SECURED INFORMATION INTEGRATION WITH A
SEMANTIC WEB-BASED FRAMEWORK

by

Pranav Parikh

APPROVED BY SUPERVISORY COMMITTEE:

___________________________________________
Dr. Bhavani Thuraisingham, Chair

___________________________________________
Dr. Latifur Khan

___________________________________________
Dr. Murat Kantarcioğlu
SECURED INFORMATION INTEGRATION WITH A
SEMANTIC WEB-BASED FRAMEWORK

by

Pranav Parikh, B.E.

THESIS
Presented to the Faculty of
The University of Texas at Dallas
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
MAJOR IN SOFTWARE ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS
December, 2009
ACKNOWLEDGEMENTS

I take this opportunity to thank Dr. Bhavani Thuraisingham, Dr. Murat Kantarcioglu and Dr. Latifur Khan for their constant support, motivation and encouragement. I am also grateful to Scott Streit and Lance Byrd for their extensive feedback and discussions during various phases of the project. Finally, I appreciate the unwavering support of my family members, lab mates and friends who were always there to help in need.

November, 2009
RESTful web services are widely used in industry by Amazon, Yahoo, Google and other companies. Cloud computing services like Amazon S3 aim to provide storage as a low-cost, highly available service with a simple ‘pay as you go’ charging model. Most of the calls made to such services are via RESTful web services. This thesis work makes two contributions. First, we incorporated RESTful web services in a semantic web-based framework and compared the different approaches for building web services. Second, we evaluated Amazon’s Simple Storage Service’s ability to provide storage support for large-scale semantic data used by a semantic web-based framework. We describe cryptographic techniques for enforcing the protection of our published data on Amazon S3 and provide a performance analysis using real data sets. We also explore access control issues associated with such services and provide a solution using Sun’s implementation of eXtensible Access Control Markup Language (XACML).
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS........................................................................................................... iii

ABSTRACT................................................................................................................................. iv

LIST OF FIGURES.......................................................................................................................... viii

LIST OF TABLES........................................................................................................................... ix

CHAPTER 1 INTRODUCTION......................................................................................................... 1

CHAPTER 2 BACKGROUND........................................................................................................... 5

  Semantic Web ................................................................................................................................ 5
  Architecture ................................................................................................................................. 6
  Application Areas ........................................................................................................................ 7

  Blackbook.................................................................................................................................... 7
  Overview...................................................................................................................................... 7

  Objectives .................................................................................................................................... 8

  Business Functions .................................................................................................................... 9

  Interfaces ..................................................................................................................................... 10

  Technologies Used ................................................................................................................... 12
  Processing of a text search .......................................................................................................... 18

  RESTful Web Services............................................................................................................... 20
  Overview ..................................................................................................................................... 20

  REST and Semantic Web .......................................................................................................... 23
Upload Object........................................................................................................69
Download Object ..................................................................................................69
Credentials Properties File..................................................................................70
Initializing S3Service .........................................................................................70
Sample XACML Request .......................................................................................71
BIBLIOGRAPHY ..................................................................................................76
VITA
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Semantic Web Layers</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Blackbook Architecture</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Web Service</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>Processing of Text Search</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>XACML Architecture</td>
<td>28</td>
</tr>
<tr>
<td>2.6</td>
<td>Policy Language Model</td>
<td>30</td>
</tr>
<tr>
<td>4.1</td>
<td>Cloud Computing Infrastructure</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>System Overview</td>
<td>42</td>
</tr>
<tr>
<td>5.1</td>
<td>Upload Statistics</td>
<td>51</td>
</tr>
<tr>
<td>5.2</td>
<td>Overhead Upload</td>
<td>52</td>
</tr>
<tr>
<td>5.3</td>
<td>Download Statistics</td>
<td>53</td>
</tr>
<tr>
<td>5.4</td>
<td>Overhead Download</td>
<td>54</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 8.1 ........................................................................................................60
Table 8.2 ........................................................................................................64
Table 8.3 ........................................................................................................66
CHAPTER 1

INTRODUCTION

The current web represents information using natural languages, graphics and multimedia objects which can be easily understood and processed by a common user. But machines cannot perform tasks which require combining and processing data from different sources. Semantic web is an initiative by World Wide Web Consortium (W3C) in this direction to be able to make machines process such tasks. Semantic web emphasizes integration and combination of data from different data sources. While the current web focuses on documents, semantic web extends the principles of web from documents to data. The framework used to represent information in the web is called Resource Description Framework (RDF), which is a building block of semantic web.

Blackbook is an initiative by IARPA (Intelligence Advanced Research Project Activity) towards building a semantic web-based data integration framework [BLBK]. The main purpose of the Blackbook system is to provide intelligence analysts an easy to use tool to access data from disparate data sources, make logical inferences across the data sources and share this knowledge with other analysts using the system. Besides providing a web application interface, it also exposes its services by means of web services.

A number of protocols and standards designed to build web services are recommended by W3C (called WS-* stack) and referred to as SOAP web services. SOAP web service, based on RPC-style (Remote Procedural Call) accepts an envelope, containing method and scoping information,
from its client and sends a similar envelope back. Some of the advantages of using SOAP web services are: industry wide support and ease of use (due to availability of many tools), interoperability (in form of XML) and support for extensibility (in form of SOAP headers). But such web services sometime become a bottleneck because of its tight coupling property. Nevertheless, most of the existing web services are based on SOAP. Blackbook also supports SOAP web services (Now after the completion of this work, it also provides RESTful web services).

In his PhD dissertation, Roy Fielding proposed a new concept of implementing web services using the Resource Oriented Architecture (ROA) approach [W3ROM] and named it Representational State Transfer (REST) [FIELDING]. REST applies the architecture of web to web services. Everything that can be referenced with a URI is treated as a resource. It allows clients to manipulate resource state by sending a resource’s representation as a part of a PUT or POST request. The server can manipulate client state by sending representations in response to the client’s GET requests.

Experts argue that REST based web services are loosely coupled than SOAP web services with respect to interface orientation, model, generated code and conversation [CPEW2009]. One of the goals of building web-based systems is to achieve loose coupling or no coupling because more interdependencies (tight coupling) make the systems brittle and complicated.

Since Blackbook is a web-based system, implementing RESTful web services for this system eliminates the disadvantages of tight coupling associated with SOAP. Moreover, since
everything is identified with a resource in the REST architectural style, RESTful web services are more suitable to work with RDF data. Blackbook is a semantic web-based infrastructure and semantic data is a collection of different vocabularies. Because of REST’s inherent simplicity, it allows visualizers (clients to the Blackbook services) to show semantic data in an easy manner as compared to SOAP. We tried to leverage these facts to build web services using the REST architectural style for Blackbook.

Blackbook integrates data from different data sources so it is a good idea to store the data sources in a shared environment like the one provided by cloud computing services. But storing shared data in cloud environment in a secure manner is a big challenge. The second part of our work focuses on solving this problem. Cloud computing services like Amazon S3 [AS3] are gaining huge popularity because of factors like cost efficiency and ease of maintenance. We evaluated the feasibility of using S3 storage services for storing semantic web data. Blackbook uses several semantic data sources to produce search results. In our approach, we stored one of the Blackbook data sources on Amazon S3 in a secure manner, thus leveraging cloud computing services within a semantic web-based framework. We encrypted the data source using Advanced Encryption Standard (AES) [AES] before storing it on Amazon S3. Also, we do not store the original key anywhere in our system. Instead, the keys generated by two separate components called “Key Server” are xored to generate the actual key used to encrypt the data.

To prevent replay attacks, we used the Lamport One Time Password [LAMP] scheme to generate the passwords which are used by the client for authentication with the “Key Servers”. We used the Role Based Access Control (RBAC) model [REHC] to restrict system access to
authorized users and implemented the Role Based Access Control policies using Sun’s implementation of eXtensible Access Control Markup Language (XACML) [OAS].

Contributions from this work can be summarized as follows:

1) Incorporated RESTful Web services in Blackbook, a semantic web-based framework, using the JBoss RESTEasy API

2) Securely utilized the cloud computing services like Amazon S3 for semantic web data sharing
CHAPTER 2
BACKGROUND

In this section, we describe the basic concepts about the semantic web technologies and also outline the tools and technologies we have used towards the fulfillment of our thesis work.

Semantic Web

Semantic web provides a common framework that allows data to be shared and reused across applications, enterprise, and community boundaries [W3SW]. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners.

