign-in Web page.....	89
Figure 57.	Main menu Web page	90
Figure 58.	Store assertion I.....	91
Figure 59.	Store assertion II	91
Figure 60.	Store assertion III.....	92
Figure 61.	Transfer assertion I	93
Figure 62.	Transfer assertion II	94
Figure 63.	Edit assertion.....	95
Figure 64.	Edit assertion II.....	96
Figure 65.	Select the status.....	97
Figure 66.	User account I	98
Figure 67.	User account II.....	98
Figure 68.	Modify account	99
Figure 69.	Update account.....	99
Figure 70.	Main menu	101
Figure 71.	Store assertion.....	102
Figure 72.	Transfer assertion.....	103
Figure 73.	Transfer assertion II	103
Figure 74.	Edit assertion.....	104
Figure 75.	Set capability statuses	105
Figure 76.	User accounts.....	105
Figure 77.	MySQL status table.....	108
Figure 78.	MySQL assertionData table.....	109
Figure 79.	MySQL assertion-related tables.....	110
Figure 80.	Executing NPS_SCIL.jar	111
Figure 81.	Method call to write the statuses.....	112
Figure 82.	Connection to the database	112
Figure 83.	Status table after execution	112
Figure 84.	Testing StatusReportWS.....	113
Figure 85.	Deploying UMD	114
Figure 86.	Storing an assertion.....	115
Figure 87.	Assertion in the database	115

Figure 88.	External Web server.....	117
Figure 89.	External Web server.....	118
Figure 90.	External ID in the database	118
Figure 91.	Testing ExplanationGetWS	119

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Local user stores assertion	29
Table 2.	Local user transfers assertion to knowledge base	31
Table 3.	Local user replaces assertion	33
Table 4.	Local user edits assertion	36
Table 5.	External user queries ExplanationGetWS.....	39
Table 6.	External user queries StatusReportWS	41
Table 7.	Administrator sets capability statuses	42
Table 8.	Administrator modifies a user account	44
Table 9.	Stereotyped associations (From Conallen, 2003)	101

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

BAA	Broad Agency Announcement
CDM	Conceptual Data Model
DBMS	Database Management System
DMV	Department of Motor Vehicles
DTG	Date Time Group
ER	Entity Relationship
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IARPA	Intelligence Advanced Research Projects Activity
JAR	Java Archive
JDBC	Java Database Connectivity
LAN	Local Area Network
ODBC	Open Database Connectivity
SCIL	Social-Cultural Content in Language
UMD	University of Maryland
NPS	Naval Postgraduate School
RDS	Relational Database Schema
REST	Representational State Transfer
RPC	Remote Procedure Call
SOA	Service-Oriented Architecture
SQL	Structured Query Language
SRS	Software Requirements Specification
SSD	System Sequence Diagram
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAE	Web Application Extension

WS	Web Service
WSDL	Web Service Description Language
WWW	World Wide Web
XML	Extensive Markup Language
XSD	Extensive Markup Language Schema Definition

ACKNOWLEDGMENTS

Thanks to my loving wife, Elsa. Without her unwavering support and patience, I could not have accomplished this report. I am equally grateful to my kids, who have stood by me and made adjustments to their lives in order to accommodate my mission here at the Naval Postgraduate School.

I would also like to extend my sincere gratitude to professors Man-Tak Shing, Bret Michael, and Craig Martell. Gentlemen, thank you for your guidance and sound advice.

Finally, I would like to dedicate this report in memory of my friend and former colleague, U.S. Air Force Maj. Ken Bourland. Ken, a graduate of the Naval Postgraduate School, passed away in the Haiti earthquake of January 2010. Godspeed, my friend.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. THE PROBLEM

In December of 2008, the Intelligence Advanced Research Projects Activity (IARPA) released a Broad Agency Announcement (BAA) soliciting proposals for the Social-Cultural Content in Language (SCIL) Program. The intent behind the SCIL program is to investigate methodologies, designs, and technologies that can contribute in the understanding of the social goals of persons or groups of people by demonstrating a relationship between these goals and their particular language use (IARPA, 2008). By *language use*, we mean their “manner of speaking” as opposed to the language itself (e.g., Spanish, French). Anyone contributing to SCIL would be required to use Natural Language Processing techniques to provide information to Intelligence analysts so that they could advise high-level decision makers. Insight into the social dynamics of a group would allow analysts to better understand the strengths and weaknesses of a group, help identify the group’s goals and motivation, and to reduce Anglo-centric assumptions about their behavior (IARPA, 2008).

At a later date, the University of Maryland (UMD) responded to IARPA’s solicitation with a plan for research in identifying social goals pertaining to persuasion. UMD subsequently subcontracted University of California at Santa Cruz and the Naval Postgraduate School (NPS) to work on the project. Each institution involved will serve as a performer team that will interact and contribute to the program via an aggregate system based on a service-oriented architecture (SOA). The information generated by each performer team will be stored into a core knowledge repository for analysts to examine for future events.

The issue that needs to be addressed involves the design and development of an automated system that the Naval Postgraduate School (NPS) performer team will employ. The design of a system will describe how the software is to be constructed and is based on the requirements set forth by the users, in this case the stakeholders of the SCIL program. It is critical that a careful examination of these requirements be conducted and

validated by a UMD representative to subsequently provide our performer team with an effective automated system to execute their component in this program. The system to be developed must be capable of performing the following:

- Allow for local access for the collection, storage, and modification of the data to be forwarded.
- Allow for remote access to the data.
- Allow for the transfer of data to the external knowledge repository over the Internet.

B. THE SOLUTION

In order to address these system requirements, our research focuses on answering the following three questions:

1. What are the requirements for system to be implemented by the NPS performer team?
2. What is the appropriate design of a modular framework to effectively manage the natural language assertions in a knowledge base repository and the sharing of the knowledge via the World Wide Web?
3. What is the appropriate Web-service design to allow for multiple users to update the knowledge base repository of natural language assertions from multiple sites?

In this thesis, we addressed the first question by generating a software requirements specification that makes use of various use cases that reinforce our understanding of the system functionality. The specification was validated by a UMD stakeholder. After a careful review of the system requirements, we used an object-oriented analysis and design approach to address question number two. We developed a preliminary design of the system as a whole using various Unified Modeling Language (UML) notations. We also analyzed and developed a database schema required for the

local storage. Lastly, we implemented a prototype system using a Web Service Description Language file (WSDL) and an Extensive Markup Language Schema Definition (XSD).

C. ORGANIZATION

- Chapter II—Background information on the concepts behind the SCIL program, the Assertion Data, SOA, and Web Services.
- Chapter III—Requirements specification.
- Chapter IV—System design.
- Chapter V—Case study using a prototype of the performer team system.
- Chapter VI—Summary of the thesis and recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. SOCIAL CONTENT IN LANGUAGE

1. Introduction

As mentioned in the Introduction, the intent behind the SCIL program is to employ appropriate theory and to develop practical technology in order to understand the social goals of groups of interest. IARPA identifies three dimensions of the Program to be of utmost importance: the social features and activities of the groups; the linguistic features that serve as evidence of social goals; and the social science theories that help define the social features (IARPA, 2008). The central medium of analysis is the human language and how it serves as evidence of social activity.

The following three domains of knowledge are listed in the BAA as some examples in which any organization submitting a proposal can research:

Social Constructs and Activities of the Group / Members

- Goals, such as power, solidarity, group supremacy, religious supremacy, actions, manipulation strategies (e.g., persuasion, coercion, threats, intimidation, oppression, abuse, and exhortation), and recruitment

Linguistic Features and Their Form, Meaning and Strength

- Sacred language
- Conversational patterns (e.g., turn-taking; conversational cues and markers)

Social and Cultural Themes and Institutions

- Coercion
- Recruitment
- Loyalties (e.g., family, government, land, religion)

The UMD team will analyze persuasion attempts to contribute to the overall program. More specifically, the UMD performer teams will seek to provide information to the analysts by using a multidisciplinary approach to analyze the language obtained over an online textual-based communication system. Miller defined persuasion as “any message that is intended to shape, reinforce, or change the responses of another, or others” (Miller, 1980). Since the Internet has quickly developed into a primary tool of communication, it has since served as an ideal means for individuals or groups of individuals affiliated with terrorist organizations to express their beliefs and to initiate persuasive dialog with the intent to either convince others to take some kind of action or to accept some proposition or belief. There are many services and massive multi-person online environments that facilitate online communication (e.g., Facebook, MySpace, Twitter, World of Warcraft), providing a medium over which to influence and persuade individuals in such a manner that would negatively affect our national security.

A key supporting idea in theories of persuasion deals with research in *compliance gaining*. Compliance gaining involves a situation in which one person seeks to convince another person to do something for him/her. Marwell and Schmitt identified 16 compliance-gaining strategies. Their paper on this subject is seminal because the majority of research up to that point was concentrated on why people complied with persuasion, instead of how they went about complying (Marwell & Schmitt, 1967). Another key aspect of persuasion concerns *framing effects*, which are different from compliance gaining in that they are related to the way in which language is used in persuasion as opposed to the actual decision process used in persuasive arguments. Framing effects focus on the intentional linguistic selection a person makes in order to solicit a certain response or choice.

2. SCIL Architecture

For UMD to accomplish their work in the analysis of persuasion, they require a multidisciplinary approach, which includes disciplines in linguistics, natural language processing, and communications working together in order to form an assertion based on raw data. The raw data will be communication text found online from various social

forums. In the long run, the raw data will be processed at the lowest level into an automated system that performs the natural language processing using artificial intelligence to generate the assertions and subsequently forward the assertions to the knowledge base. For UMD, the assertions will deal with persuasion events. Figure 1 depicts the SCIL program architecture as intended, which demonstrates UMD and other notional performer teams. Intelligence analysts will execute queries to the knowledge base for information about certain groups of interest without having to go through the raw data. With the information at hand they, in turn, would advise higher level decision makers.

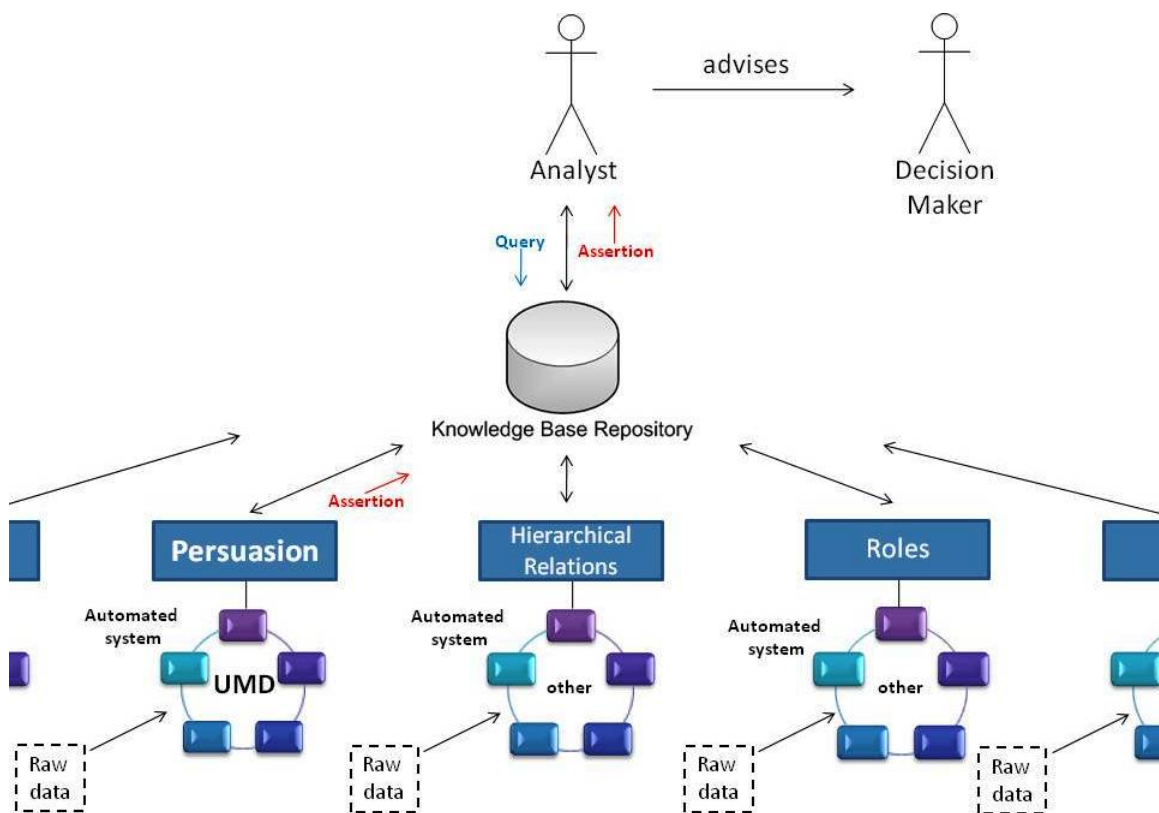


Figure 1. SCIL architecture

Currently, the UMD team is in the prescheduled Base Period, which consists of ongoing discussions about the content of an assertion, the definitions of language use, and performing system prototype testing. During this testing period, the system that we

will design will allow for local UMD researchers to manually store, manage, and forward sample or real assertions into the repository. The next sub-section describes the current agreed upon contents of an assertion.

3. **Assertions**

Assertions about groups of interest are what the analysts will query from the knowledge base. An assertion is defined as a declaration about something with or without facts. One can assert that the sky is green, or even that computers can talk, but without any evidence or support those assertions are meaningless. For our purposes, we will further define an assertion to be made up of the following three elements:

a. Claim

A claim is what the assertion is about. It contains a proposition about social phenomena in conjunction with a qualifier that reflects inherent uncertainty. The claim is based on and substantiated by language use. For example: “Phillip is a well-established leader of the group.”

b. Evidence

The evidence serves as the basis from which the claim is derived. Evidence is composed of sets of statements about social-linguistic features and/or social-cultural phenomena exhibited by the core data. For example: “90% of the topic discussions are initiated by John” or “Phillip is often referred to as sir.”

c. Support

Support is the rationale for asserting the claim based on the evidence. Support is an explanation, contextualization, and framing of the claim via one or more context statements. For example: “People who initiate discussions are typically leaders,” or “Men that are referred to as ‘sir’ are well respected in their culture.”

While we do expect that the definitions and concepts behind the generated assertions to evolve, our proposed system, along with the system tasked to perform the

natural language processing, will serve as a component deployed in a service-oriented architecture (SOA). The SOA architecture will utilize Web services communicating between UMD teams and the SCIL program repository. Section B describes the background of SOA and section C will provide information on Web services.

B. SERVICE-ORIENTED ARCHITECTURE

1. SOA Introduction

The purpose behind SOA is to allow for an enterprise to efficiently build applications that provide a service or processes to users while overcoming distributed computing challenges (Papazoglou & van den Heuvel, 2007). The impetus behind SOA has been the limited and complex nature of an organization's legacy IT architecture and its inability to implement solutions in support evolving business goals. More so, it is the lack of integration between an organization's internal IT systems and their business processes, business partners, and customers that is the driving force for change from the current system architecture to one that is service-oriented (Marks & Bell, 2006).

SOA is a term that represents a model or a design in which automation logic, which seeks to provide a solution to a given concern, is decomposed into smaller, self-sufficient, and distinct units of logic each of which contribute to the completion of the overall function (Erl, 2005). These decomposed units of logic, known as *services*, can range in size and scope. SOA, in and of itself, is not a technology. The key behind the SOA concept is that the services must be self sufficient. By avoiding inter-dependency, each individual unit can evolve on its own (Erl, 2005). The basic method in which an SOA looks to provide benefit is through the *separation of concerns*, decomposition if you will, to allow for smaller entities to complete their respective functions in order solve the overall concern.

The theory behind the separation of concerns, which Dijkstra references in his manuscript EWD 447: *On the role of scientific thought*, claims that there are benefits in decomposing a large problem into a set of smaller individual concerns where each

individual concern is addressed by a unit of logic (Dijkstra, 1982, pp.60–66). Object-oriented programming, as an example, implements this concept with its use of objects, classes and components.

Ideally, the services in a SOA should not only be aware of other services but be able to communicate with them. The services utilize what is called a *service description* to identify, locate, and manage the communication between services. The communication between the services takes place via a framework called *messaging*. Each message while underway from service to service is also self-sufficient and not dependent on anything but itself (Erl, 2005).

So far, what has been described is the basic architecture of what SOA is about. Figure 2 provides a visual representation of the basic architecture thus far.

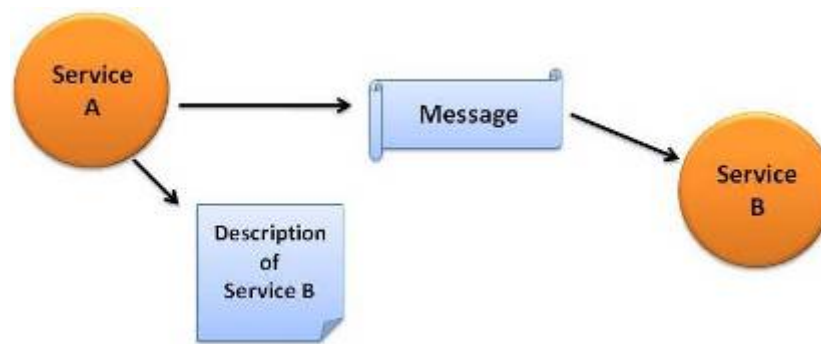


Figure 2. SOA architecture

Erl provides us with a simple analogy of the concept behind SOA, comparing it to a business community. Every city has some business community that consists of entities ranging in size from a Wal-Mart to a local mom and pop store. These businesses provide services to us as consumers, similar to how the units of logic (services) from a system perform a function and, if you take the business community as a whole, it seeks to solve a particular demand, much like how the automation logic provides a solution for an overall concern. The services still have certain established regulations that must be followed. In SOA, these regulations are the principles that drive service-orientation (Erl, 2005). Continuing with the business community analogy, the analogous regulations in place

could consist of a common currency that must be used for a transaction or even a common language that must be spoken. The services follow a set of common service-orientation principles through a process.

2. Principles of SOA

The following are commonly accepted principles of service-orientation as described by Thomas Erl:

a. Services Are Reusable

Reusability benefits by reducing the need for future development efforts. An analogy would be similar to how the Department of Motor Vehicles (DMV) services all requestors for a driver's license, but an even more reusable service would be a DMV that distributed licenses to users of all types of vehicles (ground, rail, aviation, boat, etc.).

b. Services Share a Formal Contract

Similar in concept to how a consumer would need to fill out an application for the vehicle license mentioned above, the contract defines the service, the operations/activities, and the messaging that are required to consume the service.

c. Services Are Loosely Coupled

Loose coupling facilitates agility. The factors that influence change in an IT environment are external, and so it is important that the underlying logic behind the individual services be independent. After we have waited in line, filled out our application, and taken the test at the DMV, the responsibility is on the DMV to process and provide us with the license.

d. Services Abstract Away Underlying Logic

The details (underlying logic) that make the service work are hidden, and of no concern to the user as long as the service works. Not many people know or care exactly what the DMV does in order to process their request for a license.

e. Services Are Composable

Composable means that services can have subservices. A service-oriented process is one that has a parent process in which it calls its underlying services to perform a subfunction. This principle reinforces reusability. The local law enforcement agency would require the use of the new DMV to license their drivers as well.

f. Services Are Autonomous

Autonomy allows for self-governance of all its processing, which subsequently allows for a service that is free from any dependencies that would restrain its deployment and evolution (Erl, 2005). Let us assume that the new DMV requires a background check on people applying for a license. The DMV normally delegates that task to another agency, and is dependent on their results. If the DMV were to perform its own background check, this independence would allow for the DMV to evolve its services without being held back by a dependent functionality.

g. Services Are Stateless

If a service is tied up by managing information, it reduces its own ability to receive other requests from other requestors. A stateless service promotes reusability and scalability by forwarding the message received and forgetting that it ever had the message, let alone what it was about. A services state is dependent on how well-designed the individual operations are, and how they function. The self-sufficient messages mentioned previously support statelessness. After completing the processing for an applicant's license, the details behind the application are saved in a database and there is no reason for the DMV to linger on your request. By doing so, it would inhibit the ability to service other applicants.

h. Services Are Discoverable

Discoverability prevents redundancy in service development. A discovery mechanism should be in place that enables potential clients who need your service to locate and consume it. The DMV information (location, phone number, etc.) can be found in the yellow pages, the Internet, or even signage on the highway, which is how we know to go there to begin with.

3. Benefits of Using SOA

We have vaguely touched on the benefits of using a SOA for any particular organization. Not all organizations require a change or, specifically, a shift in architecture, but for those organizations that are finding it costly to make changes to their current IT structure in response to a dynamic business environment, an SOA solution may be worth exploring. SOA provides developers with a means to overcome many distributed enterprise computing challenges, such as application integration, transaction management, and security policy enforcement, while allowing the concurrent use of multiple platforms and protocols and leveraging numerous access devices and legacy systems (Alonso et al., 2004).

Depending on how it is used, the benefits of SOA relate to the aforementioned principles and include the following: the costs of integrating your applications will be reduced, your returns on investment will increase due to the reusability principle, a reduced processing overhead and reduced skill-set requirements are experienced due to the services being composable, the need to replace legacy systems is reduced, which in turn saves money, the use of a standardized language like Extensive Markup Language (XML) reduces the cost and efforts of application development, and the costs involved in responding to changes via external sources are also reduced (Erl, 2005).

C. WEB SERVICES

A Web Service (WS) is defined by the World Wide Web Consortium (W3C) as:

A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. (W3C, 2004)

Basically, a WS is a method of implementing a distributed system, which allows for objects on one computer to interact with those of another computer over the Internet. There are currently two prominent models used for creating Web Services: the Representational State Transfer (REST) model and the SOAP-based model.

1. RESTful Web Services

RESTful WS allow for access to resources on a network referenced via a Uniform Resource Locator (URL). REST uses HTTP as the primary transportation protocol to support the execution of one of the four methods: GET, PUT, DELETE, and POST. REST services have limited support, in that REST has no standardization, few toolkits, and little by means of software library support. Message exchange can be performed in various formats (e.g., XML, HTML). Figure 3 shows an example request and associated response for a RESTful service.

Request	Response
GET /StockPrice/IBM HTTP/1.1 Host: example.org Accept: text/xml Accept-Charset: utf-8	HTTP/1.1 200 OK Content-Type: text/xml; charset=utf-8 Content-Length: nnn <?xml version="1.0"?> <s:Quote xmlns:s="http://example.org/stock-service"> <s:TickerSymbol>IBM</s:TickerSymbol> <s:StockPrice>45.25</s:StockPrice> </s:Quote>

Figure 3. REST Web service request / response (From Spies, 2008)

2. SOAP-Based Web Services

SOAP is the accepted standard method of communication between computers over the Internet that specifically uses XML to represent the information passed. SOAP is supported with toolkits and various software libraries. The earliest version of SOAP, version 1.1, was adopted by the W3C after its submittal in May of 2000. SOAP, which once stood for *Simple Object Access Protocol*, is currently in version 1.2 and is based on messages taking the form of documents. The encoding behind the *request* and *response* operations of SOAP WS is in XML format and the network transportation protocol can be by any means (e.g., IBM MQSeries, MSMQ, or SMTP) but the most common protocol is HTTP. For this research, we will focus on SOAP-based Web Services. Figure 4 shows an example request and associated response for a SOAP-based service.

Request	Response
<pre>GET /StockPrice HTTP/1.1 Host: example.org Content-Type: application/soap+xml; charset=utf-8 Content-Length: nnn <?xml version="1.0"?> <env:Envelope xmlns:env = "http://www.w3.org/2003/05/soap-envelope" xmlns:s="http://www.example.org/stock-service"> <env:Body> <s:GetStockQuote> <s:TickerSymbol>IBM</s:TickerSymbol> </s:GetStockQuote> </env:Body> </env:Envelope></pre>	<pre>HTTP/1.1 200 OK Content-Type: application/soap+xml; charset=utf-8 Content-Length: nnn <?xml version="1.0"?> <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" xmlns:s="http://www.example.org/stock-service"> <env:Body> <s:GetStockQuoteResponse> <s:StockPrice>45.25</s:StockPrice> </s:GetStockQuoteResponse> </env:Body> </env:Envelope></pre>

Figure 4. SOAP Web service request / response (From Spies, 2008)

3. Web Service Components

As mentioned previously, SOA consists of units of logic known as services that require a means either to locate other services or to advertise services. SOA also supports a means of communication that would enable interaction between services. In this section, we will elaborate more on each component that makes up a WS.

a. Services

A WS can be classified as being temporary or permanent, depending on the function that it assumes during runtime or its application logic. The basic service roles a service plays are that of a provider, intermediary, and a requestor (Erl, 2005). In the requestor role, the service will initiate the transmission of a message requesting a particular service of which the provider was designed to execute. The provider will execute the request and respond to the requestor. As an intermediary, the service will process a message, performing its own functionality, and forward the request to its service provider. Some functions an intermediary service can perform are authentication services, auditing services, and management services (Irani, 2010).

Services based on the nature of the application logic in which the service is intended to perform are classified as service models. For example, a service can be of a Business type (e.g., Accounts Payable Service) that executes one of many operations required by an organization, a Utility type (e.g., Internal Policy Service) that is completely reusable and non-application specific, or a Controller type that would be responsible for the coordinated functionality of all of the services within an organization (Erl, 2005).

b. Description

In order for services to interact, they first need to be aware of each other. Specifically, a provider needs a formal description of its services. The description acts as a contract with clients who request the provider's service. A W3C standardized WSDL document serves this purpose. WSDL is defined as:

An XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. (W3C, 2001)

The WSDL file is also written in XML and describes the data that will be passed and the method that passes it regardless of the programming language used. This means that a service written in Python can be consumed by client created in Java.

Although a WSDL is subdivided into several sections, it is generally composed of two key components: an abstract interface description (i.e., operations, operation parameters, and abstract data types) and a concrete implementation description (i.e., a network address, a protocol, and concrete data structures; Zimmermann, Tomlinson, & Peuser, 2003), the latter of which provides the means to actually invoke the service. Let us briefly look at the main elements found in a standard .wsdl file. The text and sub-tags within a .wsdl file are encapsulated inside of a <definitions> element and from top to bottom the five main elements within are: <types>, <messages>, <portTypes>, <binding>, and the <service>.

- <types>—This element includes the abstract information set using an *XML Schema* that defines the data types used in the communication process between the service provider and the requestor. If there are an excessive number of data types described, the schema can be imported into this section as a separate document.
- <messages>—The message element defines the abstract content of the messages to be communicated. The content includes the name of the message part (what the message is called), the element attribute that refers to the XML schema (what the message is), and the type (the type of data it holds).
- <portTypes>—The port type element identifies and describes a specific service interface. It is a named-set of abstract operations and their abstract messages that come in two varieties, input and output. The operations are made up of the messages described in the messages element.
- <binding>—The purpose of the binding element is to connect the abstract port type description to a concrete service implementation. The protocol details to send the messages are defined here.

- **<service>**—The service element, which is also known as the *endpoint*, specifies where and how to send the information. The service element works with the binding element by connecting a port type to a particular port defined within the service element.

Figures 5 and 6 are excerpts from two WSDL documents entitled stepGovServices_12.wsdl and stepPortTypes_12.wsdl (Tong, 2009). These files are referenced again in Chapter IV.

