

School of Computer Science
Carleton University

COMP 4905
HONOUR PROJECT

Semantic Online-Store Aggregator (SOSA)

Luan Nguyen
100337193

Supervisor: Dr Tony White

Date: April, 2008

Abstract: Semantic Online Stores Aggregator (SOSA) is a Semantic Web portal in which a user can search for products they want to buy in a semantic way and know where the stores are that have the searched items available on the Google Map. The product provider can also register for the service via the web portal. The portal front-end has been developed with Java Server Faces (JSF), and using REST web service to communicate with the back-end which has been built using the Resource Description Framework (RDF) and Web Ontology Language (OWL). All of these technologies are combined in order to provide a new user experience for searching the descriptions of the Web-accessible databases.

Acknowledgment

I would like to give special thanks to my professor Tony White for all the help he provided during the development of this project.

I would also like to acknowledge all the open source projects that made possible this implementation specially SMART¹ (Alexander De Leon's Honors Project) and Protege². Sources from these projects have been integrated directly into my source code. Original package names were use in order to differentiate from the new code of this project. Those packages are `com.dumontierlab.smart.owl.model.cache.*`, and `org.protege.editor.owl.model.*`.

¹ <http://smart.googlecode.com>

² <http://protege.stanford.edu/>

Table of Contents

I	Introduction	7
	1. Motivation	7
	2. Project Description	8
II	Background	8
	1. Semantic Web	8
	1.1 Resource Description Framework (RDF)	9
	1.2 Web Ontology Language (OWL)	11
	2. RESTful Web Services	12
III	Packages Reused From Smart Project	13
	1. com.dumontierlab.smart.owl.model.cache.*	13
	2. org.protege.editor.owl.model.*	13
IV	Project Design	14
	1. High Level Architecture	15
	2. Features and Functionality	15
V	Project Implementation	16
	1. Components Used for Development	16
	2. Web Services Implementation	17
	3. Web Portal Implementation	24

4.	Semantic Query Model	30
5.	Deploying Application and Testing Scenarios	32
VI	Conclusion	43
1.	Technologies Learned	43
2.	Future work	43
3.	Summary	44

List of Figures

Figure 1: High level architecture	15
Figure 2: Main use cases	16
Figure 3: Web Services class diagram	23
Figure 4: Navigation Rules	27
Figure 5: SOSA's query answering sequence diagram	31
Figure 6: Store Registers Service	33
Figure 7: Query Suggestion	35
Figure 8: Filtering Items by Drag and Drop Property	36
Figure 9: Selected Item and Suggested Items	37
Figure 10: Add Item into Cart and View Map	38
Figure 11: Shopping Cart Check out	39

List Of Tables

Table 1. Catalog Endpoint HTTP API	20
Table 2. Orders Endpoint HTTP API	21

I. Introduction

1. Motivation

The Internet has changed the way people purchase goods and services. In Canada alone, the statistics have shown that “almost 7 million Canadians aged 18 and over placed an order on-line in 2005”. “The people who made an on-line purchase represented about 41% of all adults who used the Internet in 2005” [1].

Online shopping can be efficient and pleasant for some. However, it imposes a few limitations that are worthwhile analyzing. First, finding a desired product may require visiting several online stores. This may seem natural since we do the same when we shop at physical stores. Nevertheless, online store aggregators that combine multiple stores into a common centralized interface can really improve the efficiency of online shopping. Another important limitation of online shopping is the lack of semantics for searching. It is sometimes hard to express exactly what we looking for; for example, “I want to buy a blue long sleeved T-shirt, size small and made of cotton”. Today, no search engine will be able to answer the former query, instead they will return a long list of unrelated items that mach the words in the query. Semantic Web technologies are staring to show potential solutions to this problem by providing machine-understandable semantics to web content. This way machines can “understand” resources on the web and provide precise answer to questions by means of automated reasoning.

2. Project Description

This paper presents the design and implementation of SOSA (Semantic Online-Store Aggregator). SOSA is a web-based portal where providers can sell their products and where consumers can transparently browse, search, and buy products made available by multiple registered providers. SOSA collects information from multiple e-commerce web services and information from these different sources is aggregated under a common web portal. Each product description is annotated with a semantic vocabulary provided by common ontology. Semantic metadata and location information is used for aggregation and creating personalized user interface. SOSA provides a semantic query answering interface where users can post queries similar to the following: “*Shirt that hasColor value brown and isDesignForSeason value winter*”. In addition, purchased orders are divided into smaller orders, which will be dispatched to the corresponding suppliers.

II. Background

SOSA is based on two important web paradigms: the Semantic Web and the REST architecture. This section presents some background information for these technologies.

1. The Semantic Web

The Semantic Web is the next generation of the Web; it is an extension of the current Web where information is given well-defined semantics in a machine-readable manner [2]. This can be achieved by augmenting web content with data that facilitates its manipulation by computers [3]. To date, new languages have been developed to

represent this semantic metadata. Examples of such languages are XML (eXtensible Markup Language) and RDF (Resource Description Framework), which have become the foundation of the Semantic Web.

1.1 Resource Description Framework (RDF)

“RDF is a general-purpose language for representing information on the Web” [4]. RDF provides a data model for describing resources and the relationships between them. This model is constructed using statements in the form of a *subject-predicate-object* known as a *triple*. The subject of the triple is the resource being described, the predicate is the particular resource's property that is being described, and finally the object is the value of the resource's property [3]. RDF documents are usually rendered using an XML-based syntax. Let's look at the example statements to get a better understanding:

```
<?xml version="1.0"?>
<RDF>
  <Description about="http://www.ontoshop.org/item001">
    <size>small</size>
  </Description>
</RDF>
```

The combination of a Resource, a Property, and a Property value forms a Statement (known as the subject, predicate and object of a Statement).

Statement: "The size of <http://www.ontoshop.org/item001> is small".

- * The subject of the statement above is: <http://www.ontoshop.org/item001>
- * The predicate is: size
- * The object is: small

RDF triples represent relationships among resources. These relationships are described by the RDF property (predicate). "RDF however, does not provides the mechanism for describing these properties. This is the role of the RDF vocabulary description language, *RDF Schema*. RDF Schema defines classes and properties that are used to describe classes, properties and other resources" [5]. RDF Schema provides a vocabulary for modeling object hierarchies by the use of primitives such as subclass and sub-property relationships, and domain and range restrictions [6].

The term *ontology* has its origin in Philosophy where it is a branch of metaphysics concerned with the basic properties and relationships of existence [6]. In computer science, this term has been adopted by the artificial intelligence community. In this field, *an ontology* is defined as "an explicit and formal specification of a conceptualization"[6]. In general, ontologies are data models for representing concepts within a domain and the relationships between those concepts [7]. The significance of ontology is in its capacity to enable machine *reasoning* using rules of logic. Such reasoning, allows the machine to, at a certain level, "understand" part of the world that is being modeled within the ontology [3]. For example, using an ontology describing the world of universities and given a Person that is taking a course at Carleton University, the machine could deduce that this person must be a student by understanding the explicit and "inferring" the implicit relations between concepts being described. That

includes the rule that every “Person” taking a course at a University is considered a “Student”. Furthermore, it could infer that this is a student studying in Ottawa. The ontology framework provides a key ingredient for the foundations of the Semantic Web. Ontologies offer the Semantic Web the necessary constructs to express domain-based terminologies and their semantics such that computers can interpret and reason.

