DESIGN and Implementation of Restful Web Services for Blackbook

Technical Report UTDCS-25-09

Department of Computer Science
The University of Texas at Dallas

August 2009

Pranav Parikh, Murat Kantarcioglu, Latifur Khan and Bhavani Thuraisingham

DESIGN AND IMPLEMENTATION OF RESTFUL WEB SERVICES FOR BLACKBOOK

- TECHNICAL REPORT

Pranav Parikh
Murat Kantarcioglu
Latifur Khan
Bhavani Thuraisingham

The University of Texas at Dallas

August 2009

ABSTRACT

The main objective of the Blackbook project is to improve intelligence analysis by coordinated exposition of multiple data sources across intelligence community agencies. The Blackbook system is a JEE server-based RDF processor that provides an asynchronous interface to back-end data sources. It is 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, and D2RQ among others to promote robustness and interoperability.

At the University of Texas at Dallas, under the KDD (Knowledge Discovery and Dissemination) Program funded by IARPA (Intelligence Advanced Research Projects Activity), we are (1) conducting research on semantic web (which includes ontology alignment and RDF query processing), (2) making enhancements to the Blackbook system (e.g., REST Interface, Geospatial Proximity techniques, and Hbase/Lucene integration) and making contributions to open source products (e.g., Large RDF Graph management techniques into JENA open source system).

In this report we will describe the enhancements we have made to Blackbook with respect to RESTful web services. The main objective of this sub-project was to integrate RESTful Web services in BLACKBOOK. Previously, BLACKBOOK supported only the SOAP web services. BLACKBOOK is a semantic web based infrastructure. Since semantic data is a collection of different vocabularies, REST allows visualizers to show semantic data in an easy manner as compared to SOAP. Hence, a pressing need was felt to incorporate the RESTful Web services in BLACKBOOK. We will describe the design and implementation of the Blackbook RESTful web services we have carried out.

ACKNOWLEDGEMENT

We thank IARPA for their support for this effort. We also thank Scott Streit and Lance Byrd for their extensive feedback and discussions of our work for IARPA.

This work will be part of Pranav Parikh's MS Thesis at the University of Texas at Dallas.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Project Objective

The main objective of the project was to integrate RESTful Web services in BLACKBOOK. Previously, BLACKBOOK supported only the SOAP web services. BLACKBOOK is a semantic web based infrastructure and since, semantic data is a collection of different vocabularies, REST allows visualizers to show semantic data in an easy manner as compared to SOAP. Hence, a pressing need was felt to incorporate the RESTful Web services in BLACKBOOK.

The organization of this report is as follows: In section 2 we discuss briefly the semantic web concepts. In section 3 we provide an overview of Blackbook. The design of our system is discussed in section 4. Implementation details are given in section 5. We discuss directions in section 6. References are given in section 7.