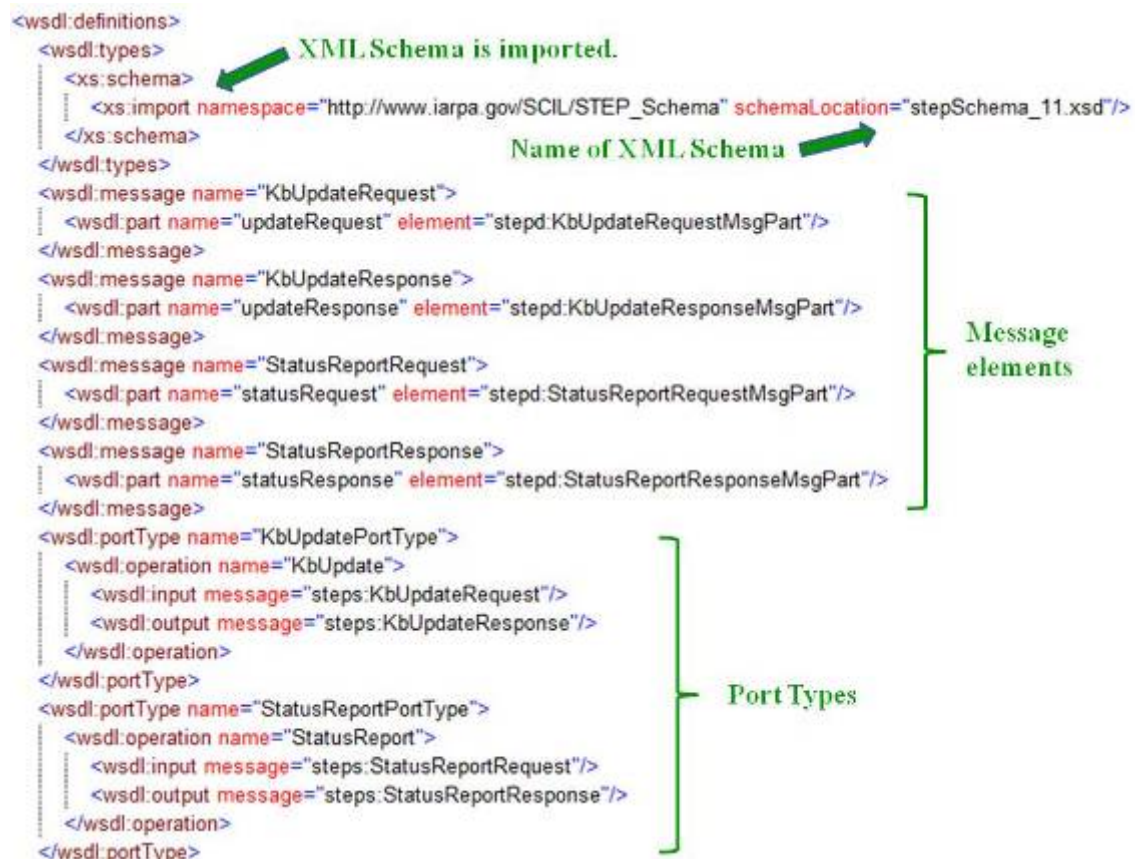


Figure 5. WSDL, abstract interface description


```

<wsdl:binding name="KBUpdateSoapBinding" type="steps:KbUpdatePortType">
  <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="KbUpdate">
    <soap12:operation soapAction="kbUpdate" soapActionRequired="true" style="document"/>
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="StatusReportSoapBinding" type="steps:StatusReportPortType">
  <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="StatusReport">
    <soap12:operation soapAction="statusReport" soapActionRequired="true" style="document"/>
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="UMD_KbUpdate">
  <wsdl:port name="KbUpdatePortType" binding="steps:KBUpdateSoapBinding">
    <soap12:address location="http://localhost:8080/UMDSCIL/UMD_KbUpdate"/>
  </wsdl:port>
</wsdl:service>
<wsdl:service name="UMD_StatusReport">
  <wsdl:port name="StatusReportPortType" binding="steps:StatusReportSoapBinding">
    <soap12:address location="http://localhost:8080/UMDSCIL/UMD_StatusReport"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Binding elements

Service elements

Figure 6. WSDL, concrete implementation description

c. XML Schema (XSD)

An XSD is a model document that defines the structure of a separate XML document. The XML document will contain a reference to the XSD that defines its structure, much like how the schema is imported to support the WSDL document in Figure 5. The schema's syntax is entirely in XML, and it serves to validate the XML document's adherence to the schema's structure. An XSD can contain several subordinate element types, similar to a WSDL, and tends to be very complex. However, some of the basic elements that you will find in an XSD are: **<elements>**, **<attributes>**, **<simpleType>**, and **<complexType>**; all of which serve to define the text data as strings, integers, dateTime, or data types. Figures 7 and 8 are excerpts from an XSD named stepSchema_12.xsd (Tong, 2009).



Figure 7. XSD I

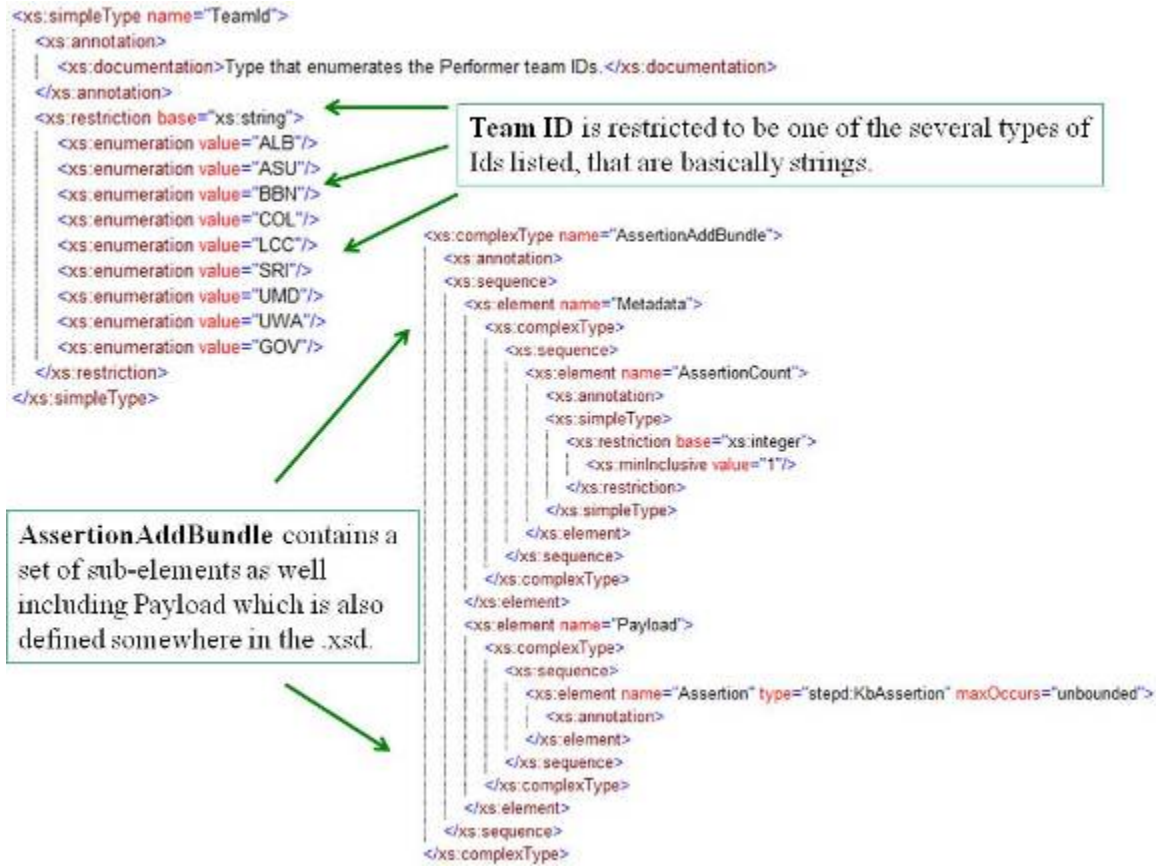


Figure 8. XSD II

d. Messaging

The communication protocol shared between services is the standard HTTP application layer used by clients and servers over the Internet. However, since the communication between services is message-based, the framework used should be standardized, flexible, and highly extensible (Erl, 2005). The SOAP messaging framework meets these requirements. A SOAP message contains the following elements: **<Envelope>**, **<Header>**, and **<Body>**.

- **<Envelope>**—The header is a mandatory element that acts as a container for the header and body elements. This element defines itself (an XML document) as a SOAP message.

- **<Header>**—The header element is optional and provides a means to pass any kind of additional processing or control information to recipients of the message.
- **<Body>**—The body is mandatory in that it holds the actual SOAP message intended for the service provider.

Figure 9 demonstrates a basic example of a client service requesting a connection identification number by sending a SOAP message containing a user name and password. The service provider responds to the requestor with the connection number.

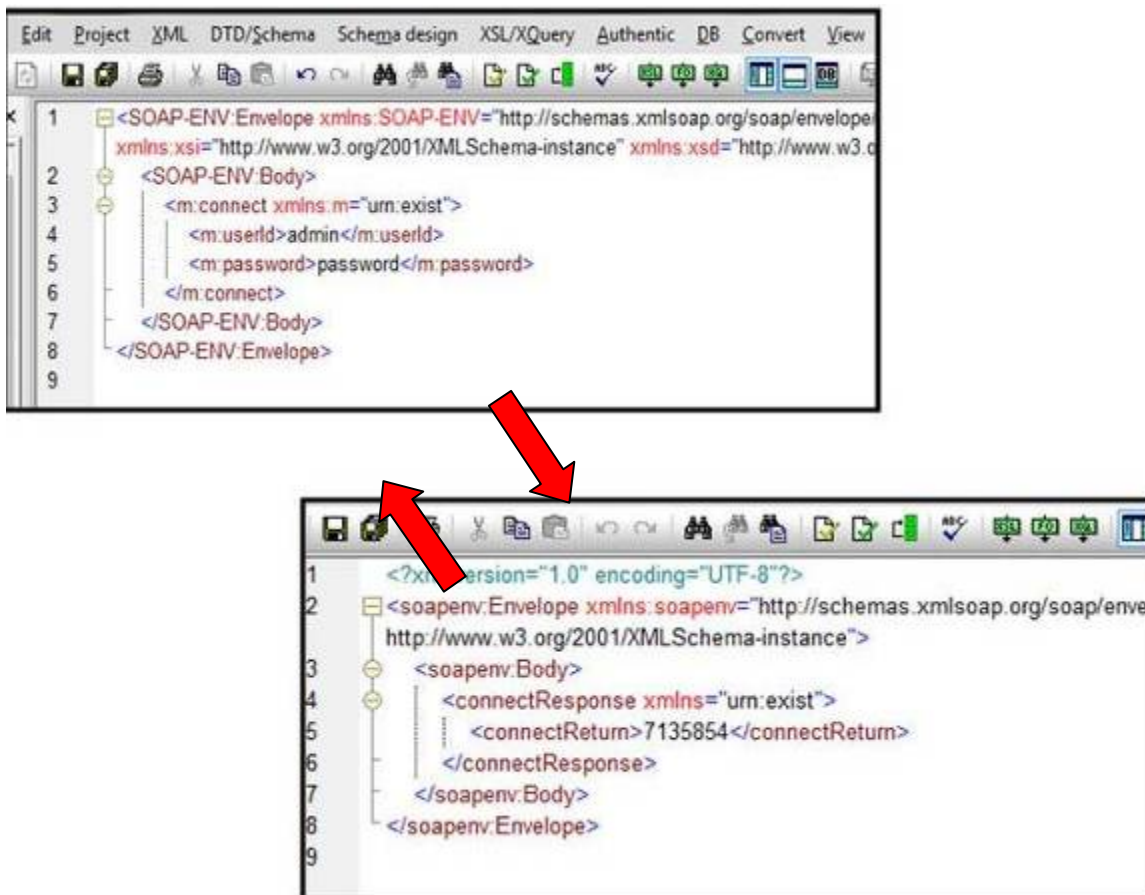


Figure 9. SOAP message

D. SUMMARY

This chapter began with an overview of the SCIL program intended to identify the social goals of a group of interest and its members by analyzing language features. UMD has proposed a research supporting the program that involves addressing the social phenomenon of persuasion. UMD will conduct a multidisciplinary approach to analyze the language and its use to determine if the intent to persuade is present. In order to make this happen, online text dialogue needs to be collected and analyzed to create assertions identifying persuasion attempts. These assertions must be processed by a system capable of performing the following: collect, modify, and locally store the data, allow for remote access to the data, and allow for the transfer of data to the external knowledge base repository over the Internet. We also covered some of the background on SOA and its benefits and we finished the chapter with a discussion on Web Services along with their common protocols (i.e., WSDL, XSD, SOAP). In Chapter III, we will address the key features and requirements necessary for the development of our proposed system.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SYSTEM REQUIREMENTS SPECIFICATION (SRS)

A. OVERVIEW

A software requirement is defined as a “software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documentation” (Leffingwell & Widrig, 2003). The methodology used to derive the following requirements for our proposed system were based on a face to face interview with a SCIL representative and an agreed upon set of use cases to describe the general system functionality.

1. Purpose

The purpose for this SRS is to determine the functional and nonfunctional requirements necessary to effectively store, process, and transfer assertion data generated by the local UMD performer team in support of the SCIL program.

2. System Perspective

This software is a new and self-contained product that is a separate component of a larger system design that includes other performer clients and services from various learning institutions and a central application server that provides a WS endpoint. Other performer systems will interact with the application server in the same way.

3. System Features and Domain Model

The major features of this system include two Web services: one that will display a local service status when queried by an external client; and another that will allow for an external client to retrieve assertion data, a WS client that will be capable of updating the external knowledge base, a software application that will be used to manage the assertion data and services, and a database used to store the data. All of the aforementioned features and interfaces will be written in or interact with the JAVA programming language.

Figure 10 is a domain model, which displays a visual perspective of the concepts relative to the proposed system. The local users will use the system to create assertions to be stored locally. The assertion text is composed of a claim, evidence, and support data. The assertion text combined with a local system-generated assertion ID (local ID), and IARPA's external system-generated assertion ID (external ID), and a date and time group (DTG) form a knowledge base assertion that is stored locally.

Our system will deploy two Web services: **ExplanationGetWS** and **StatusReportWS**. The ExplanationGetWS will allow for clients to retrieve assertion data from the local system. Our system will also execute one WS client: **KbUpdate** client. KbUpdate client will interact with the external system's WS to transfer, replace, or delete assertion data. Our KbUpdate client and ExplanationGetWS will have a status associated with them that will be returned in response to queries to the StatusReportWS. The core operations will be explained in more detail in Chapter IV. Both Web services will be deployed on an application server that will allow for their consumption by the external system. The domain model also depicts a DataPush client. The purpose behind the DataPush client has not been agreed on by the UMD stakeholders at this time; therefore, aside from its place in the domain model, this research will not include the DataPush functionality.

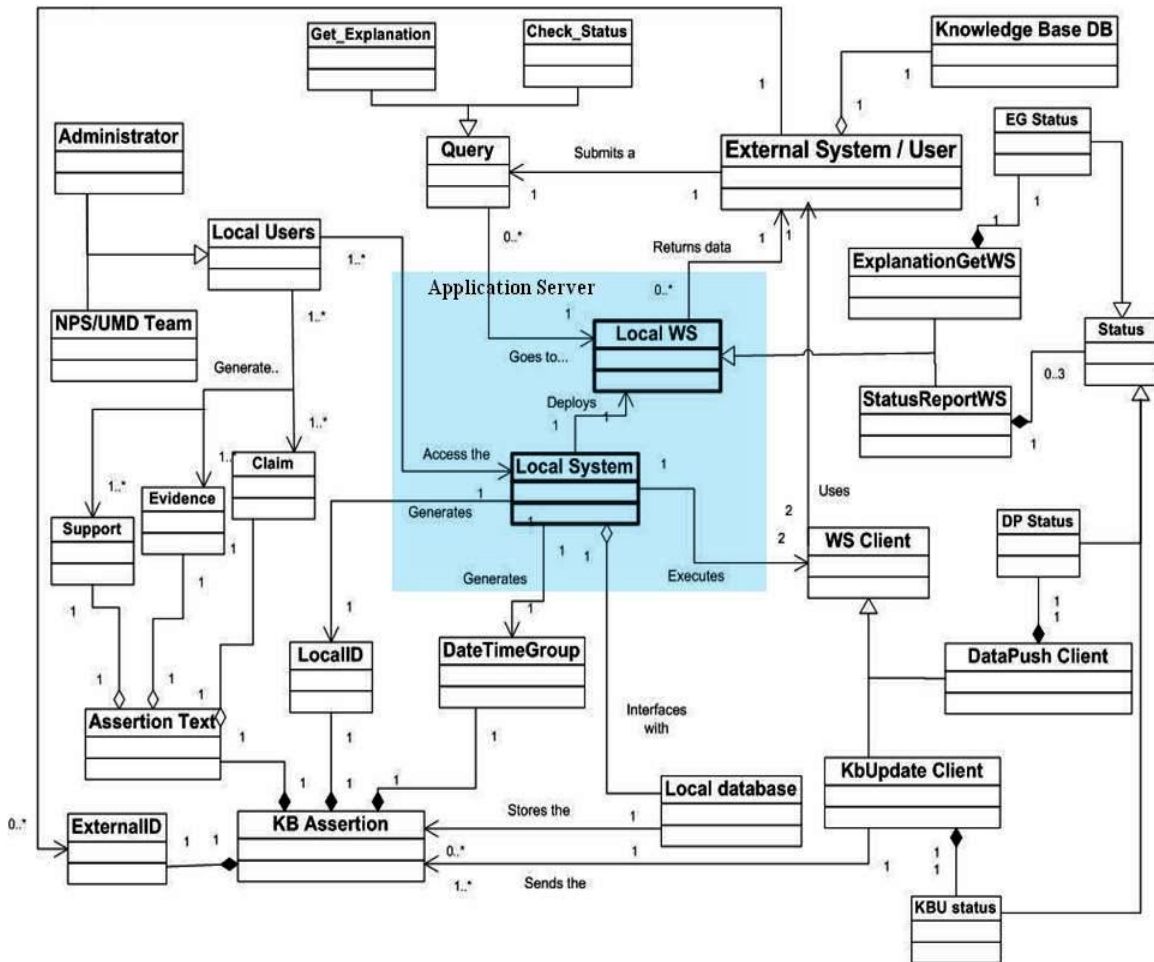


Figure 10. Domain model

4. Intended Audience

This SRS is intended to be read by the local users and project managers who represent the UMD team in support of SCIL.

B. FUNCTIONAL REQUIREMENTS

This section begins the description of the system features utilizing several use cases that serve as functional requirements. The use cases are scenario-driven, meant to provide us with both an outside-in view of the system functionality, and a critical tool in the analysis process. Figure 11 is a use-case diagram, which introduces the system actors

and their respective interactions with the system. The elements listed under UMD System represent the individual use cases, and will be explained in more detail. As a reminder, the local users will be replaced by an automated system that creates the assertion from the input of raw data in the future.

Following each use case, we have provided a system sequence diagram (SSD). An SSD is “a picture that shows, for one particular scenario of a use case, the events that external actors generate, their order and inter-system events” (Larman, 2005). The point behind the SSD is to identify the particular events that will transpire during execution giving us a clearer picture of the system behavior.

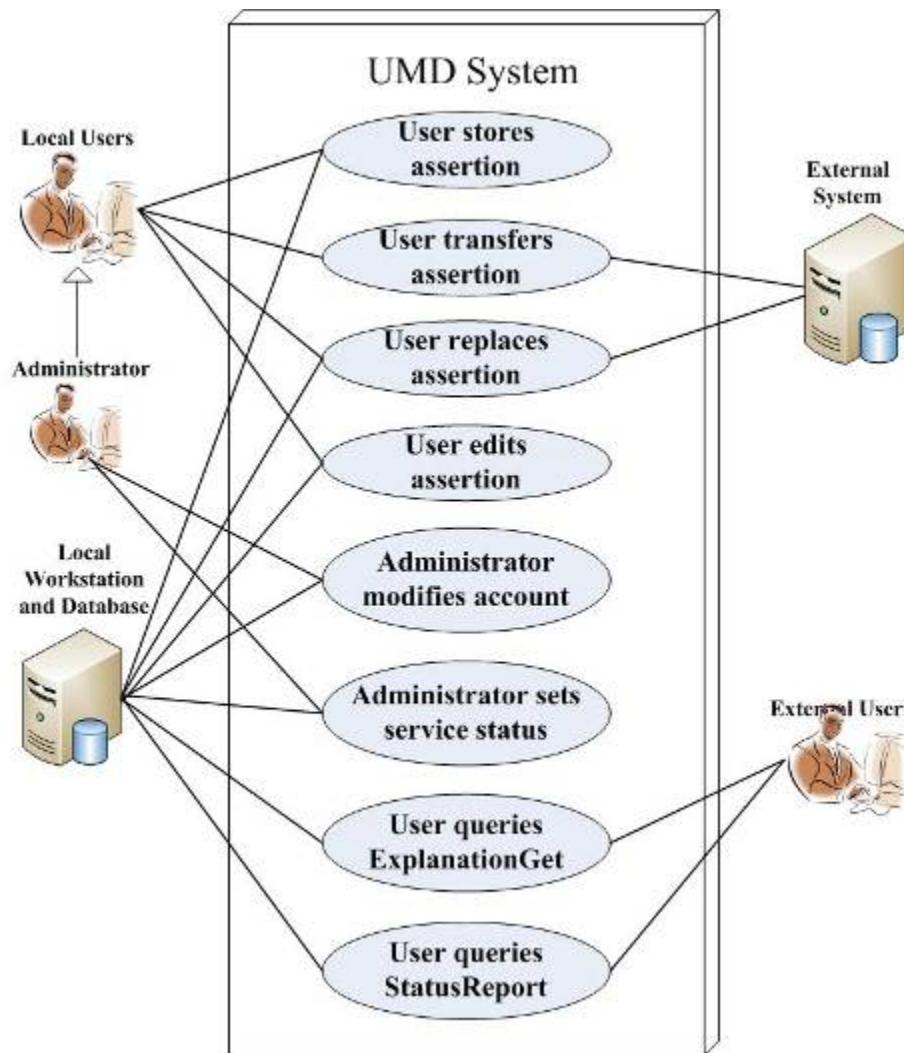


Figure 11. Use case diagram

1. *Local User Stores Assertion*

Table 1. Local user stores assertion

Primary actor	Local user
Stakeholders and interest	- Local user wants to store the assertion in to the local database. - External user needs the assertion to be accessible.
Entry conditions	- The local user workstation needs to be operational. - The local user has already logged in. - The local user can access the assertion text via the local workstation.
Exit conditions	- The assertion has been assigned an external ID. - The assertion is stored within the local database.
Main scenario	- The system displays a menu option, which includes the option to store the assertion. - Local user selects the option to store assertion. - The system displays an interface that allows for the manual entry of the data along with the option to store the assertion. - Local user manually enters the assertion data into provided text fields. - The local user selects the option to store. - The system generates an local ID for the assertion. - The system generates the DTG for this instance. - The system stores the assertion, respective DTG, and local ID into the local database. - The system displays the local ID to the user along with a message stating that the operation is complete. <i>The User repeats steps 2–9 until complete</i>

	10. The local user logs off of the workstation.
Extensions	*a. Workstation System failure: 1. The local user restarts the system and logs in. 2. The system assumes its state prior to step 1 in the main scenario. 3. The local user re-attempts the storing process 4a. Invalid input: 1. The system displays an “invalid input error” message stating that the data entered was incorrect. 2. The system returns to the state at step 3 in the main scenario.

Local User Stores Assertion

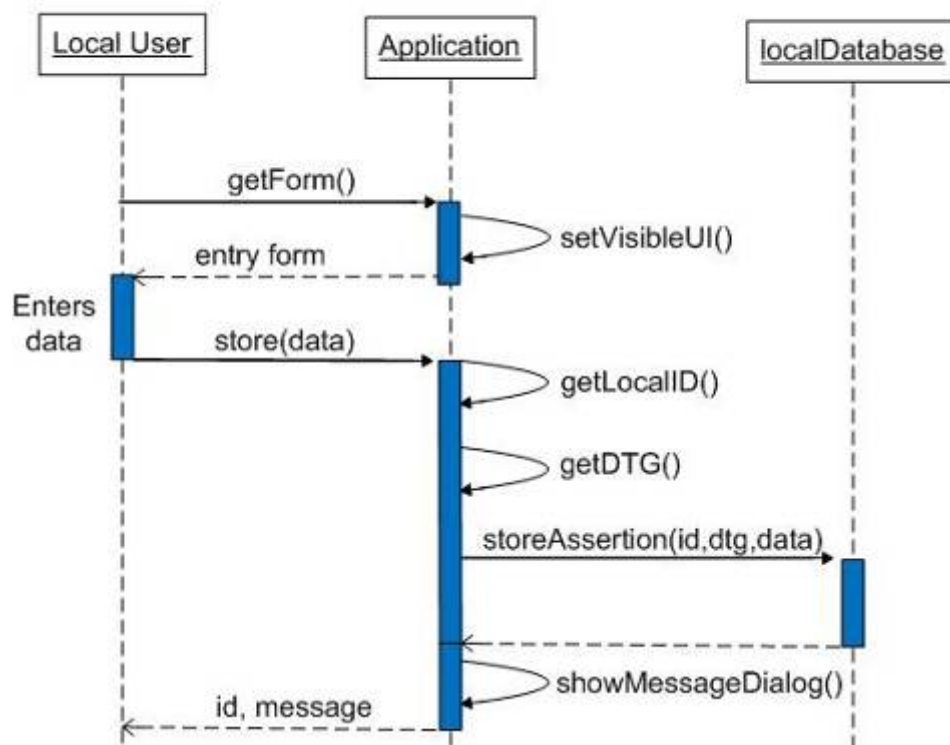


Figure 12. Store assertion SSD

2. *Local User Transfers Assertion to Knowledge Base*

Table 2. Local user transfers assertion to knowledge base

Primary actor	Local user
Stakeholders and interest	- Local user wants to transfer an assertion from the local database to the external system/database. - External system stores the assertions
Entry conditions	- The assertion data has been previously stored within the local database. - The assertion data is assigned a local ID. - The local and external systems are operational. - The local user is logged in the system and is on the main menu.
Exit conditions	- The assertion has been successfully transferred from the local database to the external database. - An external ID is returned to the local user and stored in the local database.
Main scenario	- The system displays a menu option, which includes the option to transfer assertion data. - The local user selects the option to transfer an assertion. - The system displays an interface that allows the user to select the assertion(s) that will be transferred - The local user selects the assertion(s) and then selects submit. - The system initiates a query to the local database for the assertion data using the local ID as reference. - The system makes a copy of the assertion data. - The system executes the KbUpdate client system by sending the copied data over the internet to the

	external system. 8. The external system redirects the data to the external knowledge base for storage. 9. The external system returns an assertion external ID(s) to the sender. 10. The local system receives and displays the external ID(s) to the local user. 11. The system stores the external ID in the local database. 12. The system returns the local user back to the main menu. • <i>The User repeats steps 2-12 until complete.</i> 13. The local user logs off.
Extensions	4a. Invalid identification number: 1. The system displays an error message stating that the identification number entered was mal-formed or non-existent. 2. The system returns to the state at step 3 in the main scenario 7a. Network failure before the assertion reaches the external database: 1. The local system displays an error message stating that a connection was not completed. 2. The system returns to the state at step 6 in the main scenario prior to the transfer attempt. 9a. Network failure after the transfer and before the return of the external ID. 1. The system displays an error message stating the network condition. 2. The system returns to the state at step 6 in the main scenario prior to the transfer attempt.

User Transfers Assertion Scenario

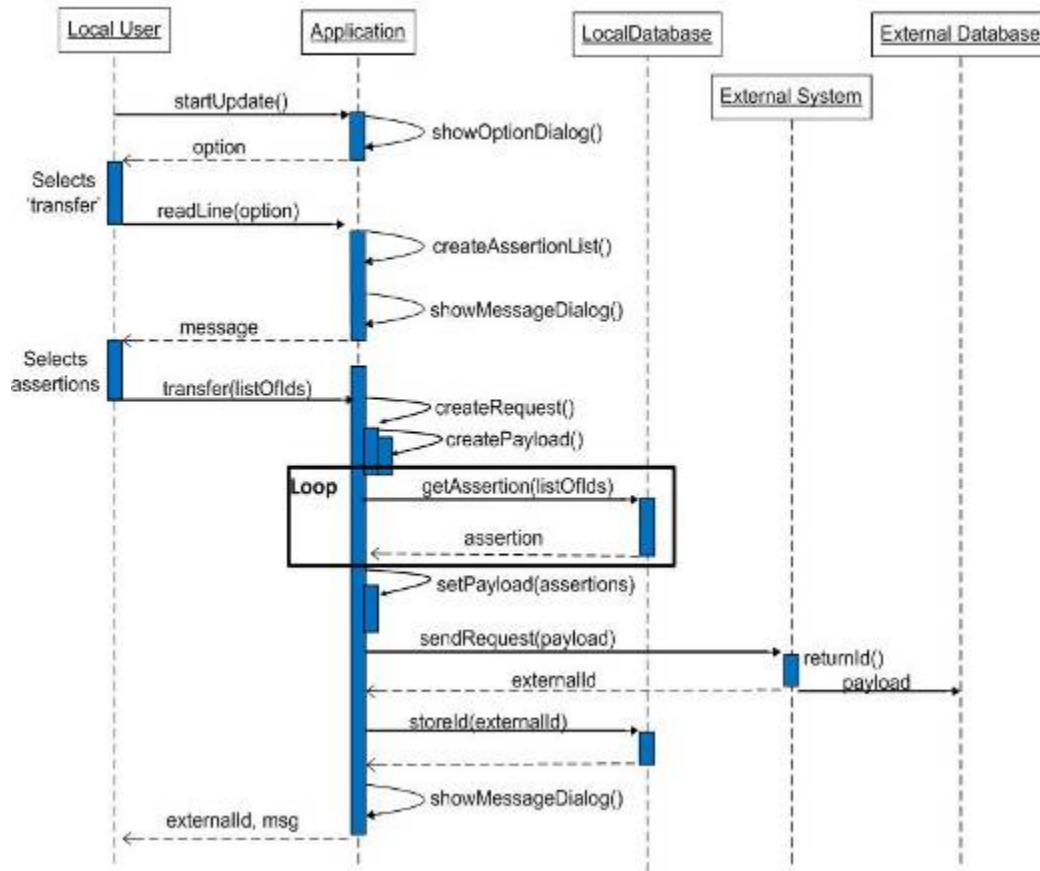


Figure 13. User transfers assertion SSD

3. *Local User Replaces Assertion in the External System*

Table 3. Local user replaces assertion

Primary actor	Local user
Stakeholders and interest:	- Local user wants to replace an existing assertion in the external database with one from the local database. - External System stores assertions.
Entry conditions:	The assertion data has been previously stored within the local database.

	• The assertion data is assigned a local ID. • The assertion to be replaced has been assigned an external ID and that ID has been stored in the local database. • The local and external systems are operational. • The local user is logged in the system and is on the main menu.
Exit conditions:	• The old assertion has been successfully replaced by the new assertion. • An external ID is returned to the local user and stored in the local database.
Main scenario:	1. The system displays a menu option, which includes the option to replace an existing assertion in the external database. 2. The local user selects the option to replace an assertion. 3. The system displays an interface that allows the user to select the assertion that will replace the assertion in the external database. 4. The local user selects the assertion. 5. The system displays an interface that allows the user to select the assertion that will be replaced. 6. The user selects submit. 7. The system initiates a query to the local database for the assertion data using the local ID as reference. 8. The system makes a copy of the assertion data. 9. The system executes the KbUpdate client by sending the copied data and the external ID of the assertion to be replaced over the internet to the external system. 10. The external system redirects the data to the external

	database and replaces the assertion referenced by the external ID storage with the new assertion. 11. The external system returns a new assertion external ID assigned to that assertion to the local system. 12. The local system receives and displays the external ID and a message to the local user indicating the operation is complete. 13. The system stores the new external ID in the local database. 14. The system returns the local user back to the main menu. • <i>The User repeats steps 2-14 until complete.</i> 15. The local user logs off.
Extensions	4a. Invalid identification number: 1. The system displays an error message stating that the identification number entered was mal-formed or non-existent. 2. The system returns to the state at step 3 in the main scenario. 9a. Network failure before the transfer takes place: 1. The local system displays an error message stating that a connection was not completed. 2. The system returns to the state at step 6 in the main scenario prior to the transfer attempt. 11a. Network failure after the transfer and before the return of the new identification number: 1. The system displays an error message stating the network condition. 2. The system returns to the state at step 6 in the main scenario prior to the transfer attempt.

User Replaces Assertion Scenario

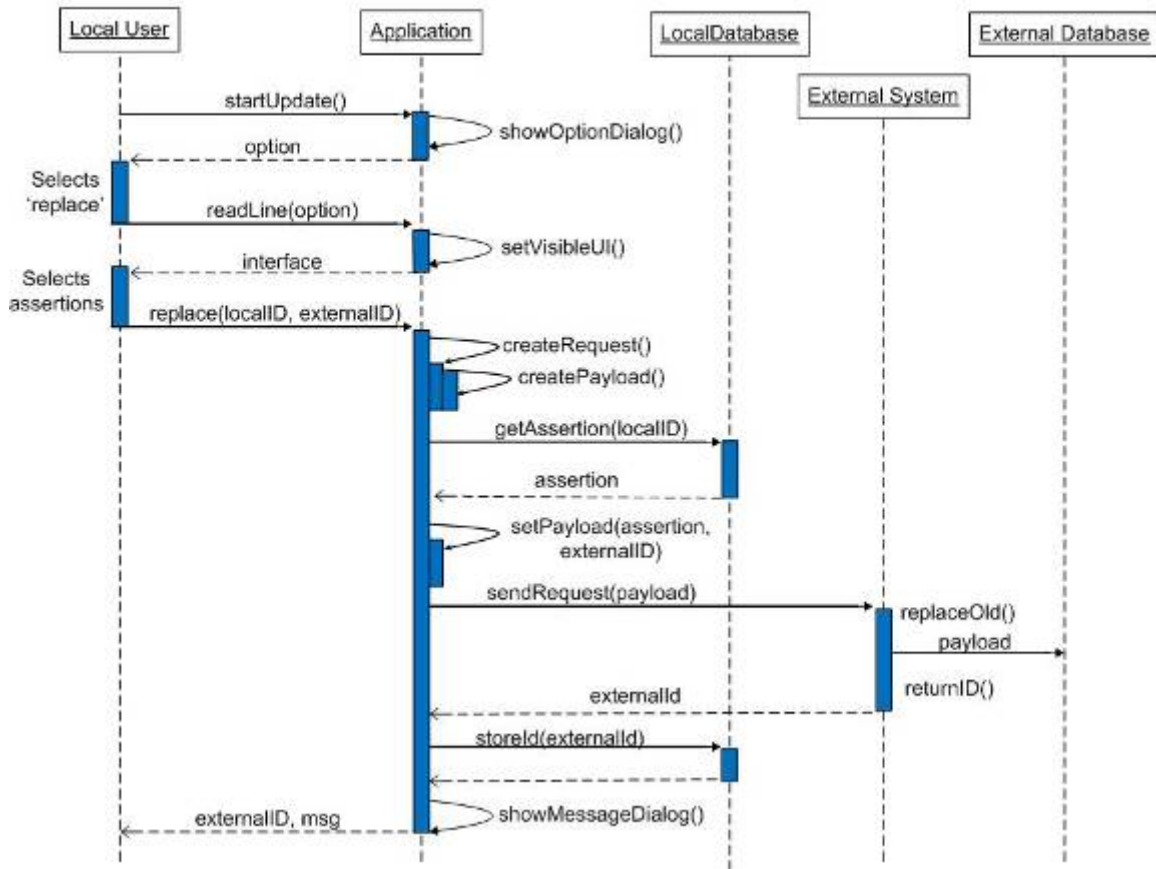


Figure 14. User replaces assertion SSD

4. *Local User Edits Assertion in Local Databas*

Table 4. Local user edits assertion

Primary actor	Local user
Stakeholders and interest:	Local user wants to ensure that the data stored in the local database is updated correctly.
Entry conditions:	- The assertion data has been previously stored within the local database. - The assertion data is assigned a unique identification

	number. • The local system is operational. • The local user is logged in to the work station.
Exit conditions:	• The data within the local database has been updated successfully.
Main scenario:	1. The system displays a menu option, which includes the option to edit a locally stored assertion. 2. The local user selects the option to edit assertion. 3. The system displays a prompt for the user to enter an assertion local ID. 4. The local user enters the local ID. 5. The system retrieves a copy of the assertion data displayed in an interface containing text fields that are populated with the current data and are capable of being changed. 6. Local user edits the data through the interface and selects the option to save. 7. The system reassigns the original assertion local ID to this instance. 8. The system generates a new DTG for this instance. 9. The system inserts the new data, DTG, and the reassigned local ID into the local database. 10. The system displays to the user the local ID and a message stating that the update is complete. • <i>The user repeats steps 2-10 until complete.</i> 11. The local user logs off the workstation.
Extensions	4a. Invalid identification number: 1. The system displays an error message stating that the identification number entered was mal-formed or non-existent.

	2. The system returns to the state at step 3 in the main scenario.
	6a. Invalid input: 1. The system displays an “invalid input error” message stating that the data entered was incorrect. 2. The system returns to the state at step 5 in the main scenario with the original assertion data remains unchanged.

Local User Edits Assertion Scenario

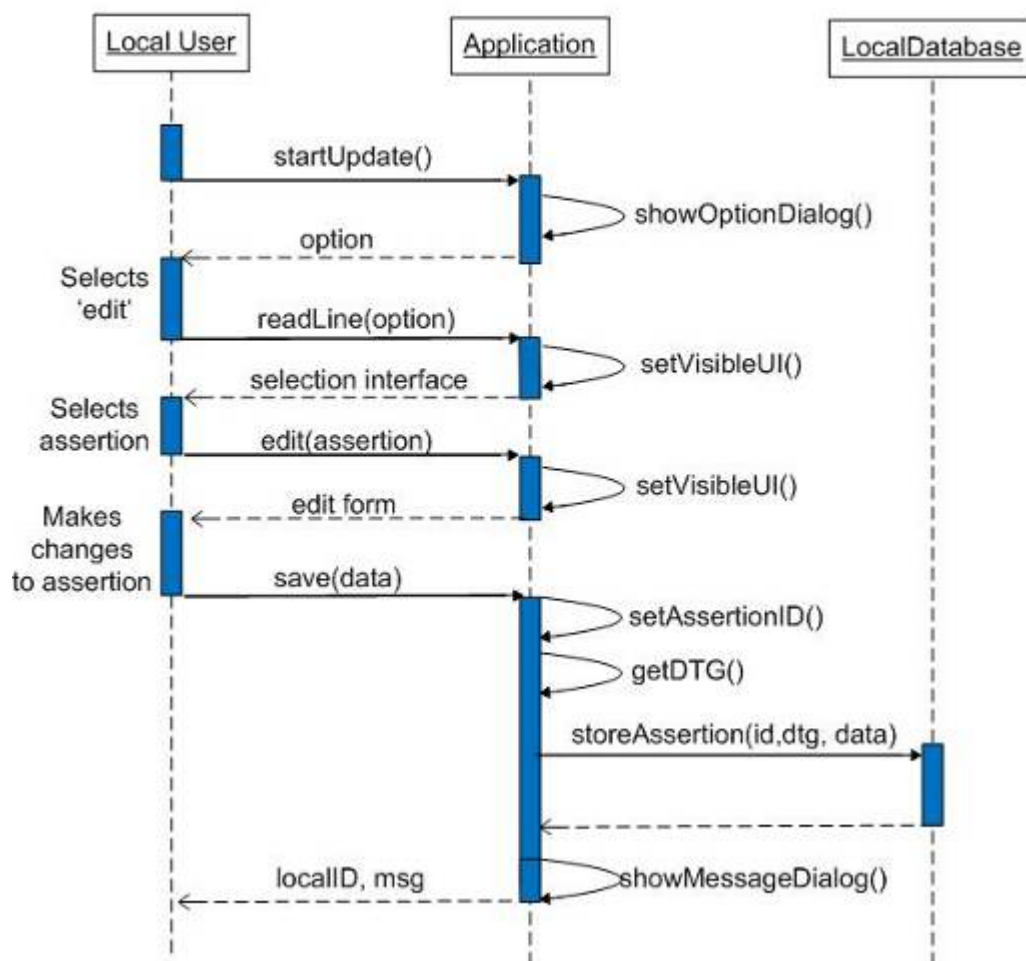


Figure 15. Local user edits assertion SSD

5. *External User Queries ExplanationGetWS*

Table 5. External user queries ExplanationGetWS

Primary actor	External user
Stakeholders and interest:	• The external user needs information from the local database. • The local database has the assertion that is being requested.
Entry conditions:	• The assertion has been previously stored in the external database. • The assertion is currently stored in the local database. • The assertion has been assigned an external ID. • The local service is operational. • The external user has already provided his/her appropriate authentication and has logged in to the external system.
Exit conditions:	• The assertion has been successfully retrieved from the local system by the external user.
Main scenario:	1. The external user (client) submits a request for assertion data. 2. The ExplanationGetWS receives the request, which includes the external ID of the assertion to be retrieved. 3. Using the external ID as a reference, the local service selects the respective data from the local database. 4. The data is returned to the external user via the internet. 5. The local service returns to the state before the initial query.

	• <i>The external user repeats steps 1-5 until complete.</i>
Extensions	1a. Network connection not available: 1. If the network connection is inoperable the external user is presented with an error. 1b. Invalid identification number: 1. The service interface displays an error message stating that the identification number entered was mal-formed or non-existent. 2. The service interface returns to step #1 in the main scenario.

External User Queries ExplanationGet

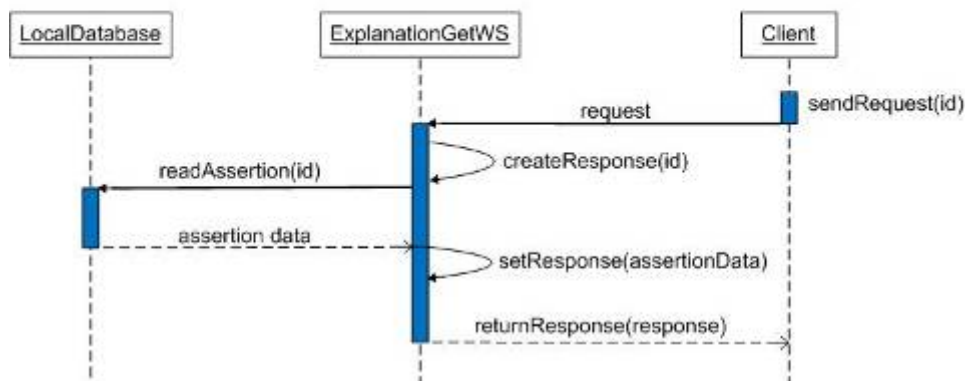


Figure 16. External user queries ExplanationGetWS SSD

6. *External User Queries StatusReportWS*

Table 6. External user queries StatusReportWS

Primary actor	External user
Stakeholders and interest:	The external user wants to verify the status of the local system capabilities.
Entry conditions:	- StatusReportWS is operational. - The local capabilities have been assigned a status. - The external user provided the appropriate authentication.
Exit conditions:	The external user successfully receives the statuses.
Main scenario:	- The external user, acting as the client, submits a request to StatusReportWS. - The local service receives the request from the external user. - The local service retrieves the operational status. - The operations status is returned to the external user via the internet. - The local system returns to the state prior to the initial query. <i>The external user repeats steps 1-5 until complete.</i>
Extensions	4a. Invalid identification number: - The system displays an error message stating that the identification number entered was mal-formed or non-existent. 1a. Network connection not available: - The external user is presented with a connection error.

External User Queries StatusReport

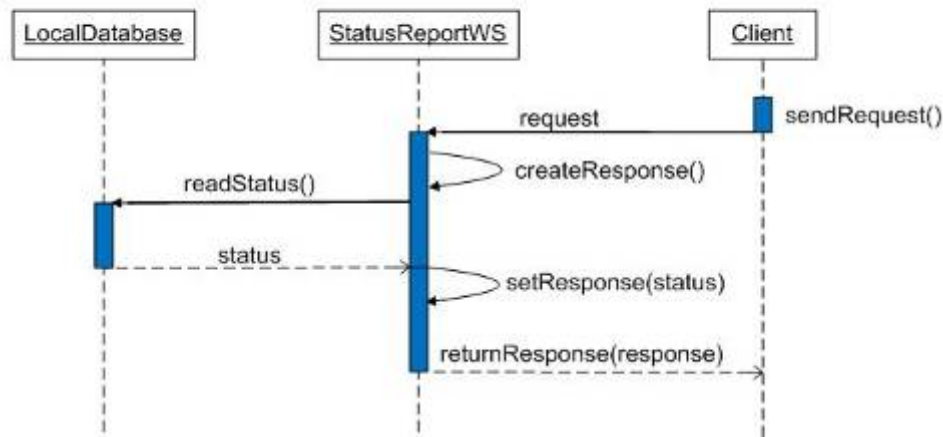


Figure 17. External user queries StatusReportWS SSD

7 *Administrator Sets the Capability Status*

Table 7. Administrator sets capability statuses

Primary actor	Administrator
Stakeholders and interest:	- The administrator wants to ensure the appropriate status is set. - The external user wants to occasionally verify the status of the local system capabilities.