1.2 Web Ontology Language (OWL)

RDF expanded with RDF Schema can be considered as a basic ontology language, it provides primitives for building hierarchies of objects and their relationships. However, expressing that objects with different identifiers refer to the same concept is not possible. For example, zip code and postal code. OWL (Web Ontology Language) was designed to overcome the limitations of RDF and RDF Schema. OWL was developed by a group of largely European researchers and it is built on top of RDF and RDF Schema adding more vocabulary for describing properties and classes. These additions include: relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes, among others [8]. OWL provides three sublanguages: OWL Full, OWL DL, OWL Lite, which are designed for use by different users.

- OWL Full is designed for the users who want no restriction on use of OWL vocabulary.

- OWL DL is designed for the users who want the benefits defined semantics, formal properties well understood, ...but still maximize the expressiveness.
- OWL Lite supports those users primarily needing a classification hierarchy and simple constraints.

2. RESTful Web Services

Representation State Transfer (REST) is an architectural style for distributed systems such as the Web. Roy Fielding introduced the idea in his PhD dissertation in the year 2000. REST is founded on the following principles:

- Every resource is identified by an URI (Uniform Resource Identifier). These resources are manipulated by its representation.
- Multiples representations can be accepted or sent.
- Messages are self described and stateless.
- Interactions with these resource representations are supported by create, read, update and delete (CRUD) operations.
- Hypertext provides the engine for application state.

REST provides a client-server architecture in which the web services are viewed as resources and can be identified by their URLs. Web service clients that want to use these resources access a particular representation by transferring application content.

The HTTP operations GET, POST, PUT, and DELETE provides the necessary CRUD operations for REST web services. [9].

III. Packages Reused From Smart Project

1. com.dumontierlab.smart.owl.model.cache

This package contains the codes for maintaining the index of entities that exist on the active ontology. Entities can be classes, properties, or individual in the ontology. The LuceneOWLEntityCache.java class eliminates the need for traversing the entire ontology data structure when looking for particular entity. The OWLEntityKeyGenerator.java class is used to add the prefix to each short name of the entity in order to avoid ambiguity.

2. org.protege.editor.owl.model

This package provides the language for writing OWL class expression – the Manchester OWL Syntax. This syntax is based on OWL abstract syntax but is more readable and easier to understand. For example, the following class description is expressed with the Manchester OWL Syntax

“Shirt that hasColor value blue”

SOSA integrates those packages and adopts the Manchester OWL Syntax in order to build the Semantic Query Model (more about this later on).

IV. Project Design

1. High Level Architecture

SOSA is an online store aggregator. From the users perspective, it looks like any other online store, in which the products are offered on the site are sold by some third-party provider. However, SOSA gives users a new and simplified experience to find the items in a semantic way. SOSA finds items that are for sale by communicating with providers over the Web. Each provider publishes a set of RESTful web services that enable B2B communication with SOSA. Fig. 1 illustrates the RESTful interfaces that are exposed by providers. These include two standard endpoints *Catalog* and *Orders*. The catalog endpoint is used to access and to query the items that the particular providers want to make available. The main functionality of the orders endpoints is to allow SOSA to post orders to the provider when a customer completes a purchase. Every resource in the provider's domain is also accessible by its own RESTful interface. This allows SOSA to obtain detailed descriptions of resources such items for sale.

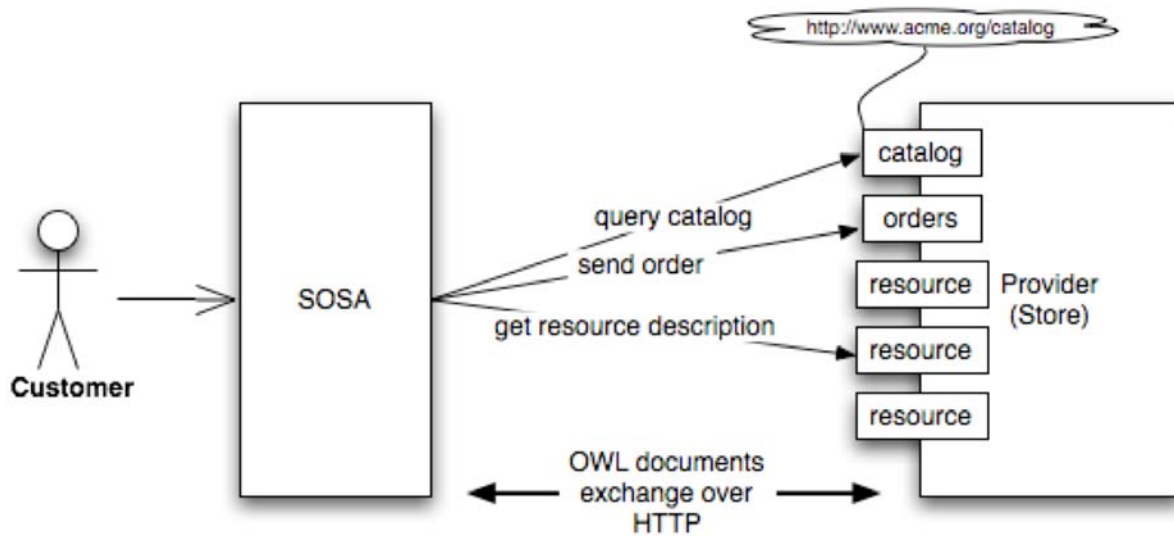


Figure 1: High-level architecture.

The interactions between SOSA and the providers involve interchanging representations of resources. These representations are expressed using the Web Ontology Language (OWL). By using OWL DL, we can make use of well-defined semantics to build object representations. This approach enables automated reasoning capabilities at both ends of the communication channel.

Semantic interoperability between SOSA and providers is possible because they both use a common semantic vocabulary published by SOSA. This vocabulary is contained in two main OWL ontologies <http://www.ontoshop.org/Clothing.owl> and <http://www.ontoshop.org/Store.owl>. These ontologies consist of a collection of OWLAxioms that provide an explicit context within which we can assert information about classes and properties. The following use cases which mainly used in the application are described in the figure below.