The current web represents information using natural languages, graphics and multimedia objects which can be easily understood and processed by an average user. Some tasks on the web require combining data on the web from different sources e.g. travel and hotel information may come from different web sites when booking for a trip. Humans can merge this information and process them quite easily. However, machines can not combine such information and process it.

So, we need to have the data that should be available for machines for further processing. Data should be possibly combined and merged on a web scale. Data may describe other data and machines may also need to reason about that data. So, we need a web of data. The vision of semantic web is to extend the principles of the web from documents to data [OWL]. Data should be accessed using the general web architecture e.g. using URI’s. Data should be related to one
another like documents are. This also means creation of a common framework that allows data to be shared and reused across applications, enterprise and community boundaries, to be processed automatically by tools as well as manually, including revealing possible new relationships among pieces of data.

Architecture

The Semantic web principles are implemented in the layers of web technologies and standards as shown in Figure 2.1 [OWL]. The Unicode and URI layers ensure that we use international character sets and provide means for identifying objects in semantic web. With the XML layer, with namespace and schema definitions, we can integrate the semantic web definitions with the other XML based standards. With RDF and RDFS Schema, it is possible to make statements about objects with URI’s and define vocabularies that can be referred to by URI’s. This is the layer where we can give types to resources and links. The Ontology layer supports the evolution of vocabularies as it can define relations between the different concepts. The Logic layer enables the writing of rules while the Proof layer executes the rules and evaluates together with the Trust layer mechanism for applications whether to trust the given proof or not.

Figure 2.1 Semantic Web Layers
Application Areas

Semantic Web can be used in a variety of application areas [OWL]:

- Data Integration – whereby data in various locations and various formats can be integrated in one seamless application e.g. Blackbook
- Resource Discovery and Classification – to provide better, domain specific search engine capabilities e.g. Blackbook
- Cataloging – For describing the content and content relationships available at a particular web site, page or digital library
- By Intelligent Software Agents – to facilitate knowledge sharing and exchange
- Content Rating
- In describing collections of pages that represent a single “logical” document and describing intellectual property rights of web pages

Blackbook

The main objective of the Blackbook [BLBK] project is to improve intelligence analysis by coordinated exposition of multiple data sources across intelligence community agencies.

Overview

The Blackbook system is a JEE server-based RDF processor that provides an asynchronous interface to back-end data sources. It’s an integration framework based on semantic web technologies like RDF, RDF Schema, OWL and SPARQL.
It relies on open standards like Jena, Jung, Lucene, JAAS, D2RQ etc to promote robustness and interoperability. Blackbook provided a default web application interface and a SOAP interface. Now, after completion of this work, it provides a RESTful interface as well.

Blackbook connects several data sources [BLBK]

- 911 Report (Unstructured transform via NetOWL -> RDF)
- Monterey – Terrorist incidents (RDBMS -> RDF transform)
- Medline – Bio-data (XML -> RDF)
- Sandia – Terrorist profiles (RDBMS -> RDF transform)
- Anubis – Bio-equipment and bio-scientists (RDBMS -> RDF transform)
- Artemis – Bio-weapons proliferation (RDBMS via D2RQ)
- BACWORTH – DIA (web-services)
- Google-Maps – NGA (via Google-map API)
- CBRN Proliferation Hotlist – CIA (RDBMS -> RDF transform)
- Global Name Recognition service and 3 DBs - JIEDDO
- ICRaD Mediawiki w/ Semantic extension – CIA (dbPedia-like adapter)
- CPD Hercules – CIA (RDBMS via D2RQ)

Objectives

The purpose of Blackbook is to provide analysts with an easy-to-use tool to access valuable data. The tool federates queries across data sources. These data sources may be local or remote databases or applications. Blackbook allows analysts to make logical inferences across the data sources, add their own knowledge and share that knowledge with
other analysts using the system. Also, one of the goals is to leverage industry standards to speed development and to maximize interoperability between Blackbook and external system.

**Business Functions**

In this section, we describe the various business functions provided by Blackbook [BLBK].

**Text Search**

A user can perform a text search against all available data sources, including those available through web services. Text searches seek matching values in the data base. For example, if a text search is for “McCullum,” the results may be for a person with the same surname or a street named “McCullum Street”. The results from a text search bring back the URI of the RDF document.

**Dip**

Dips perform searches on user-specified data sources. These searches look for name-value pairs, so that a Dip for a person named “McCullum” will not return a street named “McCullum Street”. The Dip analogy is to take a value from a text search and “dip” that value into other data sources to see what will stick.
Materialize

Text searches also return the Uniform Resource Identifier (URI) which provides the source of the RDF document. The source may be a RDF or a non-RDF document stored locally or in a remote location.

For example, a URI may point to a MS Word Document (.doc) stored in a database located across the network. The URI goes across the network as an HTTPS link. This facilitates an encrypted data exchange via SSL. The user’s web browser knows how to visualize the document returned based on its MIME [MIME] type. In this case, the web browser will visualize the .doc file with MS Word.

Interfaces

In this section, we outline the various interfaces to Blackbook.

Import Process

The import process allows an analyst to manipulate the OWL representation of an RDF document. Analysts build their own logical inferences through a user interface.

This interface also includes importing algorithms developed to perform social network analysis. The algorithms run against the data sources as a batch process, without any analyst input.

MIME type of RDF/XML

The purpose of this interface is to plug-and-play open source visualizers. The system sends a RDF/XML document, with a MIME [MIME] type of “RDF/XML,” back to the
user’s web browser. The web browser will then know to visualize the RDF/XML document. If the web browser does not know what to do with the RDF/XML document, it asks the user to download it as a file.

**Business Process Execution Language (BPEL)**

BPEL [BPEL] lets the user build a sophisticated query for the workflow of the Text Search and Dips. Using BPEL the user may specify the search order of data sources.

**Blackbook Architecture**

![Figure 2.2 Blackbook Architecture](image)
Figure 2.2 shows a high-level diagram of Blackbook system architecture [BLBK]. The figure shows how two agencies can use the Blackbook system to share and transfer data via web services. The technologies involved in building the Blackbook system are described in detail in this section.

Technologies Used

This section discusses briefly the different technologies used in this project.

Web Services

Blackbook uses web services to automate the data exchange mechanism with any capable enterprise application belonging to organizational partners. Other technologies, such as RMI or JMS [JMS], are capable in building the data exchange mechanism. However, web services give three features that other technologies do not provide:

1. Two-way SSL
2. Use of the HTTP protocol
3. No dependency on JEE server implementation

Figure 2.3 [BLBK] compares the two approaches which the client-server systems can use to communicate with each other. They can interact by means of Enterprise Java Beans. But that would mean to have identical implementation for sending and receiving on both the systems to understand the serialized message.

Another approach is to use web services. This helps to provide an implementation independent of communication but also incurs extra overhead.
Resource Description Framework

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web [W3RDF]. It is the W3C standard for knowledge encoding. RDF provides the ability to express any fact in small, structured pieces and represent the knowledge as a network graph, a set of statements or even as XML.

The design of RDF is suitable for meshing together distributed knowledge sources. The applications use RDF files from different sources to derive new facts. The RDF standard enables this by providing logical descriptions of inferences between facts and instructions.
on how to search for facts in other RDF documents. These facts may occur in the local document, external document, or a combination of both. The RDF standard not only links documents together by a common vocabulary, but also allows any document to use any vocabulary. A vocabulary is a set of consistently and carefully defined terms used to construct RDF statements in conformance with the RDF format.

In Blackbook [BLBK], the RDF format is used to

- Integrate data from different data sources without custom programming
- Offer local data for re-use by other off-site organizations
- Decentralize data such that no “siloing” of data occurs among organizations
- Browse, query, match and extract facts from large amounts of data without developing separate tools for each data source

**Jena**

Jena is an open source framework which provides a Java programming environment for the Resource Description Framework [JENA]. Blackbook project is Java based and Jena provides the Java interface to RDF.

On the web tier, Jena provides Java-based visualizers (clients to Blackbook services) with the ability to manipulate RDF.

On the enterprise tier, it allows Enterprise Java Beans to filter the data in the RDF model based on security credentials.
Lucene

Lucene is an open source project that provides high performance, scalable indexing and powerful, accurate and efficient search algorithms all through a simple API [LUCENE]. It creates an index in a file on the file system and an application using Lucene processes the queries against the indexed file.

