2008

Personalized Shopping Agent for Semantic Web

Ramani Arunachalam
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the [Computer Sciences Commons](#)

Recommended Citation

Arunachalam, Ramani, "Personalized Shopping Agent for Semantic Web" (2008). *Master's Projects*. 78.
DOI: <https://doi.org/10.31979/etd.a3j7-qk2g>
https://scholarworks.sjsu.edu/etd_projects/78

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Personalized shopping agent for Semantic web

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Ramani Arunachalam

Spring 2008

Copyright © 2008

Ramani Arunachalam

All Rights Reserved

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Robert Chun

Dr. Chris Tseng

Senthilkumar Narayanaswamy, Brocade Communications Inc

ABSTRACT

Current online shopping experience has limitations due to lack of uniform, inter-linked machine readable data across various shopping sites. This project attempts to improve the experience by employing semantic web concepts such as OWL and SPARQL. A new OWL ontology is defined for product data and real-life data from shopping sites such as Amazon, Yahoo shopping are imported and inter-linked using this ontology. A web-based personalized shopping agent is implemented to query this inter-linked database by specifying the preferences for various product attributes. Screenshots of the resulting system and its usage have been provided.

ACKNOWLEDGEMENTS

I thank my advisor, Dr. Robert Chun, whose guidance and support has been invaluable. He always encouraged me to come up with something innovative and improve it further. I am also thankful for Dr. Chris Tseng and Senthil Kumar's participation as committee members. They provided enlightening insight that has been helpful throughout the project.

Thank you.

Table of Contents

1.0 Introduction	1
1.1 Limitations of online shopping	1
1.2 Related literature	3
2.0 Semantic Web standards	9
2.1 Resource Description Framework (RDF)	9
2.2 Web Ontology Language (OWL)	9
2.3 SPARQL query language	11
2.4 Jena framework	11
3.0 Design	12
3.1 Ontology design	12
3.2 Data collection	15
3.2.1 XML to OWL Translator	15
3.2.2 Algorithm	18
3.2.3 RSS to OWL Translator	19
3.3 User interface	20
3.3.1 Login interface	20
3.3.2 Query/filter interface	21
3.3.3 Query results interface	22
4.0 Implementation details	23
5.0 Usage Example	25
5.1 Comparison with other shopping sites	29
6.0 Conclusion	31
7.0 Future Work	31
Appendices	
Appendix A. Source Code	33
References	67

List of Tables

Table 1: Properties of Product class.....	14
Table 2: Properties of Offer class	14
Table 3: Properties of Review class.....	15
Table 4: Properties of Seller class.....	15
Table 5: Amazon XML to OWL.....	16
Table 6: Node to class translation.....	17
Table 7: Node to property translation	17

List of Figures

Figure 1: Purchaser agent.....	5
Figure 2: Seller agent.....	6
Figure 3: Semantic E-Business	7
Figure 4: PIM Architecture	8
Figure 5: Relationship between ontology classes	14
Figure 6: Amazon XML example	17
Figure 7: Amazon translation result.....	18
Figure 8: Data collection implementation.....	23
Figure 9: Shopping agent implementation.....	24

1. Introduction

The World Wide Web is predominantly a large collection of human readable documents that are linked together with hyperlinks. While it serves the purpose of browsing, the data behind many websites are not available easily to software programs. This limits the usage of such data and thereby limiting the usage of the World Wide Web itself. To improve the usability of the web, WWW Consortium proposed Semantic Web a few years ago. Wikipedia defines the semantic web as “an evolving extension of the World Wide Web in which web content can be expressed not only in natural language, but also in a format that can be read and used by software agents, thus permitting them to find, share and integrate information more easily.” WWW Consortium has defined standards such as Resource Description Framework (RDF) and Web Ontology Language (OWL) for Semantic web implementation.

Online shopping sites such as Amazon, eBay etc have become quite popular due to the ease with which one can browse through the product catalog without leaving the comfort of his/her home. There are also numerous other small websites selling products online. But, not all the data is exposed in such a way that software agents can effectively integrate and search for products matching one’s specific needs. This project is an attempt to make use of the semantic web concepts such as RDF and OWL for improving online shopping experience.

1.1 Limitations of online shopping

This section summarizes the limitations of current online shopping experience.

1.1.1 Lack of machine readable uniform data representation

A large number of shopping websites are only human readable in a browser. Some websites like Amazon and Yahoo shopping provide machine readable data in the form of XML web services. But, each of them uses different XML schema (elements and attributes) and so there's no uniformity in the data representation. This requires different pieces of software to be written to interpret each website's data.