Entry conditions:	- The local service is operational. - The administrator has already provided proper authentication and is logged in.
Exit conditions:	- The administrator successfully sets the capability status. - The administrator receives confirmation of the status update.
Main scenario:	1. The system displays a menu option that includes the

	option to set the status of the system capabilities. 2. The administrator selects the option to set the status. 3. The system displays an user interface to set the appropriate status. 4. The administrator sets the status and selects to the option to save. 5. The system updates the local database with the changes to the statuses. 6. StatusReportWS is redeployed to the application server reflecting the new statuses. 7. The system returns a message to the administrator confirming the status settings. 8. The local system returns to the state at step 1 in the main scenario. • <i>The administrator repeats steps 1-8 (if necessary)</i>
Extensions	1a. Connection to the database is not available: 1. The system displays a database connection error. 2. The system returns to the state prior to step 1.

Administrator Sets Capability Status

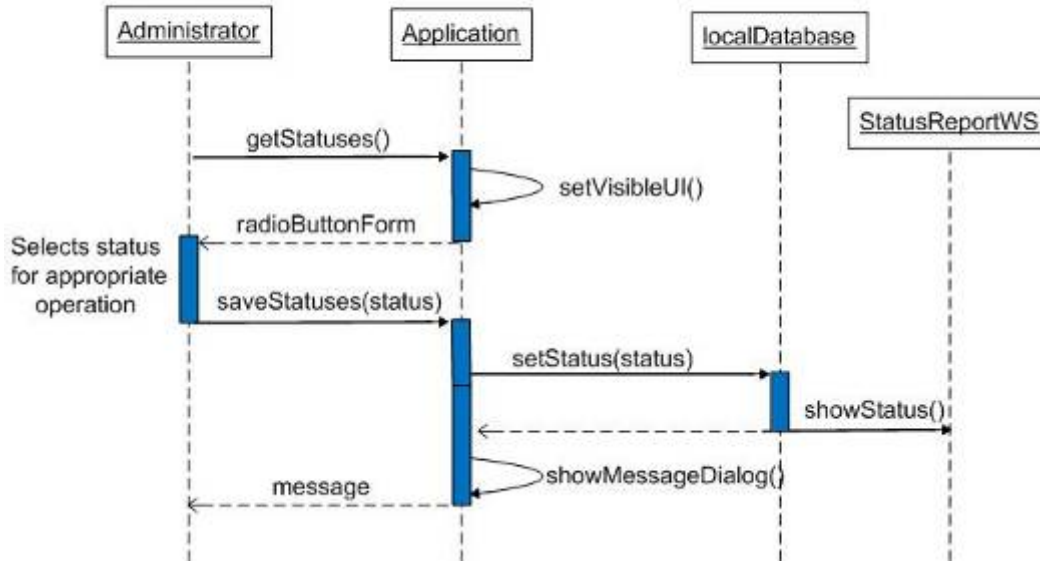


Figure 18. Administrator sets statuses SSD

8. *Administrator Modifies a User Account*

Table 8. Administrator modifies a user account

Primary actor	Administrator
Stakeholders and interest:	• A local user needs an account to execute the system operations. • The Administrator can create, delete, or edit a user account.
Entry conditions:	• The local system is operational. • For account creation, the new user must not have an existing account. • For account deletion or editing, the user must have existing account. • The Administrator is already logged on to the system.

Exit conditions:	• A user account is either successfully created, changed, or deleted.
Main scenario:	1. The system displays a main menu interface that includes the option to modify user accounts. 2. The administrator selects the option to modify user accounts. 3. The system re-prompts the system administrator for a password. 4. The administrator enters the password. 5. The system displays an option to either create a new user account or edit a user account. 6. The administrator selects the option to create a new user account. 7. The system displays an interface prompting for the new user's name, account username, and account password. 8. The administrator enters the new username and password. 9. The administrator selects the option to save the new account creation. 10. The system processes the request. 11. The system returns a message stating that the account has been successfully created. 12. The system returns to the state described in step 5. • <i>The system administrator repeats steps 5-12 until complete.</i> 13. The system administrator returns to the main menu and logs off.
Extensions	4a. Incorrect Password: 1. The system re-prompts for the system administrator's

	password. 6a. Administrator wants to edit a user account: 1. The administrator selects the option to edit a user account. 2. The system displays a list of users that are selectable and the option to edit or delete the selected account. 3. The administrator selects the user and chooses the option to edit. 4. The system displays an interface with text fields populated with the users account information and capable of being manipulated. 5. The administrator makes the changes to the user account and selects the option to save. (Continue with step #10 in the main scenario). 6b. Administrator wants to delete a user account. 1. Steps 1 and 2 from 6a are performed. 2. The administrator selects the user(s) and chooses the option to delete. 3. The system prompts the administrator for confirmation on the delete operation. 4. The administrator confirms the operation. (Continue with step #10 on the main scenario).
--	--

Figure 19 displays the SSD for the case in which the administrator modifies a user's personal account. Figure 20 extends from Figure 19 displaying the administrator's choice of delete and edit. Both continue after the administrator selects the *option(s)* as depicted within the red box.

Administrator Modifies a User Account

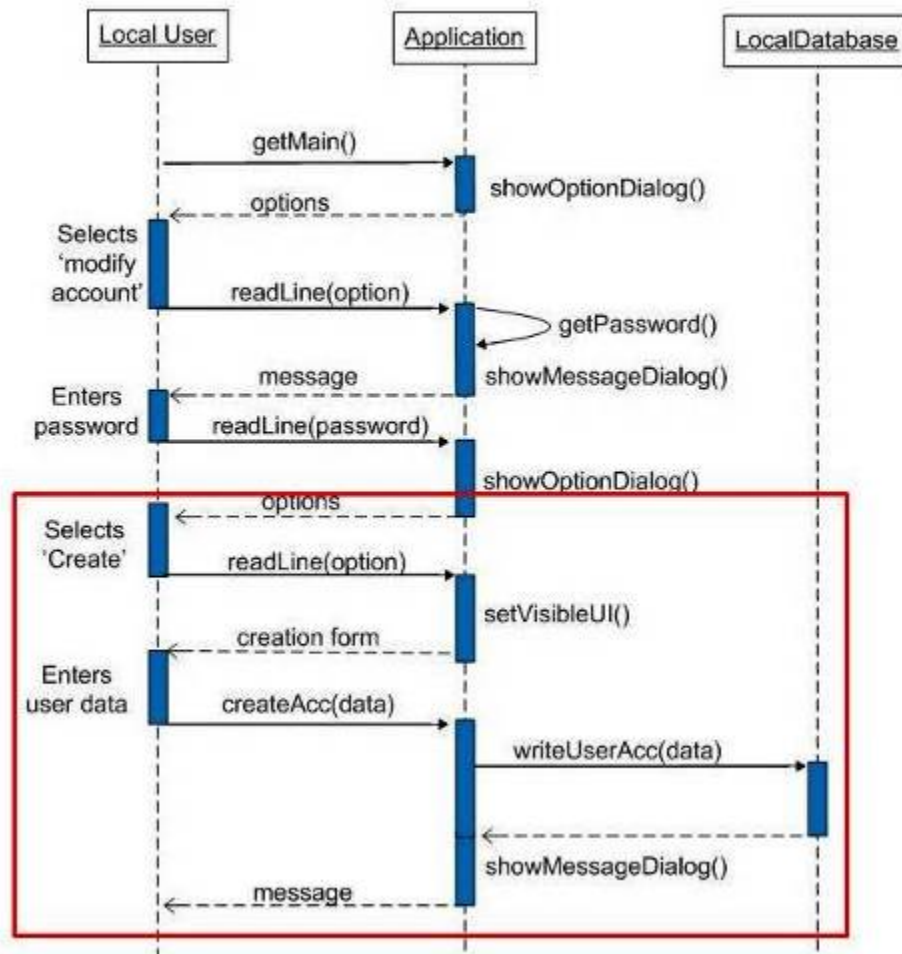


Figure 19. Modify user account (create) SSD

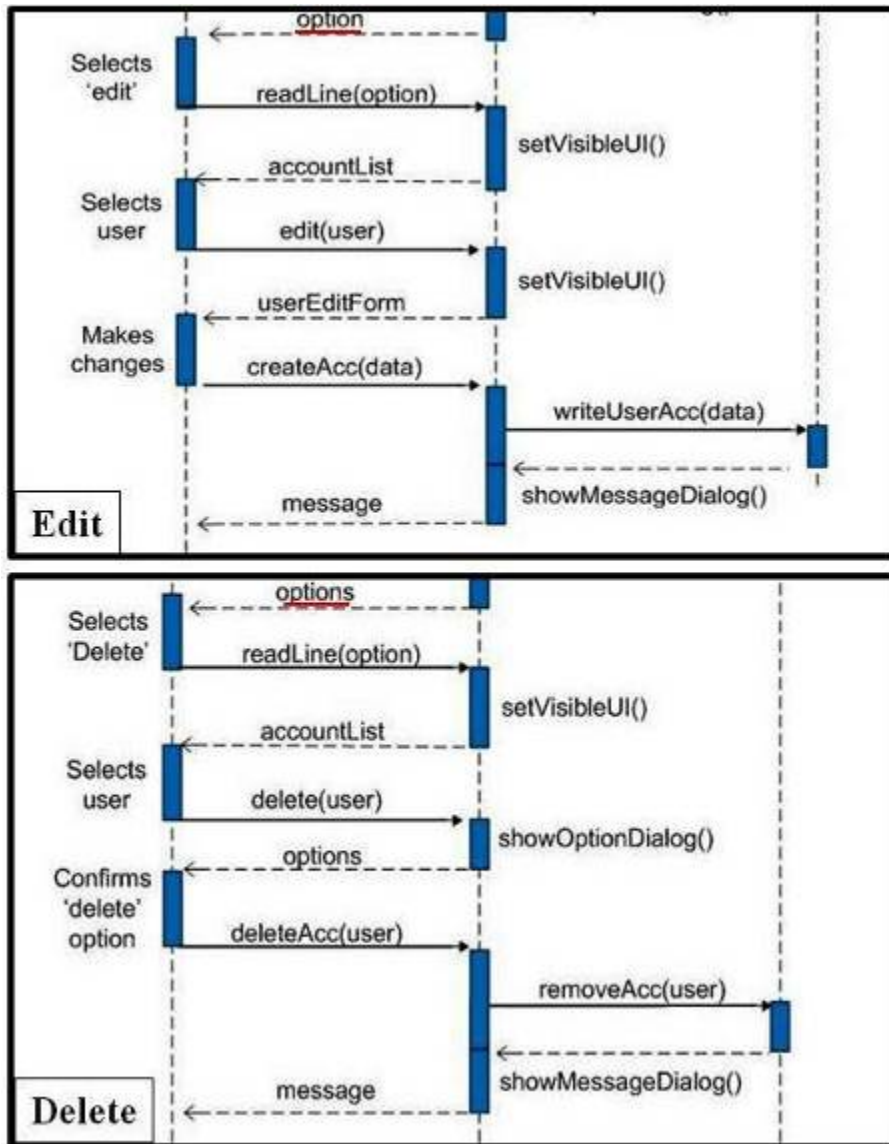


Figure 20. Modify user account (edit & delete) SSD

C. OTHER FUNCTIONAL REQUIREMENTS

The requirements listed in this section are those that were not specified in the use cases described in Subsection B.

1. *System Access*

1.1 A local user is required to have an account in order to use the system.

- 1.2 The user account will include a username and password.
- 1.3 Usernames will be the user's school email address.
- 1.4 Users will not be able to change their passwords.
- 1.5 User passwords will be up to 15 characters in length.
- 1.6 User passwords can contain any ASCII character.
- 1.7 User passwords will be case sensitive.
- 1.8 The ability to manage the local user accounts will be restricted to the system administrator.
- 1.9 The system administrator will have the ability to grant or remove system privileges to users, change user passwords, change the system status, and to modify new user accounts.

2. *Database*

- 2.1 The Database Management System (DBMS) will use a server database type.
- 2.2 The DBMS will support a relational model.
- 2.3 The DBMS will utilize a Structured Query Language (SQL) engine.
- 2.4 The DBMS will utilize an interface driver (e.g., JDBC, ODBC).
- 2.5 The DBMS should be able to run on a Windows, Macintosh, or Linux operating system.
- 2.6 The DBMS should have a minimum database size of 4 Gigabytes.
- 2.7 The DBMS should be capable of executing, at minimum, the following Data Types: integers, decimal, strings, and date & time.
- 2.8 The primary key will be used as the local database assertion identification.
- 2.9 The DBMS will need to allow for more than one row of data per assertion.
- 2.10 Any changes or service requests made to the local database must be logged and subsequently be capable of being queried.

3. *Local System Functionality*

3.1 When creating and subsequently storing an assertion, the user will also have the option to either Store and Replace or Store and Transfer.

3.1.1 The Store and Replace function will store the assertion in the local database and replace an assertion that currently exists in the external database.

3.1.1.1 When a Store and Replace function is executed the local copy of the assertion that is replaced in the external database will remain in the local database with a note replacing the respective external ID stating a replacement has occurred.

3.1.2 The Store and Transfer function will store the assertion in the local database and forward the assertion to the external database.

3.2 When editing an assertion, the user will have the option to either 'edit' an existing assertion or 'delete' an existing assertion.

3.3 When editing an assertion, the user will be shown an interface that displays selectable assertions with fields that show the assertions have a local ID and / or an external ID.

3.3.1 When the function to 'edit' is selected and both an local ID and external ID exist, the selected assertion will be changed in the local database and the copy that exists in the external database will be replaced by the new version.

3.3.2 When the function to 'edit' is selected and only a local ID exists for the assertion then the selected assertion will be changed in the local database only.

3.3.3 When the function to 'delete' is selected and only a local ID exists for the selected assertion, the assertion will be removed from the local database.

3.3.4 When the function to 'delete' is selected and both an local ID and external ID exist, the user will be given an option to either delete the selected assertion from only the external database or from both databases.

3.3.4.1 If the user selects to delete an assertion from the external database only, the local copy of the assertion that is removed from the external database will remain in the local database with a note replacing the respective external ID stating a deletion has occurred.

4. *Services and Clients*

4.1 The core capabilities of the system are named: KbUpdate, ExplanationGetWS, DataPush, and StatusReportWS.

4.2 KbUpdate will be a client application that will allow for the transfer, replacement, or deletion of assertion data to the external system.

4.2.1 The local ID generated for the assertions stored in the local database will be different from the external ID generated in response to a KbUpdate operation.

4.2.2 An assertion can only be successfully transferred once in order to avoid overwriting the external ID that was originally returned. This does not include replacements.

4.2.3 Feedback will be returned in the form of a message to the user either during or following the completion of an operation.

4.3 ExplanationGetWS will be a local port type WS that will allow for an external user to request assertion data from the local database.

4.3.1 For the external user to request assertion data from the local database, the external ID needs to be received.

4.3.2 ExplanationGetWS will be deployed on the local application Server.

4.4 DataPush will be a client application that is currently undefined.

4.5 StatusReportWS will be a local port type WS that will allow for an external user to request the status of the local system main capabilities.

4.5.1 The status returned to the external user consists of a status message and a DTG for the time when the operation was last performed.

4.5.2 A status will be assigned to each of the three key capabilities: DataPush, ExplanationGet, and KbUpdate.

4.5.3 The Web service status message can be a choice between being 'Available,' 'Unavailable,' or 'Unsupported.'

4.5.4 StatusReportWS will be deployed on the local application server.

5. *User Interface (UI)*

5.1 The UI will be a Web application based on a local client / server system that consists of web pages a user can access with a Web browser to execute the main system operations.

5.2 The use of the UI will be restricted to the local users.

5.3 The UI will consist of a sign-in page and a main menu page with connecting Web pages to facilitate executing five main operations: storing assertions, transferring assertions, editing assertions, setting the capability statuses, and managing user accounts.

5.3.1 The sign-in page will provide text fields to enter user sign-in data and a means to cancel the operation.

5.3.2 The main menu will allow the user to navigate to all five main operations described in 3.2.

5.3.3 The main menu will provide the user the means to log out of the system.

5.3.4 The main menu will display a list of the 10 most recently executed activities associated with core system capabilities along with their respective DTG.

5.3.5 The main menu will allow the user to check executed activities that are not among the 10 most recent.

5.3.6 The main menu will display the current capability statuses.

5.3.7 The Transfer Assertion and Edit Assertion pages will provide a browsing capability for the user to look up and select assertions.

5.3.8 The functions under the Store Assertion, Transfer Assertion, and Edit Assertion Web pages that either send assertion data to the external database, replace an existing assertion in the external database, or delete an existing assertion from the external database will execute the KbUpdate client.

6. *Application Server*

6.1 The application server must be JAVA Enterprise Edition compliant.

6.2 The application server must provide a runtime for Web-based applications.

6.3 The application server must allow for the deployment of Java Server Pages and Servlets.

D. NONFUNCTIONAL REQUIREMENTS

These requirements express specific nonfunctional attributes of the system's environment.

1. *Usability*

1.1 The intended user is one which should be comfortable with the basic functionality of a Web browser, which will allow for the manipulation of an assertion.

1.2 The software must be consistent with standard Web browser-based applications.

2. *Reliability*

2.1 The system services must be consistently available 24 hours a day during the project period.

2.2 Mean time to repair will be less than 1 hour.

2.2 Upon a system crash and recovery the database data must be safe and represent what it had prior to the crash.

3. *Portability*

3.1 The software must be portable, meaning that the program must be capable of being executed from a Java Archive file (.jar) and / or a Web application archive file (.war).

3.1.1 The portable executable files must be compatible with the Glassfish version 3 Application Server.

4. *Supportability*

4.1 The software must be capable of accommodating updates to the XSD, WSDL, or general changes.

4.2 The program code must be well commented to allow for future analysis and modification.

4.3 The program code must be well commented modular code to support testing procedures.

E. SUMMARY

In this chapter, we provided an SRS for our proposed distributed system. The SRS utilized use cases to demonstrate the functional requirements and those specific requirements not mentioned in the use cases were explained in sub-section C. The sequence diagrams were created to demonstrate the interaction of the processes which would be needed to execute the core operations and also the order in which they would perform. We used these requirements to help us design the proposed system that is covered in Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SYSTEM DESIGN

This chapter presents a design of the proposed distributed system based on the requirements defined in Chapter III. We have broken down the system into four main tiers to be developed as depicted in Figure 21. The four tiers represents the fundamental architecture that our client application and service will use for deployment.

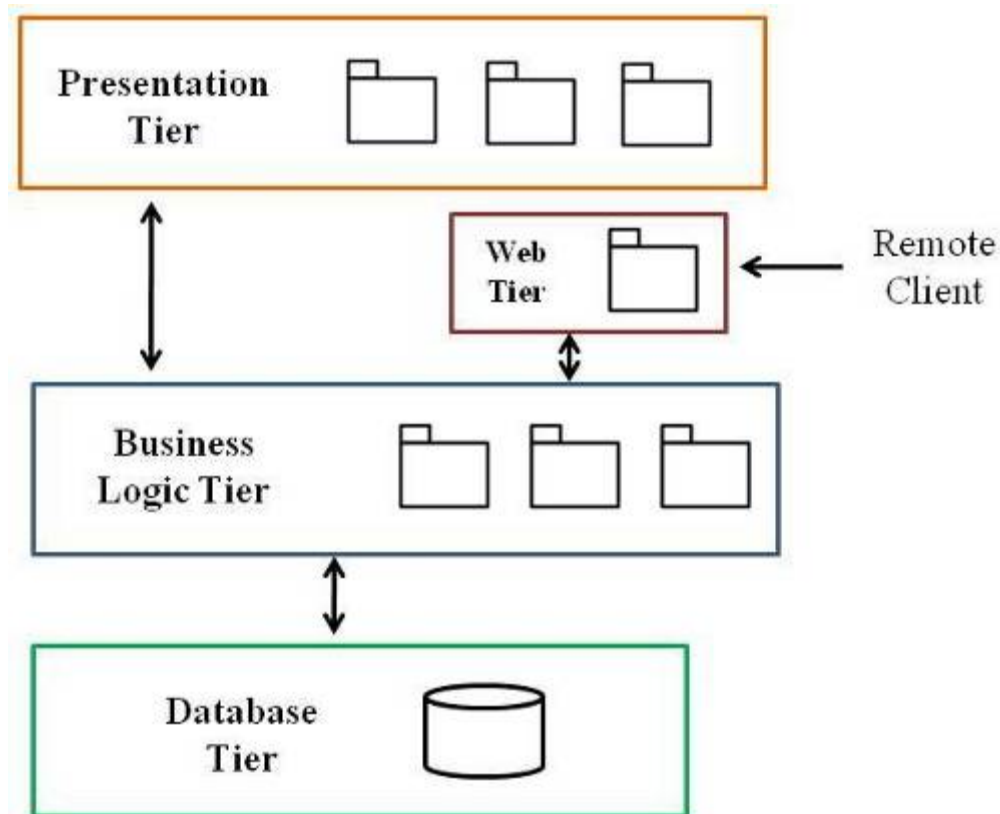


Figure 21. Four tiers of development

A. DATABASE TIER

Based on the stakeholder's requirement of a relational database mentioned in Chapter III, our first step was to build a conceptual data model (CDM) of the underlying database schema. A CDM is high-level visual representation that uses concepts such as *entities*, *attributes*, and *relationships* to describe a database application (Elmasri &

Navathe, 2007). Specifically, we will use an entity-relationship (ER) diagram, which is a popular type of CDM, to represent our database. To avoid redundancy while ensuring scalability, we normalized our database by using an algorithm that performs several steps to map an ER diagram into a Relational Database Schema (RDS) (Elmasri & Navathe, 2007). Figure 22 shows our ER diagram. To clarify what our ER diagram depicts, we will briefly discuss what the aforementioned concepts found in CDMs mean.

- Entities represent a thing (conceptual or physical) in the real world that exists independently. In our diagram entities are depicted as the green rectangular boxes. Entities that have an additional outline around the rectangle are called *weak entities*, which means that they do not have a *key attribute* and are related to a specific entity called the *owner entity* (Elmasri & Navathe, 2007).
- Attributes are specific properties of an entity. Attributes that are underlined are considered to be key attributes, which means that they are unique in value. Attributes are depicted as yellow circles. Multi-valued attributes are depicted with a double circle and represent data that can be greater than one set. Composite attributes are those that have additional sub-attributes (e.g., the sub-attributes for a person's address are: street number, street name, zip code, city, and state). Composite attributes are displayed with extra lines attached to their sub-attributes.
- Relationships can exist between entities and serve to describe an association between them. Relationships are depicted as orange diamonds and those having an additional outline around the diamond relate weak entities to its owner entity (Elmasri & Navathe, 2007). The number 1 or letters N and M adjacent to the relationship represent a cardinality ratio that depicts the maximum number of instances that an entity can participate in (e.g., 1:N means a one-to-many relationship).

1. ER Diagram

In our ER diagram, we listed **LocalUserAccounts** as a regular entity with the user email attribute as the key attribute. **LocalUserAccounts** is an owner entity that shares a 1:N relationship 'Sets_the' with the weak entity called **CapabilityStatus**. Since only one user (system administrator) can set the capability status this type of association between entities is called *partial participation* and is displayed as a single line connecting the relationship to the entities. **LocalUserAccounts** shares a 1:N relationship 'Manages' with itself, which represents the system administrator's ability to manage user accounts. **LocalUserAccounts** also shares a M:N (many-to-many) relationship 'Creates_an' with a regular entity **Assertion**, whose key attribute is Local_ID. The double lines extending from **LocalUserAccounts** to **Assertion** represents an association between entities called *total participation*, which means that all local users can create an assertion (Elmasri & Navathe, 2007). **Assertion** is an owner entity type that shares a 1:1 relationship with **DataSource**, **KbEvidence**, **KbSupport**, and a **KbClaim**. These weak entities are the major elements of an assertion that have been determined by the SCIL stakeholders thus far. **DataSource** is a new element that is currently under evaluation so we decided to include this element into our design.

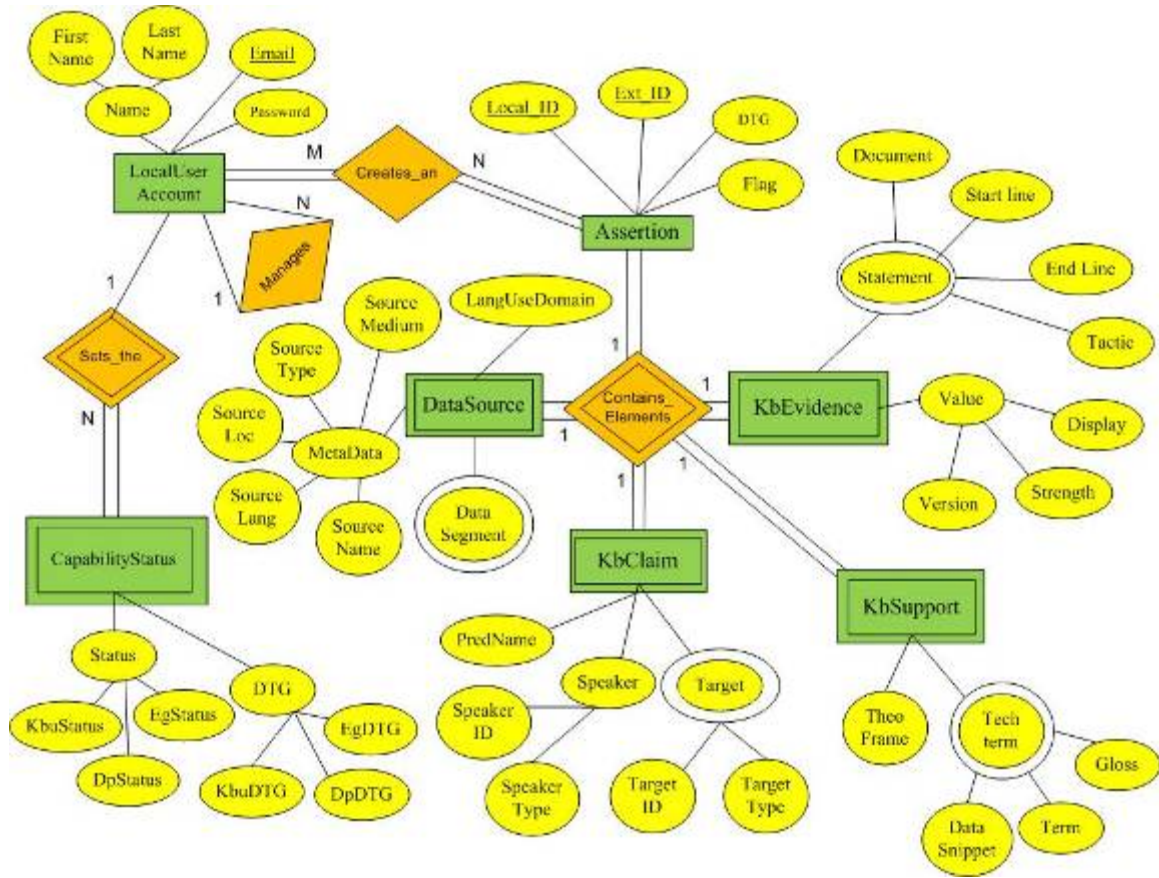


Figure 22. Database ER diagram

2. Relational Database Schema

Taking the ER diagram developed in our first step we used the following steps to map the ER diagram into a Relational Database Schema (RDS). The schema in turn guides the design of the database tables.

a. Mapping of Regular Entity Types

Figure 23 displays the RDS, which initially includes **LocalUserAccount** and **Assertion**. They are both assigned their own relation with email and localID, respectively, as their primary keys.

LocalUserAccount	firstName	lastName	<u>email</u>	password
Assertion	<u>localID</u>	ext_ID	dtg	flag

Figure 23. Mapping of regular entity types

b. Mapping of Weak Entity Types

In step 2, relations that are created for the weak entities, which include only the non-multivalued attributes are added to the RDS.

LocalUserAccount	firstName	lastName	<u>email</u>	password		
Assertion	<u>localID</u>	ext_ID	dtg	flag		
CapabilityStatus	kbuStatus	dpStatus	egStatus	kbuDTG	dpDTG	egDTG
KbSupport	theoFrame	localID				
KbClaim	predName	speakerID	speakerType	localID		
KbEvidence	version	strength	display	localID		
DataSource	sourceMedium	sourceType	sourceLoc	sourceLang		
	sourceName	langUseDomain	localID			

Figure 24. Mapping of weak entity types

c. Mapping of Binary 1:1 Relationship Types

In step 3, we merged all of the weak entity relations created in step 2 with their owner entity (Assertion) keeping localID as the primary key.

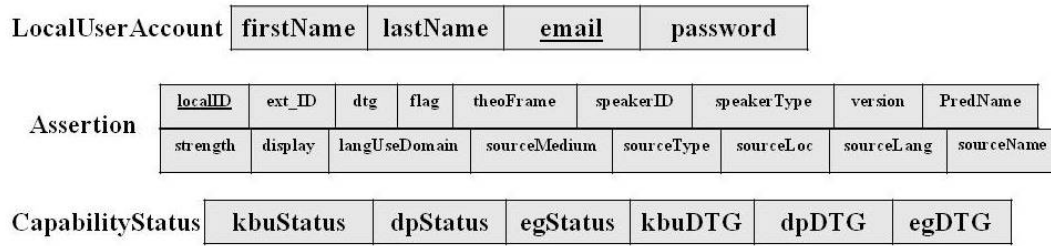


Figure 25. Mapping of binary 1:1 relationship types

d. Mapping of Binary 1:N Relationship Types

In step 4, we added foreign keys in both **CapabilityStatus** and **LocalUserAccount**; both of which refer to **LocalUserAccount**'s primary key: email.

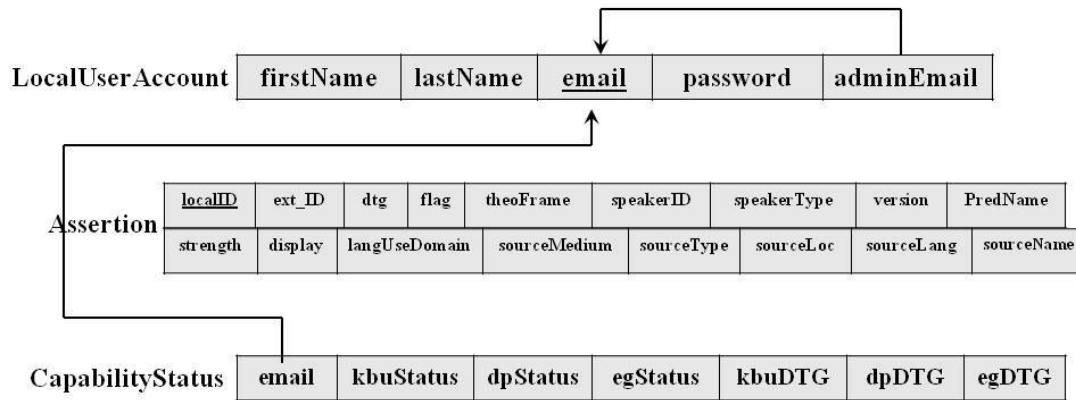


Figure 26. Mapping of binary 1:N relationship types

e. Mapping of Binary M:N Relationship Types

In step 5, we create a new relation called 'Creates_an' that houses two foreign keys referring to the primary keys in **LocalUserAccount** and **Assertion**.

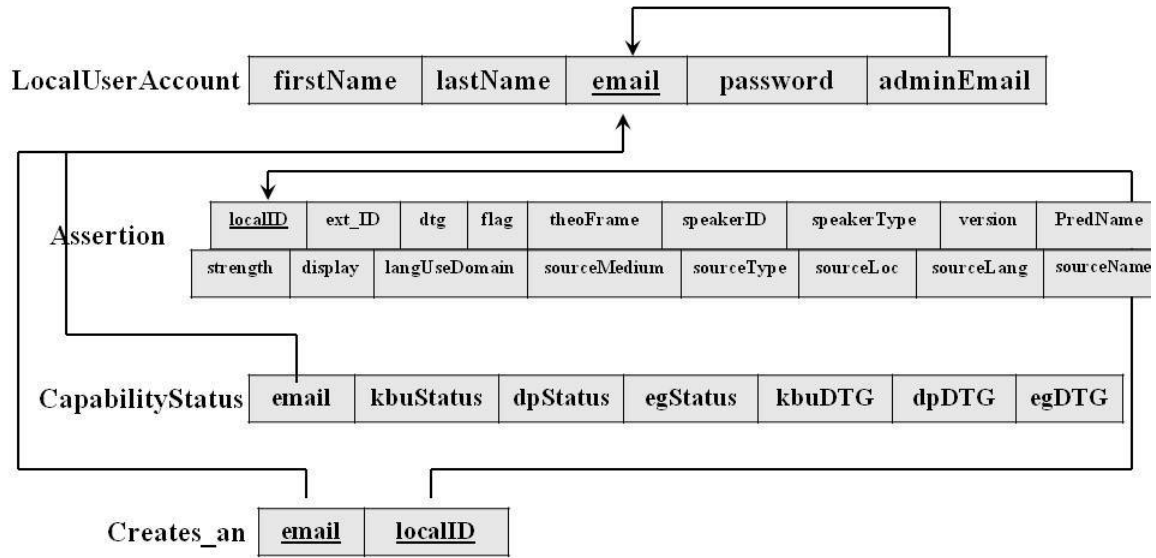


Figure 27. Mapping of binary M:N relationship types

f. Mapping of Multivalued Attributes

Finally, in step 6, we map the multivalued attributes to their own relations containing their respective attributes along with a foreign key that refers to the primary key of **Assertion**. Figure 30 depicts the proposed database tables that will be created for the system. The table names are the relation names on the side of the attributes and the column names are the attributes themselves. We populated the database with sample data and we will discuss that process in Chapter V.

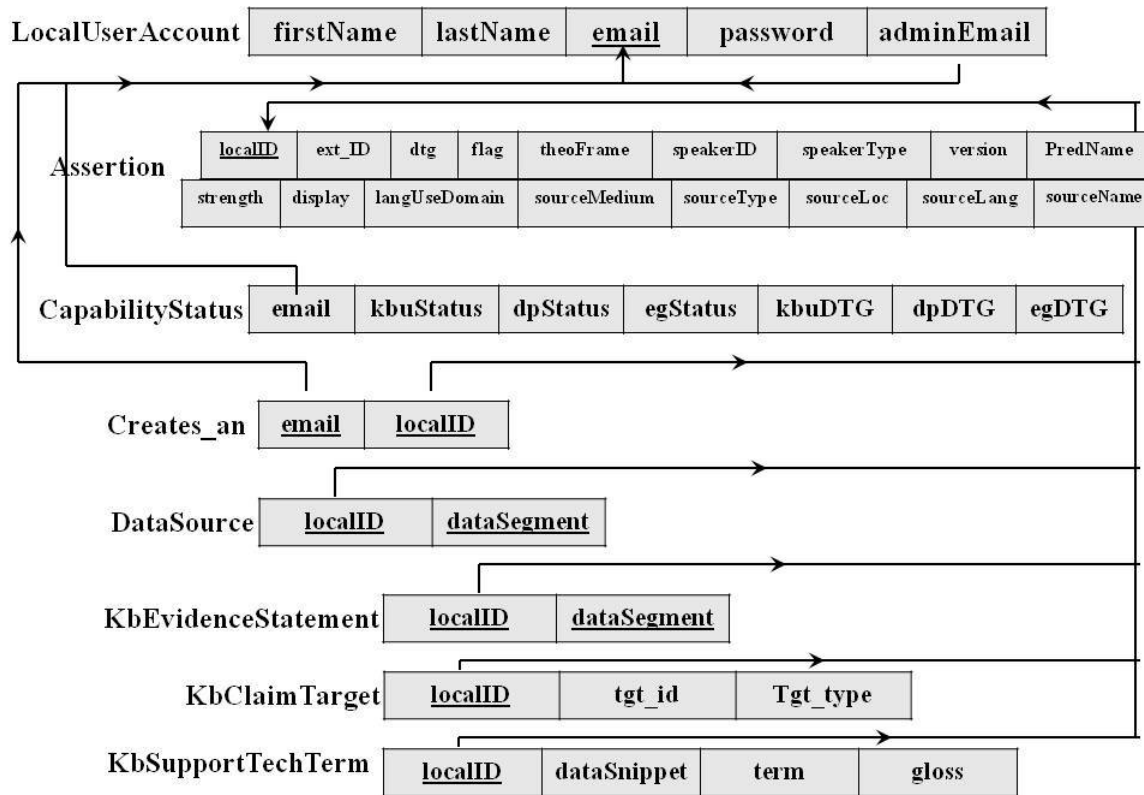


Figure 28. Mapping of multivalued attributes

B. BUSINESS LOGIC TIER

1. XML Schema

In Chapter II, we discussed XML schemas and their significance. In this section we present aspects of the XML schema related to our client and services. The schema was generated by Richard Tong, a representative from IARPA-Scientific, Engineering and Technical Assistance branch. As of May 4, 2010 the current version of the schema is 1.2. We modified and renamed the schema in order to accurately reflect the structure of the assertions agreed on by the UMD stakeholders. We also employed two versions of the schema, one for our Web services and one for our client. By doing so, we were able to remove the XML syntax that did not pertain to the respective operations, which made the lines of XML easier to manage and sped up the program compilation process. The

StatusReportWS and ExplanationGetWS services utilize UMDSchema_12b.xsd. The KbUpdate client uses STEPSchema_12b.xsd. In the Appendix, we have listed one schema that includes the data types and elements defined in both versions.

a. *StatusReportWS*

The schema element StatusReportResponseMsgPart, shown in Figure 29, represents the service response to the incoming request. It is constructed with two sub-elements: Metadata and Payload.