Lucene allows the use of multiple indexing algorithms against a data source. This allows multiple query types for a text search. For example, a data source containing news articles may have an index created by an analyzer that ignores common English words (“a”, “an”, “the”) that are usually not useful for searching, and another index that reduces the data to a phonetic encoding. Thus a text search for “Smith” will return results for “Smith” and its phonetic equivalent “Smyth”.

Enterprise Java Beans (EJB)

EJB’s encapsulate the business logic of an application into distributable, usable components [EJB]. The Blackbook project uses the EJB 3.0 specification, a part of the Java platform standard as Java Specification Request #220 [JSR220].
Different types of EJB’s are

1) Stateful Session Beans

Stateful session beans perform logic for the client and can save data across multiple interactions with a single client.

2) Stateless Session Beans

Stateless session beans also perform logic for the client but can’t save data across interactions with its client

3) Entity Bean

Entity beans are responsible for inserting, updating, selecting and removing data within the data source.

4) Message Driven Bean

Message Driven Beans allow the applications to handle asynchronous messages sent by the Java Messaging Service (JMS).

**Java Server Faces (JSF)**

JSF is a Java-based web application framework that simplifies the development of user interfaces for Java enterprise applications [JSF]. Blackbook uses JSF to build the user interface. It uses Oracle ADF faces, which implements the JSF standard. The Oracle ADF
Faces technology uses AJAX to modify web pages dynamically without flickering. The JSF is rendered to HTML before being sent to the user’s browser.

RESTEasy

RESTEasy is a portable implementation of JAX-RS, JSR-311 specification that provides a Java API for restful web services over the HTTP protocol [JSR311]. RESTEasy is a JBoss [JBOSS] project that provides various frameworks to build RESTful web services and RESTful Java applications. It is a fully certified and portable implementation of the JAX-RS specification. JAX-RS is a new JCP specification that provides a Java API for RESTful web Services over the HTTP protocol.

RESTEasy can run in any Servlet container running JDK 5 or higher, but tighter integration with the JBoss Application Server is also available. While JAX-RS is only a server-side specification, RESTEasy has innovated to bring JAX-RS to the client through the RESTEasy JAX-RS Client Framework. This client-side framework allows you to map outgoing HTTP requests to remote servers using JAX-RS annotations and interface proxies.

Features [RESTEASY]:

- Fully certified JAX-RS implementation
- Portable to any application server/Tomcat that runs on JDK 5 or higher
- Embeddable server implementation for JUnit testing
- Rich set of providers for: XML, JSON, YAML, Fastinfoset, Atom, etc.
• JAXB marshalling into XML, JSON, Fastinfoset, and Atom as well as wrappers for arrays, lists, and sets of JAXB Objects.
• Asynchronous HTTP (Comet) abstractions for JBoss Web, Tomcat 6, and Servlet 3.0
• EJB, Spring, and Spring MVC integration
• Client framework that leverages JAX-RS annotations so that you can write HTTP clients easily (JAX-RS only defines server bindings)

JetS3t

JetS3t [JET] is a Java (1.4+) toolkit for Amazon S3 and Amazon CloudFront. Building on the Java library provided by Amazon, the toolkit aims to simplify interaction with Amazon S3.

Processing of a text search

This section shows how a text search is processed in Blackbook.

Figure 2.4 [BLBK] shows the processing of text search in Blackbook system. The steps can be explained as follows [BLBK]:

1) When a user issues a query, the application server (JBoss) invokes the interface method of the Query manager (Stateful session bean) and passes the query to it.

2) A separate query for each data source is placed in the query queue (managed by Java Messaging Service)
3) The container instantiates the message driven bean (MDB) to process query stored on the queue. Each data source has its own message driven bean.
4) The MDB will issue query for each data source and wait for all results from the data sources.

5) When all the results are retrieved, MDB places them in the temporary queue. There is one MDB for each result.

6) Query manager pulls the messages from the temporary queue, one message at a time and the results are displayed to the user from the temporary queue.

RESTful Web Services

In this section, we describe RESTful web services, its relationship with semantic web and compare this approach with the SOAP web services.

Overview

REST is a term coined by Roy Fielding in his Ph.D. dissertation to describe an architectural style of network systems. REST is an acronym for Representational State Transfer.

REST is not a standard but an approach to developing and providing services on the Internet and is thus also considered an architectural style for large-scale software design.

Roy Fielding's explanation of the meaning of Representational State Transfer is:

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user
progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use." [FIELDING]

REST emphasizes [FIELDING]

- The scalability of component interactions
- The generality of interfaces
- The independent deployment of components
- The existence of intermediary components, reducing interaction latency, reinforcing security and encapsulating legacy systems

The present day web has certainly achieved most of the above mentioned goals. The fundamental way how REST achieved these goals is by imposing several constraints:

- **Identification of resources** with Uniform Resource Identifier (URI) means that the resources identified by these URI’s are the logical objects that messages are sent to.

- **Manipulation of resources through representations** means that resources are not directly accessed or manipulated, but instead their representations are used.

- **Self-descriptive messages** refer to the fact that the HTTP messages should be as self-descriptive as possible in order to enable intermediaries to interpret messages and perform services on behalf of the user. This in turn is achieved by standardizing several HTTP methods (e.g. GET, POST etc), many headers and the addressing mechanism.
• Also, HTTP being a stateless protocol allows the interpretation of each message without any knowledge of the preceding messages.

• **Hypermedia as the engine of application state**, enabling the current state of a particular web application to be kept in one or more hypertext documents, residing either on the client or the server. This enables a server to know about the state of its resources, without having to keep track of the states of the individual clients.

REST uses standards such as

• **HTTP**, the Hypertext Transfer Protocol

• **URL**, as the resource identification mechanism

• **XML / HTML / PNG etc** as different resource representation formats

• **MIME types** such as text/xml, text/html, image/png etc

The use of these standards is based on the fundamental characteristics of REST:

• **Client-server**: a pull-based interaction style: consuming components pull representations.

• **Stateless**: each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.

• **Cache**: to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.

• **Uniform interface**: all resources are accessed with a generic interface (e.g. HTTP GET, PUT, POST, DELETE).
• **Named resource**: the system is comprised of resources which are named using a URL

• **Interconnected resource representations**: the representations of the resources are interconnected using URL’s, thereby enabling a client to progress from one state to another.

• **Layered components**: intermediaries, such as proxy servers, cache servers, gateways etc can be inserted between clients and resources to support performance, security etc.

The RESTful systems follow the principles of REST, which evolve around resources, their addressing and the manipulation of their representation. It is still argued whether the distinction between resources and their representations is too impractical for normal use on the web, even though it is popular in the RDF community. For more detailed information, we refer the reader to [LRSR].

**REST and Semantic Web**

RDF resources are perfect candidates for publication via RESTful interfaces. The RDF specification appears to have been conceived with REST in mind. All RDF resources have an URI. If the URI for a resource is actually a live URL that responds with the RDF statements for that resource, you have a RESTful service. Implementing all the HTTP methods (GET, PUT, POST and DELETE) gives a complete interface. [BLBK]

These applications require the identifier of the resource and the action it wishes to invoke. There is no need to know whether there are any intermediaries, such as caching mechanisms, proxies, gateways, firewalls, tunnels etc between it and the server actually holding the information. Applications still have to be able to understand the format of the
information (representation) returned, which is typically an HTML or XML document. Currently, most resources are intended for consumption by humans and hence are represented by HTML. But in areas like semantic web, where machine-to-machine communication becomes more important, the representation of the resources can be done in different formats such as RDF.

Adherence to REST will enable the reference of resources available on other machines, using resource identification mechanisms, such as URL. While a URL represents the noun, the operations such as GET, POST etc represent the verbs that can be applied to them. These basic functionalities are provided by the HTTP protocol and form the basis of the web and its functioning.

**REST vs. SOAP**

Both SOAP and REST are the ways to implement web services.

SOAP applies the Remote Procedure Call (RPC) approach [W3SOAP]. In RPC, the emphasis is on the diversity of protocol operations or verbs. For example, an RPC application might define operations such as the following:

```
getUser()
addUser()
removeUser()
updateUser()
```

REST emphasizes the diversity of resources or nouns. So a REST application might define the following two resource types:
In REST each resource has its own location, identified by its URL. Clients can retrieve representation of these resources through the standard HTTP operations, such as GET, manipulate it and upload a changed copy, using the PUT command, or use the DELETE command to remove all representations of that resource. Each object has its own URL and can be easily cached, copied and bookmarked. Other operations, such as POST can be used for actions with side-effects, such as placing an order, or adding some data to the collection.