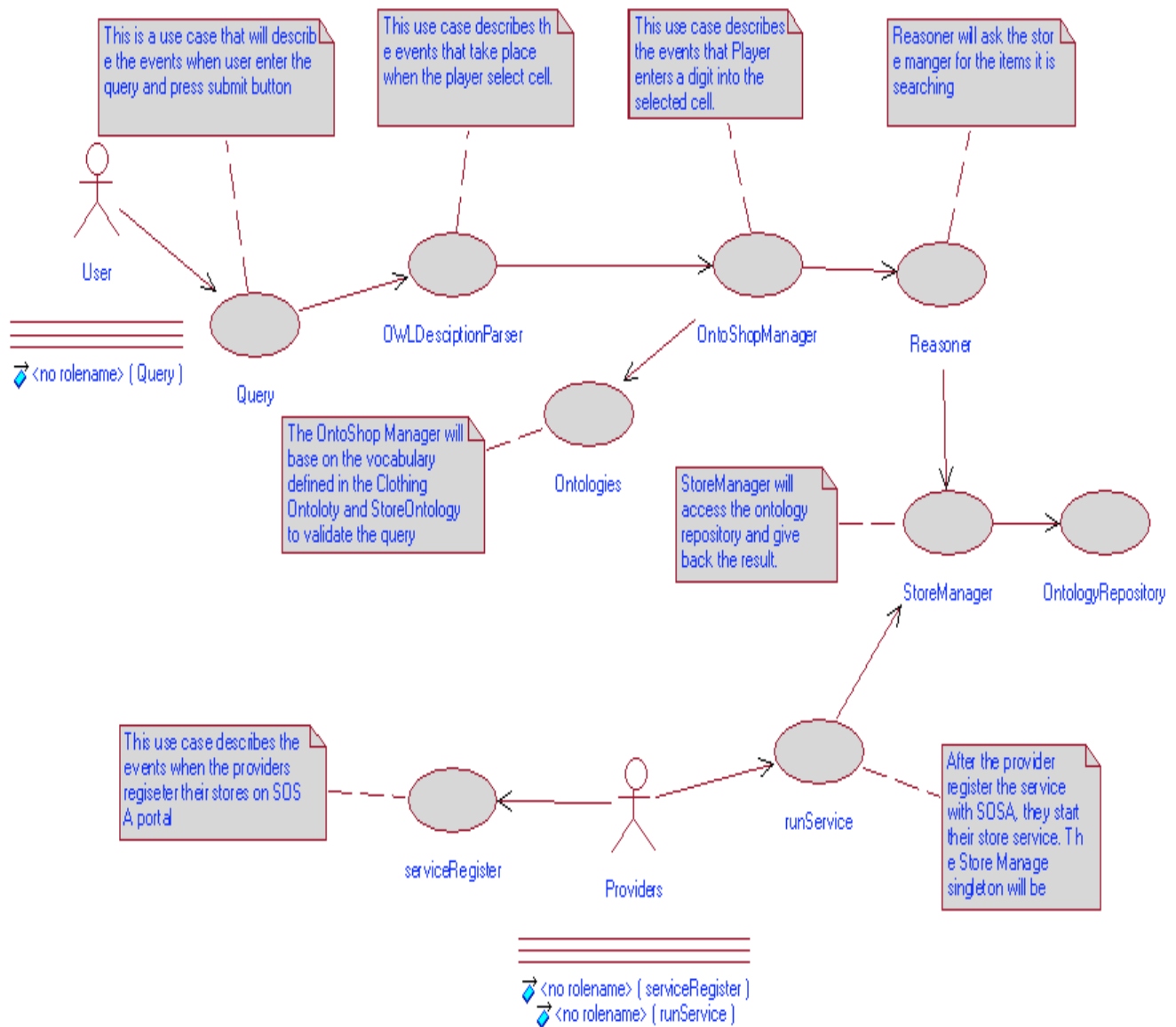


Figure 2: Main use cases

2. Features and Functionality

- SOSA provides a common web-portal such that the providers can register and start selling their products in a few steps.

- SOSA provides friendly interface so that the customer can easily find the items they want in a semantic way.
- Ajax suggestion vocabulary is implemented to help the user complete their query for item searching.
- Recommended items will be shown when an item is selected.
- Integrating Google Map, user can see the store locations which selling the items.
- Filtering the items, by drag and drop of an item's properties.
- Even user may buy the items from different providers they only need to pay as one order. Those purchased items will be dispatched to the corresponding suppliers.

V. Project Implementation

1. Components Used For Development

All the components and tools are used to develop the project are open source, well supported and documented. They are also available for downloading from the Web.

- Development Tools: Eclipse Platform version 3.3.1.1. Can be downloaded from <http://www.eclipse.org/>
- Application Server: Apache Tomcat version 6.0.14. Can be downloaded from <http://www.apache.org/>

- Ontology Editor: Protégé platform version 4.0 alpha. Can be downloaded from <http://protege.stanford.edu/>

2. Web Services Implementation

Generic framework is implemented such that allows providers to automatically instantiate and publish the necessary RESTful web services needed for communicating with SOSA. The input is an OWL ontology that describes the on-line store (e.g. store name, address, endpoints URIs, etc) and items that the store sells. This OWL ontology must use and extend the classes, properties, and instances of the vocabulary ontologies that SOSA provides (i.e. Clothing.owl and Store.owl). To avoid confusion I will refer to this input ontology as *O*.

The framework extracts all resources from *O*. These resources include the store, items, and addresses. For each of these resources it creates a RESTful web service that allows one to resolve the URI of the resource to a subset of *O* that contains only the description of the requested resource. The snippet from *O* that described the store representation in OWL looks like the following:

```
<Store:Store rdf:about="http://localhost:8084/alexLittleStore">
  <rdfs:label rdf:datatype="&xsd:string">
    Alex's Little Store
  </rdfs:label>
  <Store:hasAddress
    rdf:resource="http://localhost:8084/alexLittleStoreAddress" />
  <Store:ordersEndpoint rdf:datatype="&xsd:anyURI"
```

```

    >http://localhost:8084/orders
</Store:ordersEndpoint>

<Store:catalogEndpoint rdf:datatype="&xsd:anyURI"

    >http://localhost:8084/catalog

</Store:catalogEndpoint>

</Store:Store>

```

From the descriptions the framework extracts the asserted *catalogEndpoint* and *ordersEndpoint* URIs, and creates two special RESTful web services for these. The HTTP APIs for these endpoints described in tables 1 and 2.

HTTP Method	Request Parameters	Response
GET	none	All instances are classifiable under the class <i>CatalogItem</i> . These are basically all items for sale.
	<i>query</i> , <i>queryLang</i>	All instances in the catalog that satisfy the query provided by the <i>query</i> parameter. The <i>queryLang</i> is optional and specifies the language used to compose the query. Currently we only support the Manchester OWL Syntax language.

HTTP Method	Request Parameters	Response
POST		HTTP 403 response (forbidden)
PUT		HTTP 403 response (forbidden)
DELETE		HTTP 403 response (forbidden)

Table 1. Catalog Endpoint HTTP API

HTTP Method	Request Parameters	Response
GET	none	(Not implemented in current version) All instances are classifiable under the class Order. These are basically all orders that have been submitted.

HTTP Method	Request Parameters	Response
	<i>query , queryLang</i>	All orders that satisfy the query. For example one may request all orders for a specific customer.
POST	An OWL document containing the representation of an Order using vocabulary from Store.owl	HTTP 202 Response (If the order is accepted)
PUT	HTTP 403 response (forbidden)	
DELETE	HTTP 403 response (forbidden)	

Table 2. Orders Endpoint HTTP API

The *org.ontoshop.store.** package contains the implementation classes for this the RESTful web services framework. Fig. 2 shows the main classes and interfaces used for building RESTful web services. We can group classes into tree layers the *communication layer*, the *business logic layer* and the *model layer*. In the

communication layer, we found classes that extend the JAX-WS framework for creating instances of web services. The JAX-WS abstracts the bindings to the HTTP protocol and let the implementation of the Provider interface to handle the requests. The business logic layer contains the StoreService interface and implementation this service object provides the business functions that are expected from a Store. This service object uses the StoreManager object to access the information coming for the OWL ontology. The StoreManager uses the OWL API, a set of programming interfaces, which allows access and manipulation of OWL ontologies in Java applications. The primary goal of this API is to allow application programmers to work at a higher level of abstraction, without concern over some of the problematic issues related to serialization and parsing of OWL data structures. In addition, this API enables interoperability among the different applications that make use of the API.