```

<xs:element name="StatusReportResponseMsgPart">
  <xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Metadata">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="MessageId" type="xs:string">
              <xs:annotation>
            </xs:element>
            <xs:element name="RequestorId" type="xs:string">
              <xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Payload">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="StatusReturnBundle" type="stepd:StatusReturnBundle"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Figure 29. XML schema StatusReportResponseMsgPart

Metadata is composed of the message and requestor identification that is submitted in the request and is subsequently returned. The Payload element holds a sequence of sub-elements and types, including StatusReturnBundle, that eventually provide the capability status.

Figure 30 shows the ServiceStatusReport type that holds the individual client and service capability status. Although not depicted for brevity, each sub-element capability is composed of a State and a LastDtg. The State is of type ServiceState that is

the simple type described on the top. LastDtg is of type dateTime, which is an integer-valued year, month, day and time structure (W3C, 2004).

```
<xs:simpleType name="ServiceState">
  <xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="available"/>
    <xs:enumeration value="unavailable"/>
    <xs:enumeration value="unsupported"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="ServiceStatusReport">
  <xs:annotation>
  <xs:sequence>
    <xs:element name="KbUpdateCapability">
      <xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="State" type="stepd:ServiceState"/>
          <xs:element name="LastDtg" type="xs:dateTime"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="DataPushCapability">
    <xs:element name="ExplanationGetCapability">
  </xs:sequence>
</xs:complexType>
```

Figure 30. XML schema ServiceState and ServiceStatusReport

b. ExplanationGetWS

Figure 31 shows the ExplanationGetResponseMsgPart, which is the root element that comprises the message to be returned to the client. In similar fashion to StatusReport, it contains a Metadata and Payload. The Payload houses an ExplanationReturnBundle that, although not depicted, also has a set of Metadata and Payload. This sub-Metadata holds the AssertionId that references the particular data to be obtained from our database. The sub-Payload subsequently contains the main elements that make up an assertion that are listed under the complex type KbAssertion in Figure 32. Although, in concept, the key elements to an assertion remain as the claim, evidence, and support, the KbAssertion is made up of two elements, the AssertionMetadata and

AssertionContext. However, AssertionClaim, AssertionSupport, and AssertionEvidence are of separate individual types that are further defined in Figures 34, 35, and 36.

```

<xs:element name="ExplanationGetResponseMsgPart">
  <xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Metadata">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="MessageId" type="xs:string">
              <xs:annotation>
            </xs:element>
            <xs:element name="RequestorId" type="xs:string">
              <xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Payload">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ExplanationReturnBundle" type="stepd:ExplanationReturnBundle"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Figure 31. XML schema ExplanationGetResponseMsgPart

AssertionMetadata is composed of three elements including AssertionFlag, which is also defined in the schema. The AssertionFlag is designed as an enumeration of the value: 'Public' and 'Private,' which represent viewing authorization labels for the assertion. AssertionContext has an element called DataSet that is of type DataSource. Figure 33 shows DataSource. The elements and types defined within KbClaim, KbEvidence, and KbSupport were specified by the UMD teams.

```

<xs:complexType name="KbAssertion">
  <xs:sequence>
    <xs:element name="AssertionMetadata">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="AssertionId" type="xs:string">
          <xs:element name="AssertionDtq" type="xs:dateTime">
          <xs:element name="AssertionFlag" type="stepd:AssertionFlag">
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="AssertionContext">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="LanguageUseDomain" type="xs:string" default="Persuasion Attempt">
          <xs:element name="DataSet" type="stepd:DataSource">
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="AssertionClaim" type="stepd:KbClaim"/>
    <xs:element name="AssertionEvidence" type="stepd:KbEvidence"/>
    <xs:element name="AssertionSupport" type="stepd:KbSupport"/>
  </xs:sequence>
</xs:complexType>

```

Figure 32. XML schema KbAssertion

```

<xs:complexType name="DataSource">
  <xs:annotation>
  <xs:sequence>
    <xs:element name="DataMetadata">
      <xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="SourceName" type="xs:string">
          <xs:element name="SourceLocation" type="xs:anyURI">
          <xs:element name="SourceLanguage" type="xs:language">
          <xs:element name="SourceType" type="xs:string">
          <xs:element name="SourceMedium" type="xs:string">
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="DataSegment" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="SourceDataSegment" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

Figure 33. XML schema DataSource

```

<xs:complexType name="KbClaim">
  <xs:sequence>
    <xs:element name="PredicateClaim">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="PredicateName" type="xs:string" default="Persuasion Attempt"/>
          <xs:element name="Speaker">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Entity" type="stepd.Entity"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="Target">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="Entity" type="stepd.Entity" maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="type" type="xs:string" default="directed">
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

Figure 34. XML schema KbClaim

```

<xs:complexType name="KbEvidence">
  <xs:sequence>
    <xs:element name="EvidenceValue" type="xs:string">
    <xs:element name="EvidenceStatement" type="stepd.EvidenceStatement"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="EvidenceStatement">
  <xs:sequence>
    <xs:element name="ConstituentMultiset">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="PersuasionTactic" maxOccurs="unbounded">
            <xs:complexType>
              <xs:group ref="stepd.PersuasionTacticGroup"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:group name="PersuasionTacticGroup">
  <xs:all>
    <xs:element name="tactic" type="xs:string"/>
    <xs:element name="startline" type="xs:integer"/>
    <xs:element name="endline" type="xs:integer"/>
    <xs:element name="doc" type="xs:string"/>
  </xs:all>
</xs:group>

```

Figure 35. XML schema KbEvidence

```

<xs:complexType name="KbSupport">
  <xs:sequence>
    <xs:element name="TheoreticalFrame" type="xs:string"/>
    <xs:element name="TechnicalTerm" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="TechnicalTermGloss" type="xs:string"/>
          <xs:element name="DataSnippet" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="term" type="xs:string" default="redefinition"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

Figure 36. XML schema KbSupport

c. *KbUpdate Client*

KbUpdate is a client and the KbUpdateRequestMsgPart, pictured below, is the root element comprising our request message to the external system. The request is sent to the external system to update the external knowledge base with assertion data. The request can either add a new assertion, delete a existing assertion, or replace an existing assertion from the external database. As depicted, the request contains Metadata and Payload. The Metadata contains a message identification and the requesting team identification. The Payload is composed of an option of element bundles representing the desired type of update. Figure 38 displays the AssertionAddBundle. The AssertionAddBundle element has Metadata that addresses the number of assertions to be updated and is called AssertionCount. The Payload contains the element Assertion and references the same elements and types previously defined in the ExplanationGet description.

```

<xs:element name="KbUpdateRequestMsgPart">
  <xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Metadata">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="MessageId" type="xs:string">
            <xs:element name="TeamId" type="stepd:TeamId">
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Payload">
        <xs:complexType>
          <xs:choice>
            <xs:element name="AssertionAddBundle" type="stepd:AssertionAddBundle"/>
            <xs:element name="AssertionReplaceBundle" type="stepd:AssertionReplaceBundle"/>
            <xs:element name="AssertionDeleteBundle" type="stepd:AssertionDeleteBundle"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Figure 37. XML schema KbUpdateRequestMsgPart

```

<xs:complexType name="AssertionAddBundle">
  <xs:annotation>
  <xs:sequence>
    <xs:element name="Metadata">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="AssertionCount">
            <xs:annotation>
            <xs:simpleType>
              <xs:restriction base="xs:integer">
                <xs:minInclusive value="1"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Payload">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Assertion" type="stepd:KbAssertion" maxOccurs="unbounded">
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

```

Figure 38. XML schema AssertionAddBundle

The AssertionDeleteBundle (Figure 39) also contains Metadata and Payload elements. The Metadata is similar to the AssertionAddBundle, however now the AssertionCount refers to the number of assertions to be deleted. The Payload on the other hand refers to the particular identification number of the assertion(s) to be deleted. The AssertionReplaceBundle (Figure 40) uses the same Metadata and Payload structure where the AssertionCount refers to the number of pairs of assertions to be swapped. The Payload's sub-elements describe both the identification number of the assertion from the external database to be replaced and the particular assertion from the local database that will replace it. Again, this Assertion is of type KbAssertion described in the ExplanationGet.

```
<xs:complexType name="AssertionDeleteBundle">
  <xs:annotation>
  <xs:sequence>
    <xs:element name="Metadata">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="AssertionCount">
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Payload">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="AssertionId" type="xs:string" maxOccurs="unbounded">
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:sequence>
</xs:complexType>
```

Figure 39. XML schema AssertionDeleteBundle