To update for instance a user’s address, a REST client would first download the XML record using HTTP GET, modify the file to change the address and upload it using HTTP PUT. The “generality of interfaces” in REST makes it a better basis for a web services framework than the SOAP-based technologies. In contrast to SOAP, where all the method names, addressing model and procedural conventions of a particular service must be known, HTTP clients can communicate with any HTTP server, without knowing any configuration. This is because HTTP is an application protocol whereas SOAP is a protocol framework.

It is noteworthy that the HTTP operations do not provide any standard method for resource discovery. Instead, REST data applications work around the problem by treating a collection or set of search results as another type of resource, requiring application
designers to know additional URL’s or URL patterns for listing or searching each type of resource. As per Berner-Lee’s point of view, the first goal of web is to establish a shared information space. Legacy systems can participate by publishing objects and services into this space. The core of the web’s shared information is the URI. The SOAP-based web services specifications have not adopted the notion of web as a shared information space and thus have not fully adopted the web’s model of URI usage.

They have always rather presumed that every application would set up its own unique namespace from scratch, instead of using URI’s as an addressing mechanism. Each WSDL describes only one web resource and provides no way to describe links to other resources. SOAP and WSDL use URI’s only to address endpoints, which in turn manage all of the objects within them. Technologies like semantic web can only work with web services that identify resources with URI’s and hence REST is an ideal platform for implementing web services for semantic web-based systems.

However, whether to use REST or SOAP to build web services depends on many factors like the application being built, available tools, end users etc. [PZL2008] compares these two approaches of building web services to help decision makers assess the two integration styles and technologies more objectively and select the one that best fits their needs.

**Web Feeds**

A web feed or news feed is a data format used for providing users with frequently updated content. The requested content from the web sites or web services is rendered as a web feed for the user [WSYND]. Content distributors syndicate a web feed, allowing users to subscribe it,
hence web feed is also known as syndicate feed. Making a collection of web feeds accessible in one spot is known as aggregation, which is performed by an Internet aggregator.

A content provider publishes a feed link on their site which end users can register with an aggregator program (also called a 'feed reader' or 'newsreader') running on their own machines. When instructed, the aggregator asks all the servers in its feed list if they have new content; if so, the aggregator makes a note of the new content or downloads it. Aggregators can be scheduled to check for new content periodically. Atom Feed and RSS (Really Simple Syndication) are the most commonly used formats of web feeds

**Access Control Using XACML**

eXtensible Access Control Markup Language (XACML) is an XML-based language for access control that has been standardized in OASIS [OAS]. It describes both an access control policy language and a request/response language. The policy language is used to express access control policies (i.e. who can do what and when). The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses). Benefits of XACML over other access control policy languages [SUN-XACML]:

1. One standard access control policy language can replace dozens of application-specific languages.

2. Administrators and developers save time and money as they don’t need to rewrite the policies in different languages or invent new policy languages and write code for them.
3. XACML is flexible to accommodate most access control policy needs and extensible so that new requirements can be supported. One XACML policy can cover many resources.

4. XACML allows one policy to refer to another. For e.g. a site specific policy can refer to company-wide policy and country-specific policy.

![XACML Architecture](image)

**Figure 2.5 XACML Architecture**

Figure 2.5 shows the XACML architecture. [IBM-XACML]

The components of XACML architecture can be described briefly as follows:

**Policy Enforcement Point (PEP)**

The PEP creates a request based on the requester’s attributes, the resource in action and other information.
**Policy Decision Point (PDP)**

The PDP arrives at a decision after evaluating the relevant policies and the rules within them based on the policy target. The policy target contains information about the subject, action, and other environmental properties.

**Policy Access Point (PAP)**

The PDP uses the Policy Access Point (PAP) which writes the policies and policy sets and makes them available to PDP. In our case, we don’t use the PAP to write the policies.

**Policy Information Point (PIP)**

The PDP may invoke the Policy Information Point (PIP) service to retrieve the attribute values related to subject, resource or environment.

XACML has three top-level components:

- Policy
- PEP
- PDP

The process of creating XACML infrastructure for the request is managed by these components. Figure 2.6 [IBM-XACML] shows how these components are related to each other.
Figure 2.6 Policy Language Model
CHAPTER 3
RESTFUL INTERFACE – IMPLEMENTATION

We implemented the RESTful interface in the following three modules in the Blackbook system:

• Workspace-Blackbook
• Workspace-Workflow
• Workspace-Workspace

This section briefly explains the steps we followed to implement the RESTful interface in each of the three modules.

Workspace-blackbook

BLACKBOOK uses Resteasy API for implementing RESTful Web Services.

1) Dependencies

We injected the dependencies required to implement RESTful web services in the Project Object Model file (an XML representation of a Maven project). We added the required dependencies in blackbook-war/pom.xml

e.g.  <dependency>  <groupId>resteasy</groupId>
        <artifactId>resteasy</artifactId>
        <version>1.0.1.GA</version></dependency>

2) Jar files

The following jar files are required under the .m2/repository/resteasy directory:

jaxrs/1.0.1.GA.jar
jaxrs-api/1.08beta/jaxrs-api-1.0.1.GA.jar
scannotation/1.0.2/scannotation-1.0.2.jar
slf4j-api/1.5.2/slf4j-api-1.5.2.jar
slf4j-simple/1.5.2/slf4j-simple-1.5.2.jar

3) Web.xml settings

RESTEasy is deployed as a WAR archive and thus depends on a servlet container. It is implemented as a ServletContextListener and a Servlet and deployed within a WAR file. The servlet parameters are added in web.xml as shown in Appendix (Workspace-blackbook).

The ResteasyBootstrapListener initializes some basic components of RESTEasy as well as scannotation classes in the WAR file.

4) RESTFUL Servlet

The Restful Servlet “Blackbook.java” is placed in the blackbook-war directory under the package blackbook.web.restful.

The @javax.ws.rs.Path annotation must exist on either the class and/or resource method. If it exists on both the class and method, the relative path to the resource method is a concatenation of the class and method.

The servlet class is annotated with the following annotation: @Path ("/rest")

This maps to the url-pattern we defined in web.xml ("/rest/*").
The setup() method gets a reference to the remote EJB

The getAllAlgorithmClasses() method is annotated with

@GET
@Path("algorithms/ \{feedtype\}")

This means that the URL https://localhost:8443/blackbook/rest/algorithms/{feedType} via HTTP GET method invokes the method getAllAlgorithmClasses(). The value of feedType can be atom_1.0 or rss_0.93.

We get the list of all the algorithm classes by invoking the DataManager bean's getAllAlgorithmClasses(). We use the Java Syndication utilities for the generating the ROME feed for the output. We use the ROME feed for the output because any application can consume the output and utilize the result in its own way.

@PathParam is a parameter annotation which allows mapping variable URI path fragments into the method call.

public String getAllAlgorithmClasses(@PathParam("feedtype") String feedType)

This allows embedding variable identification within the URI of the resources. The “feedtype” parameter is used to pass the feed type the user wants the output. Appendix (Workspace-blackbook) shows the list of all the methods and its corresponding URL's with the arguments.
Workspace-workflow

1) Dependencies

We injected the dependencies required to implement RESTful web services in the Project Object Model file (an XML representation of a Maven project). We added the required dependencies in workflow-war/pom.xml

e.g.  

```xml
<dependency>
  <groupId>resteasy</groupId>
  <artifactId>jaxrs</artifactId>
  <version>1.0.1.GA</version>
</dependency>
```

The complete list can be found in Appendix.(Workspace-workflow)

2) Jar files

The following jar files are required under the .m2/repository/resteasy directory:

- jaxrs/1.08beta/jaxrs-1.0.1.GA.jar
- jaxrs-api/1.08beta/jaxrs-api-1.0.1.GA.jar
- scannotation/1.0.2/scannotation-1.0.2.jar
- slf4j-api/1.5.2/slf4j-api-1.5.2.jar
- slf4j-simple/1.5.2/slf4j-simple-1.5.2.jar

3) web.xml settings

RESTEasy is deployed as a WAR archive and thus depends on a servlet container. It is implemented as a ServletContextListener and a Servlet and deployed within a WAR file.