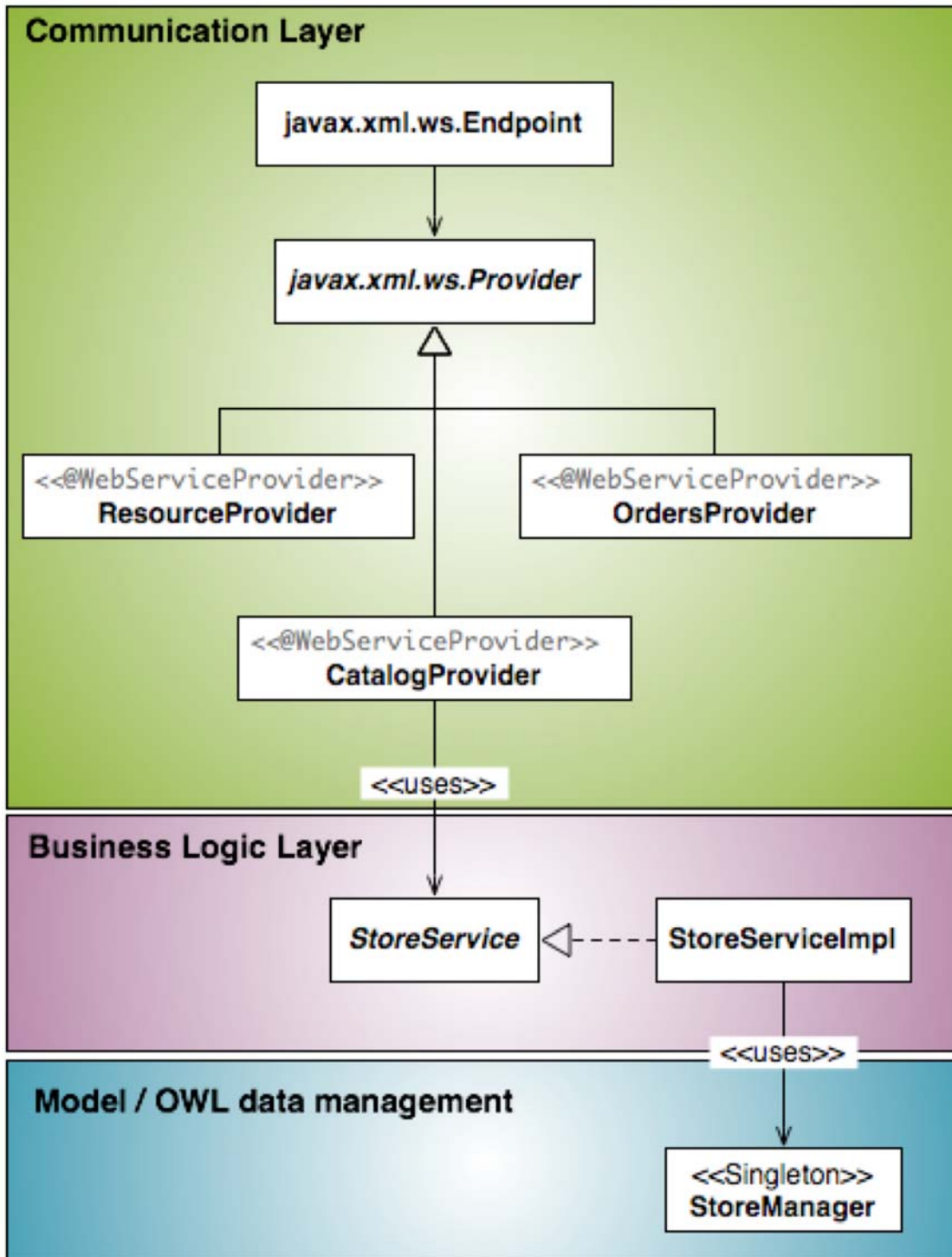


Figure 3: Web Services class diagram

3. Web Portal Implementation

The SOSA application consists of a core component and a web-based user interface component. At the boundary of core component is the *OntoshopService* interface and its corresponding implementation class. This service provides the set of functionalities needed to support the interaction with the providers' web service. This class implements most of the business logic of the application such that it can be abstracted from the user interface code. Following the same architecture as with the web services, there is a singleton *OntoshopManager* object that abstracts most of the OWL processing logic. For example, this manager provides the functionality to bind the OWL representations to the corresponding java domain objects.

The SOSA user interface was implemented using the JSF (Java Server Faces) framework. JSF provides a component-based framework for creating rich server-side user interfaces. This technology is based on the Model-View-Controller design pattern for separating logic from presentation. This architecture enables reusable server-side code that follows object-oriented practices. An important feature of JSF is that it provides the mechanism for binding the user events to server-side event handlers, making the user interface design similar to that on a desktop application.

The view of the application is represented by home page, item listing page, checkout page, user register page, selected item page and register service page. These pages are written in JSP using JSF built in components. In combination with these

components are the beans that are used to bind value of the component to the property of the bean or to refer method of the bean from component tag. For example, searching input field is bound to the query attribute in the Query class using the syntax as below

```
<h:inputText id="query" value="#{Query.query}" />
```

In order to use the bean we need to declare it in the faces-config.xml

By specify the bean name, the bean class and the scope for the bean.

```
<managed-bean>
    <managed-bean-name>Query</managed-bean-name>
    <managed-bean-class>com.sosa.beans.Query</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

In the faces-config.xml we also need to specify the navigation rules from one page to another. For example, we want the search result items will be display in the listing item page when we enter the query into the searching input field in the home page and press on the search button. We need to specify the navigation rules as the following:

```
<navigation-rule>
    <navigation-case>
        <from-view-id>/home.jsp</from-view-id>
    </navigation-case>
    <from-outcome>stay</from-outcome>
```

```
        <to-view-id>/home.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>listItems</from-outcome>
        <to-view-id>/listingItems.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

The outcome “stay” and “listItems” are the string returned by the method name search executed by the action on the search button.

```
<h:commandButton type="submit" action="#{Query.search}"
value="#{bundle.search_button_label}"></h:commandButton>
```

The figure below shows the navigation rules of a number of pages in the application.

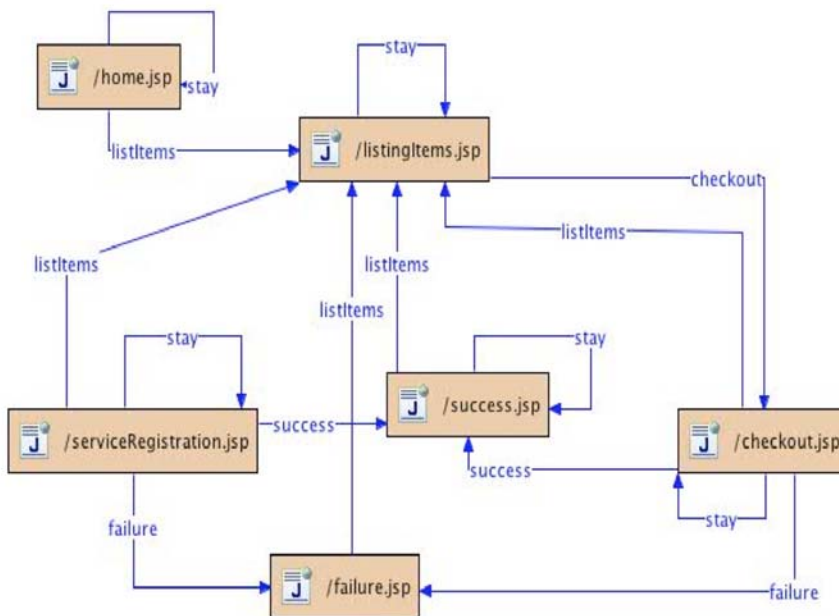


Figure 4: Navigation Rules

In addition, Rich Faces, an open source framework, is added on top of the JSF in order to enable the Ajax capability for vocabulary suggestion and filtering. Rich Faces provides two component libraries -- Core Ajax, and UI -- with components that require no JavaScript coding. For example, the code below are used for the suggestion query box

```

<h:inputTextarea id="expressionBox" value="#{Query.query}" styleClass="solidBorder"
style="width: 442px">
    <a4j:support id="expression_ajax" event="onkeyup" />
</h:inputTextarea>
<rich:suggestionbox for="expressionBox" suggestionAction="#{Query.findSuggestions}"