```

<xs:complexType name="AssertionReplaceBundle">
  <xs:annotation>
  <xs:sequence>
    <xs:element name="Metadata">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="AssertionCount">
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Payload">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="AssertionPair" maxOccurs="unbounded">
              <xs:annotation>
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="ReplacedAssertionId" type="xs:string">
                  <xs:element name="Assertion" type="stepd:KbAssertion">
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:sequence>
</xs:complexType>

```

Figure 40. XML schema AssertionReplaceBundle

2. Class Diagrams

Earlier, we showed the readers our proposed domain model that provided a conceptual perspective of the system as a whole. In this section, we used the same UML modeling notation to provide class diagrams that provide the software-focused structural view of the system, with the main elements indicated by boxes. Each box contains three sections: name (top section), attributes (middle section), and functions (bottom section). In the following sections, we will walk through the class diagrams for our StatusReport service, ExplanationGet service, and KbUpdate Client.

a. *StatusReport*

Figures 41 and 42 display the class diagram for the StatusReport service. Class StatusReport implements the client interface SR_PortType and is dependent on the request message received that is of the type StatusReportRequestMsgPart. The message will contain both a message ID and a requester ID (Metadata), which are eventually returned to the client as part of the StatusReportRequestMsgPart along with the Payload. This Payload class refers to a StatusReturnBundle class, which also has a sub-class called Payload that refers to the class ServiceStatusReport. ServiceStatusReport has the three subclasses, which hold the methods that work closest to the Database class in order to get the information from the database. KbUpdateCapability, ExplanationGetCapabiltiy, and DataPushCapability refer to the ServiceState class, which holds an enumeration of string literals: *Available*, *Unavailable*, or *Unsupported*. Finally the Database class implements the Connection interface, which provides the methods for interacting with our database. The Connection class is a Java 2 platform object (http://download.oracle.com/docs/cd/E17476_01/javase/1.3/docs/api/java/sql/Connection.html).

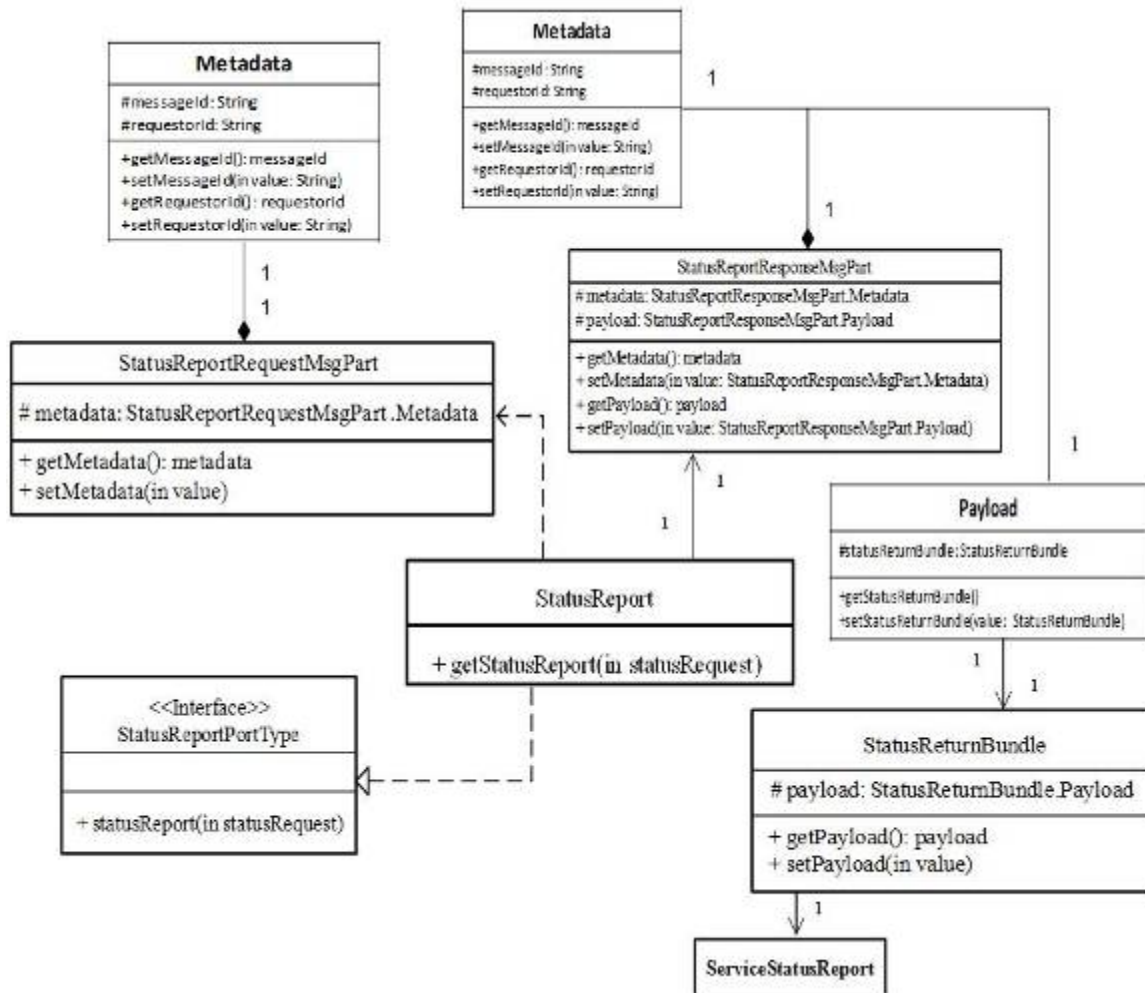


Figure 41. StatusReport class diagram part I

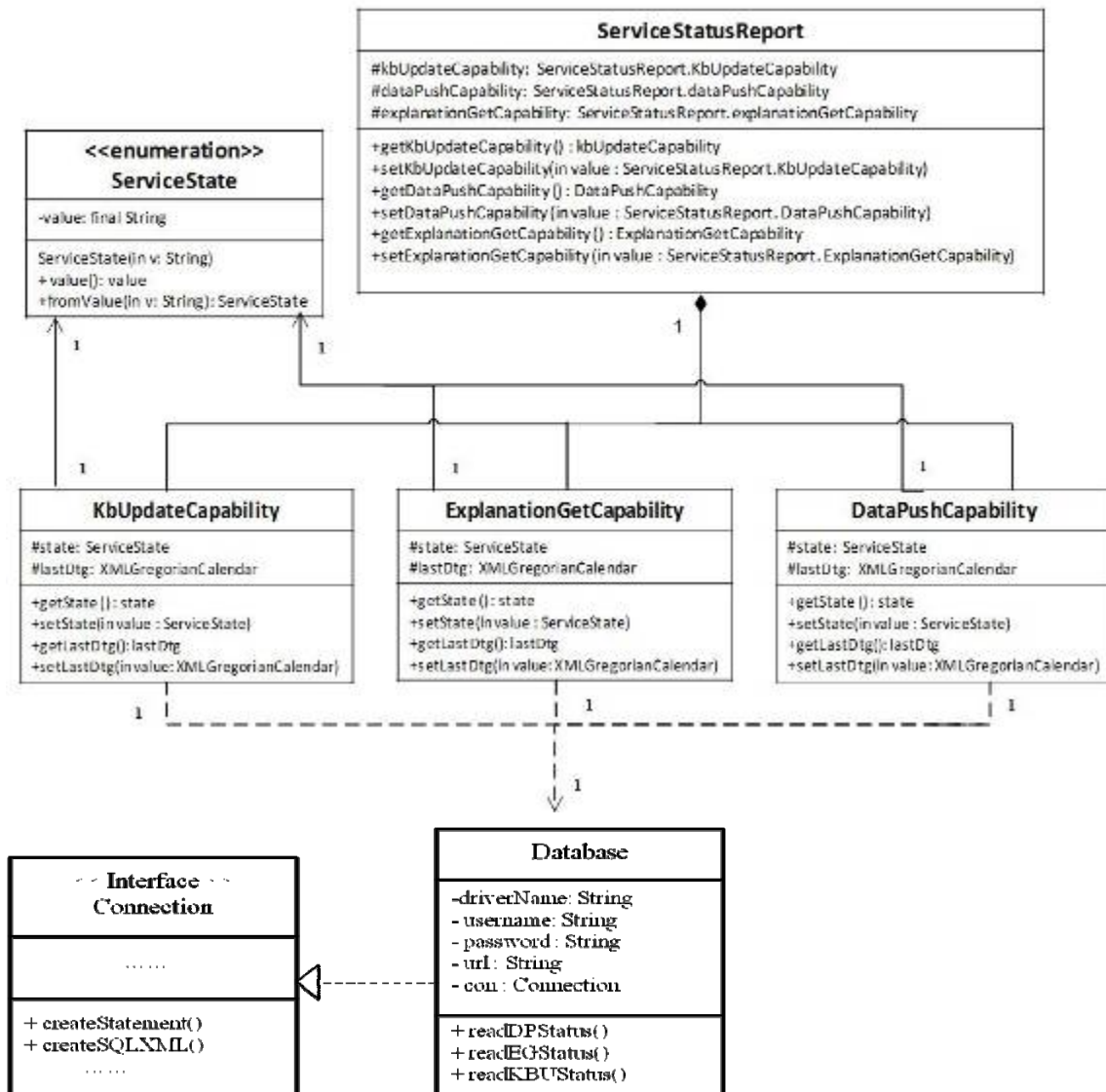


Figure 42. StatusReport class diagram part II

b. ExplanationGet

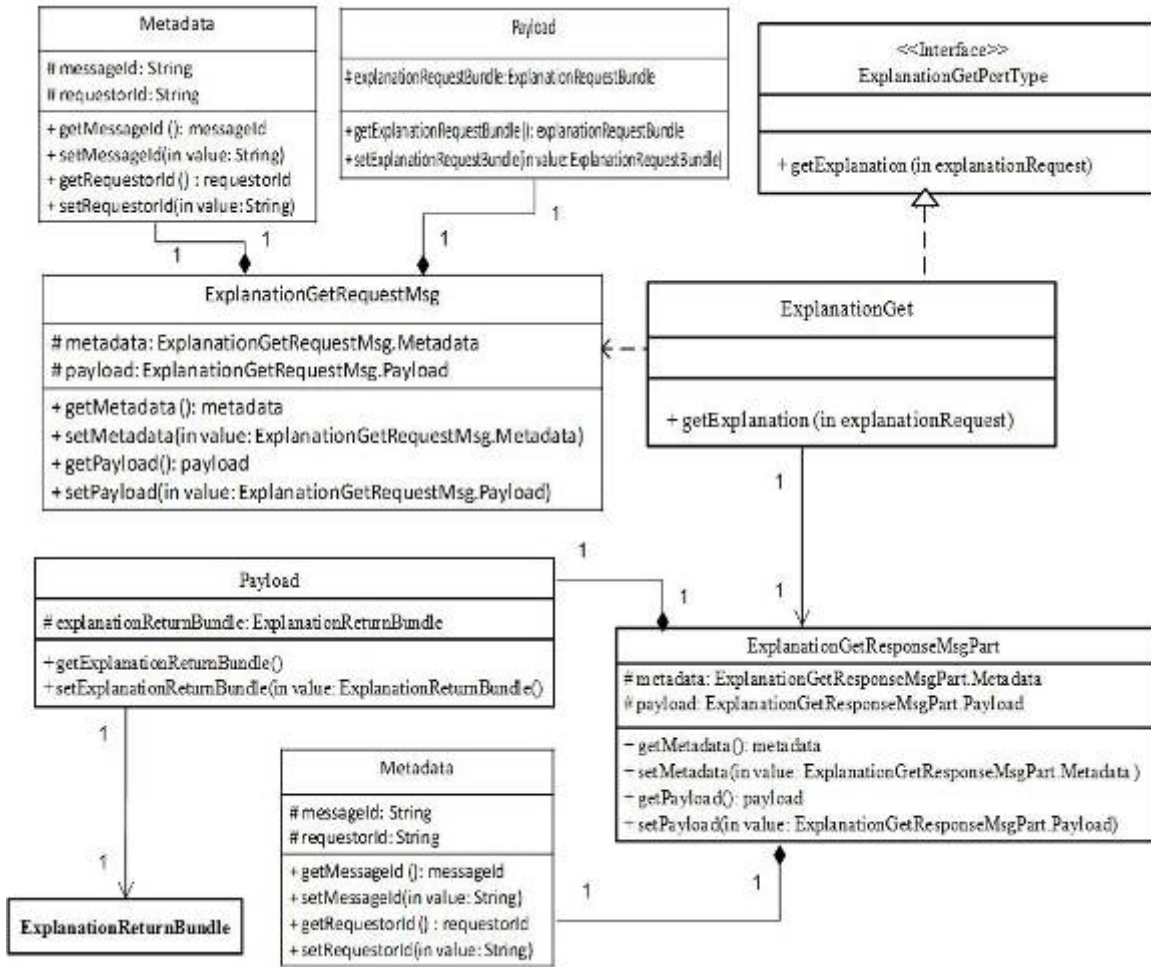


Figure 43. ExplanationGet class diagram part I

Figures 43 through 46 display **ExplanationGet**. **ExplanationGet** implements the **ExplanationGetPortType** and is dependent on the request message of type **ExplanationGetRequestMsg**. Similar to **StatusReport** the same **Metadata** that comes in the message is returned to the client along with the **Payload**. The **Payload** refers to an **ExplanationReturnBundle** class that contains a **Payload** and **Metadata**. The **Payload** is actually a list of assertions, which are instances of the **KbAssertion** class. **KbAssertion** has two components called **AssertionMetadata** and **AssertionContext**. **AssertionMetadata** calls the **DataBase** class directly while **AssertionContext** refers to **DataSource**.

KbAssertion also refers to KbClaim, KbSupport, and KbEvidence shown on Figure 45. Each of these classes either contain instances of classes that call the DataBase class or call the DataBase class directly in order to obtain the assertion referenced by the assertion ID used in Payload.

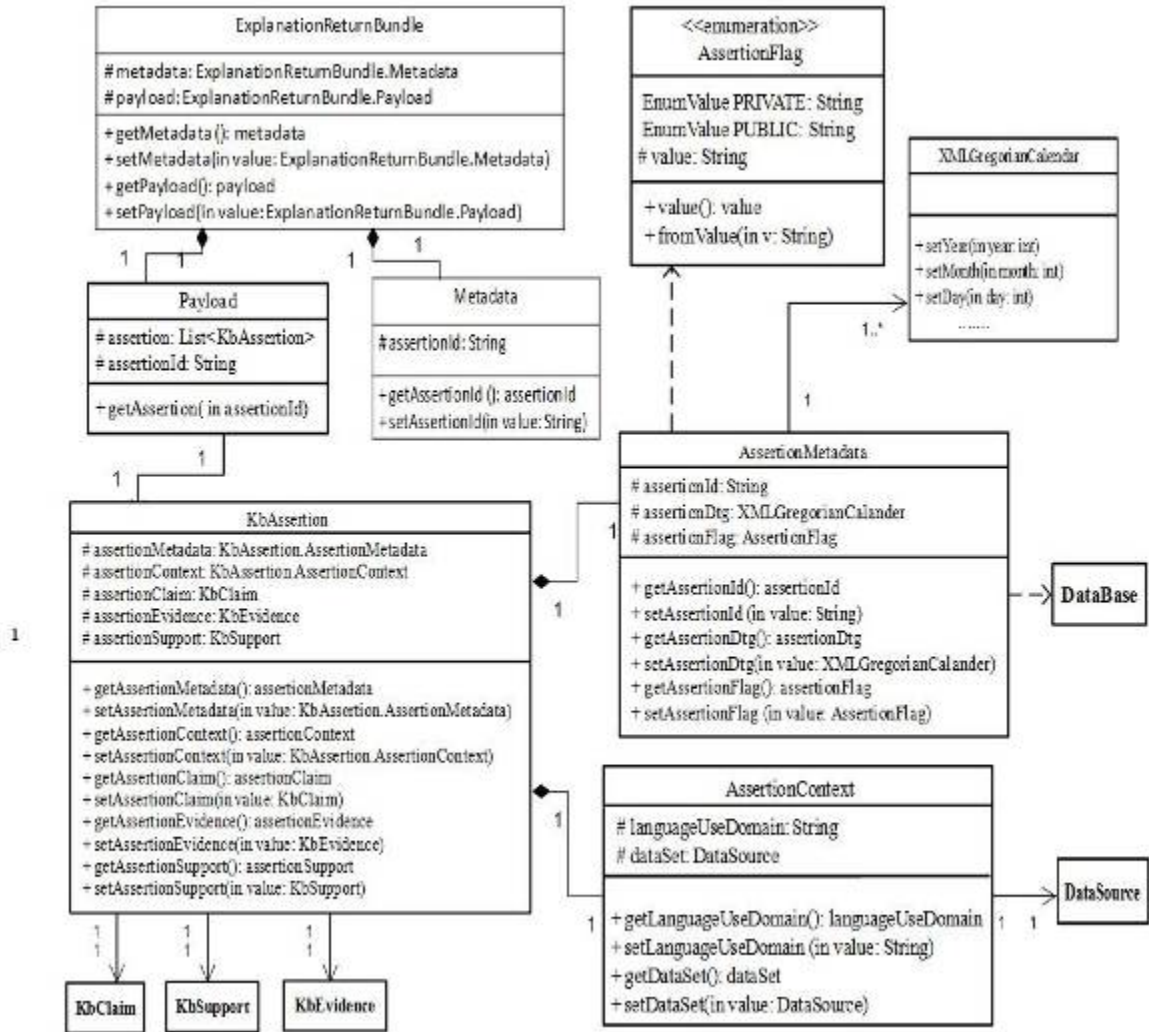


Figure 44. ExplanationGet class diagram part II

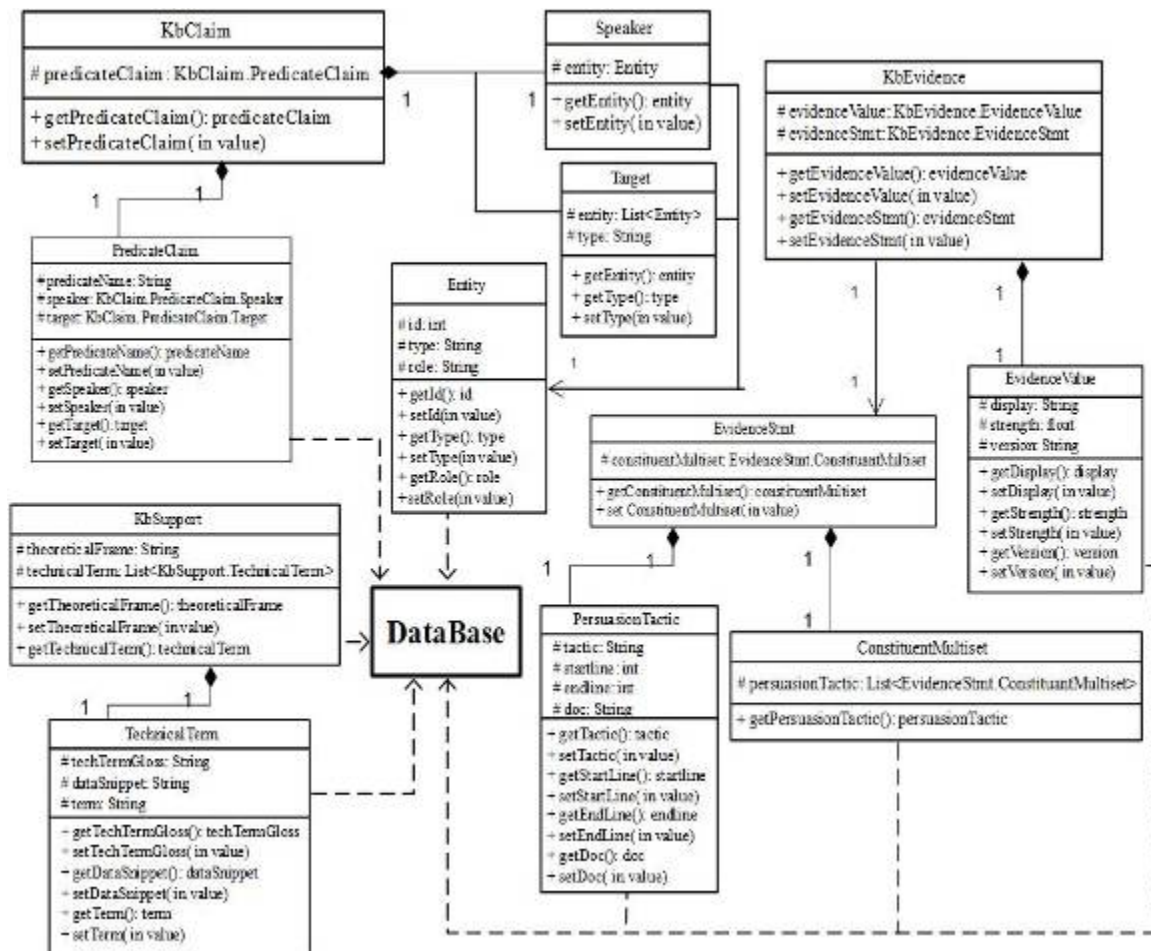


Figure 45. ExplanationGet class diagram part III

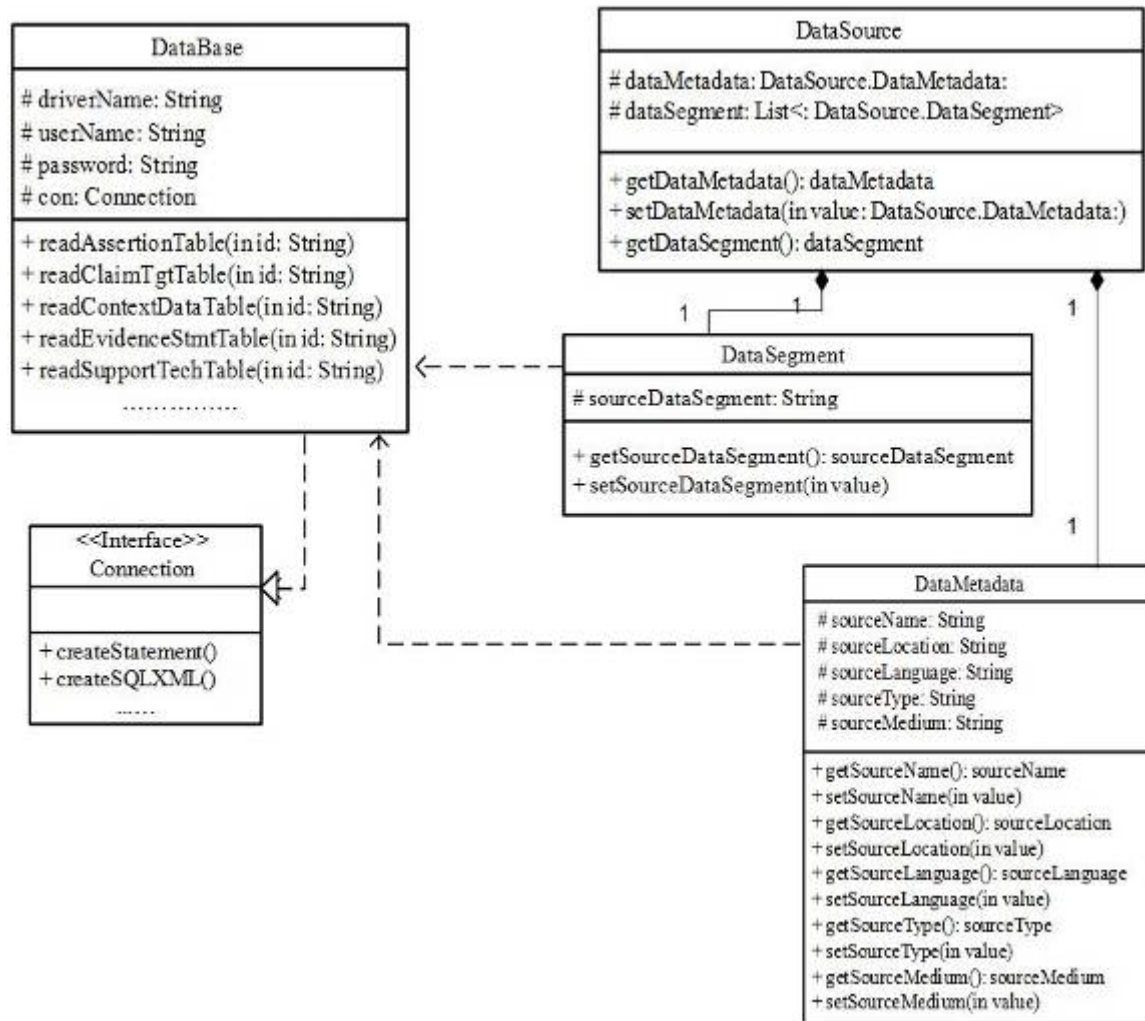


Figure 46. ExplanationGet class diagram part IV

c. *KbUpdate*

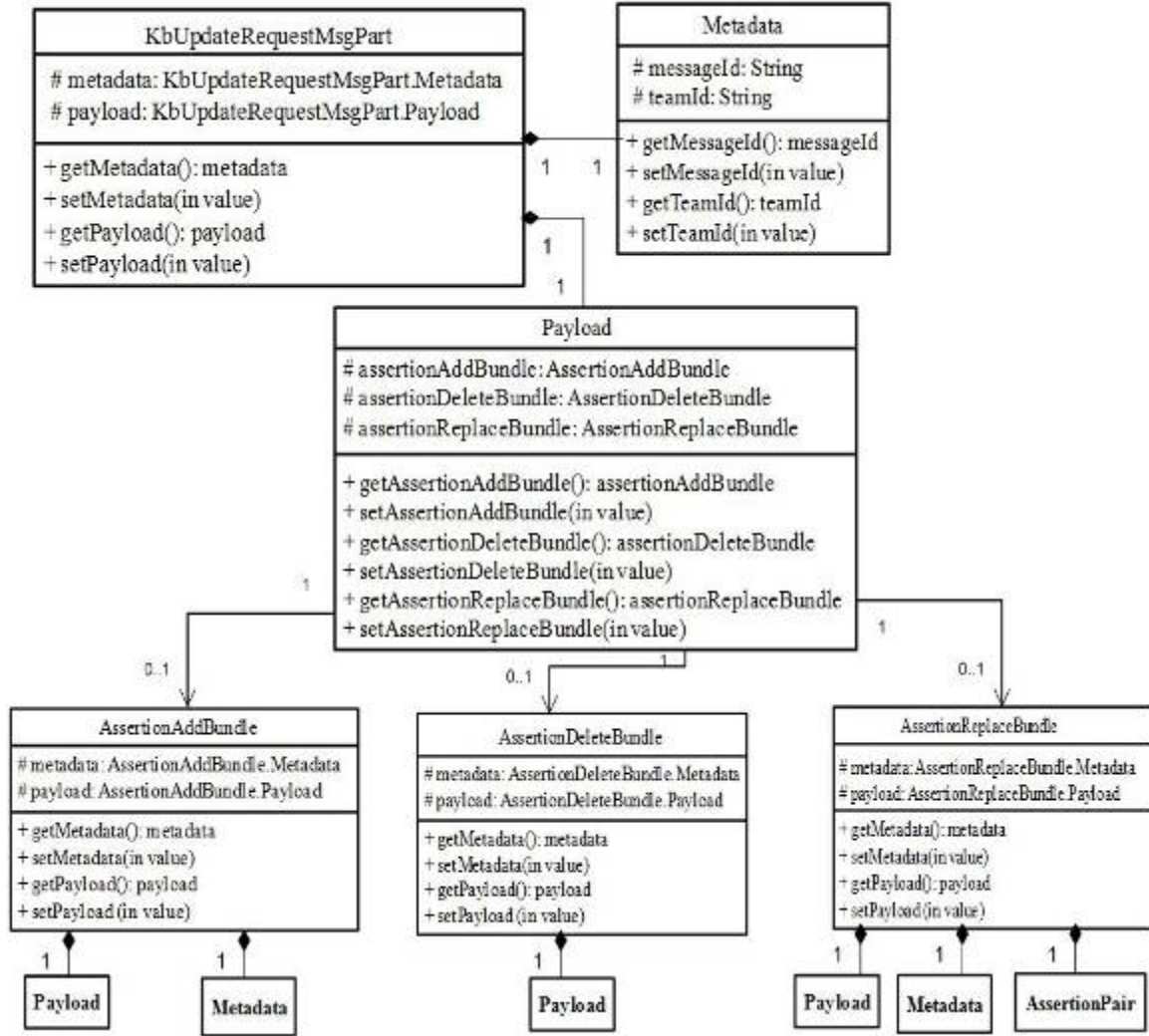


Figure 47. *KbUpdate* class diagram part I

Figure 47 above shows the first half of the *KbUpdate* client class diagram. Our request message contains *Metadata* and *Payload*. The *Payload* holds a bundle depending on the operation desired. Each of the bundle classes have their own components that continue in Figure 48. The *AssertionAddBundle* and *AssertionReplaceBundle* eventually use the same class path structure to obtain the assertion data as the *ExplanationGet* service. *AssertionDeleteBundle* only has one class that handles the list of assertion IDs to be removed from the external database.

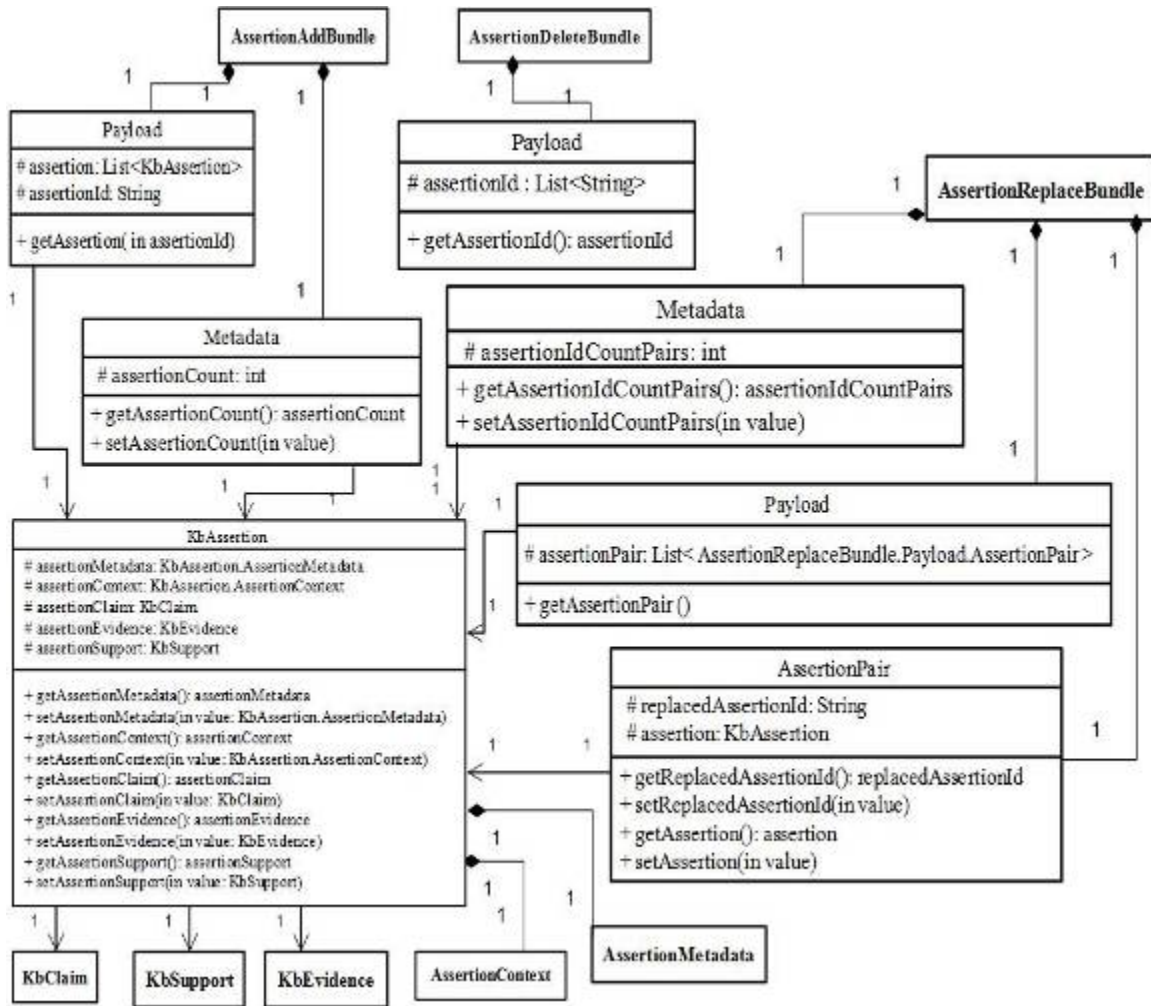


Figure 48. KbUpdate class diagram part II

C. WEB TIER

In this section we will describe the design of the WSDL documents for our client and services. We employed two WSDL files; one for KbUpdate called STEPPortTypes_12b.wsdl and another for both ExplanationGet and StatusReport called UMDServices_12b.wsdl. We did this because our prototype, discussed in Chapter V, consists of two separate applications; one for the Web services and the other for the Web client. The WSDL files were modified and renamed from the documents originally generated as stepGovServices_12.wsdl and stepPortTypes_12.wsdl (Tong, 2009). The

binding and *service* sections of the UMD Services WSDL were left for us to define since they pertained to our network protocol details. Sub-sections 1 and 2 show the main sections of the WSDL files.

1. UMD Services

```
<wsdl:types>
  <xs:schema>
    <xs:import namespace="http://www.iarpa.gov/SCIL/STEP_Schema" schemaLocation="UMDSchema_12b.xsd"/>
  </xs:schema>
</wsdl:types>
<wsdl:message name="ExplanationGetRequest">
  <wsdl:part name="omnia" element="stepd:ExplanationGetRequestMsgPart"/>
</wsdl:message>
<wsdl:message name="ExplanationGetResponse">
  <wsdl:part name="omnia" element="stepd:ExplanationGetResponseMsgPart"/>
</wsdl:message>
<wsdl:message name="StatusReportRequest">
  <wsdl:part name="omnia" element="stepd:StatusReportRequestMsgPart"/>
</wsdl:message>
<wsdl:message name="StatusReportResponse">
  <wsdl:part name="omnia" element="stepd:StatusReportResponseMsgPart"/>
</wsdl:message>
```

Figure 49. UMD services WSDL part I

Figure 49 above shows the *types* and *messages* sections. The Schema document was generated separately and is referred to in the types section by name. The messages section shows the part name 'omnia' to represent the name of the request and response messages to be used between service and client. Figure 50 shows StatusReportPortType and ExplanationGetPortType as the *portTypes* to be used by the client. The message types previously defined are referenced here as the input and output messages.

```

<wsdl:portType name="StatusReportPortType">
  <wsdl:documentation>Core service. Hosted at the Performer site.</wsdl:documentation>
  <wsdl:operation name="StatusReport">
    <wsdl:documentation>Request-Response. STEP AppServer requests a status report from Performer team service.</wsdl:documentation>
    <wsdl:input message="steps:StatusReportRequest"/>
    <wsdl:output message="steps:StatusReportResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="ExplanationGetPortType">
  <wsdl:documentation>Optional service. Hosted at the Performer site. </wsdl:documentation>
  <wsdl:operation name="ExplanationGet">
    <wsdl:documentation>Request-Response. STEP AppServer requests an explanation from Performer team service.</wsdl:documentation>
    <wsdl:input message="steps:ExplanationGetRequest"/>
    <wsdl:output message="steps:ExplanationGetResponse"/>
  </wsdl:operation>
</wsdl:portType>

```

Figure 50. UMD services WSDL part II

```

<wsdl:binding name="StatusReportSoapBinding" type="steps:StatusReportPortType">
  <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="StatusReport">
    <soap12:operation soapAction="statusReport" soapActionRequired="true"/>
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="ExplanationGetSoapBinding" type="steps:ExplanationGetPortType">
  <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="ExplanationGet">
    <soap12:operation soapAction="explanationGet" soapActionRequired="true"/>
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

```

Figure 51. UMD services WSDL part III

Figure 51 shows the binding between the service interface descriptions (StatusReportPortType and ExplanationGetPortType) to the their respective service implementations (StatusReport and ExplanationGet). Note that we are using SOAP over HTTP.

Finally, Figure 52 shows the service section combining the previously defined port names and binding names to our network address creating the end-point.