We change the servlet mapping as shown below:

```xml
<service-mapping>
```
For complete listing, see appendix (Workspace-workflow)

The ResteasyBootstrapListener initializes some basic components of RESTEasy as well as scannotation classes in the WAR file.

### 4) RESTFUL Servlet

The Restful Servlet “Workflow.java” is placed in the workflow-war directory under the package “restful”.

The `@javax.ws.rs.Path` annotation must exist on either the class and/or resource method. If it exists on both the class and method, the relative path to the resource method is a concatenation of the class and method.

The servlet class is annotated with the following annotation: `@Path("/rest")`

This maps to the url-pattern we defined in web.xml (“/rest/*”).

The setup() method gets a reference to the remote EJB.

Appendix (workspace-workflow) shows the list of all the methods and its corresponding URL's with the arguments.

**Workspace-workspace**

### 1) Dependencies

We injected the dependencies required to implement RESTful web services in the Project Object Model file (an XML representation of a Maven project). We added the following dependencies in workspace-war/pom.xml.
<dependency>
  <groupId>resteasy</groupId>
  <artifactId>jaxrs</artifactId>
  <version>1.0.1.GA</version>
</dependency>

Complete listing can be found in Appendix (Workspace-workspace)

2) Jar files

We added the following jar files under the .m2/repository/resteasy directory:

jaxrs/1.08beta/jaxrs-1.0.1.GA.jar
jaxrs-api/1.08beta/jaxrs-api-1.0.1.GA.jar
scannotation/1.0.2/scannotation-1.0.2.jar
slf4j-api/1.5.2/slf4j-api-1.5.2.jar
slf4j-simple/1.5.2/slf4j-simple-1.5.2.jar

3) web.xml settings

RESTEasy is deployed as a WAR archive and thus depends on a servlet container. It is implemented as a ServletContextListener and a Servlet and deployed within a WAR file.

We change the servlet mapping as shown below:

<servlet-mapping>
  <servlet-name>Workspace</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
The ResteasyBootstrapListener initializes some basic components of RESTEasy as well as scannotation classes in the WAR file.

4) RESTFUL Servlet

The Restful Servlet “Workspace.java” is placed in the workspace-war directory under the package “restful”.

The @javax.ws.rs.Path annotation must exist on either the class and/or resource method. If it exists on both the class and method, the relative path to the resource method is a concatenation of the class and method.

The servlet class is annotated with the following annotation: @Path("/rest")

This maps to the url-pattern we defined in web.xml ("/rest/*"). The setup() method gets a reference to the remote EJB. Appendix (workspace-workspace) shows the list of all the methods and its corresponding URL's with the arguments.
CHAPTER 4
INTEGRATING BLACKBOOK WITH AMAZON S3

Introduction
REST is widely used to implement web services in the industry currently. For example, Amazon.com relies heavily on REST for its cloud computing services like Amazon S3. There are certain security issues like access control techniques that need to be designed and implemented for such services. Our current research is focusing on designing and developing access control for cloud computing services. We will also integrate the security technology we develop into Blackbook.

Cloud computing is a paradigm of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet [CLOUD]. The concept incorporates the following combinations:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

Economic advantage is a main motivation behind cloud computing paradigm since it promises the reduction of capital expenditure (CapEx) and operational expenditure (OpEx). [MJNL09]
As shown in Figure 4.1 [CLOUD], various organizations can share data and computational power using the cloud computing infrastructure. For instance, salesforce.com is an industry leader in Customer Relationship Management (CRM) products and one of the pioneers to leverage the cloud computing infrastructure on such a huge scale.

Since Blackbook is a data integration framework, it can search and integrate data from various data sources which may be located on local machines or remote servers.

We utilized the data storage services provided by Amazon S3 to store the data sources.

The reasons we chose Amazon S3 are as follows:

- Cost Effective - Storage price as low as 15 cents per GB per month
- Ease of use - Can be invoked via both REST and SOAP web services
- Reliability - Amazon is big player in Cloud Computing and is known for providing reliable cloud computing services
Privacy risks

Privacy is an important issue for cloud computing services in terms of legal compliance and user trust. The main privacy risks involved are as follows [SP2009]:

- For the cloud service user – being forced to be tracked or give personal information against will
- For the organization using cloud service – non compliance to enterprise policies, loss of reputation and credibility
- For implementers of cloud platforms – exposure of sensitive information stored on the platforms, loss of reputation and credibility
- For providers of applications on top of cloud platforms – legal non-compliance, loss of reputation
- For the data subject – exposure of personal information

Amazon S3

“Amazon S3 is storage for the Internet. It is designed to make web-scale computing easier for developers.

Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers.” [AS3]

Many organizations are using the services like Amazon S3 for data storage. Some important questions arise here –
Is the data we store on S3, secure? Is it accessible by any user outside our organization?

How do we restrict access to the files to the users within the organization?

To keep our data secure, we propose to encrypt the data using AES (Advanced Encryption Standard) before uploading the data files on Amazon S3.

To restrict access to the files to the users within the organization, we propose to implement the Role-based access control policies using XACML. In Role Based Access Control (RBAC), permissions are associated with roles and users are made members of appropriate roles. This simplifies management of permissions. [REHC]
System Overview

The data sources are stored on Amazon S3 server in an encrypted form. The two keys used to encrypt the data source are stored on two servers – Key server 1 and Key server 2. The policies associated with the data sources for different users are also stored on these servers.

Authentication

The system uses the One Time Password (OTP) for authentication. It is a password that is only valid for a single session or transaction. OTPs avoid the shortcomings associated with static passwords [OTP]. Unlike static passwords, they are not vulnerable to replay attacks. So if an
intruder manages to get hold of an OTP that was used previously to log into a service or carry a transaction, the system’s security won’t be compromised since that password will no longer be valid. The only drawback of OTP is that humans cannot memorize them and hence require additional technology in order to work.

**How OTP’s are generated and distributed**

OTP generation algorithms make use of randomness to prevent prediction of future OTPs based on the previously observed OTPs. Some of the approaches to generate OTPs are as follows:

- Use of a mathematical algorithm to generate a new password based on the previous passwords
- Based on time-synchronization between the authentication server and the client providing the password
- Use of a mathematical algorithm where the new password is based on a challenge (e.g. a random number chosen by the authentication server or transaction details) and / or a counter. We use Lamport’s One Time Password scheme for authentication. The Lamport OTP approach is based on a mathematical algorithm for generating a sequence of “passkey” values and each successor value is based on the value of predecessor.

The core of Lamport OTP scheme requires that co-operating client/service components agree to use a common sequencing algorithm to generate a set of expiring one-time passwords (client side) and validate client-provided passkeys included in each client-initiated request (service side). In our case, the client is the Blackbook system and the service components are the “Key Servers”.
The client generates a finite sequence of values starting with a “seed” value and each successor value is generated by applying some transforming algorithm (or F(S) function) to the previous sequence value:

\[ S_1 = \text{Seed}, S_2 = F(S_1), S_3 = F(S_2), S_4 = F(S_3) \ldots S[n] = F(S[n-1]) \]

We use the “password” of the user which is salted [Appendix (Salting)] with some random generated bytes (using SHA1PRNG) as a key to generate the seed value using SHA-256[SHA]. The next values in sequence are generated using the obtained seed value using SHA-256. All these generated values are stored in a stack on the client machine. The topmost value on the stack is stored on both the “Key Servers” (1 & 2).

If the client sends a request for the first time, the topmost value of the client stack is compared with the value on the “Key Servers” (1&2). If the values match, the client is authenticated and the topmost value on the client stack is removed.

For subsequent requests, the topmost value on the client stack is used to compute the successor value using the hash function (used to build the stack). If the generated value and the value on the “Key Servers” match, the user is authenticated; the topmost value on the client stack is stored on the “Key Servers” and subsequently removed from the client stack.

If the client stack gets exhausted, a new stack is generated and the topmost value on the stack is stored on the “Key Servers”.
Once the user is authenticated using the One Time Password scheme, the user request is evaluated against the policies applicable for the resource (data source in our case) requested by the user to access. The pre-defined policies are stored in the “Policy Server” component of the “Key Servers”. If the policies for the resource are applicable for the user request, the “Key Servers” sends the keys used to encrypt the resource requested by the user.

**Authorization**

We use XACML (eXtensible Access Control Markup Language), an XML-base language for access control to implement the access controls using the policies defined in the XML file.