1.1.2 Lack of inter-linked data

Though many websites may contain data about the same product, they're not inter-linked in an effective manner. For example, Amazon provides detailed catalog data (product specifications) for all its products. Other small shopping sites may offer the same products as Amazon but they may not provide catalog data. Since this offer data is not linked with catalog data, deep search/filtering, involving both catalog data and offer data is not possible.

1.1.3 Lack of personalization

Though some websites like Amazon attempt to remember your past purchases and suggest new products, it is not applicable to other websites. There's no single place where one can store his personal preferences for brand, price range etc in a way that is applicable across shopping sites.

1.1.4 Lack of update notification

When a new product is added or a product's price or other attributes change, one doesn't get to know about it until he/she visits the website again. Thus, if one wants to

follow a particular product until its price comes down to his comfort range, he has to visit multiple websites repeatedly.

1.2 Related Literature

In this section, we highlight some of the literature related to Internet shopping and usage of intelligent agents for improving shopping experience.

Ziming Zeng and Bo Meng [12] list the following as required characteristics for an intelligent shopping system:

(1) Autonomy: The system should have autonomous transaction facilities. The system should generate shopping lists from multiple suppliers on Internet. The system should have comparison mechanism to help customers make the best possible selection of supplied commodities.

(2) Adaptation to customer preferences: The system should learn and imitate the customer's individual preferences automatically by referring to the customer's past shopping behavior

(3) Multiple supplier access: the shopping system should gather commodities information from several suppliers and evaluate it with multiple attributes of the commodities (not just price).

They had also implemented a multi-agent based intelligent system using JADE [4]. As per their paper, the following agents were implemented:

“Interface agent

The main work of the interface agent is bidirectional communication between the shopping system and customers. It interacts with the customer to respond to his/her shopping request.

Buyer manager

This agent responds to the purchase requirements of customers. It is responsible for generating the commodity shopping lists and searching the commodity information from multiple suppliers on Internet.

Buyer agent

When buyer agents are created, they will be dispatched to search for the commodity information from the respective supplier. After they find the respective supplier, each buyer agent will ask for an offer about the commodity from the respective supplier

Evaluation agent

The evaluation agent is an important component of the online shopping system. After receiving the offers of all commodities from the suppliers, the evaluation agent will have comparison mechanism to evaluate each commodity in order to make the best possible selection of all the supplied commodities

Preference agent

If the customer is still not satisfactory to the recommended commodity, the preference agent will acquire the customer's purchasing preference from the preference knowledge base and perform preference analyzing."

Nikos and Pavlos [3] developed generic and re-usable agent architecture.

Figure 1: Purchaser agent

A purchaser agent is composed of three modules (namely, the *communication*, *coordination* and *decision making* modules). These module exchange intra-agent messages and run concurrently. A purchaser agent remains idle while no messages arrive at its communication module. As soon as a message arrives, the communication module transforms it to an intra-agent message and sends it to the coordination module using a queuing mechanism. The decision making module implements the behavior of the agent by using appropriate rules and performs a comparison of the offers sent by the seller agents.

Seller agent architecture is similar to purchaser agent. The *communication module* behaves the same as its counterpart in the purchaser agent. The coordination module handles interaction between (i) the seller and the purchaser agents, and (ii) the seller and its merchant. A *selling coordinator* manages the message exchange. The *selling database* keeps records of the products' specification, and potential customers to be informed about the release of a new product.

Figure 2: Seller agent

Yun Wan and Satya Menon [9] developed a formal classification system for comparison shopping agents that often operate in different ways. They define “comparison” as a thinking process that involves two modes of thinking: associative reasoning and rule-based reasoning. The associative reasoning operates by grouping objects with similar other objects [5]. The rule-based reasoning operates using abstract symbols based on logical rules that have a certain well specified symbolic logical structure [7]. They have also defined three different processes involved in comparison shopping.

- *Differentiation* abstracts the products into symbols like price, weight, quality, etc. by rule-based reasoning
- *Evaluation* uses both rule-based reasoning (from quantified experience about the product or the service) and associative reasoning (from past experience).
- *Preference Identification* is mainly achieved through associative reasoning.

Whenever it is difficult to choose based on evaluation, specific preference will be used to make the decision.

Rahul Singh et al [6] describe semantic e-business as “an approach to managing knowledge for the coordination of e-business processes through the systematic application of Semantic Web technologies”. Figure 1 shows the high-level components for creating a semantic e-business.

Figure 3: Semantic E-Business

Jingzhi Guo and Chengzheng Sun [2] highlight the type of conflicts that could arise in the representation of product data in XML by different websites:

- “*Product classification conflicts* arise from the same product being classified in different ways.
- *Product definition conflicts* arise from different definitions of product identifier to definition mapping.
- *Product structure conflicts* arise from the same product being defined by different number of and kinds of attributes.