2 SEMANTIC WEB CONCEPTS

2.1 Overview

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

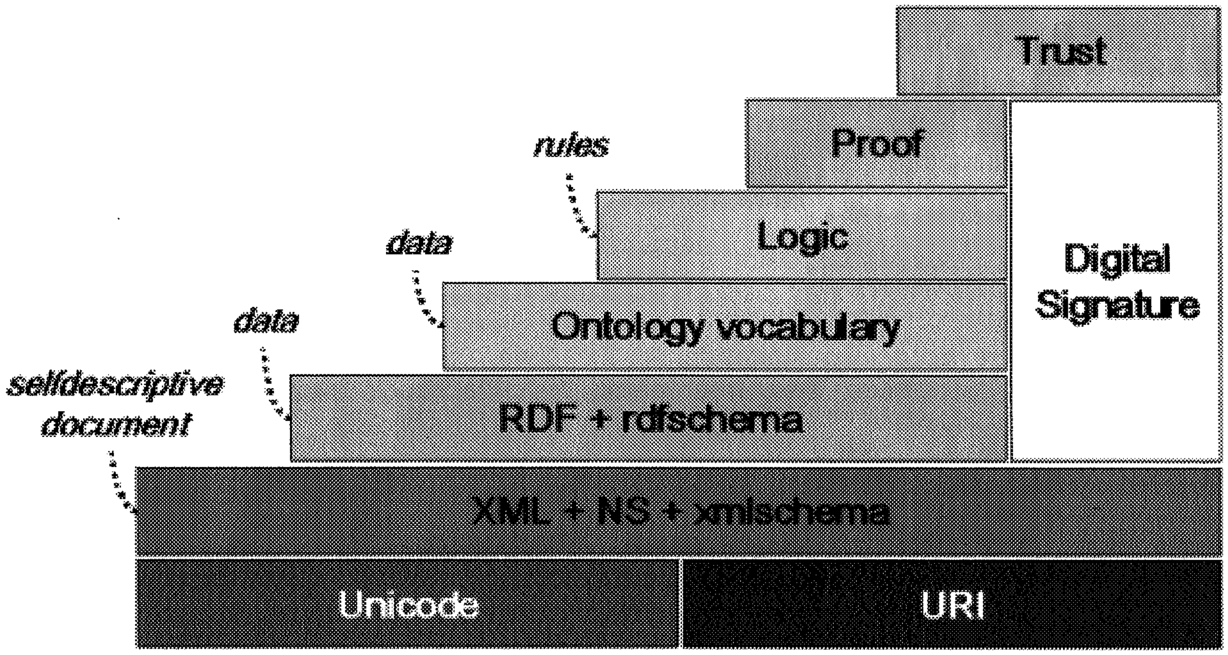
The current web represents the information using natural languages, graphics and multimedia objects which can be easily understood and processed by an average user. Some tasks on the web require combining data on the web from different sources e.g hotel and travel information may come from different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

2.2 Architecture

The Semantic Web principles are implemented in the layers of Web technologies and standards. 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 RDFSchema, 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.



Semantic Web Layers

2.3 Application Areas

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

- Data Integration – whereby data in various locations and various formats can be integrated in one seamless application
- Resource Discovery and Classification – to provide better, domain specific search engine capabilities
- 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
- For describing intellectual property rights of Web pages and many others.

3 BLACKBOOK

3.1 Overview

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

The Blackbook system is a JEE server-based RDF processor that provides an asynchronous interface to back-end datasources.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 provides a default web application interface, SOAP interface and RESTful interface.

Blackbook connects several datasources

- 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)

3.2 Objectives

The purpose of BLACKBOOK is to provide analysts with an easy-to-use tool to access valuable data. The tool federates queries across datasources. These datasources are databases or applications located either locally or remotely on the network. BLACKBOOK allows analysts to make logical inferences across the datasources, add their own knowledge and share that knowledge with other analysts using the system.

3.3 Business Functions

3.3.1 Text Search

A user performs a text search against all available datasources. These datasources include those available through Web Services. Text searches search for matching values in the database. For example, if a text search is for "Smith," the results may be for a person with the same surname or a street named "Smith Street".

The results from a text search bring back the URI of the RDF document.

3.3.2 Dip

Dips perform searches on user-specified datasources. These searches look for name-value pairs, so that a Dip for a person named "Smith" will not return a street named "Smith Street".

The Dip analogy is to take a value from a text search and "dip" that value into other datasources to see what will stick.

3.3.3 Materialize

Text searches also return the Uniform Resource Identifier (URI). These URIs return the source of the RDF document. The source may be a RDF or 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 allows an encrypted data exchange via SSL. The user's web browser knows how to visualize the document returned based on its MIME type. In this case, the web browser will visualize the .doc file with MS Word.

3.4 Interfaces

3.4.1 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 by researchers. These algorithms perform social network analysis. The algorithms run against the datasources as a batch process, without any analyst input. Text Search, Dip, Materialize the Text Search, Dip and Materialize interfaces are the Business functions of BLACKBOOK. The description of these functions is in the previous topic.