After the user gets authenticated with the system, the system checks if the user is authorized to access the requested resource. The user request is handled by the Policy Enforcement Point (PEP) which creates the request into an XACML request and sends it to the Policy Decision Point (PDP) for further evaluation. The PDP evaluates the request and sends back a response, which can be either “access permitted” or “access denied”, with the appropriate obligations. (We are not considering obligations for our system).

A policy is a collection of several subcomponents: target, rules, rule-combining algorithm and obligations.

**Target:**

Each policy has only one target which helps in determining whether the policy is relevant for the request. The policy’s relevance for the request determines if the policy is to be evaluated for the request, which is achieved by defining attributes of three categories in
the target – subject, resource and action. For e.g. we’ve specified the value “testadmin@blackbook.jhuapl.edu” for the subject and “amazons3” for the resource.

**Rules:**

We can associate multiple roles with the policy. Each rule consists of a condition, an effect and a target.

**Conditions** are statements about attributes that return True, *False* or *Indeterminate* upon evaluation

**Effect** is the consequence of the satisfied rule which assumes the value *Permit* or *Deny*.

We’ve specified the value as *Permit*.

**Target** helps in determining if the rule is relevant for the request.

**Rule Combining Algorithms:**

As a policy can have various rules, it is possible for different rules to generate conflicting results. Rule combining algorithms resolve such conflicts to arrive at one outcome per policy per request. Only one rule combining algorithm is applicable to one policy.

**Obligations:**

Obligations provide the mechanism to give much finer-level access control than mere permit and deny decisions. They are the actions that must be performed by the PEP in conjunction with the enforcement of an authorization decision.

After successful authentication and authorization, the Amazon File Manager downloads the requested resource from Amazon S3 server.

More specifically, Key Server – 1 sends key1 and the Key Server – 2 sends key2

The keys are XORED to get $key_{org}$ i.e.

\[ key_{org} = key1 \ XOR key2 \]
key$_{org}$ is used to decrypt the resource by the Encryption / Decryption Service Provider.

**Why two key servers are used?**

The main motive behind using two key servers is to avoid a single point of failure. If any of the key servers gets hacked, the data is not compromised as two keys, one from each of the key servers are needed to decrypt the data sources.

In case one of the key servers is hacked and the keys stored on that server are compromised, we run into the risk of rendering the data source stored on Amazon useless as we need two keys, one from each key server, to retrieve the original key used to encrypt the data source. To avoid this, we propose to take periodic backups of the keys on each of the key server.

**Scenario**

In this section, we describe a sample scenario, depicting the interaction with the Amazon S3 storage service, with respect to the Blackbook system.

1. The user U fires a search query to Blackbook (Step 1 in figure 4.2).

   Blackbook federates the queries across various data sources along with data source F securely on Amazon S3.

2. We follow the One Time Password (OTP) scheme to authenticate the client (Blackbook in this case) for using the AWS S3 services. The client machine sends the topmost value on the OTP stack along with the user
credentials and the request to the key server 1 & 2. (Steps 2a and 2b in figure 4.2)

3. If the value passed by the client matches with the value on the OTP stack on the key server and the policies applicable for the user are valid for the request, the key server sends the “key” used to decrypt the data source. (Step 3a and 3b in figure 4.2)

4. The keys key1 and key2 obtained from the key servers 1 & 2 are xored to obtain the original key used to decrypt the data source F (Step 4 in figure 4.2)

5. Amazon File Manager passes the Amazon account credentials and the data source name to retrieve the data source. (Steps 5 and 6 in figure 4.2)

6. The Encryption / Decryption Service Manager retrieves the encrypted data sources and using the XOR-ed key, decrypts the data source. (Steps 7 & 8 in figure 4.2)

7. Blackbook performs search on the data source retrieved from Amazon along with other data sources and returns the results to the user. (Step 9 in figure 4.2)

A sample XACML request

The subject, testadmin@blackbook.jhuapl.edu, which belongs to users group (attribute of the subject), is trying to perform a read action on the resource amazons3. To create such a request, we need two subject attributes, one resource attribute and one action attribute. The two subject attributes are rfc822Name (e-mail ID) and the group to which the subject belongs. The one
resource attribute is the URI of the resource, and the one action attribute is the read action on the resource. The complete listing which demonstrates the creation of the PEP with all of these attributes can be found in Appendix.
CHAPTER 5
EXPERIMENTAL RESULTS

In our approach, we have used the Advanced Encryption Standard to encrypt the data before storing it on Amazon S3 server. Uploading the data on the Amazon server is a one-time process. The data source needs to be uploaded again only when the stored data needs to be modified. But the data source stored on Amazon S3 needs to be downloaded every time the user issues a search query to the Blackbook system. Since, the data source needs to be decrypted every time a query is issued, it may affect performance since encryption and decryption are costly operations.

We ran the experiments on a Dell desktop computer running on Ubuntu Gutsy 7.10 with the following hardware configuration:

Intel® Pentium® 4 CPU 3.00 GHZ, 1 GB RAM

The network bandwidth while running the experiments varied between 250 and 300 Mbps. We generated the data files using the triple generation program provided by SP2B, the SPARQL Performance Benchmark [SP2B]. We experimented with 30 files of different sizes, ranging from 1 MB to 30 MB.
Figure 5.1 Upload Statistics

Figure 5.1 shows the upload statistics represented in the form of a Time vs. Size graph. We experimented by uploading all the 30 sample data sources on Amazon S3, with and without encryption and compared the resultant time. The results indicate that for small data sets, the difference between the time taken to upload with and without encryption is negligible.
Figure 5.2 shows the overhead incurred due to encryption of data files before storing on Amazon S3. The results show that the total overhead due to encryption does not exceed 10%. Moreover, we observed that some inconsistency in the results is because of fluctuating network traffic.
Figure 5.3 Download Statistics

Figure 5.3 shows the download statistics represented in the form of a Time vs. Size graph. We experimented by downloading all the 30 sample data sources on Amazon S3, with and without decryption and compared the resultant time. The results indicate that for small data sets, the difference between the time taken to upload with and without decryption is negligible.
Figure 5.4 Overhead Download

Figure 5.4 shows the overhead incurred due to encryption of data files before storing on Amazon S3. The results show that the overall overhead due to decryption does not exceed 10% . We observed that some inconsistency in the results is because of fluctuating network traffic.
CHAPTER 6
RELATED WORK

Yahoo!’s Web services uses the REST architecture, where specially constructed URL’s return an XML response in a unique format [YREST]. Apart from web search results, it also includes the ability to fetch results from images, local information, news and video. In our work, we also make the calls to the RESTful web services and return the response in the form of Atom / RSS feeds which are built on top of XML.

The Google AJAX search API provides simple web objects to allow embedding Google Search in our web pages and other web applications [GREST]. The API exposes a raw RESTful interface that returns JSON encoded results that are easily processed by most languages and runtime. In our approach, we also use AJAX to handle the results and use the ROME feeds to display results.

In [GMPH], the authors propose a new, distributed architecture that allows an organization to outsource its data management to two untrusted servers while preserving data privacy. In our work, we propose the use of two key servers to store the keys whereas the data is stored entirely on Amazon S3. In [PIRG], the authors evaluate whether S3 is a feasible and cost effective alternative for offloading storage from in house maintained mass storage systems for scientific collaborations. In [GEDA], authors propose a framework to enforce access control policies on published XML documents using cryptography techniques. The owner publishes a single data
instance, which is partially encrypted and which enforces all access control policies. The data owner enforces an access control policies by granting keys to users. Comparing this to our approach, we provide two keys to clients which are xored to generate the original key.

In [AML09], the authors explore how to use the search-as-a service and present a new file system search technique that integrates access control and indexing/searching mechanisms into a unified framework to support access control aware search. In our approach, we are using the Amazon service as a storage service and rely entirely on our system to search from the data we store on Amazon. In [VFJPS07], the authors propose a novel architecture that makes use of two distinct repositories - one repository to store the encrypted representation of resources and other repository dedicated to storage and management of the access policy. Each user is assigned a key by the owner and then a set of tokens is used to allow the derivation of one key from another key. The tokens can be used only by the users having the access to secret key. In our approach, we don’t provide any secret key to the user. Instead, two keys exist to decrypt the data stored at Amazon. The users can access those keys when they are authenticated and authorized by our key servers.
CHAPTER 7

CONCLUSIONS AND FUTURE WORK

SOAP-based web services and the REST architectural style have been the topic of many debates. The more prescriptive style of SOAP approach and the descriptive style of REST approach have their roots in different scenarios. The SOAP approach assumes closed worlds and contractual relationships whereas the REST style caters to an open world with ad-hoc interactions [SVIN08].