```
<wsdl:service name="UMD_Service">
  <wsdl:port name="StatusReport" binding="stepn:StatusReportSoapBinding">
    <soap12:address location="http://10.3.21.97:8080/UMD/UMD_Service"/>
  </wsdl:port>
  <wsdl:port name="ExplanationGet" binding="stepn:ExplanationGetSoapBinding">
    <soap12:address location="http://10.3.21.97:8080/UMD/UMD_Service"/>
  </wsdl:port>
</wsdl:service>
```

Figure 52. UMD services WSDL part IV

2. UMD Client

Figure 53 shows the KbUpdate WSDL. Aside from the name changes in the messages, ports, and binding sections that reflect the KbUpdate operation, the structure of this document is the same as the UMD Services WSDL. The service section provides the combination of the binding and port names and also specifies the network address of the external service.

```

<wsdl:types>
  <xs:schema>
    <xs:import namespace="http://www.iarpa.gov/SCIL/STEP_Schema" schemaLocation="stepSchema_12b.xsd"/>
  </xs:schema>
</wsdl:types>
<wsdl:message name="KbUpdateRequest">
  <wsdl:part name="omnia" element="stepd:KbUpdateRequestMsgPart"/>
</wsdl:message>
<wsdl:message name="KbUpdateResponse">
  <wsdl:part name="omnia" element="stepd:KbUpdateResponseMsgPart"/>
</wsdl:message>
<wsdl:portType name="KbUpdatePortType">
  <wsdl:documentation>Core service. Hosted on the STEP AppServer.</wsdl:documentation>
  <wsdl:operation name="KbUpdate">
    <wsdl:documentation>Request-Response. Performer team service pushes assertions to STEP AppServer.</wsdl:documentation>
    <wsdl:input message="steps:KbUpdateRequest"/>
    <wsdl:output message="steps:KbUpdateResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:import namespace="http://www.iarpa.gov/SCIL/STEP_PortTypes" location="stepPortTypes_12b.wsdl"/>
<wsdl:binding name="KbUpdateSoapBinding" type="steps:KbUpdatePortType">
  <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="KbUpdate">
    <soap12:operation soapAction="kbUpdate" soapActionRequired="true"/>
    <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="StepAppServerService">
  <wsdl:port name="KbUpdate" binding="stepn:KbUpdateSoapBinding">
    <soap12:address location="http://10.3.21.1:8084/STEP/StepAppServerService"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Figure 53. UMD client WSDL

D. PRESENTATION TIER

We now shift our focus to the presentation tier, which contains the components that create and execute the UI necessary for our users to communicate with the other tiers in the system.

Based on the requirements from Chapter III, a closed Web application is necessary to implement the UI. Using a closed Web application to serve as our UI means that only our users will be presented with the Web pages to interface with and execute the system functionality; similar to how a person would use the Internet to manage his/her

personal online banking account. The UMD performer team users will perform the system operations by using the Web pages to interface with the local system's business logic through an HTTP connection. The interaction is initiated with a request for a Web page by the user's Web browser. The local system's Web server will return the requested Web pages to the users for execution. The File system manages the HTML files for the Web server and the Application server executes the server-side business logic (Conallen, 2003). Figure 54 shows the interplay among the components.

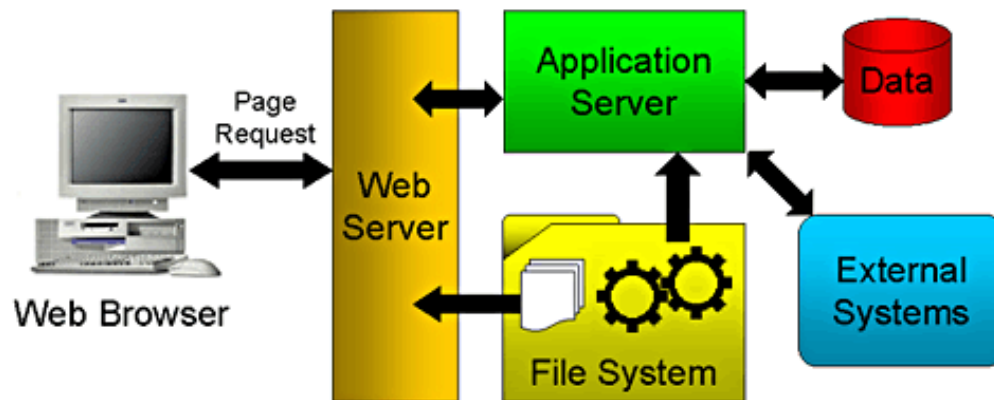


Figure 54. Web architecture (From Booch, 2001)

The remainder of this section is separated into two sections. In Section 1, we used the Microsoft PowerPoint 2007® application to develop a conceptual design of the UI Web pages. We also discuss the intended functionality behind each Web page and step through what a user would encounter while using the Web application. Section 2 shows the respective Web pages' UML class diagrams.

1. UI Web Pages

We begin our design of the UI with another UML modeling tool called an Activity Diagram. This diagram displays the workflow associated with our UI. As depicted in Figure 55, the basic structure of the UI allows a user to sign in and view the main menu. The main menu provides the user an option to perform one of several system operations. Finally, the user can sign out of the system when complete.

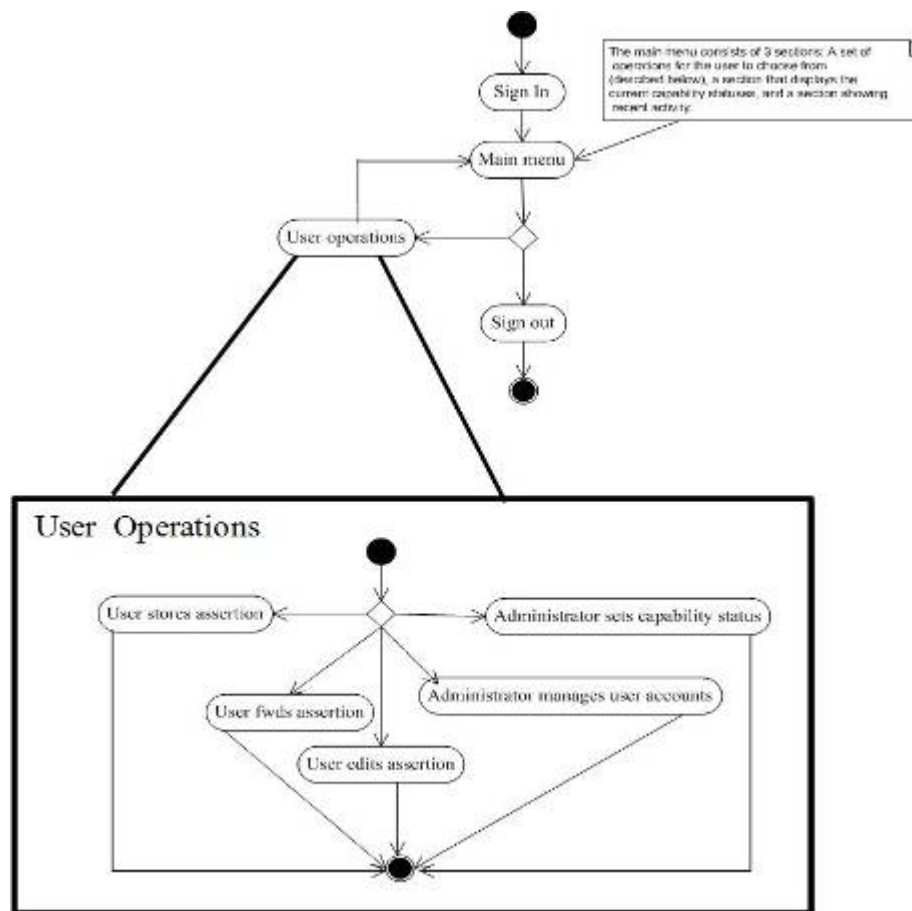


Figure 55. UI workflow

a. Sign In

The first step for our users would be to sign into our system through a sign in page that requires the user to input a username and password. Figure 56 displays our proposed design of what that page would look like.

The image shows a web page titled "SOCIO-CULTURAL CONTENT IN LANGUAGE UMD PERFORMER TEAM". Below the title, there are two input fields: "UserID" and "Password", each with a blue rectangular text box. At the bottom of the page, there are two yellow buttons: "Sign In" and "Cancel".

Figure 56. Sign-in Web page

b. Main Menu

Figure 57 represents a preliminary design of the main menu, which is displayed following user authentication. The main menu will display the status of the core system capabilities, show the ten most recent activities, provide the user a means to exit the system, and provide the user hyperlinks to navigate to the desired Web pages to perform the system functions. For clarity purposes, we designed the main menu image to show only four activities instead of ten. A DTG is associated with the activity. The newer and older buttons under the recent activity will allow the user to view additional activity not displayed. A color-labeled display on the top right corner shows the current state of the core capabilities. The buttons in yellow serve as hyperlinks to the Web pages.

UMD SCIL					Current Status: KbUpdate – Available ExplanationGet – Unavailable DataPush – Unsupported											
Home	Store assertion	Transfer assertion	Edit assertion	User accounts	Capability Status											
Recent Activity: <table border="1"> <thead> <tr> <th>Date & Time</th> <th>Activity</th> </tr> </thead> <tbody> <tr> <td>2010-01-10T09:35:12-05:00</td> <td>Capability Status queried</td> </tr> <tr> <td>2010-01-05T17:00:00-05:00</td> <td>Explanation for assertion A123Q4 queried</td> </tr> <tr> <td>2009-12-11T10:45:41-05:00</td> <td>Local assertion 2 forwarded and returned as A453X4</td> </tr> <tr> <td>2009-12-08T13:00:01-05:00</td> <td>Assertion 3 stored</td> </tr> </tbody> </table>							Date & Time	Activity	2010-01-10T09:35:12-05:00	Capability Status queried	2010-01-05T17:00:00-05:00	Explanation for assertion A123Q4 queried	2009-12-11T10:45:41-05:00	Local assertion 2 forwarded and returned as A453X4	2009-12-08T13:00:01-05:00	Assertion 3 stored
Date & Time	Activity															
2010-01-10T09:35:12-05:00	Capability Status queried															
2010-01-05T17:00:00-05:00	Explanation for assertion A123Q4 queried															
2009-12-11T10:45:41-05:00	Local assertion 2 forwarded and returned as A453X4															
2009-12-08T13:00:01-05:00	Assertion 3 stored															
***** Newer Older																
Sign Out																

Figure 57. Main menu Web page

c. *Store Assertion*

If the user wishes to store an assertion that has not been previously stored into the local database, the user would select the *Store assertion* tab. The application would then provide the user a Web interface to input the assertion data. For demonstration purposes, we provide two pages shown in Figures 58 and 59. However, the data could be input by using just one Web page. When complete, the user is given one of three *store* options to execute:

- **Store**—Stores the assertion in the local database only.
- **Store and Transfer**—Stores the assertion in the local database and then executes the KbUpdate client to add that assertion to the external database.
- **Store and Replace**—Stores the assertion in the local database and then executes the KbUpdate client to transfer that new assertion over to the external database in order to replace an existing assertion.

Store Assertion	
CONTEXT	
Language Use Domain:	Source Name:
Assertion Flag: ☐ Public ☐ Private	Source Language:
Source Location:	Source Medium:
	Source Type:
Data Segment:	
CLAIM	
Speaker Entity Type:	Target Entity Type:
Speaker Entity ID:	Target Entity ID:
Target Attribute Type:	
Continue Cancel	

Figure 58. Store assertion I

EVIDENCE		
Strength:	Display Type: ☐ Weak Confidence ☐ Strong Confidence ☐ None	
Version:		
EVIDENCE DOCUMENT		
Start line:	End line:	Tactic:
SUPPORT		
Theoretical frame:	Technical term gloss:	
Technical term:	Data snippet:	
Store Store & Replace Store & Transfer Back Cancel		

Figure 59. Store assertion II

When the user selects the option to Store and Replace, the application would display the Web interface depicted in Figure 60. The interface allows the user to search for the assertion by using one of two search options: by external ID or author. The output of the search would be displayed in a form that allows the user to select only one assertion that is currently in the external database. Upon selection, a short summary is displayed for the user to view before replacement.

Replace Assertion

Search for the assertion that will be replaced by:

External ID

Search

Author

Search

Select the assertion that will be replaced:

	External ID	Local ID	Author	DTG
☐	A12567	3	Thomas Jones	2010-01-05T17:00:00-05:00
☒	A78B64	10	Thomas Jones	2010-01-10T18:00:00-05:00
☐	B78R45	15	Thomas Jones	2010-01-15T19:00:00-05:00

10

will replace

A78B64

Replace

Back

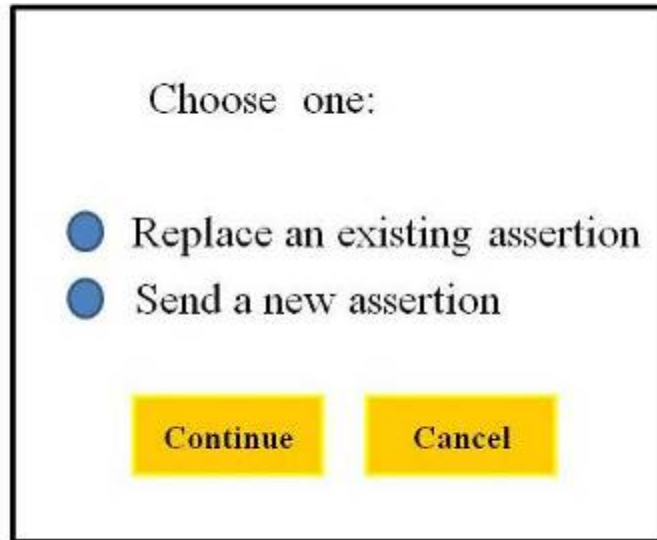
Cancel

Figure 60. Store assertion III

d. *Transfer Assertion*

The *Transfer assertion* operation will execute the KbUpdate client for assertions that have been previously stored in the local database only. Figure 61 shows, upon selection, that the application will provide the user an option between sending an assertion not currently in the external database or to transfer an assertion meant to replace

an existing one. Figure 62 shows the interface upon selection of the *Send a new assertion* option. The user is provided a means to search for and to select the assertion(s) to be transferred. If the user selects the option to *Replace an existing assertion* the user will first choose the assertion that will do the replacing using a Web page similar to Figure 62. The user will then select the assertion to be replaced through a Web page that looks similar to Figure 60.



Choose one:

- ☒ Replace an existing assertion
- ☐ Send a new assertion

Continue Cancel

Figure 61. Transfer assertion I

Transfer Assertion

Search for the assertion(s) that will be sent by:

Local ID

Search

Author

Search

Select the assertion(s) that will be forwarded:

	Local ID	Author	DTG
☐	4	Robert Uecker	2010-01-10T18:00:00-05:00

Transfer

Back

Main Menu

Figure 62. Transfer assertion II

e. Edit Assertion Data

The Web interface under the *Edit assertion* operation will allow the user to select an assertion based on a search for either the author or the assertion local ID. The user would then have the option to either *edit* or *delete* the selected assertion.

After a user selects an assertion and chooses the option to *edit*, an interface capable of being edited, similar to Figures 58 and 59 above, will appear with the text fields populated with the current assertion data. The user could change the data in the text fields and save the new data. If an external ID was associated with the assertion selected for editing, then the KbUpdate client will execute and replace the assertion in the external database with the newly modified version; otherwise the newly edited assertion will be stored locally with the same local ID. Figure 63 displays the Edit Assertion Web interface.

If the user selects the *delete* option, and there is not an external ID associated with the assertion, then the assertion will only be removed from the local database. However, if there is an external ID associated with the assertion to be deleted, the user will receive a prompt, as depicted in Figure 64, to decide between deleting an assertion from the external database only, or deleting the assertion from both the local and external databases. In either case, the external ID for that assertion will no longer be recognized by the external system, and will be removed from the local database. The user can also select the help button to understand the ramifications of his/her decision. In either case, the KbUpdate client will be executed to delete the respective assertion referred to by the external ID.

Edit Assertion

Search for the assertion to be edited by:

Local ID

Search

Author

Search

Select the assertion to be edited:

	External ID	Local ID	Author	DTG
☐	A12567	3	Thomas Jones	2010-01-05T17:00:00-05:00
☐	A123BTR	16	Robert Uecker	2010-01-10T18:00:00-05:00
☐	B65QGX	12	Gladys Knight	2010-01-15T19:00:00-05:00
☐	C567QX	8	Frederick Fender	2010-01-01T20:00:00-05:00

Edit

Delete

Main Menu

Figure 63. Edit assertion

Choose one:

- ☒ Delete from external database only
- ☐ Delete from both databases

Delete **Back** **Help**

Figure 64. Edit assertion II

e. Set Capability Status

The *Set Capability Status* operation will only allow an administrator to set the capability status. If a non-privileged user would attempt to access this operation, the application would generate a message for the user stating that the user does not have those privileges. The Web interface will show the administrator three sets of radio buttons per capability. The administrator would select the appropriate status per respective capability, and then save the operation (see Figure 65).

Select the capability status

KBUPDATE

- ☐ Available
- ☐ Unavailable
- ☐ Unsupported

DATAPUSH

- ☐ Available
- ☐ Unavailable
- ☐ Unsupported

EXPLANATIONGET

- ☐ Available
- ☐ Unavailable
- ☐ Unsupported

Save **Cancel**

Figure 65. Select the status

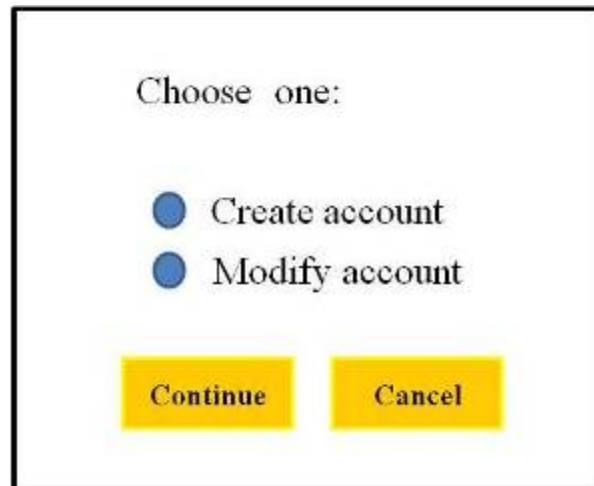
f. User Accounts

The *User Accounts* operation is also restricted to administrators in the same way as setting the capability statuses. When selected, the administrator will be given the option (Figure 66), to either *create account* or to *modify account*.

If the administrator selects the option to *create account*, the application will display an interface with text fields for entering the user's personal data. When complete, the administrator would save the user account information, thereby storing the data in the local database. See Figure 67.

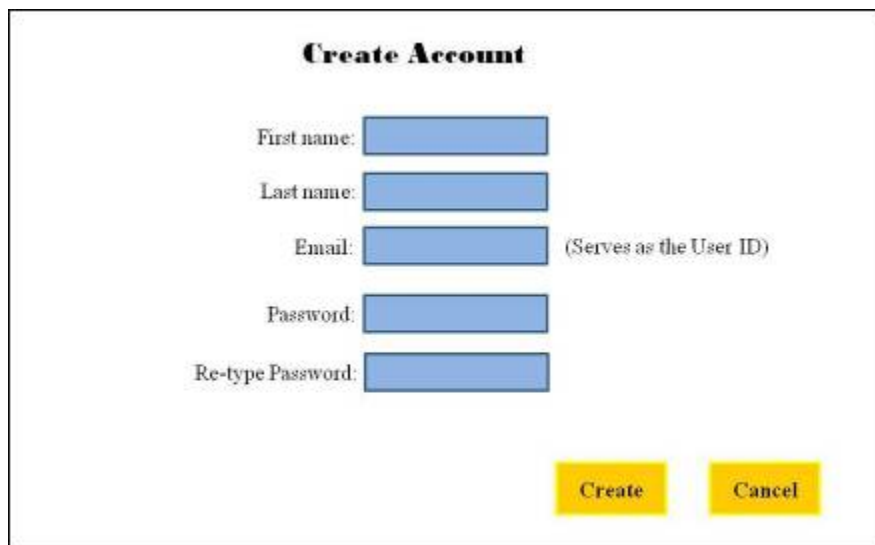
If the administrator selects the option to *modify account*, a Web interface listing all of the users will be shown. The administrator would select the user account to perform either a *modify* or *delete* function (Figure 68). If *modify* is selected, the administrator will be presented with an interface capable of being edited (Figure 69), but with the text fields populated with the user's account data. The administrator can make changes as necessary and save the changes, thereby updating the local database. If the

administrator selects *delete*, the application will prompt the administrator for confirmation. After confirmation, the user account data will be permanently removed from the local database.



A dialog box titled "Choose one:" with two radio button options: "Create account" and "Modify account". At the bottom, there are two yellow buttons labeled "Continue" and "Cancel".

Figure 66. User account I



A dialog box titled "Create Account" with five text input fields: "First name:", "Last name:", "Email:", "Password:", and "Re-type Password:". The "Email:" field has a note "(Serves as the User ID)" to its right. At the bottom right, there are two yellow buttons labeled "Create" and "Cancel".

Figure 67. User account II

Modify Account

	Last Name	First Name	User ID
☐	Thomas	Jones	jThomas@myEmail.com
☐	Robert	Uecker	rUecker@myEmail.com
☐	Gladys	Knight	gKnight@myEmail.com
☐	Frederick	Fender	ffender@myEmail.com

Figure 68. Modify account

User Account

First name:

Last name:

Email: (Serves as your User ID)

Password:

Re-type Password:

Figure 69. Update account

2. UI Class Diagrams.

We now present the models of the UI Web application using UML class diagrams. It is important to note that the original UML standard notations were never intended to represent Web pages. It was not until the Web Application Extension (WAE)

for UML was adopted that Web pages were capable of being represented by new modeling notations. Jim Conallen, co-founder of UML, states that the extension to UML is expressed using the following mechanisms: *stereotypes*, *tagged values*, and *constraints* (Conallen, Modeling Web Application Architectures with UML, 1999). In his book, *Building Web Applications With UML Second Edition*, Conallen describes these mechanisms as follows:

Stereotype, an extension to the vocabulary of the language, allows us to attach a new semantic meaning to a model element. Stereotypes can be applied to nearly every model element and are usually represented as a string between a pair of guillemets: « ». However, they can also be rendered by a new icon.

Tagged value, an extension to the property of a model element, is the definition of a new property that can be associated with a model element. Most model elements have properties associated with them. Classes, for instance, have names, visibility, persistence, and other attributes associated with them. A tagged value is rendered on a diagram as a string enclosed by brackets.

Constraint, an extension to the semantics of the language, specifies the conditions under which the model can be considered well formed. A constraint is a rule that defines how the model can be put together. Constraints are rendered as strings between a pair of braces: {}.

Conallen defines three core class stereotypes: Server page, Client page, and HTML form; all three of which comprise the aforementioned mechanisms. These classes, and the stereotype class associations described in Table 9, were used to design the class diagrams in the following pages. Figure 70 shows the WAE class diagram for the main menu.

Table 9. Stereotyped associations (From Conallen, 2003)

Stereotype	Description
«link»	A relationship between a client page and a Web page. The target may be a client page class or a server page class.
«build»	A directional relationship between a server page and a client page. This relationship identifies the HTML output of a server page's execution.
«submit»	A directional relationship between an «HTML form» and a server page. It references a server-side resource. However, when the resource is requested from the server, all the form's fields attributes are submitted, along with the request where they are processed.
«redirect»	A directional relationship between one server page and another server page or a client page. This association indicates a command to the client to request another resource.

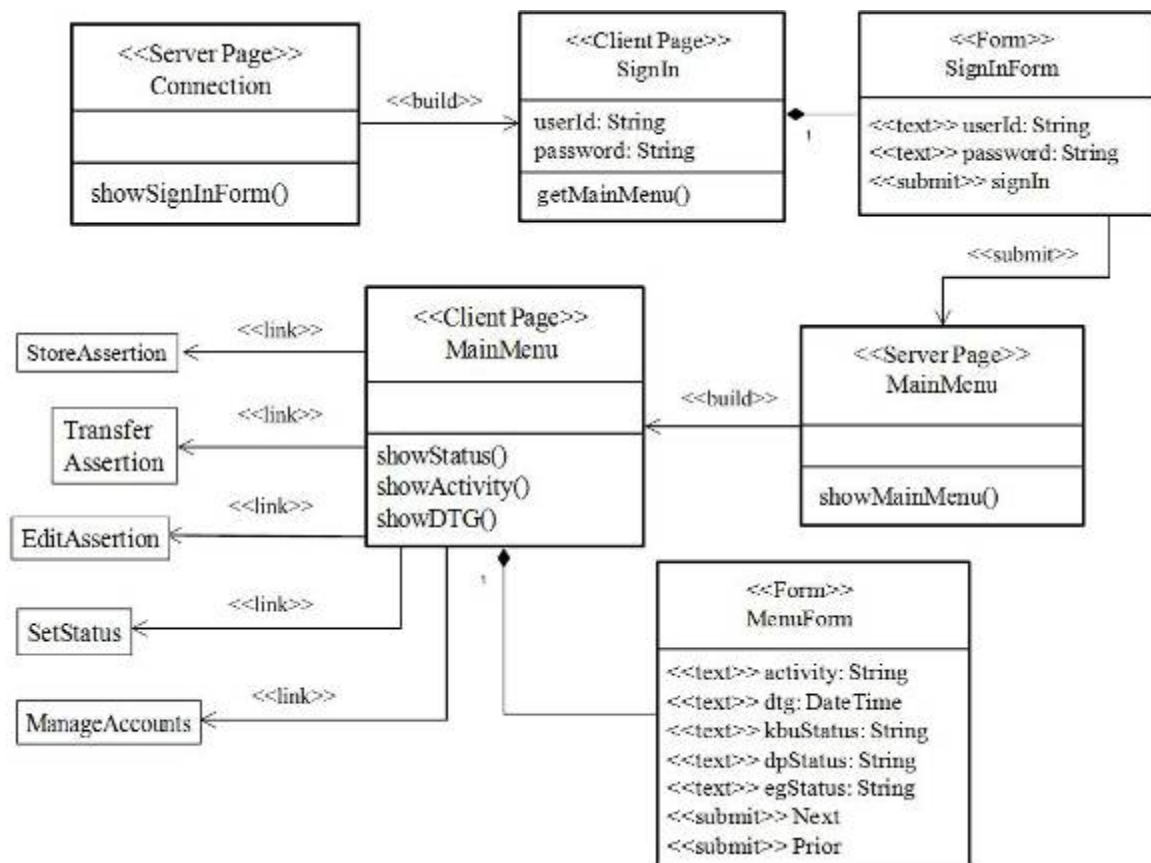


Figure 70. Main menu

From the main menu, the user could choose among the five operations. Figure 71 shows the interaction between the classes for the case when the user chooses to store an assertion. This model demonstrates that a user can still replace a current assertion in the external database after its initial creation at the local level. As per the requirements in Chapter III, the diagrams depict feedback in the form of a Web page after the successful completion of the respective operation.

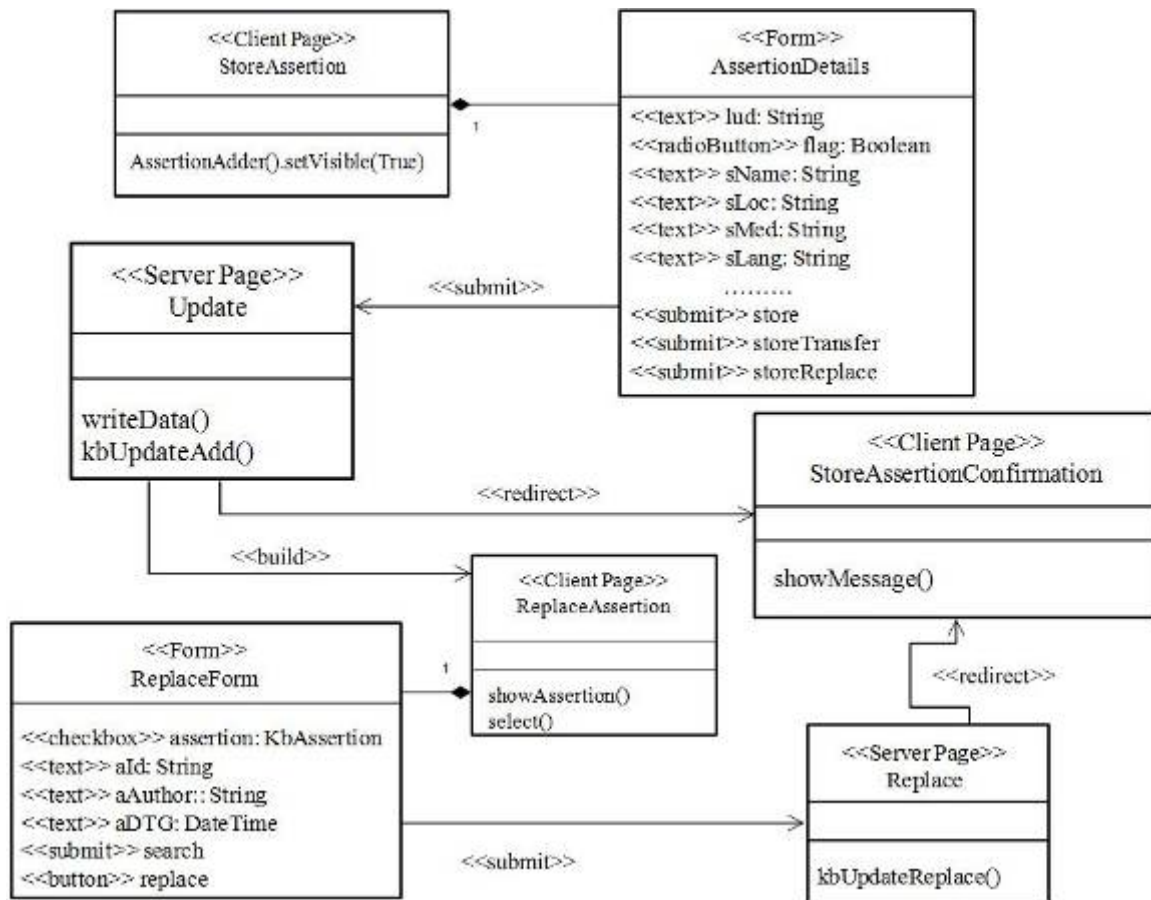


Figure 71. Store assertion

Figure 72 shows the user's choice in transferring an assertion that has only been stored locally. Again, the user does have the option to replace an assertion currently in the external database; that option is a continued in Figure 73.