- *Attribute description conflicts* are similar to product definition conflicts but for attributes.
- *Measurement conflicts* arise from different units being used for the value of an attribute without explicit declaration of the unit being used.”

Brunner et al [1] describe their experiences in using semantic web technologies for Product Information Management (PIM). They use RDF for synchronization of product information management utilities to other core business entities, such as customer data integration (CDI). They use OWL for its richer properties and relationships. According to the paper: “The expressivity of OWL allows the definition of logical classes (*intersection, union and complement* operators), which enables automatic classification for product items. OWL Restrictions support the creation of dynamic categorization of product lines.” The following figure shows the architecture of their system.

Figure 4: PIM Architecture

2.0 Semantic web standards

It was clear that the only way to achieve machine-readable and uniform data representation was to standardize the terms used to represent product data. We could possibly come up with a standard XML schema for representing the data from multiple websites. But XML doesn't naturally provide a way for defining semantics and inter-linking the data. This is where the Semantic web standards such as RDF and OWL come handy.

2.1 Resource Description Framework (RDF)

RDF is a data model for representing resources and their properties. RDF uses subject, property, object triplets where subject is a Uniform Resource Identifier (URI), property is one of its attributes and object is a URI or a literal value. RDF is a directed named graph whereas XML is represented as an undirected tree.

2.2 Web Ontology Language (OWL)

OWL [10] defines more vocabulary for defining domain specific ontology using classes and properties. A class is defined as “a group of individuals that belong together because they share some properties”. Individuals are instances of classes. A property is used to “state relationships between individuals or from individuals to data values”. As per the standard, one can define:

- Relationships between classes
 - Two classes may be stated to be equivalent using `equivalentClass`
 - Two properties may be stated to be equivalent using `equivalentProperty`

- Two individuals may be stated to be the same using `sameAs`
- An individual may be stated to be different from other individuals using `differentFrom`
- Cardinality of classes
 - `minCardinality`: If a *minCardinality* of 1 is stated on a property with respect to a class, then any instance of that class will be related to at least one individual by that property
 - `maxCardinality`: If a *maxCardinality* of 1 is stated on a property with respect to a class, then any instance of that class will be related to at most one individual by that property
- Property types
 - `ObjectProperty` is used to define relationships between individuals
 - `DatatypeProperty` is used to define relationships between individuals and data values
- Property characteristics
 - One property may be stated to be the inverse of another property using `inverseOf`
 - Properties may be stated to be transitive using `TransitiveProperty`.
If a property is transitive, then if the pair (a, b) is an instance of the transitive property P, and the pair (b, c) is an instance of P, then the pair (a, c) is also an instance of P
 - Properties may be stated to be symmetric using `SymmetricProperty`. If a property is symmetric, then if the pair (a,

b) is an instance of the symmetric property P, then the pair (b, a) is also an instance of P

- Properties may be stated to have a unique value using
FunctionalProperty
- Properties may be stated to be inverse functional using
InverseFunctionalProperty

2.3 SPARQL query language

SPARQL [8] is a SQL-like query language for querying RDF graphs. It provides facilities to:

- extract information in the form of URIs and literals.
- extract RDF subgraphs.
- construct new RDF graphs based on information in the queried graphs.

2.4 Jena framework

Jena [11] is HPLabs' semantic web framework that supports RDF/OWL and supports persistent storage of these in well known databases such as MySQL and Oracle.

As per their website, it supports the following:

- A RDF API
- Reading and writing RDF in RDF/XML, N3 and N-Triples
- An OWL API
- In-memory and persistent storage
- SPARQL query engine

3.0 Design

This section describes the design aspects of this project. The goal of this project is to design a shopping agent which supports

- Deep Search – Search by various properties in catalog data
- Wide Search – Search across the data from various sites
- Personalized Search – Personalize the search by obtaining and storing user's search criteria

The overall approach for this project is as follows:

- Define an OWL ontology to standardize the classes and properties that can be used to describe catalog and offer data
- Import data from multiple websites (Amazon, Yahoo shopping, Buy.com) into this ontology's classes and properties. Inter-link the offer data from all three of them with catalog data from Amazon and store them persistently
- Develop a form interface to get the personalized filter criteria from user
- Display the query results from the database in a user-friendly format
- Store the filter criteria as a SPARQL query in database so that updated results can be shown every time the user logs in.

The following sub-sections describe the design and implementation of each of these steps.

3.1 Ontology design

An OWL ontology called Ontology for Product Data was developed using the Protégé ontology editor. Protégé is an open source ontology editor and knowledge base

framework written in Java developed by Stanford University. It has a plug-in for creating OWL ontologies. A screenshot of the editor is shown below.