Blackbook, a semantic web based data integration framework, allows data integration from various data sources. We argue that RDF resources are perfect candidates for publication via RESTful web services. Since RESTful web services and semantic web, both deal with resources, it makes sense to expose RDF resources via RESTful interface. Technologies like semantic web can only work with web services that identify resources with URI’s and hence REST is an ideal platform for implementing web services for semantic web-based systems.

Cloud computing paradigm is becoming increasingly important in today’s world. So, the issues like data security and privacy have also gained lot of attention along with it. In [SP2009], the author provides some interesting insights about how privacy issues should be taken into consideration when designing cloud computing services.

We suggest techniques to protect our data by encrypting it before storing on cloud computing servers like Amazon S3. Our approach is novel as we propose to use two key servers to generate and store the keys. Also, we assure more security than some of the other known approaches as
we don’t store the actual key used to encrypt the data. This assures the protection of our data even if one or both the key servers are compromised.

In our current approach, we download the data source for every request from the user. In future, we can make the provision to cache the data source requested by the user on our local server and provide the results to the user from the cached data source by treating it as a local data source. This approach will help to enhance performance.

Also, we can divide the data source in chunks and then upload it. Since the search process takes place in an asynchronous manner, we can download the chunks, search for results and display them to the user, one at a time. Meanwhile, the application can keep downloading other chunks. We can also keep track of search history for a user. When the user logs in to the system, we can download those chunks from where the user is most likely to perform search on.
CHAPTER 8
APPENDIX

Workspace-blackbook

Web.xml – settings
This section shows the servlet parameters to be added in the web.xml file

    <context-param>
        <param-name>resteasy.scan</param-name>
        <param-value>true</param-value>
    </context-param>

    <listener>
        <listener-class>
            org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap
        </listener-class>
    </listener>

    <servlet>
        <servlet-name>Blackbook</servlet-name>
        <servlet-class>
            org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher
        </servlet-class>
    </servlet>
<servlet-mapping>
    <servlet-name>Blackbook</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>

The table 8-1 gives the information about the method name corresponding to the URL invoked and the HTTP method.

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>URL</th>
<th>Method Name</th>
<th>HTTP Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="https://localhost:8443/blackbook/rest/algorithms/%7BfeedType%7D">https://localhost:8443/blackbook/rest/algorithms/{feedType}</a></td>
<td>getAllAlgorithmClasses()</td>
<td>GET</td>
</tr>
<tr>
<td>2</td>
<td><a href="https://localhost:8443/blackbook/rest/getFieldNames/datasources/%7BfeedType%7D">https://localhost:8443/blackbook/rest/getFieldNames/datasources/{feedType}</a></td>
<td>getAllDataSources()</td>
<td>GET</td>
</tr>
<tr>
<td>3</td>
<td><a href="https://localhost:8443/blackbook/rest/localdatasources/%7BfeedType%7D">https://localhost:8443/blackbook/rest/localdatasources/{feedType}</a></td>
<td>GetLocalDataSource()</td>
<td>GET</td>
</tr>
<tr>
<td>4</td>
<td><a href="https://localhost:8443/blackbook/rest/fieldnames/%7Bdatasources%7D/%7BfeedType%7D">https://localhost:8443/blackbook/rest/fieldnames/{datasources}/{feedType}</a></td>
<td>getFieldNames()</td>
<td>GET</td>
</tr>
<tr>
<td>5</td>
<td><a href="https://localhost:8443/blackbook/rest/keyword/%7Bdatasource%7D/%7Bsearch%7D/%7BfeedType%7D">https://localhost:8443/blackbook/rest/keyword/{datasource}/{search}/{feedType}</a></td>
<td>getKeyword()</td>
<td>GET</td>
</tr>
<tr>
<td>6</td>
<td><a href="https://localhost:8443/blackbook/rest/lucenekeyword/%7Bdatasource%7D/%7Bkeyword%7D/%7BfeedType%7D">https://localhost:8443/blackbook/rest/lucenekeyword/{datasource}/{keyword}/{feedType}</a></td>
<td>luceneKeyword()</td>
<td>GET</td>
</tr>
</tbody>
</table>
Dependencies to be added in pom.xml [Same for workspace-workflow and workspace-workspace]

This section shows the dependencies to be added in the project’s pom.xml file.

```
<dependency>
  <groupId>resteasy</groupId>
  <artifactId>scannotation</artifactId>
  <version>1.0.2</version>
</dependency>

<dependency>
  <groupId>resteasy</groupId>
  <artifactId>jaxrs-api</artifactId>
  <version>1.0.1.GA</version>
</dependency>

<dependency>
  <groupId>resteasy</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.5.2</version>
</dependency>

<dependency>
  <groupId>resteasy</groupId>
  <artifactId>slf4j-simple</artifactId>
</dependency>
```
Web.xml settings

This section shows the servlet parameters to be added in the web.xml file

<context-param>
    <param-name>resteasy.scan</param-name>
    <param-value>true</param-value>
</context-param>

<listener>
    <listener-class>org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap</listener-class>
</listener>
<servlet>
    <servlet-name>Workflow</servlet-name>
</servlet>
<servlet-class>
    org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher
</servlet-class>    </servlet>

<servlet-mapping>
    <servlet-name>Workflow</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>

Table 8.2 gives the information about the method name corresponding to the URL invoked and the HTTP method.

### Table 8.2

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>URL</th>
<th>Method Name</th>
<th>HTTP Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="https://localhost:8443/workflow/rest/processdefinition/">https://localhost:8443/workflow/rest/processdefinition/</a></td>
<td>CreateProcessDefinition()</td>
<td>PUT</td>
</tr>
<tr>
<td>2</td>
<td><a href="https://localhost:8443/workflow/rest/processdefinition/%7Bfeedtype%7D/%7Bprocessdefinitionid%7D">https://localhost:8443/workflow/rest/processdefinition/{feedtype}/{processdefinitionid}</a></td>
<td>readProcessDefinition()</td>
<td>GET</td>
</tr>
<tr>
<td>3</td>
<td><a href="https://localhost:8443/workflow/rest/processdefinition/%7Bprocessdefinitionid%7D">https://localhost:8443/workflow/rest/processdefinition/{processdefinitionid}</a></td>
<td>DeleteProcessDefinition()</td>
<td>DELETE</td>
</tr>
<tr>
<td>4</td>
<td><a href="https://localhost:8443/workflow/rest/processinstance/%7Bprocessdefinitionid%7D">https://localhost:8443/workflow/rest/processinstance/{processdefinitionid}</a></td>
<td>StartProcessDefinition()</td>
<td>PUT</td>
</tr>
<tr>
<td>5</td>
<td><a href="https://localhost:8443/workflow/rest/processdefinition">https://localhost:8443/workflow/rest/processdefinition</a></td>
<td>updateProcessDefinition()</td>
<td>POST</td>
</tr>
<tr>
<td>6</td>
<td><a href="https://localhost:8443/workflow/rest/processinstance/atom_1.0/%7BprocessInstanceId%7D">https://localhost:8443/workflow/rest/processinstance/atom_1.0/{processInstanceId}</a></td>
<td>getProcessInstance()</td>
<td>GET</td>
</tr>
</tbody>
</table>
Workspace-workspace

web.xml - settings

This section shows the servlet parameters to be added in the web.xml file

<context-param>
    <param-name>resteasy.scan</param-name> <param-value>true</param-value>
</context-param>

<listener>
    <listener-class>
        org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap
    </listener-class></listener>

<servlet>
    <servlet-name>Workspace</servlet-name>
    <servlet-class>
        org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher
    </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>Workspace</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
Table 8.3 gives the information about the method name corresponding to the URL invoked and the HTTP method.