```

```

var="suggestion" width="200" height="150" border="1" tokens=" ({" shadowOpacity="4"
shadowDepth="4" fetchValue="#{suggestion.suggestedSentence}" ajaxSingle="true">

    <h:column>

        <h:outputText value="#{suggestion.suggestedWord}"/>

    </h:column>

</rich:suggestionbox>

```

The web portal has been integrated with Google Maps. When the user finds the items that they are interested in, they can check the location of the stores that has those items. If there is more than one store, the map will display multiple makers corresponding to the location of the stores. In order to implement this feature, we need to register for the Google Maps API key for the page that will be displayed the map. The JavaScript below has been added:

```

<script
src="http://maps.google.com/maps?file=api&v=2&key=ABQIAAAAOOv1\_3ecZ9OvyFbTyMit3xS32-EYe2hdSt5FnzEztlHpbTXYKBSzAQNxWBtfRnUZ3HrqawwYdN9SNQ"
type="text/javascript"></script>

<script type="text/javascript">

function load(addresses) {

    if (GBrowserIsCompatible()) {

        var map = new GMap2(document.getElementById("map"));

        var geocoder = new GClientGeocoder();

        map.addControl(new GSmallMapControl());

```

```

var arr = addresses.split(",");
for (var i=0;i<arr.length;i++)
{
    var postalCode = arr[i];
    geocoder.getLatLng(arr[i],function(point) {
        alert(arr[i] + " not found");
    } else {
        var marker = new GMarker(point);
        marker.value = i;
        map.setCenter(point, 9);

        map.addOverlay(marker);

        GEvent.addListener(marker, "click", function() {
            var myHtml = "<b>" + postalCode + "<br/>";
            map.openInfoWindowHtml(point, myHtml);
        })
    }
});
}
}
</script>

```

The load function above is the main function which gets the addresses and finds the locations of those addresses and display the corresponding points onto the map. The expected parameter passing to the load function is a string with more than many

addresses separated by the comma. The first important class we need to use in the Google Map API is the GMap2. The GMap2 class constructor creates a new map inside the given HTML container normally is a DIV element. The next important class used in the JavaScript is the GClientGeocoder. The instance geocode will send a request to Google servers to get the longitude and latitude of the specified address. If the address was successfully located, the user-specified callback function is invoked with a GLatLng point. Otherwise, the callback function is given a null point. Lastly, the GMarker is used to display the marker on the map at the corresponding points. And the markers can have an event listener, so every time you click on the marker it can show a message.

4. Semantic Query Model

Base on the Manchester OWL syntax, SOSA adopts the Manchester OWL Syntax query functionality of Protégé 4.0.1 so that users can describe what they are looking for, as a class expression. The OWLDescriptionParser will parse the query and then infer the individuals of the ontology, which can be classified under the user's descriptions. These individual instances are used with the Reasoner for querying as it shown in the figure below.

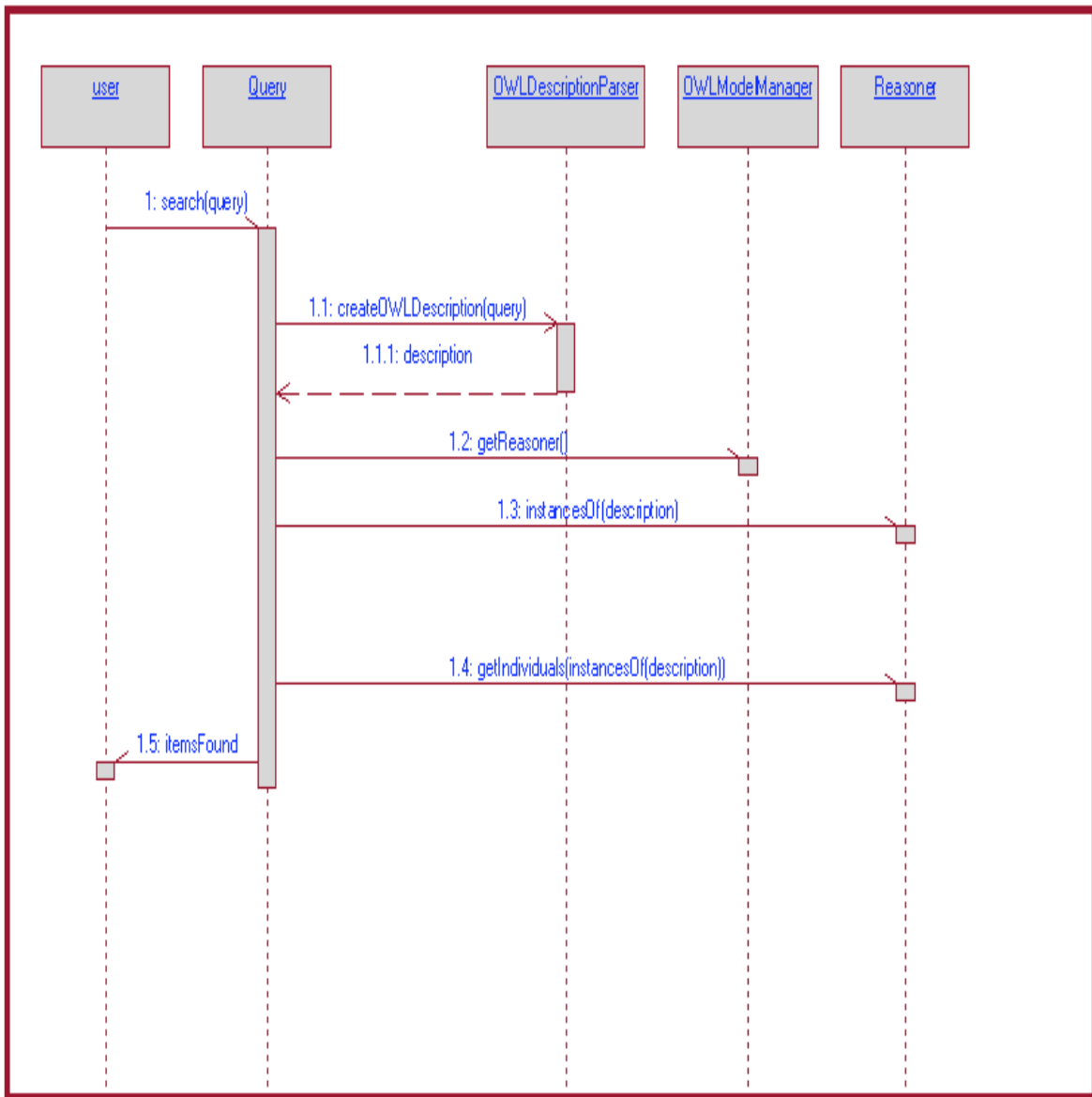


Figure 5: SOSA's query answering sequence diagram.