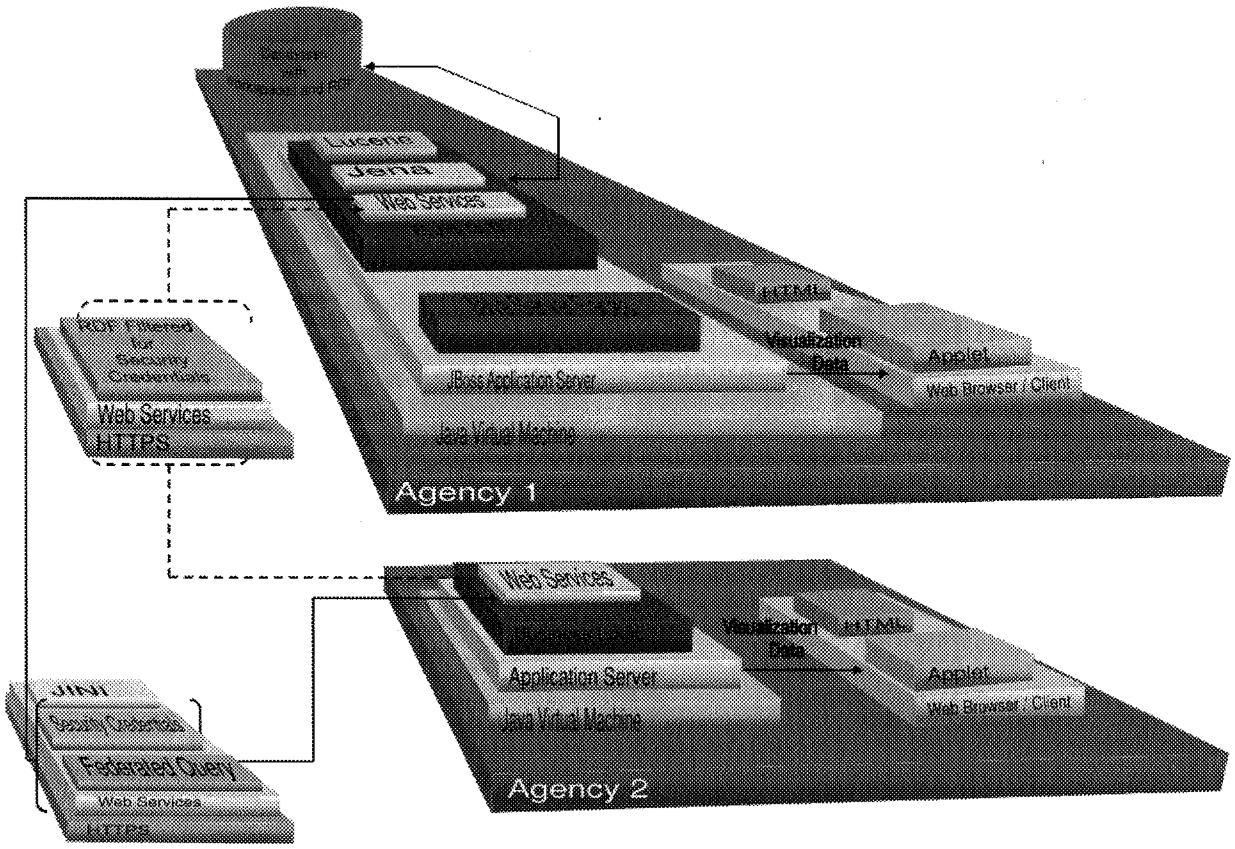
3.4.2 MIME type of RDF/XML

The purpose of this interface is to plug-and-play open source visualizers. The system sends a RDF/XML document, with a MIME 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.