### Table 8.3

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>URL</th>
<th>Method Name</th>
<th>HTTP Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="https://localhost:8443/workspace/rest/rootfolder/">https://localhost:8443/workspace/rest/rootfolder/</a></td>
<td>getRootFolder()</td>
<td>GET</td>
</tr>
<tr>
<td>2</td>
<td><a href="https://localhost:8443/workspace/rest/subfolder/%7Bsubfoldername%7D">https://localhost:8443/workspace/rest/subfolder/{subfoldername}</a></td>
<td>CreateSubFolder()</td>
<td>PUT</td>
</tr>
<tr>
<td>3</td>
<td><a href="https://localhost:8443/workspace/rest/subfolder/%7Bparentfolderid%7D/%7Bsubfoldername%7D">https://localhost:8443/workspace/rest/subfolder/{parentfolderid}/{subfoldername}</a></td>
<td>CreateSubFolder()</td>
<td>PUT</td>
</tr>
<tr>
<td>4</td>
<td><a href="https://localhost:8443/workspace/rest/item/%7BfeedType%7D/%7BitemId%7D">https://localhost:8443/workspace/rest/item/{feedType}/{itemId}</a></td>
<td>getChildItems()</td>
<td>GET</td>
</tr>
<tr>
<td>5</td>
<td><a href="https://localhost:8443/workspace/rest/processdefinition/%7BparentFolderId%7D/%7BProcessDefinitionName%7D">https://localhost:8443/workspace/rest/processdefinition/{parentFolderId}/{ProcessDefinitionName}</a></td>
<td>CreateProcessDefinition()</td>
<td>PUT</td>
</tr>
<tr>
<td>6</td>
<td><a href="https://localhost:8443/workspace/rest/processdefinition/%7Busername%7D">https://localhost:8443/workspace/rest/processdefinition/{username}</a></td>
<td>GetProcessDefinitionID()</td>
<td>GET</td>
</tr>
<tr>
<td>7</td>
<td><a href="https://localhost:8443/workspace/rest/item/%7BitemId%7D">https://localhost:8443/workspace/rest/item/{itemId}</a></td>
<td>RemoveItem()</td>
<td>DELETE</td>
</tr>
</tbody>
</table>
Salting

This section lists the method used for salting as discussed in Chapter 4

```java
public String generateSalt() {

    SecureRandom random = null;
    String s = null;

    try {
        random = SecureRandom.getInstance("SHA-256");
        byte[] seed = random.generateSeed(20);
        s = new String(seed);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }

    return s;
}
```

Seed Generation

This section lists the method used for salting as discussed in Chapter 4

```java
public String genSeed(String key) {

    MessageDigest md;

    try {
        md = MessageDigest.getInstance("SHA-256");
    }
```
```java
md.reset();
md.update(key.getBytes(), 0, key.length());

String salt = generateSalt();
if(salt == null) {
    salt = "";
}

return ""+ new BigInteger(1,md.digest()).toString(16)+salt;
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
    return null;
}

Password Generation

This section lists the method used for generating one time passwords as discussed in Chapter 4.

public void genValues(int n,String username,String password,String authKeyServer,
        String clientKeyServer,OTPStorage otpStorage) {
    String value = null;
    value = genSeed(password);
    passwordLifo.add(value);
    for (int i=0; i < n-1 ; i++) {
        value = nextValue(value);
        passwordLifo.add(value);
    }
    otpStorage.storeServerStackValue(username, authKeyServer, value);
    otpStorage.storeStackValues(username,clientKeyServer,passwordLifo);
}
Upload Object

This section lists the method used for uploading object on Amazon S3 as discussed in Chapter 4

/**
 * This method uploads a file specified by
 * the File Object on Amazon S3 Server
 * @param file
 */
public boolean uploadObject(File file)
throws IOException{
    initialize();
    File encFile = crypto.encryptFile(file);
    S3Bucket bucket = null;
    S3Object fileObject = null;
    try{
        bucket = s3Service.getBucket(bucketName);
        fileObject = new S3Object(bucket,encFile);
        s3Service.putObject(bucket, fileObject);
    }
    catch(Exception e){
        e.printStackTrace();
        return false;
    } return true;
}

Download Object

This section lists the method used for downloading object from Amazon S3 as discussed in Chapter 4

/** * @param fileName
 * @return */
public InputStream downloadObject(String fileName){
    initialize();
S3Bucket bucket = null;
S3Object s3Obj = null;
InputStream in = null;
InputStream newIn = null;
try {
   bucket = s3Service.getBucket(bucketName);
   s3Obj = s3Service.getObject(bucket, fileName);
   in = s3Obj.getDataInputStream();
   if (encrFlag)  
      newIn = crypto.decryptFile(fileName, in);
   else  
      newIn = in;
} catch (Exception e) {
   e.printStackTrace();
   return newIn;
}

**Credentials Properties File**

This section lists the contents of credentials properties file as discussed in Chapter 4

awsAccessKey = [The AWS access key provided by Amazon S3]
awsSecretKey = [The AWS secret key provided by Amazon S3]
bucketName = researchutdallas [ Our bucket name on Amazon S3]

**Initializing S3Service**

This section lists the method used for initializing S3 service as discussed in Chapter 4

private void initialize() {
   try {
      Properties credentials = new Properties();
   
   }
credentials.load(getClass().getResourceAsStream("/"+Constants.AWS_CREDENTIALS_FILEPATH));

awsAccessKey = credentials.getProperty("awsAccessKey");
awsSecretKey = credentials.getProperty("awsSecretKey");
awsCredentials = new AWSCredentials(awsAccessKey, awsSecretKey);
bucketName = credentials.getProperty("bucketName"); s3Service = new
RestS3Service(awsCredentials); catch(Exception e){ e.printStackTrace();

} }

**Sample XACML Request**

This section lists the sample XACML request as discussed in Chapter 4

<Request>

<Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">

<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="urn:oasis:names:tc:xacml:1.0:datatype:rfc822Name">

<AttributeValue>testadmin@blackbook.jhuapl.edu</AttributeValue>

</Attribute>

<Attribute AttributeId="group" DataType="http://www.w3.org/2001/XMLSchema#string">

<AttributeValue>users</AttributeValue>

</Attribute>

</Subject> <Resource>
A typical policy file for our system looks like this:

```
<Policy PolicyId="AmazonAccessPolicy"
     RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
                 algorithm:ordered-permit-overrides">
    <Description>This policy file is used to authorize users with Amazon S3
                 datasource</Description>

    <Target>
        <Subjects>
           <Subject>
```

<SubjectMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">testadmin@blackbook.jhuapl.edu</AttributeValue>
  <SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="urn:oasis:names:tc:xacml:1.0:datatype:rfc822Name"/>
</SubjectMatch>
</Subject>
</Subjects>
<Resources>
  <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#anyURI">amazons3</AttributeValue>
      <ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </ResourceMatch>
  </Resource>
</Resources>
<Resources/>

<Actions>

<AnyAction/>

</Actions>

</Target>

<Rule RuleId="ProjectPlanAccessRule" Effect="Permit">

<Target>

<Subjects>

<AnySubject/>

</Subjects>

<Resources>

<AnyResource/>

</Resources>

<Actions>

<Action>

<ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">

<AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue

> <ActionAttributeDesignator

AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"

DataType="http://www.w3.org/2001/XMLSchema#string"/>

</Action>

</Actions>

</Rule>
<ActionMatch/>
</Action>
</Actions>
</Target>

<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
    <SubjectAttributeDesignator AttributeId="group"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
  </Apply>
  <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">users</AttributeValue>
</Condition>  </Rule></Policy>
# BIBLIOGRAPHY


15. [JET] http://jets3t.dev.java.net/


27. [OWL] OWL: Representing information using the Web Ontology Language – By Lacy Lee
33. [SP2B] http://www.openlinksw.com/dataspace/vdb/weblog/vdb%27s%20BLOG%20%5B136%5D/1423


37. [URI] http://www.w3.org/Addressing/


41. [W3SOAP] http://www.w3.org/TR/soap/

42. [W3SW] http://www.w3.org/2001/sw/


VITA

Pranav attended higher-secondary school in Vadodara, India. After completing his work at the school, he entered the reputed L.D.College of Engineering in Ahmedabad to pursue a Bachelors in Computer Engineering. After completing under graduation in June 2004, he joined a start-up company called Actax Infotech as a Software Programmer in his hometown of Vadodara. A year later, he became associated with TATA Consultancy Services at Mumbai as an Assistant Systems Engineer and also worked on software development projects for General Electric. Additionally, he was involved in leadership and mentorship roles. After working in the industry for 3 years, Pranav decided to pursue further studies and enrolled in the Graduate program in Software Engineering at The University of Texas at Dallas in Fall 2007. There he worked on some interesting research projects in the Semantic Web Research Lab in the Computer Science department under the guidance of Dr. Bhavani Thuraisingham, Dr. Murat Kantarcioglu and Dr. Latifur Khan.