To demonstrate how the diagram above is working, let consider the class expression as the following:

T-Shirt that hasColor value blue

The definition of T-Shirt matches exactly the description of the query by finding the individuals that is instance of T-Shirt class and has properties hasColor. The properties hasColor blue which also matched to the instance of class Color. If there is no instance of class T-Shirt or Color matched, there will be no individuals returned.

The query model has been build with a limited time, so there are some limitations such as

- The query must be enter in exact syntax (e.g. case sensitive, must have keyword “value” before the value of the properties)

“Shirt that hasColor value blue”

- The user can search for the item with exact price, but not from max or min price.

“Shirt that price value 19.9”, but not “Shirt that price value max 19.9”

5. Deploying Application and Testing Scenarios

The application can be run on Apache Tomcat server version 6.0.14. You just simply deploy the SOSA_Ajax. War onto Tomcat, and everything will be ready for running. Assume that you are using port 8080 for your local host.

4.1 Service Register

Before you can search for the items, the SOSA portal needs have the stores register their service. In order to register the store service, open the browser and go to http://localhost:8080/SOSA_Ajax/serviceRegistration.faces

or you can go to SOSA home page http://localhost:8080/SOSA_Ajax/home.faces and click on Service Register . On the store URL input box enter <http://localhost:8084/alexLittleStore> and hit submit. You will get the information for the store registered.

The screenshot shows a web browser window titled "SOSA Service register" with the URL http://localhost:8080/SOSA_Ajax/serviceRegistration.faces. The page features the SOSA logo (Semantic Online Stores Aggregator) and a navigation menu with "Home" and "Service Register" options. The main content area is titled "Service Register" and contains a form for registering a new store. The form includes a "Store URL" input field with the value "http://localhost:8084/catalog" and a "Submit" button. Below this, a "Confirm" section prompts the user to confirm the information below. The confirmation form consists of two sets of input fields for store details:

Field	Value
Name	Alex's Little Store
Catalog URL	http://localhost:8084/catalog
Order URL	http://localhost:8084/orders
Address	43 Haxvy Pvt.
City	Ottawa
Province	Ontario
Postal Code	K1T 3B7
Country	Canada
Address	1125 Colonel By Drive
City	Ottawa
Province	Ontario
Postal Code	K1S 5B6
Country	Canada

Figure 6: Store Registers Service

5.2 Searching Items with Ajax suggestion

Let say you want to search for a T-Shirt that has color black. In the search box you type: "T-Shirt that hasColor value black". When you are typing the suggestion box will appear.

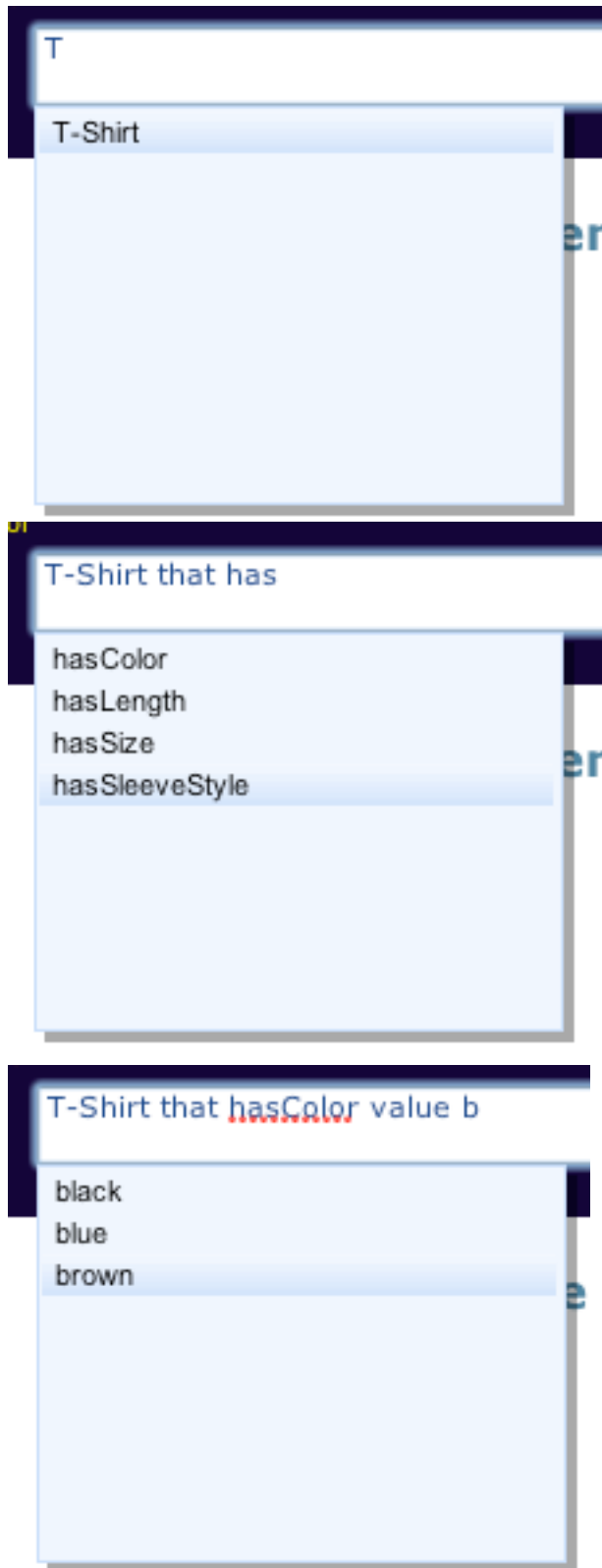


Figure 7: Query Suggestion

4.3 Filtering Items

For example, when you have searched for T-Shirt and now you want to filter items based on their properties, instead of entering the query in the search box again, you can simply drop and drag the properties into the Drop Zone and select the value of the properties you want to filter.