3.4.3 Business Process Execution Language

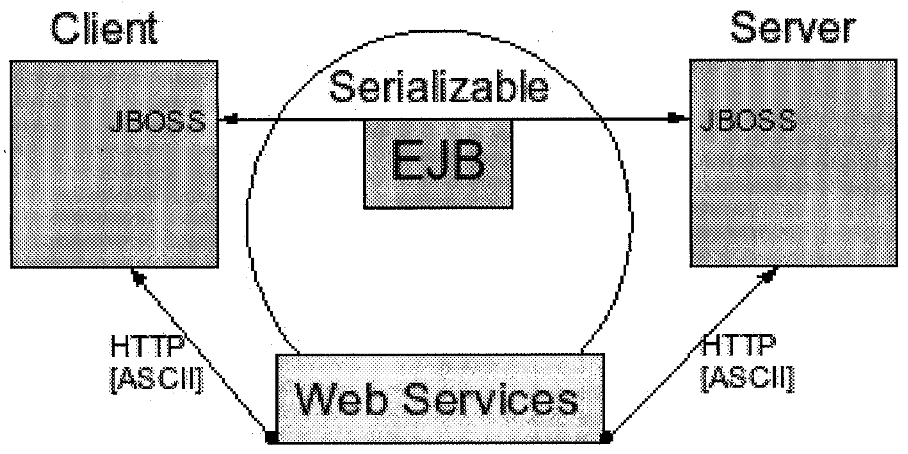
BPEL lets the user build a sophisticated query for the workflow of the Text Search and Dips. Using BPEL the user may specify the search order of datasources.

3.5 Web Services



BLACKBOOK is using Web Services to automate the data exchange mechanism with any capable enterprise application belonging to organizational partners. Other technologies, such as RMI or JMS, are capable in building the data exchange mechanism. However, Web Services gives three features other technologies do not provide:

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



The Client / Server could communicate without Web Services using the Serialized Enterprise Java Beans (EJB), but this would force an identical implementation for sending and receiving on all systems to be able to understand the serialized message. Web Services provides an implementation independent form of communication allowing systems to run on other servers, but since it is basically serializing the data a second time (into HTTP [ASCII]) it adds a considerable overhead.

4 RESTFUL INTERFACE - DESIGN

4.1 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 **Re**presentational **S**tate **T**ransfer.

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."

REST emphasizes

- 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 that are 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 the normal use on the web, even though it is popular in the RDF community.

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, depending on the further use. 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.

4.2 REST vs. SOAP

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

SOAP applies the Remote Procedure Call (RPC) approach. 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:

user()

location()

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 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 one and 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.

The requested content is rendered as a web feed for the user. A web feed or news feed is a data format used for providing users with frequently updated content. 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 note of the new content or downloads it. Aggregators can be scheduled to check for new content periodically.

5 RESTFUL INTERFACE – IMPLEMENTATION

The RESTful interface has been implemented for the following modules in the Blackbook project:

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

5.1 RESTEasy

BLACKBOOK uses Resteasy API for implementing RESTful Web Services.

Resteasy is a portable implementation of JAX-RS, JSR-311 specification that provides a Java API for Restful Web services over the HTTP protocol. RESTEasy is a JBoss project that provides various frameworks to help you 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 to make the user experience nicer in that environment. 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:

- Fully certified JAX-RS implementation
- Portable to any app-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)

5.2 Workspace-blackbook

1) Dependencies

Add the following dependencies in blackbook-war/pom.xml

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

<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>
        <version>1.5.2</version>
</dependency>
<dependency>
        <groupId>georss</groupId>
        <artifactId>georss</artifactId>
        <version>0.9.8</version>
</dependency>

<dependency>
        <groupId>rome</groupId>
        <artifactId>rome</artifactId>
        <version>0.9</version>
</dependency>
```

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) Add the following in web.xml

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.

```
<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 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.

Here is the list of all the methods and its corresponding URL's with the arguments.

Sr. No	URL	Method Name	HTTP Method
1	https://localhost:8443/blackbook/rest/algorithms/{feedType}	getAllAlgorithmClasses()	GET
2	https://localhost:8443/blackbook/rest/getFieldNames/datasources/{feedType}	getAllDataSources()	GET
3	https://localhost:8443/blackbook/rest/localdatasources/{feedType}	GetLocalDataSources()	GET
4	https://localhost:8443/blackbook/rest/fieldnames/{datasources}/{feedType}	getFieldNames()	GET
5	https://localhost:8443/blackbook/rest/keyword/{datasource}/{search}/{feedType}	getKeyword()	GET
6	https://localhost:8443/blackbook/rest/lucenekeyword/{datasource}/{keyword}/{feedType}	luceneKeyword()	GET
7	https://localhost:8443/blackbook/rest/keyword/{datasource}/{feedType}	postKeyWord()	POST
8	https://localhost:8443/blackbook/rest/lucenekeyword/{datasource}/{feedType}	postLuceneKeyword()	POST