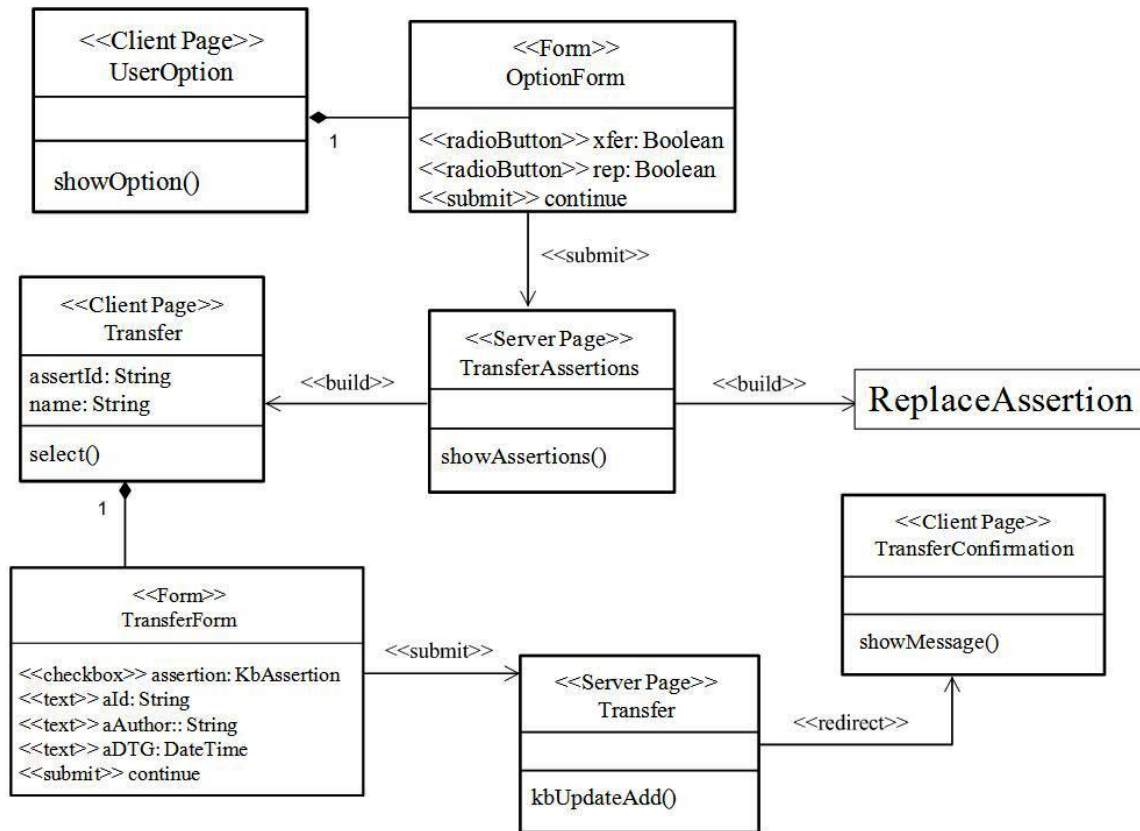


Figure 72. Transfer assertion

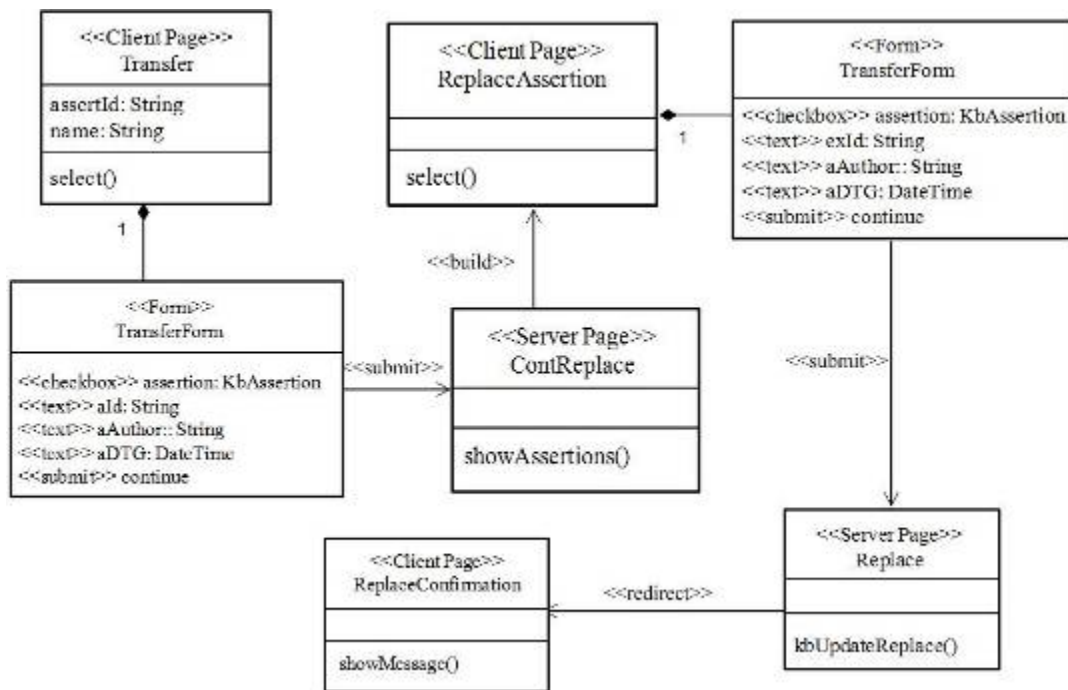


Figure 73. Transfer assertion II

Figure 74 shows the user's option to either edit or delete an assertion from the databases. The form DeleteForm provides the user another option, to remove the assertion locally only or from the external database, too.

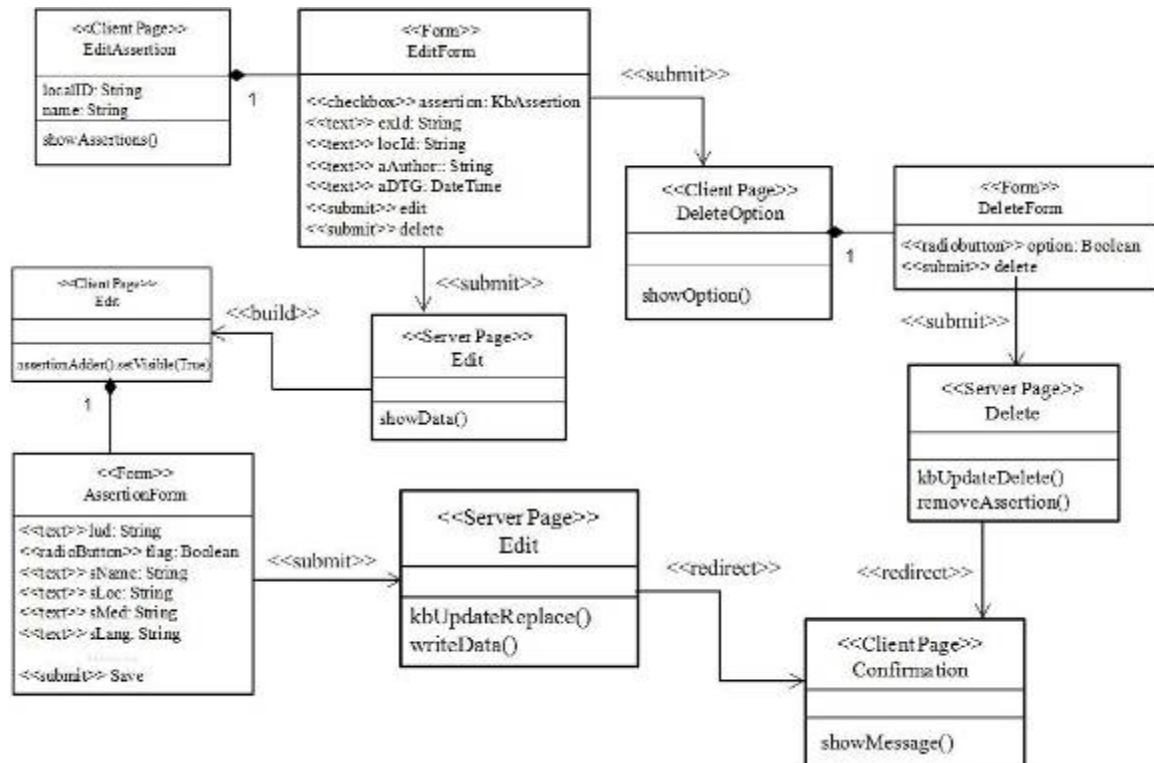


Figure 74. Edit assertion

Setting the capability status requires the administrator to choose from one of three sets of radio buttons per capability. The selected button is represented in the form DeleteForm. See Figure 75.

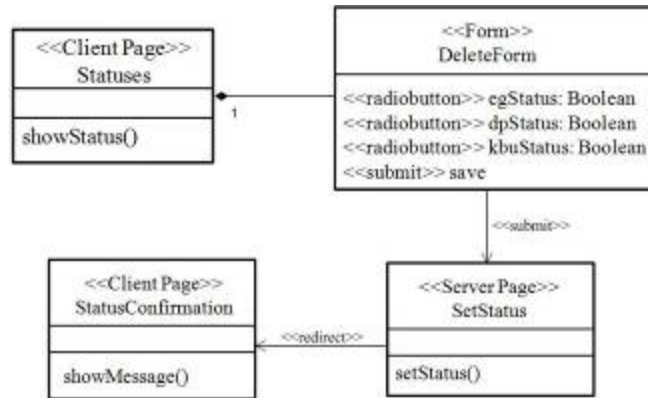


Figure 75. Set capability statuses

Finally, the administrator is given the option either to create a new user or modify an existing user account. See Figure 76.

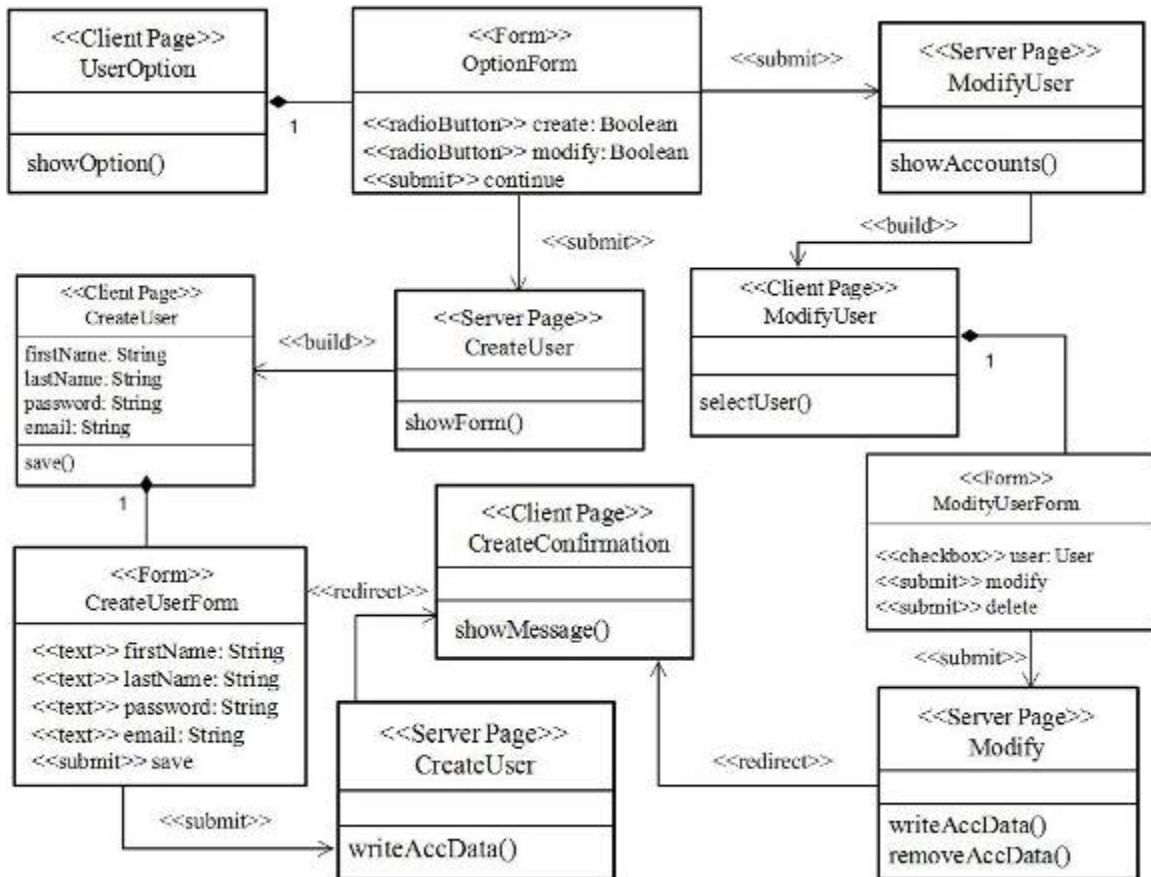


Figure 76. User accounts

E. SUMMARY

In this chapter, we took the requirements from Chapter III and used them to provide the readers our proposed system design. We began our design with the Database tier, wherein we summed up and displayed the attributes, entities, and relationships of the system in an ER diagram. We followed up by using an algorithm to map the ER diagram into an RDS, which we will use to create our database tables. Our next step was to design the Business Logic tier by presenting aspects of our XML Schema and generating the UML class diagrams that express the functionality of the core operations of our system: the StatusReport WS, the KbUpdate client, and the ExplanationGet WS. For our Web tier, we presented the WSDL documents that we will use as our endpoints. Finally, for the Presentation tier, we showed the reader our design for the Web pages to be used by the local user to operate the system, followed by the Web application UML class diagrams. In Chapter V, we present our prototype of the system to be used by the UMD performer team.

V. PROTOTYPE

In Chapter I, we described the basic system features that the UMD performer team would need in order to contribute to the SCIL program. The first focused on the local user's management of the assertion data that would allow for the collection, storage, and modification of the assertion prior to distribution. The second was to allow for remote access to the data. This ability is realized with the use of our core Web services described in Chapter III (StatusReportWS and ExplanationGetWS). The third feature was the ability to transfer the data to the external knowledge repository over the Internet. Our Web service client, KbUpdate, has been designed to manage the transfer of assertion data to the external database; this includes the ability to replace and delete assertion data.

In this chapter, we will describe the prototype of the system. Our prototype consists of a database store that supports our two core Web services (StatusReportWS and ExplanationGetWS), and our Web service client (KbUpdate). The prototype also involves the application server in which the Web services are deployed. Our objective for this prototype was two-fold. First, we wanted to provide a proof of concept implementation for both Web services and WS client in support of the preliminary engineering tests. Second, we wanted to provide the stakeholders with a working system capable of being modified and upgraded for their future use.

Our system was developed in an Apple® Macintosh desktop computer running OS X. This chapter begins with the implementation of our local database. We will describe the database tables we developed to interact with the business logic of our system. Next, we present our Web services followed by our client. Using screenshots of the applications, we will walk the reader through the sequence of events that transpire in order to execute each operation.

A. MYSQL DATABASE

Although the SRS in Chapter III described the DBMS to be a Relational-type, we also researched into the feasibility of using an Object-Oriented Database Management

System instead. In the end, we selected the MySQL Community Server version 5.1.49 DBMS (www.mysql.com) for our prototype because the database is, at this point in time, required to handle only simple data types (i.e., numbers and strings). This edition of the MySQL DBMS is also open-sourced, and is compatible with the NetBeans Integrated Development Environment (IDE), which we used to develop our applications. Based on the RDS we designed in Chapter IV, we constructed a total of six tables to support our system operations: *status*, *assertionData*, *claimTarget*, *contextDataSegment*, *evidenceStatements*, and *supportTechTerm*.

1. Status

Figure 77 displays a description of the status table that we generated using SQL. The Field column on the left holds the names of the columns in the table. The status table is queried for all six values whenever the StatusReportWS is consumed by the external client. The DTG columns are separately updated with the successful execution of either the KbUpdate or DataPush clients or when the ExplanationGetWS has been consumed.

```
mysql> describe status;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| kbu_status | varchar(25) | NO   |     | NULL    |       |
| dp_status  | varchar(25) | NO   |     | NULL    |       |
| eg_status  | varchar(25) | NO   |     | NULL    |       |
| kbu_dtg    | datetime   | YES  |     | NULL    |       |
| dp_dtg     | datetime   | YES  |     | NULL    |       |
| eg_dtg     | datetime   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Figure 77. MySQL status table

2. AssertionData

Figure 78 represents the assertionData table. This table holds all of the assertions that are referenced by both localID and externalID. However, not all of the assertion data can be found in this table. We needed to generate four other tables to hold the remainder

of the data because those tables would be required to hold at least one row of data per assertion. To handle this one-to-many relationship, we created a foreign key called 'localID' in claimTarget, evidenceStatements, supportTechTerm and contextDataSegment that referenced localID from assertionData. This relationship allowed for the entire assertion data to be inserted into and read from our database tables. Figure 79 shows the four additional tables.

Field	Type	Null	Key	Default	Extra
localID	int(11)	NO	PRI	NULL	auto_increment
dtg	timestamp	NO		CURRENT_TIMESTAMP	
flag	varchar(25)	YES		NULL	
externalID	varchar(25)	YES		NULL	
display	varchar(25)	YES		NULL	
strength	float	YES		NULL	
version	varchar(25)	YES		NULL	
theoretical_frame	varchar(25)	YES		NULL	
predicate_name	varchar(50)	YES		NULL	
speaker_type	varchar(25)	YES		NULL	
speaker_id	int(11)	YES		NULL	
tgtAttributeType	varchar(25)	YES		NULL	
lang_use_dom	varchar(50)	YES		NULL	
sourceName	varchar(45)	YES		NULL	
sourceLocation	varchar(45)	YES		NULL	
sourceLanguage	varchar(45)	YES		NULL	
sourceType	varchar(45)	YES		NULL	
sourceMedium	varchar(45)	YES		NULL	

Figure 78. MySQL assertionData table

```
mysql> describe claimTarget;
```

Field	Type	Null	Key	Default	Extra
externalID	varchar(25)	YES		NULL	
localID	varchar(50)	YES	MUL	NULL	
tgt_type	varchar(50)	YES		NULL	
tgt_entity_id	int(11)	YES		NULL	

```
4 rows in set (0.00 sec)
```



```
mysql> describe evidenceStatements;
```

Field	Type	Null	Key	Default	Extra
externalID	varchar(25)	YES		NULL	
localID	varchar(50)	YES	MUL	NULL	
doc	varchar(500)	YES		NULL	
start_line	int(11)	YES		NULL	
end_line	int(11)	YES		NULL	
tactic	varchar(50)	YES		NULL	

```
6 rows in set (0.00 sec)
```



```
mysql> describe supportTechTerm;
```

Field	Type	Null	Key	Default	Extra
externalID	varchar(25)	YES		NULL	
localID	varchar(50)	YES	MUL	NULL	
gloss	varchar(2500)	YES		NULL	
term	varchar(50)	YES		NULL	
data_snippet	varchar(2500)	YES		NULL	

```
5 rows in set (0.01 sec)
```



```
mysql> describe contextDataSegment;
```

Field	Type	Null	Key	Default	Extra
externalID	varchar(25)	YES		NULL	
localID	varchar(50)	YES	MUL	NULL	
d_segment	varchar(2000)	YES		NULL	

```
3 rows in set (0.00 sec)
```

Figure 79. MySQL assertion-related tables

B. UMD OPERATIONS

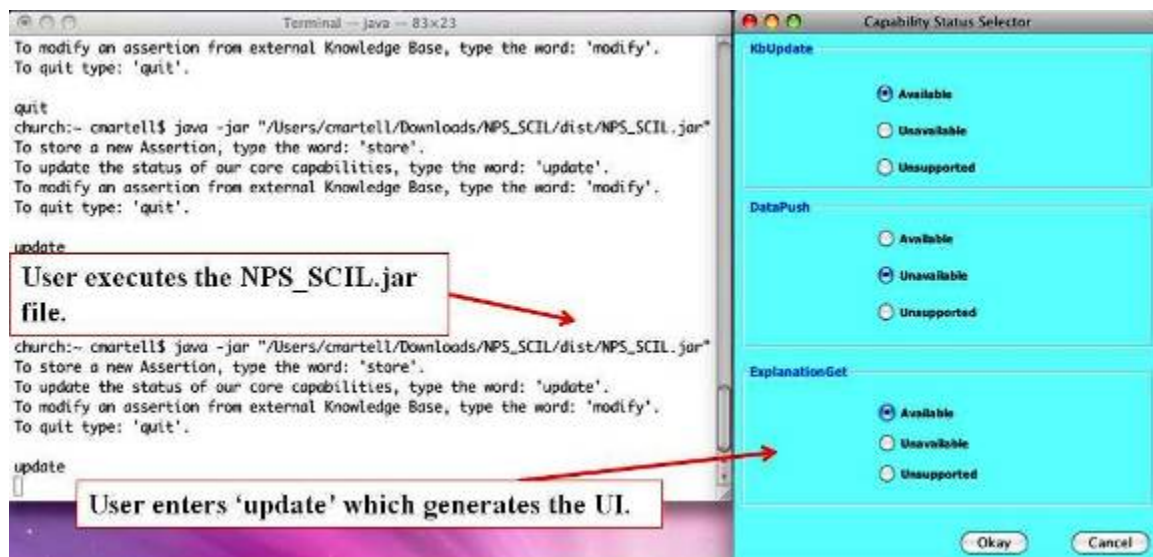
Now that the reader has seen our underlying data store, let us turn to the execution of the three core prototype operations. We designed our applications with the NetBeans IDE version 6.8, an open-source application development tool, using the Java™ Development Kit 6 update 20. Both Web services were developed under one Java Web application project, entitled UMD, and subsequently deployed to the open-source Glassfish Web application server version 3. The program required to manage the

assertion data and the capability status was developed using the Java Standard Edition application called NPS_SCIL. A rudimentary command line UI supported by Java Swing GUI components were developed to assist the local user in running the program—the ideal UI being the Web application designed in Chapter IV.

1. StatusReport

a. Setting the Status

A local user can execute the NPS_SCIL application by running the NPS_SCIL.jar file created from the compiling process. As displayed in Figure 80, upon execution, the user is presented with a set of options. Selecting the 'update' command will generate a window for the user to select the appropriate status per capability. When finished, the user clicks on the 'Okay' button that executes an excerpt of the Java code displayed in Figure 81, which calls the method named writeCapabilityStatus(), displayed in Figure 82, to insert the desired status values into the database table *status* shown in Figure 83.



```

if (jRadioButton4.isSelected()){
    dpStatus = {"Available"};
} else if (jRadioButton5.isSelected()){
    dpStatus = {"Unavailable"};
} else {
    dpStatus = {"Unsupported"};
}

if (jRadioButton7.isSelected()){
    egStatus = {"Available"};
} else if (jRadioButton8.isSelected()){
    egStatus = {"Unavailable"};
} else {
    egStatus = {"Unsupported"};
} try {

    DatabaseConnection con = new DatabaseConnection();
    con.writeCapabilityStatus(kbuStatus, dpStatus, egStatus);
    JOptionPane.showMessageDialog(null, " The local status has been updated.");
    System.exit(0);
}

```

Figure 81. Method call to write the statuses

```

public void writeCapabilityStatus(String kb, String dp, String eg) throws Exception {

    String sqlStmt1 = "UPDATE status SET kbu_status = '".concat(kb)+"', " +
        "dp_status = '".concat(dp)+"', eg_status = '".concat(eg)+"'";

    try {
        Class.forName(driverName);
        con = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        stmt1 = con.prepareStatement(sqlStmt1);
        stmt1.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {con.close();}

}

```

Figure 82. Connection to the database

```

mysql> select kbu_status, dp_status, eg_status, kbu_dtg, dp_dtg, eg_dtg from status;
+-----+-----+-----+-----+-----+-----+
| kbu_status | dp_status | eg_status | kbu_dtg | dp_dtg | eg_dtg |
+-----+-----+-----+-----+-----+-----+
| Available | Unavailable | Available | 2010-07-31 13:27:35 | 2010-07-31 11:37:44 | 2010-07-31 11:38:07 |
+-----+-----+-----+-----+-----+-----+

```

Figure 83. Status table after execution

b. The Web Service

The Web application can be executed by running the UMD.war file generated from the compiling process. Prior to deploying the WS to the Glassfish server, we tested it by using the open-source software called *soapUI* version 3.0.1 (www.eviware.com). This software allows users to act as clients and consume Web services over the Internet. This was important to us because we needed a way to determine if the correct data was returned upon a query from the external client. Figure 84 shows the soapUI split screen interaction between the client's request on the left and the StatusReportWS response on the right.

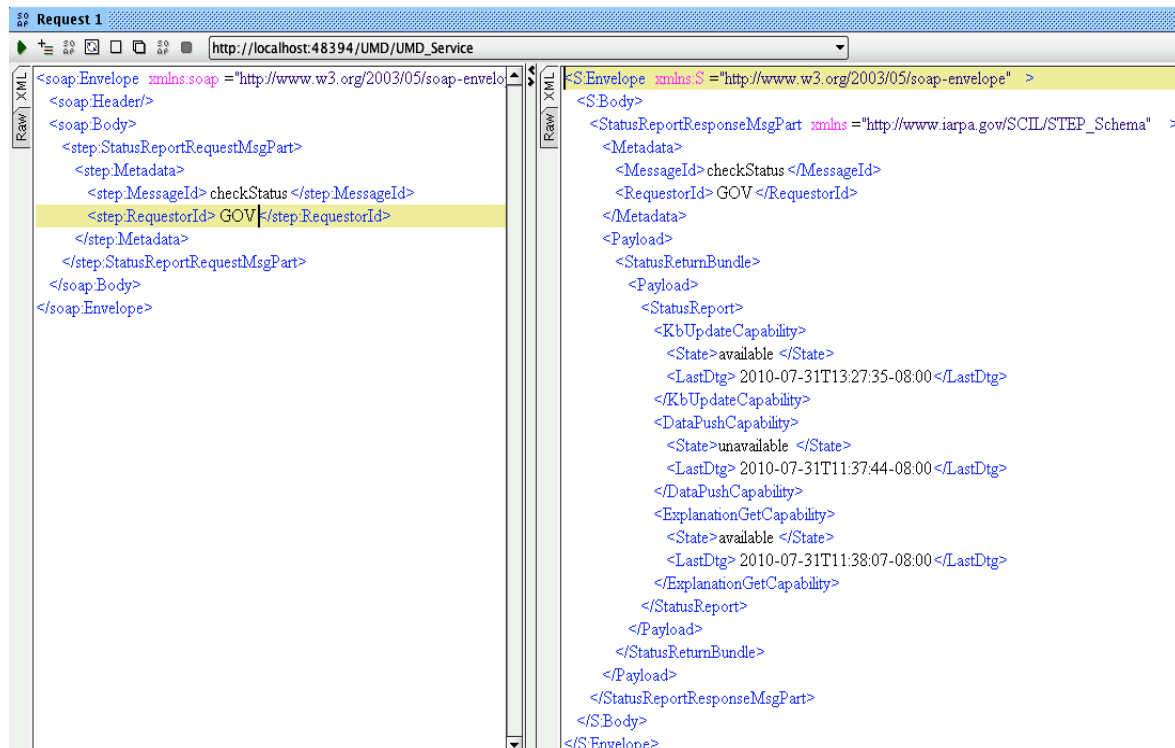


Figure 84. Testing StatusReportWS

Following the successful test, we deployed the UMD.war file to the Glassfish server to be consumed by the external client, as depicted in Figure 85.

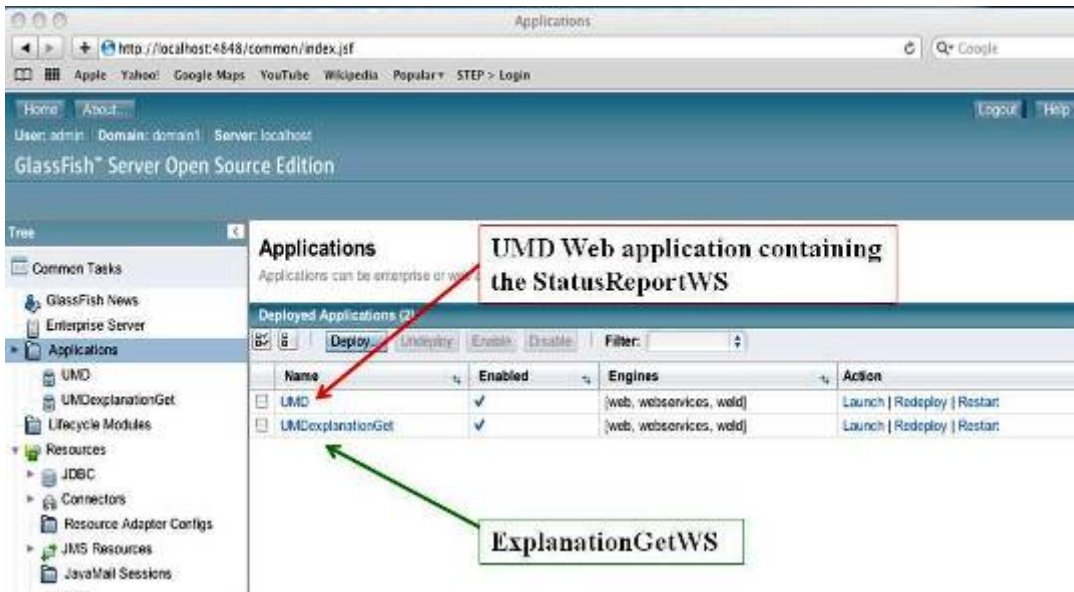


Figure 85. Deploying UMD

2. KbUpdate client

In this section, we step through the execution of the KbUpdate client. The functionality behind the options to either add, delete, or replace was discussed in detail in the two previous chapters. In this section, we will focus on the ‘adding’ function of the client. An assertion needs to be in the local database before the update so we will begin with storing the assertion.

a. Storing the assertion

Figure 86 below shows the execution of the NPS_SCIL.jar file from the command line. The user selects the option to 'store.' The user is presented with an interactive window to enter the assertion data. When the user is finished, he clicks on the 'save' button, which runs the code that executes five separate SQL statements in order to store the assertion data into its respective tables, described in section A.2 above. The system generates a local ID for the assertion and inserts that ID into all five assertion-related tables in the same row(s) that pertains to that assertion. Figure 87 shows an excerpt of the *assertionData* table that highlights the local ID and the lack of an external

ID at this point in the sequence. Note in Figure 87 that the predicate_name and lang_use_dom columns share the same data; this is because the data entered was for test purposes.

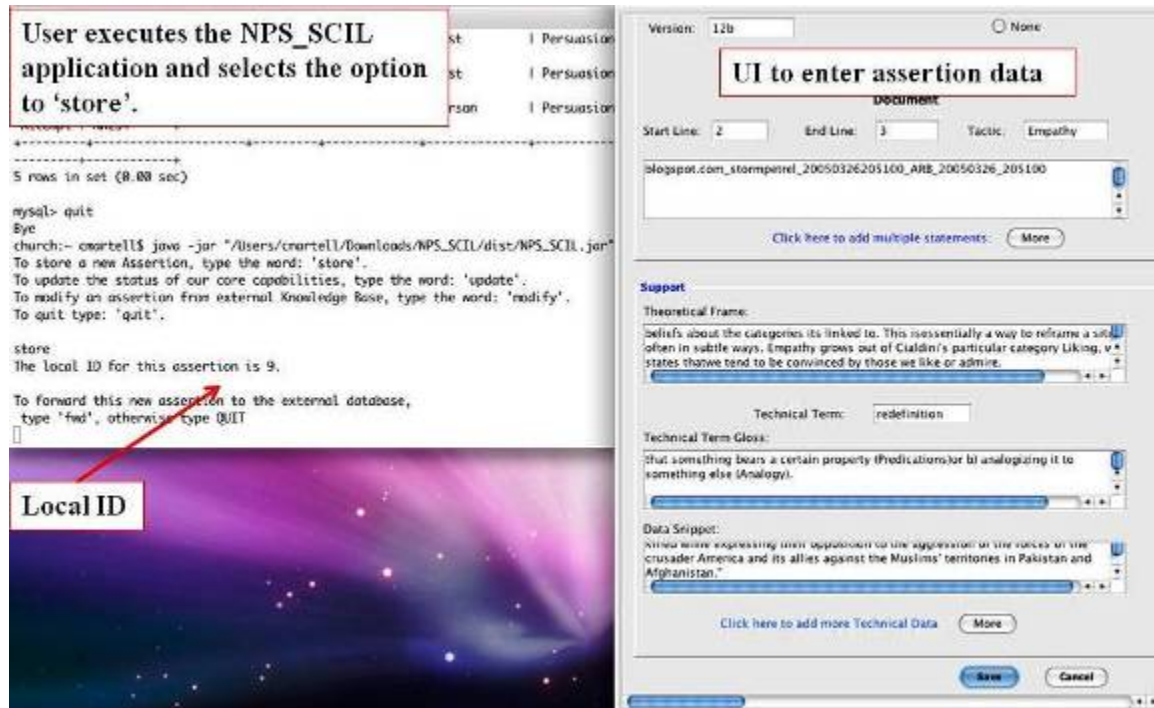


Figure 86. Storing an assertion