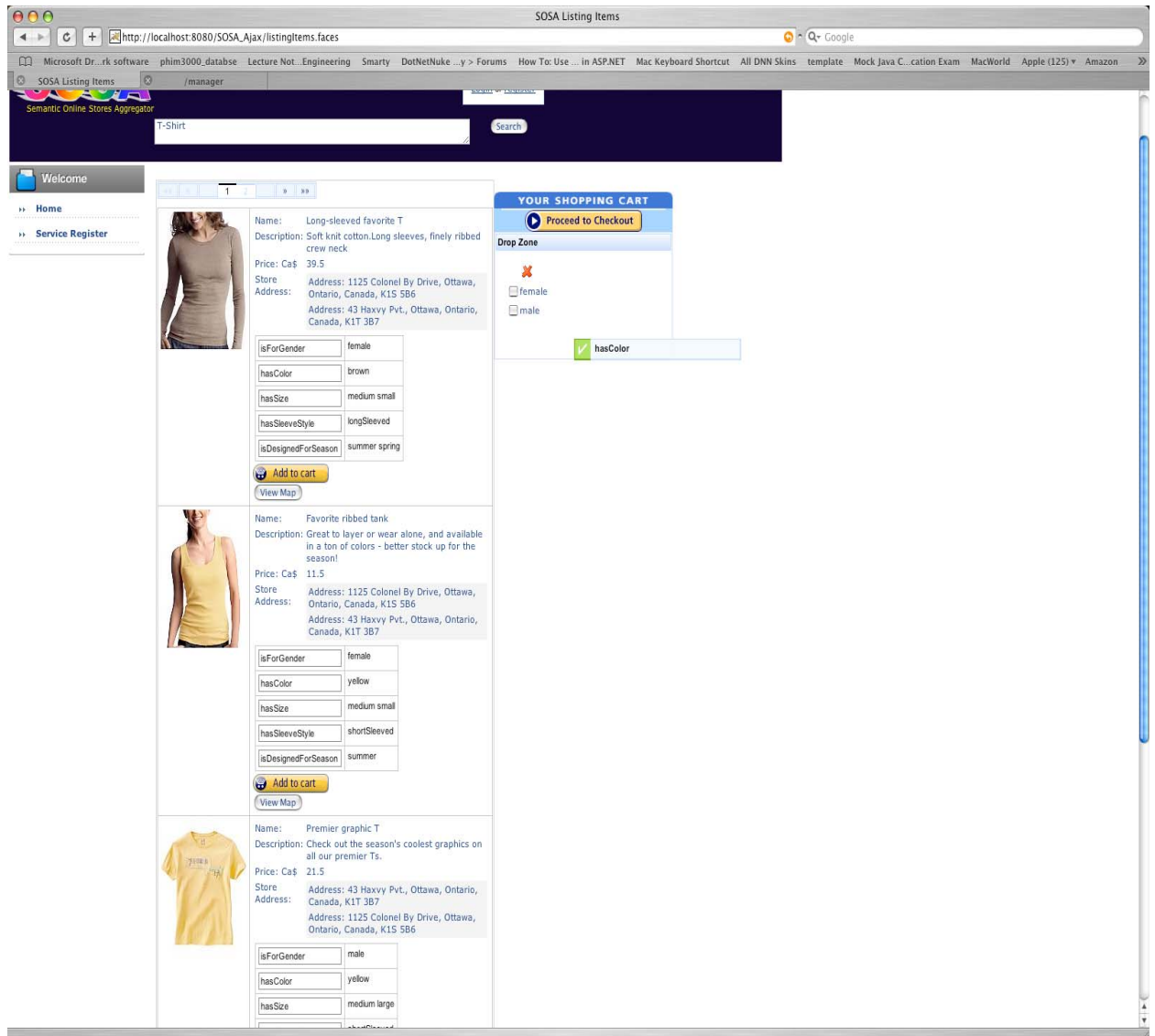



Figure 8: Filtering Items by Drag and Drop Properties

4.4 Select Item and Item Recommended

Depending on the properties of each item, the recommendation will be dynamically find the items, which are matched those properties and not the same type. For example, if you select the T-Shirt that has color green and white, has sleeve style long and is designed for spring season. The suggested items will not be T-Shirt, and has color white or green or has sleeve style long or designed for spring season.

T-Shirt Search



Name: Classic striped waffle V-neck T
Description: Who says a wardrobe staple can't be sexy?
Price: Ca\$ 22.5
Store Address: Address: 43 Haxvy Pvt., Ottawa, Ontario, Canada, K1T 3B7
Address: 1125 Colonel By Drive, Ottawa, Ontario, Canada, K1S 5B6


isForGender	female
hasColor	white green
isStyle	casual informal
hasSize	medium extra-small small
hasSleeveStyle	longSleeved
isDesignedForSeason	spring

[Add to cart](#)
[View Map](#)


YOUR SHOPPING CART

[Proceed to Checkout](#)


You might also like




Straight fit heather clean pants
\$49.5



Calle striped pique polo
\$19.5



Gathered-waist pullover top
\$0.0




Striped surplus khaki
\$44.5

Figure 9: Selected Item and Suggested Items

4.5 Add Item into Cart and View Store on Google Map

«
1
2
»



Name: Reversible T


Description: A whole new reversible. Front to back. Not inside out. Scoopneck or crewneck.

Price: Ca\$ 22.5

Store Address: 43 Haxvy Pvt., Ottawa, Ontario, Canada, K1T 3B7
1125 Colonel By Drive, Ottawa, Ontario, Canada, K1S 5B6

isForGender	female
hasColor	black
hasSize	medium small
hasSleeveStyle	shortSleeved
isDesignedForSeason	summer

[Add to cart](#)
[View Map](#)



Name: Premier graphic T

Description: Check out the season's coolest graphics on all our premier Ts.

Price: Ca\$ 21.5

Store Address: 1125 Colonel By Drive, Ottawa, Ontario, Canada, K1S 5B6
43 Haxvy Pvt., Ottawa, Ontario, Canada, K1T 3B7

isForGender	male
hasColor	yellow
hasSize	medium large
hasSleeveStyle	shortSleeved
isDesignedForSeason	summer

[Add to cart](#)
[View Map](#)

YOUR SHOPPING CART

Classic striped waffle V-neck T
22.5\$

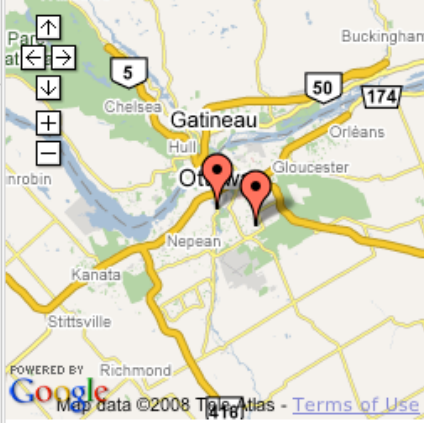
[Delete](#)

Reversible T
22.5\$

[Delete](#)

[Proceed to Checkout](#)

Drop Zone



POWERED BY Google
Map data ©2008 TMap, Atlas - [Terms of Use](#)

Figure 10: Add Item into Cart and View Map

4.5 Checking out and dispatching orders

Check Out

User Name*	<input type="text" value="luan nguyen"/>
Street*	<input type="text" value="133 Rocky Hill Drive"/>
City*	<input type="text" value="Napan"/>
Province*	<input type="text" value="Ontario"/>
Postal Code*	<input type="text" value="K2G7B2"/>
Country*	<input type="text" value="Canada"/>
Phone Number*	<input type="text" value="613-123-4567"/>
Credit Card Number*	<input type="text" value="123457890"/>
Expired Date*	<input type="text" value="20/08/2008"/>
	<input type="button" value="Submit"/>

Figure 11: Shopping Cart Check out

When you finish enter the information and press on submit, on the other end the store will receive the information of your orders in xml form which tells the provider that the items has been ordered and user's information (name, shipping address, credit card number)

**** Received Order: ****

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xml:base="http://www.ontoshop.org/order123457890.owl"
xmlns="http://www.ontoshop.org/order123457890.owl#"
xmlns:Store="http://www.ontoshop.org/Store.owl#"
xmlns:localhost="http://localhost:8084/"
xmlns:owl11="http://www.w3.org/2006/12/owl11#"
xmlns:owl11xml="http://www.w3.org/2006/12/owl11-xml#"
```

```

xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:order123457890="http://www.ontoshop.org/order123457890.owl#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
  <owl:Ontology rdf:about="">
    <owl:imports
rdf:resource="http://www.ontoshop.org/Store.owl"/>
    </owl:Ontology>
    <owl:ObjectProperty
rdf:about="http://www.ontoshop.org/Store.owl#hasAddress"/>
    <owl:ObjectProperty
rdf:about="http://www.ontoshop.org/Store.owl#hasItem"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#buyersName"/>

    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#city"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#country"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#creditcard"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#phone"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#postalCode"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#province"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#street"/>
    <owl:Class
rdf:about="http://www.ontoshop.org/Store.owl#Address"/>
    <owl:Class
rdf:about="http://www.ontoshop.org/Store.owl#Order"/>
    <rdf:Description rdf:about="http://localhost:8084/item022"/>
    <Store:Order rdf:about="#order123457890">
      <Store:hasItem
rdf:resource="http://localhost:8084/item022"/>
      <Store:phone
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">613-123-
4567</Store:phone>
      <Store:buyersName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">luan
nguyen</Store:buyersName>
      <Store:creditcard

```