5.3 Workspace-workflow

1) Dependencies

Add the following dependencies in workflow-war/pom.xml

```
<dependency>
        <groupId>resteasy</groupId>
        <artifactId>jaxrs</artifactId>
        <version>1.0.1.GA</version>
</dependency>
<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>
        <version>1.5.2</version>
</dependency>

<dependency>
        <groupId>georss</groupId>
        <artifactId>georss</artifactId>
        <version>0.9.8</version>
</dependency>

<dependency>
        <groupId>rome</groupId>
        <artifactId>rome</artifactId>
        <version>0.9</version>
</dependency>
```

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) Add the following in web.xml

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.

```
<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-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>
```

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.

Here is the list of all the methods and its corresponding URL's with the arguments.

Sr. No	**URL**	**Method Name**	**HTTP Method**
1	https://localhost:8443/workflow/rest/processdefinition/	CreateProcessDefinition()	PUT
2	https://localhost:8443/workflow/rest/processdefinition/{feedtype}/{processdefinitionid}	readProcessDefinition()	GET
3	https://localhost:8443/workflow/rest/processdefinition/{processdefinitionid}	DeleteProcessDefinition()	DELETE
4	https://localhost:8443/workflow/rest/processinstance/{processdefinitionid}	StartProcessDefinition()	PUT
5	https://localhost:8443/workflow/rest/processdefinition	updateProcessDefinition()	POST
6	https://localhost:8443/workflow/rest/processinstance/atom_1.0/{processInstanceId}	getProcessInstance()	GET

5.4 Workspace-workspace

1) Dependencies

Add the following dependencies in workspace-war/pom.xml.

```
<dependency>
        <groupId>resteasy</groupId>
        <artifactId>jaxrs</artifactId>
        <version>1.0.1.GA</version>
</dependency>
<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>
        <version>1.5.2</version>
</dependency>

<dependency>
        <groupId>georss</groupId>
        <artifactId>georss</artifactId>
        <version>0.9.8</version>
</dependency>

<dependency>
        <groupId>rome</groupId>
        <artifactId>rome</artifactId>
        <version>0.9</version>
</dependency>
```

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) Add the following in web.xml

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.

```
<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>
```

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.

Here is the list of all the methods and its corresponding URL's with the arguments.

Sr. No	URL	Method Name	HTTP Method
1	https://localhost:8443/workspace/rest/rootfolder/	getRootFolder()	GET
2	https://localhost:8443/workspace/rest/subfolder/{subfoldername}	CreateSubFolder()	PUT
3	https://localhost:8443/workspace/rest/subfolder/{parentfolderid}/{subfoldername}	CreateSubFolder()	PUT
4	https://localhost:8443/workspace/rest/item/{feedType}/{itemId}	getChildItems()	GET
5	https://localhost:8443/workspace/rest/processdefinition/{parentFolderId}/{ProcessDefinitionName}	CreateProcessDefinition()	PUT
6	https://localhost:8443/workspace/rest/processdefinition/{username}	GetProcessDefinitionID()	GET
7	https://localhost:8443/workspace/rest/item/{itemId}	RemoveItem()	DELETE

6 SUMMARY AND DIRECTIONS

In this report we have described the design and implementation of RESTful web services into Blackbook. As we have stated earlier, we are making several enhancements to Blackbook and these enhancements will be reported in future reports.

REST is widely used to implement web services in the industry recently. 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.

7 REFERENCES

http://rabasrv.jhuapl.edu/karma/

OWL: Representing information using the Web Ontology Language – By Lacy Lee

http://www.ics.uci.edu/~fielding/pubs/dissertation/

http://www.w3c.org