```
mysql> select localID, dtg, externalID, predicate_name, speaker_id, speaker_type, lang_use_dom from assertionData;
```

localID	dtg	externalID	predicate_name	speaker_id	speaker_type	lang_use_dom
2	2010-07-30 13:32:17	NULL	Persuasion Attempt	1	test	Persuasion Attempt
3	2010-07-30 14:12:10	UMD50083b8a	Persuasion Attempt	1701	person	Persuasion Attempt
6	2010-07-31 12:21:53	NULL	Persuasion Attempt	1	test	Persuasion Attempt
7	2010-07-31 12:35:58	NULL	Persuasion Attempt	1	test	Persuasion Attempt
8	2010-08-03 09:20:59	NULL	Persuasion Attempt	123	person	Persuasion Attempt
9	2010-08-03 09:43:58	NULL	Persuasion Attempt	123	person	Persuasion Attempt

Figure 87. Assertion in the database

b. Updating the External Database

From the command line, instead of selecting 'store,' the KbUpdate client is run by selecting the option to 'modify.' The application then gives the user the option to either add, delete, or replace. The user selects the option to 'add' and is then prompted for a message ID and for the localID of the assertion to be added. When the user submits the localID to be transferred, the system reads the respective assertion data in all five assertion-related tables and sends the data to the external system. Figure 88 is a screenshot of the Web page generated by the external system's Web server called the STEP server, which displays the list of assertions currently in the external database (Naval Research Laboratory, 2010). For this demonstration, the listing created on 2010-08-03 refers to the assertion with local ID #9. Figure 89 is a screenshot of another STEP server generated Web page that shows the specific ID #9 assertion elements following the successful transfer (Naval Research Laboratory, 2010). The external system generates and returns an external ID that refers to that assertion, and the local system writes that external ID into the rows of the five tables that pertain to the local copy of the assertion that was just sent. Figure 90 shows two of the five tables with external ID 'UMDb39f431a.'

The screenshot shows a web browser window displaying the STEP application. The address bar shows a URL starting with 'http://207.46.100.47/step/'. The browser's address bar and tabs are visible at the top. The application header includes the 'STEP [Powered by SCIL]' logo and navigation links: 'KB Query', 'Status', 'About', and 'Help'. The user 'Craig.Martell' is logged in, with 'Logout' and 'Quick Feedback' buttons. Below the header, there are two tabs: 'Simple Query' (selected) and 'Query Summary'. The 'Simple Query' tab shows a query: 'team: UMD, langUse:'. Below this, the results are displayed under the heading 'Results (Found 5 current assertions)'. The results are presented in a table with 7 columns: 'Language Use', 'Team', 'Created', 'Language', 'Source', 'Type', and 'Visibility'.

Language Use	Team	Created	Language	Source	Type	Visibility
Persuasion Attempt	UMD	2010-08-03	English	NRL34	Blog	Public
Persuasion Attempt	UMD	2010-07-30	English	NPS/UMD	Blog	Public
Persuasion Attempt	UMD	2010-05-25	Spanish	SourceName	blog	Private
Persuasion Attempt	UMD	2010-05-25	English	SourceName	email	Private
Persuasion Attempt	UMD	2010-05-25	Spanish	SourceName	blog	Private

Figure 88. External Web server

Assertion UMDb39f431a

STEP ID: UMDb39f431a	Performer Team: UMD	Created: 2010-08-03	Version: 1	Public
Language Use: Persuasion Attempt		Source: NRL34	Language: English	
[-] AssertionClaim [-] PredicateClaim [-] PredicateName Persuasion Attempt [-] Speaker [-] Entity [-] id 123 [-] type person [-] Target (type: directed) [-] Entity [-] id 321 [-] type person				
[-] AssertionEvidence [-] EvidenceValue Future Evidence value [-] EvidenceStatement [-] ConstituentMultiset [-] PersuasionTactic [-] tactic Empathy [-] startline 2 [-] endline 3 [-] doc blogspot.com_stormpetrel_20050326205100_ARB_20050326_205100				
[-] AssertionSupport [-] TheoreticalFrame This interaction includes use of a redefinition tactic and an appeal to empathy. The persuasive force of the redefinition tactic comes in its ability to marshal pre-existing beliefs about the categories its linked to. This is essentially a way to				

Figure 89. External Web server

```
mysql> select localID, dtg, externalID, predicate_name, speaker_id, speaker_type, lang_use_dom from assertionData;
```

localID	dtg	externalID	predicate_name	speaker_id	speaker_type	lang_use_dom
2	2010-07-30 13:32:17	NULL	Persuasion Attempt	1	test	Persuasion Attempt
3	2010-07-30 14:12:10	UMD50083b8a	Persuasion Attempt	1701	person	Persuasion Attempt
6	2010-07-31 12:21:53	NULL	Persuasion Attempt	1	test	Persuasion Attempt
7	2010-07-31 12:35:58	NULL	Persuasion Attempt	1	test	Persuasion Attempt
8	2010-08-03 09:20:59	NULL	Persuasion Attempt	123	person	Persuasion Attempt
9	2010-08-03 09:43:58	UMDb39f431a	Persuasion Attempt	123	person	Persuasion Attempt

6 rows in set (0.00 sec)

```
mysql> select externalID, localID, start_line, end_line, tactic from evidenceStatements;
```

externalID	localID	start_line	end_line	tactic
NULL	2	1	2	test
UMD50083b8a	3	15	15	Redefinition
UMD50083b8a	3	24	24	Empathy
NULL	4	1	1	test
NULL	5	1	2	test
NULL	6	1	2	test
NULL	7	1	2	test
NULL	8	2	3	Empathy
UMDb39f431a	9	2	3	Empathy

9 rows in set (0.00 sec)

Figure 90. External ID in the database

3. ExplanationGetWS

b. The Web Service

Now that we have data that is ready to be retrieved, we test the WS using the soapUI software, as shown in Figure 91. The key component to the request is the external ID. Using the external ID, the local system retrieves the respective assertion data located in the assertion-related tables. Finally, with the successful test of the ExplanationGetWS using soapUI, we deployed the .war file to the Glassfish server for its subsequent consumption by the external client, as shown in Figure 85.

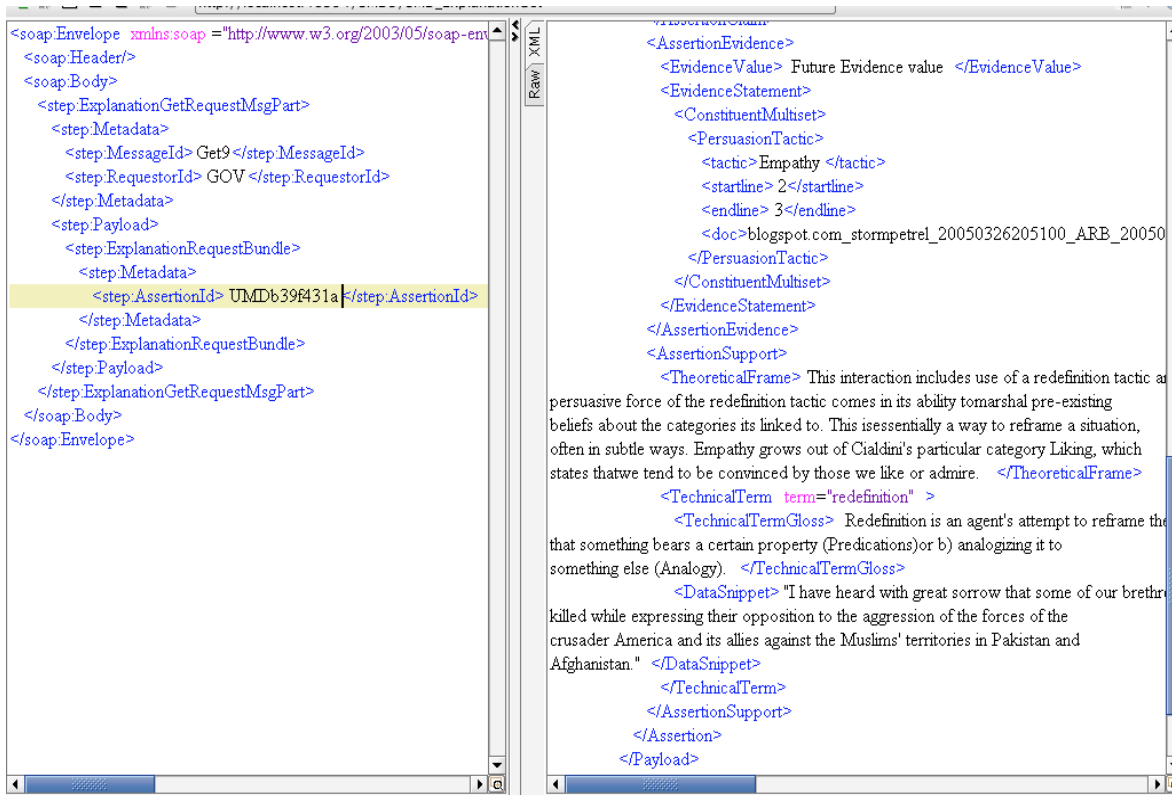


Figure 91. Testing ExplanationGetWS

C. USER FEEDBACK

With the operating prototype in hand, we obtained an acceptance test by a UMD representative. We had the representative step through the following sequence of

operations: setting the capability status, storing a new assertion, transferring an assertion, replacing an assertion, and deleting an assertion. The StatusReport WS and all three functions of the KbUpdate were fielded in order to assist in the completion of a series of engineering tests with the external system. Knowing that not all of the desired features were implemented in the prototype, the representative was satisfied with the prototype because it accomplished its intended purpose, and it allowed for future modifications as necessary.

At this point, the system has been used in multiple engineering tests following directed changes to the XSD by both IARPA and UMD. An agreed-upon structure of an assertion is still under discussion by the UMD stakeholders, so actual assertions have yet to be generated and forwarded to the external database.

D. SUMMARY

In this chapter, we used the system design discussed in Chapter IV to develop our proposed prototype automated system to manage the assertion data generated by the UMD performer team. The prototype provides a means for the stakeholders to validate the system design and a working platform for follow-on research and development.

We used the MySQL server as our database and created six tables to store the data necessary to execute the core operations. Our Web client application provides multiple functions including:

- the ability to add, replace, or delete assertions
- the ability to store an assertion into the database
- the ability to set the capability statuses

We also provided screenshots of the assertion as displayed in the external STEP application server. Both of our Web services and our Web client were developed using the Java programming language. We demonstrated to the reader the WS testing results using the soapUI software. Finally, we deployed our Web services to the Glassfish application server in order to await future external client requests.

VI. CONCLUSION

A. SYNOPSIS

The goal for this thesis was to design and develop an automated system to be used by the UMD performer team in support of the SCIL program led by IARPA. The SCIL program seeks to investigate various methodologies to help understand the social goals of people by demonstrating a relationship between these goals and their particular language use. UMD's role in the program is to identify the social goals that pertain to persuasion by analyzing chat-based Web forums. The end product of the analysis of the unstructured text is a set of assertions that declare acts of persuasion were attempted.

The system we designed enables users to locally store, manage, and transfer the assertions to the external system. Eventually, SCIL will be combined with a functionality that uses artificial intelligence techniques to process raw text. The processing will result in assertions that are then forwarded to an external knowledge base run by IARPA.

In Chapter I, we stated that the solution to the system required by the UMD performer team in the short run stemmed from the answers to the following questions:

1. *What are the requirements for system to be implemented by the NPS performer team?*
2. *What is the appropriate design of a modular framework to effectively manage the natural language assertions in a knowledge base repository and the sharing of the knowledge via the World Wide Web?*
3. *What is the appropriate Web-service design to allow for multiple users to update the knowledge base repository of natural language assertions from multiple sites?*

To address these questions, we began our research with an investigation of SOA and Web services. This background information introduces the key concepts we expanded upon in the remainder of the thesis.

Chapter III began with the introduction of an abstract system domain model that showed the components and their relationships to one another. We then presented the stakeholder-validated software requirements to answer question number one. The SRS included several use cases and SSDs to demonstrate the core system functionality. This is the initial version of the requirements specification, and is subject to iterative development based on the needs of the stakeholders.

For questions two and three, we started with an illustration of the four main tiers of the system architecture: database, business logic, Web, and the presentation. We then delved into each tier and discussed its respective design features. For the database, we presented an ER diagram and then followed a six-step algorithm to normalize the elements into a relational database design. The business logic tier was addressed by showing the particular system data types and elements of the XML schema to be used in the creation of our WS and client, along with their respective class diagrams. For the Web tier, we presented elements of our WSDL, which described our WS interfaces for our future clients. Lastly, using PowerPoint, we prototyped the layout of the graphical user Web interface that will enable the local users to perform the core system functionality including managing the assertion data and capability status. We finished the design of our presentation tier with the Web application class diagrams.

The thesis concludes with a description of the SCIL prototype that we implemented. We demonstrated the execution of the system functions by walking the reader through a scenario that involved a user setting the capability status and also performing the three main functions of the KbUpdate client.

The significance of this research is that it will support the analysis of the social dynamics behind certain groups of interest by managing the assertions generated from online chat communication. The prototype will serve as a vehicle to elicit additional requirements for SCIL.

B. FUTURE WORK

The prototype described in this thesis uses a relational database schema to organize the system data. We did not delve into the use of an object-relational or an

object-oriented database management system. Since we used an object-oriented programming language to develop the system software, it would seem to be more efficient to use a database that is designed to store objects instead of tuples. The data types used in the prototype were simple and easy to implement using MySQL, but a RDBMS does not handle complex data types as well as object-relational or object-oriented database management systems. An analysis of using either of these database management systems is a subject of future research.

The system design did not address concurrency issues that can be encountered when, for example, two or more users attempt to concurrently modify the same assertion. A *race condition* is a particular example of a concurrency issue. Stallings defines a race condition as “A situation in which multiple threads or processes read and write a shared data item and the final result depends on the relative timing of their execution” (Stallings, 2009, p. 207). An analysis of these system issues and their consequences would help to ensure that the integrity of the data is not compromised.

We designed a Web GUI for the system, but this feature was not implemented in our prototype. An analysis of some of the popular Web-development technologies (e.g., Ajax Frameworks, Java Server Faces, and Microsoft’s ASP.Net) is needed to identify which of these techniques should be used to implement the deployed system.

The intent behind the DataPush client mentioned in Chapter III is still in debate amongst the UMD stakeholders. Since the KbUpdate client already sends assertions to the external system, it would be redundant to implement another Web client alongside KbUpdate to perform the same function. We recommend that the DataPush client either be dropped from further discussion or clearly specified to warrant the development of another Web client.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. XML SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2010 (http://www.altova.com) by Javier Palomo (Naval
Postgraduate School) -->
<xs:schema xmlns:stepd="http://www.iarpa.gov/SCIL/STEP_Schema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.iarpa.gov/SCIL/STEP_Schema"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.3d"
xml:lang="EN">
  <!-- Schema Documentation -->
  <xs:annotation>
    <xs:documentation>
      XSD for STEP (the IARPA SCIL Program SOA Platform)
      Original Publication Date: 2009-10-26
      Current Date: 2010-08-27
      Current Version: 1.3d
    </xs:documentation>
    <xs:documentation>
      Change Log (from Version 1.2)
      2010-07-14 : Makes claims predicate-based; no wildcards
      2010-07-16 : Enumerates claims by team
      2010-07-21 : Changes the claim context element to SocialConstructDomain--an
      enumerated type
      2010-08-21 : Adds a SocialConstruct element to the assertions context--a simple
      string
      ==
    </xs:documentation>
  </xs:annotation>
  <!-- Global Types used in defining KB content -->
  <!-- Utility Types-->
  <xs:complexType name="DataSource">
    <xs:annotation>
      <xs:documentation>Type that defines the data source used in generating an
      assertion.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="DataMetadata">
      <xs:annotation>
        <xs:documentation>Metadata that describes the source of the data.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SourceName" type="xs:string">
          <xs:annotation>
            <xs:documentation>The name of the source.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="SourceLocation" type="xs:anyURI">
          <xs:annotation>
            <xs:documentation>A URI that allows the data to be located. Can be a dummy
            value.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:sequence>

```

```

</xs:annotation>
</xs:element>
<xs:element name="SourceLanguage" type="xs:language">
<xs:annotation>
<xs:documentation>The human language of the source.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="SourceType" type="xs:string">
<xs:annotation>
<xs:documentation>The type of source: blog, email, broadcast conversation,
etc.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="SourceMedium">
<xs:annotation>
<xs:documentation>A enumerated list. Currently just text or
speech.</xs:documentation>
</xs:annotation>
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="text"/>
<xs:enumeration value="speech"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="DataSegment" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>A segment of the source data processed in generating the claim.
</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="SourceDataSegment" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="KbClaim">
<xs:sequence>
<xs:element name="PredicateClaim">
<xs:complexType>
<xs:sequence>
<xs:element name="PredicateName" type="xs:string" default="Persuasion Attempt"/>
<xs:element name="Speaker">
<xs:complexType>
<xs:sequence>
<xs:element name="Entity" type="stepd:Entity"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Target">
<xs:complexType>
<xs:sequence>

```

```

<xs:element name="Entity" type="stepd:Entity" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="type" type="xs:string" default="directed">
</xs:attribute>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Entity">
<xs:sequence>
<xs:element name="id" type="xs:integer"/>
<xs:element name="type" type="xs:string" default="person">
<xs:annotation>
<xs:documentation> "type" refers to the entity being either a person or a
group</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="role" type="xs:string">
<xs:annotation>
<xs:documentation> "role" refers to the wether the entity is either the speaker or
target</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="KbEvidence">
<xs:sequence>
<xs:element name="EvidenceValue" type="xs:string">
<xs:annotation>
<xs:documentation>An overall assessment of the degree to which the set of evidence
statements support the claim. It can be a Bayesian probability, an interval
probability, a fuzzy number, a modal, a value on a Likert scale, etc. ; whatever
the underlying theory of evidence supports.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="EvidenceStatement" type="stepd:EvidenceStatement"/>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="Display">
<xs:restriction base="xs:string">
<xs:enumeration value="Weak Confidence."/>
<xs:enumeration value="Strong Confidence."/>
<xs:enumeration value="No Confidence."/>
</xs:restriction>
</xs:simpleType>
<xs:complexType name="EvidenceStatement">
<xs:sequence>
<xs:element name="ConstituentMultiset">
<xs:complexType>
<xs:sequence>
<xs:element name="PersuasionTactic" maxOccurs="unbounded">
<xs:complexType>
<xs:group ref="stepd:PersuasionTacticGroup"/>

```

```

</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:group name="PersuasionTacticGroup">
<xs:all>
<xs:element name="tactic" type="xs:string"/>
<xs:element name="startline" type="xs:integer"/>
<xs:element name="endline" type="xs:integer"/>
<xs:element name="doc" type="xs:string"/>
</xs:all>
</xs:group>
<xs:complexType name="KbSupport">
<xs:sequence>
<xs:element name="TheoreticalFrame" type="xs:string"/>
<xs:element name="TechnicalTerm" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="TechnicalTermGloss" type="xs:string"/>
<xs:element name="DataSnippet" type="xs:string"/>
</xs:sequence>
<xs:attribute name="term" type="xs:string" default="redefinition"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="KbAssertion">
<xs:sequence>
<xs:element name="AssertionMetadata">
<xs:complexType>
<xs:sequence>
<xs:element name="AssertionId" type="xs:string">
<xs:annotation>
<xs:documentation>Performer team generated ID for this
assertion.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="AssertionDtg" type="xs:dateTime">
<xs:annotation>
<xs:documentation>Performer team generated DTG on which this assertion was
created.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="AssertionFlag" type="stepd:AssertionFlag">
<xs:annotation>
<xs:documentation>Performer team generated flag that indicates whether this is a
"public" or "private" assertion.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="AssertionContext">
<xs:complexType>

```

```

<xs:sequence>
  <xs:element name="LanguageUseDomain" type="xs:string" default="Persuasion
  Attempt">
    <xs:annotation>
      <xs:documentation>A string that specifies the Language Use domain that the
      assertion targets. Will ultimately be an enumerated list.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="DataSet" type="stepd:DataSource">
    <xs:annotation>
      <xs:documentation>The data set from which the evidence for the claim is
      drawn.</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="AssertionClaim" type="stepd:KbClaim"/>
<xs:element name="AssertionEvidence" type="stepd:KbEvidence"/>
<xs:element name="AssertionSupport" type="stepd:KbSupport"/>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="AssertionFlag">
  <xs:restriction base="xs:string">
    <xs:enumeration value="private"/>
    <xs:enumeration value="public"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ServiceState">
  <xs:annotation>
    <xs:documentation>Type that defines the state that a service capability can be
    in.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="available"/>
    <xs:enumeration value="unavailable"/>
    <xs:enumeration value="unsupported"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="ServiceStatusReport">
  <xs:annotation>
    <xs:documentation>Type that defines the status report generated by a Performer
    team.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="KbUpdateCapability">
      <xs:annotation>
        <xs:documentation>Status of the KbUpdate capability. Current state and DTG of last
        kb update.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:sequence>
      <xs:element name="State" type="stepd:ServiceState"/>
      <xs:element name="LastDtg" type="xs:dateTime"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="DataPushCapability">
  <xs:annotation>
    <xs:documentation>Status of the DataPush capability. Current state and DTG of last
    data push.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="State" type="stepd:ServiceState"/>
      <xs:element name="LastDtg" type="xs:dateTime"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ExplanationGetCapability">
  <xs:annotation>
    <xs:documentation>Status of the ExplanationGet capability. Current state and DTG
    of last explanation returned.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="State" type="stepd:ServiceState"/>
      <xs:element name="LastDtg" type="xs:dateTime"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="StatusRequestBundle">
  <xs:annotation>
    <xs:documentation>Type that defines a STEP server request for a status check.
    Currently not used.</xs:documentation>
  </xs:annotation>
</xs:complexType>
<xs:complexType name="StatusReturnBundle">
  <xs:annotation>
    <xs:documentation>Type that defines the Performer team response to a status check
    request.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Payload">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="StatusReport" type="stepd:ServiceStatusReport"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ExplanationRequestBundle">
  <xs:annotation>
    <xs:documentation>Type that defines a STEP server request for an explanation.
    Currently minimally specified.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="Metadata">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="AssertionId" type="xs:string">

```



```

<xs:annotation>
<xs:documentation>STEP ID of the assertion for which an explanation is
requested.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ExplanationReturnBundle">
<xs:annotation>
<xs:documentation>Type that defines the Performer team response to an explanation
request. Currently a placeholder.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="Metadata">
<xs:complexType>
<xs:sequence>
<xs:element name="AssertionId" type="xs:string">
<xs:annotation>
<xs:documentation>STEP ID of the assertion that this explanation refers
to.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Payload">
<xs:complexType>
<xs:sequence>
<xs:element name="Assertion" type="stepd:KbAssertion" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:element name="StatusReportRequestMsgPart">
<xs:annotation>
<xs:documentation>Used in a message sent by the STEP server to request a Performer
team capability status check.</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="Metadata">
<xs:complexType>
<xs:sequence>
<xs:element name="MessageId" type="xs:string">
<xs:annotation>
<xs:documentation>Message ID generated by the STEP server.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="RequestorId" type="xs:string">
<xs:annotation>
<xs:documentation>ID of the requestor (typically the STEP server) of the status
report.</xs:documentation>
</xs:annotation>

```

```

</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="StatusReportResponseMsgPart">
<xs:annotation>
<xs:documentation>Used in a message sent by a Performer team in response to a
StatusReport request.</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="Metadata">
<xs:complexType>
<xs:sequence>
<xs:element name="MessageId" type="xs:string">
<xs:annotation>
<xs:documentation>ID of the status request message to which this message is a
response.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="RequestorId" type="xs:string">
<xs:annotation>
<xs:documentation>ID of the ultimate requestor of the status check. Typically the
STEP server.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Payload">
<xs:complexType>
<xs:sequence>
<xs:element name="StatusReturnBundle" type="stepd:StatusReturnBundle"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="ExplanationGetRequestMsgPart">
<xs:annotation>
<xs:documentation>Used in a message sent by the STEP server to request an
explanation for an assertion.</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="Metadata">
<xs:complexType>
<xs:sequence>
<xs:element name="MessageId" type="xs:string">
<xs:annotation>
<xs:documentation>Message ID generated by the STEP server.</xs:documentation>
</xs:annotation>
</xs:element>

```

```

<xs:element name="RequestorId" type="xs:string">
  <xs:annotation>
    <xs:documentation>ID of the requestor (typically a user) of the
    explanation.</xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Payload">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ExplanationRequestBundle"
        type="stepd:ExplanationRequestBundle"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="ExplanationGetResponseMsgPart">
  <xs:annotation>
    <xs:documentation>Used in a message sent by a Performer team in response to an
    ExplanationGet request.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Metadata">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="MessageId" type="xs:string">
              <xs:annotation>
                <xs:documentation>ID of the explanation request message to which this message is a
                response.</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="RequestorId" type="xs:string">
              <xs:annotation>
                <xs:documentation>ID of the ultimate requestor of the explanation. Typically an
                end-user.</xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Payload">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="ExplanationReturnBundle" type="stepd:ExplanationReturnBundle"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web services: Concepts, architectures and applications*. Heidelberg: Springer.
- Booch, G. (2001, June 1). *Architecture of Web applications*. Retrieved July 26, 2010, from http://www.ibm.com/developerworks/ibm/library/it-booch_web/
- Conallen, J. (1999). Modeling Web application architectures with UML. *Communications Of The ACM* , 42, 63–70.
- Conallen, J. (2003). *Building Web applications with UML; Second Edition*. Boston: Pearson Education, Inc.
- Dijkstra, E. (1982). *Selected writings on computing: A personal perspective*. New York: Springer-Verlag.
- Elmasri, R., & Navathe, S. B. (2007). *Fundamentals of database systems*. Boston: Pearson.
- Erl, T. (2005). *Service-oriented architecture: Concepts, technology, and design*. Crawfordsville, IN: Prentice Hall.
- Erl, T. (2008). *SOA: Principles of service design*. Boston, MA: Prentice Hall.
- Irani, R. (2001, November 21). *Web services architect*. Retrieved February 11, 2010, from <http://www.webservicesarchitect.com/content/articles/irani07.asp>
- Jovanovic, J. (2010, February 25). *Designing user interfaces for business Web applications*. Retrieved July 26, 2010, from <http://www.smashingmagazine.com/2010/02/25/designing-user-interfaces-for-business-web-applications/>
- Larman, C. (2005). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development. 3rd Edition*. Upper Saddle River: Prentice Hall.
- Leffingwell, D., & Widrig, D. (2003). *Managing software requirements: A use case approach*. Boston, MA: Addison-Wesley.
- Marks, E. A., & Bell, M. (2006). *Service-oriented architecture. A planning and implementation guide for business and technology*. Hoboken, NJ: John Wiley & Sons, Inc.
- Marwell, G., & Schmitt, D. R. (1967). Dimensions of compliance-gaining behavior: An empirical analysis. *Sociometry*, 30, 350–364.

- Miller, G. R. (1980). *On being persuaded: Some basic distinctions*. Thousand Oaks, CA: Sage Publishing, Inc.
- Naval Research Laboratory. (2010, May 19). *STEP[Powered by SCIL]*. Retrieved August 1, 2010, from <http://10.3.21.1:8084/STEP/login.jsp>
- Papazoglou, M. P., & van den Heuvel, W.-J. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16, 389–415.
- Schussel, G. (n.d.). *Client/Server: Past, present and future*. Retrieved November 13, 2009, from <http://www.dciexpo.com/geos/dbsejava.htm>
- Spies, B. (2008, May 2). *Web services, Part 1: SOAP vs. REST*. Retrieved March 12, 2010, from <http://www.ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest>
- Tong, R. (n.d.). *MITRE*. Retrieved July 15, 2010, from [https://partners.mitre.org/sites/SCIL/Shared%20Documents/STEP%20WEB%20SERVICES/STEP%20Web%20Services%20Specification%20\(v1.2\)/stepPortTypes_12.wsdl](https://partners.mitre.org/sites/SCIL/Shared%20Documents/STEP%20WEB%20SERVICES/STEP%20Web%20Services%20Specification%20(v1.2)/stepPortTypes_12.wsdl)
- W3C. (2001, March 15). *Web services description language (WSDL)*. Retrieved January 22, 2010, from <http://www.w3.org/TR/wsdl>
- W3C. (2004, February 11). *Web services architecture*. Retrieved March 15, 2010, from <http://www.w3.org/TR/ws-arch/>
- W3C. (2004, October 28). *W3C XML schema part 2: Datatypes second edition*. Retrieved July 14, 2010, from <http://www.w3.org>
- Zimmermann, O., Tomlinson, M., & Peuser, S. (2003). *Perspectives on Web services: Applying SOAP, WSDL and UDDI to real-world projects*. Heidelberg: Springer.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Professor Peter Denning
Naval Postgraduate School
Monterey, California
4. Professor Man-Tak Shing
Naval Postgraduate School
Monterey, California
5. Professor Bret Michael
Naval Postgraduate School
Monterey, California
6. Professor Craig Martell
Naval Postgraduate School
Monterey, California
7. Marine Corps Representative
Naval Postgraduate School
Monterey, California
8. Director, Training and Education, MCCDC, Code C46
Quantico, Virginia
9. Director, Marine Corps Research Center, MCCDC, Code C40RC
Quantico, Virginia
10. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
Camp Pendleton, California