```

rdf:datatype="http://www.w3.org/2001/XMLSchema#string">123457890</Store:
creditcard>
    <Store:hasAddress
rdf:resource="#order123457890Address"/>
    </Store:Order>
    <Store:Address rdf:about="#order123457890Address">
    <Store:city
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Nepean</Store:c
ity>
    <Store:postalCode
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">K2G7B2</Store:p
ostalCode>
    <Store:country
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Canada</Store:c
ountry>
    <Store:province
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ontario</Store:
province>
    <Store:street
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">133 Rocky Hill
Drive</Store:street>
    </Store:Address>
</rdf:RDF>
**** Received Order: ****
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xml:base="http://www.ontoshop.org/order123457890.owl"
xmlns="http://www.ontoshop.org/order123457890.owl#"
xmlns:Store="http://www.ontoshop.org/Store.owl#"
xmlns:localhost="http://localhost:8084/"
xmlns:owl11="http://www.w3.org/2006/12/owl11#"
xmlns:owl11xml="http://www.w3.org/2006/12/owl11-xml#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:order123457890="http://www.ontoshop.org/order123457890.owl#"
xmlns:owl="http://www.w3.org/2002/07/owl#">
    <owl:Ontology rdf:about="">
    <owl:imports
rdf:resource="http://www.ontoshop.org/Store.owl"/>
    </owl:Ontology>
    <owl:ObjectProperty
rdf:about="http://www.ontoshop.org/Store.owl#hasAddress"/>

    <owl:ObjectProperty

```

```

rdf:about="http://www.ontoshop.org/Store.owl#hasItem"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#buyersName"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#city"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#country"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#creditcard"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#phone"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#postalCode"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#province"/>
    <owl:DatatypeProperty
rdf:about="http://www.ontoshop.org/Store.owl#street"/>
    <owl:Class
rdf:about="http://www.ontoshop.org/Store.owl#Address"/>
    <owl:Class
rdf:about="http://www.ontoshop.org/Store.owl#Order"/>
    <rdf:Description rdf:about="http://localhost:8084/item003"/>
    <rdf:Description rdf:about="http://localhost:8084/item022"/>
    <Store:Order rdf:about="#order123457890">
        <Store:hasItem
rdf:resource="http://localhost:8084/item022"/>
        <Store:phone
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">613-123-
4567</Store:phone>
        <Store:buyersName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">luan
nguyen</Store:buyersName>
        <Store:hasItem
rdf:resource="http://localhost:8084/item003"/>
        <Store:creditcard
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">123457890</Stor
e:creditcard>
        <Store:hasAddress
rdf:resource="#order123457890Address"/>
        </Store:Order>
        <Store:Address rdf:about="#order123457890Address">
            <Store:city
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Nepean</Store:c
ity>

```

```
        <Store:postalCode
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">K2G7B2</Store:p
ostalCode>
        <Store:country
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Canada</Store:c
ountry>
        <Store:province
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Ontario</Store:
province>
        <Store:street
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">133 Rocky Hill
Drive</Store:street>
    </Store:Address>
</rdf:RDF>
```

VI. Conclusions

1. Technologies Learned

The implementation provided with this project can only be considered a proof of concept because it ignores many real world issues such as trust management, concurrency and scalability. However, working on this project has given me the opportunity to learn about RESTful web services and how they can be use for integrating distributed system on the Web. I also learn about Semantic Web and specially OWL. In addition, I have also learned using JSF in which has Ajax supporting components to build the user interface more dynamic and user interacting.

2. Future work

- Building complex suggestion model base on rules and axiom of the ontology in a way that the providers can sell more items and easy to change those

suggestion rules such as “White Shirt goes with Black Pant”, “Color Red go with Blue or Yellow.

- Considering about the concurrency and scalability so that the application can be used in the real environment.

- The application needs to be more flexible to integrate with other applications such as reuse this application not only for searching clothing but may be for others.

- Create an interactive user interface in a way that allows the user to create their own rules as to how they want to see item information every time they visit the web site.

3. Summary

I presented in this report a summary of the design and implementation of the SOSA project. SOSA is an implementation of an online-store aggregator, is built using Semantic Web technologies and RESTful web service. SOSA was designed to improve the experience of shopping online.

During the implementation of this project I explored an interesting architecture. I combined the RESTful architectural pattern and OWL to enable interoperability along distributed peers on the Web. The main advantage of this approach is that A-box ontologies can be spited into smaller pieces that can be accessed individually by RESTful interfaces. This can be beneficial when working with large knowledge bases

(KB). By interchanging representations in OWL each peer can send new assertions to another peer. The assertions can be used in combination with the KB of the receiving peer for automated reasoning, which results can be sent back to the sender when appropriated. For example one user can specify the class *Shirt* as a “*Shirt that hasColor value brown and hasSize value medium*”. This query can be sent to different stores along with the description of *Shirt*. Each store can import the description of *Shirt* into its KB and try to classify its inventory items under this class.

References

[1] Statistics Canada, E-commerce: Shopping on the Internet, The Daily, November 1, 2006, Available at: <http://www.statcan.ca/Daily/English/061101/d061101a.htm>

[2] Tim Berners-Lee and James Hendler and Ora Lassila , The Semantic Web, Scientific American, 2001, Available at: <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>

[3] Bo Leuf, The Semantic Web : Crafting Infrastructure for Agency, John Wiley & Sons, January 2006.

[4] W3C Recommendation , RDF/XML Syntax Specification (Revised), Available at: <http://www.w3.org/TR/rdf-syntax-grammar/>

[5] W3C Recommendation, RDF Vocabulary Description Language 1.0: RDF Schema, Available at: <http://www.w3.org/TR/rdf-schema>

[6] Grigoris Antoniou and Frank van Harmelen, A Semantic Web Primer, The MIT Press, 2004.

[7] Wikipedia, Ontology, computer science, Available at: [http://en.wikipedia.org/wiki/Ontology_\(computer_science\)](http://en.wikipedia.org/wiki/Ontology_(computer_science))

[8] W3C Recommendation, OWL Web Ontology Language, Available at: <http://www.w3.org/TR/owl-features/>

[9] Sameer Tyagi, RESTful Web Services, August 2006, Available at:
<http://java.sun.com/developer/technicalArticles/WebServices/restful/>

[10] W3C (Feb 10th, 2004) OWL Web Ontology Language Guide.

Retrieve from <http://www.w3.org/TR/owl-guide/> (Dec 15th, 2007)

[11] W3C (Nov 22nd, 2004) OWL-S: Semantic Markup for Web Service. Retrieve from
<http://www.w3.org/Submission/OWL-S/> (Dec, 15th, 2007)

[12] Google Google Map API Concepts.

Retrieve from <http://code.google.com/apis/maps/documentation/> (Dec, 13th, 2007)

[13] John j. Yates (Feb 1, 2006) JAX-RPC Evolves into Simpler, More Powerful JAX-WS
2.0.

Retrieve from <http://www.devx.com/Java/Article/30459> (Dec, 15th, 2007)

[14] Stanford Center for Biomedical Informatics Research Protégé

Retrieve from <http://protege.stanford.edu/> (Dec 9th, 2007)