The ontology has the following core classes:

- **Product** Class – describes the catalog entry for the product (specifications)
- **Offer** Class – describes sales related properties of the product
- **Seller** Class – describes the properties of a seller
- **Review** Class – describes the properties of a review for the product

Figure 5: Relationship between ontology classes

Product and Offer classes are linked by productType property.

Product and Review classes are linked by reviewOf property.

Seller and Offer classes are linked by hasProduct property.

Object and Datatype properties have been defined for these classes.

For Product class, the defined properties are:

Table 1: Properties of Product class

Property	Type	Description
brand	string	Brand name
family	string	Product family name
model	string	Model name
category	string	Product category
subcategory	string	Product subcategory
description	string	Description of the product
color	string	Color
weight	int	Weight in lbs
width	int	Width in inches
height	int	Height in inches
depth	int	Depth in inches

For Offer class, the defined properties are:

Table 2: Properties of Offer class

Property	Type	Description
productType	resource	Product resource URI
price	int	Price in USD
condition	string	new/old/refurbished
stockStatus	string	INSTOCK/NOSTOCK

link	string	URL describing this product
UPCcode	string	UPC code
ISBN	string	ISBN for books

For Review class, the defined properties are:

Table 3: Properties of Review class

Property	Type	Description
reviewOf	resource	Product resource URI
reviewer	string	Reviewer's name
rating	int	5-star rating(1 to 5)
reviewText	string	Text of the review
reviewDate	date	Date of the review
reviewSource	string	Source website (if any)

For Seller class:

Table 4: Properties of Seller class

Property	Type	Description
name	string	Seller's name
email	string	Contact email
hasProduct	resource	Offer resource URI

3.2 Data collection

It was decided to use real life data from Amazon, Yahoo and Buy.com to demonstrate the shopping agent. Laptops were chosen as the product category for which the data was imported. Amazon and Yahoo provide XML web services for extracting product data. Amazon provides detailed catalog data describing various properties of products and in addition, provides offer data. Offer data was also imported from Yahoo shopping (XML web service) and Buy.com (RSS feed).

3.2.1 XML to OWL Translator

Instead of using the Java client library provided by Amazon and Yahoo, a generic XML Translator was designed to translate any given XML to our OWL format based on

a mapping table that maps XPATH expressions to OWL class/property. The translation logic is performed in two phases.

Phase 1: Collect the set of nodes for each XPATH expression using Java XPATH API and create two hash maps – one mapping XML nodes to OWL classes and another mapping to OWL properties. Note that not all nodes will be mapped like this.

Phase 2: Parse the XML document tree from the root recursively and lookup the hash maps for each node encountered. If there's a mapping for that node, then create the corresponding OWL class/property.

Here's an example mapping for Amazon's XML.

Table 5: Amazon XML to OWL

XPATH expression	OWL class/property
//Item	Class http://www.owl-ontologies.com/opd.owl#Product
//Item/ItemAttributes/Brand	Property http://www.owl-ontologies.com/opd.owl#brand
//Item/ItemAttributes/Model	Property http://www.owl-ontologies.com/opd.owl#model
//Item/ItemAttributes/ItemDimensions/Length	Property http://www.owl-ontologies.com/opd.owl#length
//Offer	Class http://www.owl-ontologies.com/opd.owl#Offer Object Property (domain) http://www.owl-ontologies.com/opd.owl#productType
//Offer/OfferListing/Price/Amount	Property http://www.owl-ontologies.com/opd.owl#price
//Offer/OfferAttributes/Condition	Property http://www.owl-ontologies.com/opd.owl#condition
//Offer/Merchant	Class http://www.owl-ontologies.com/opd.owl#Seller Object Property (domain) http://www.owl-ontologies.com/opd.owl#hasProduct
//Review	Class http://www.owl-ontologies.com/opd.owl#Review

Each XML element //Item will result in an instance of the Product class being created.

Each XML element //Item/ItemAttributes/Brand will result in the 'brand' property getting created.

The element //Offer will result in both:

- Instance of Offer class getting created
- A “productType” object property will be created linking the parent Product object and the child Offer object.

As an example, consider this portion of XML tree from Amazon.

Figure 6: Amazon XML example

The following tables show how the nodes will be translated to classes and properties.

Table 6: Node to class translation

Nodes (using XPATH API)	Class name
N2, N3	http://www.owl-ontologies.com/opd.owl#Product
N5, N6	http://www.owl-ontologies.com/opd.owl#Offer
N7	http://www.owl-ontologies.com/opd.owl#Seller

Table 7: Node to property translation

Nodes (using	Property name
---------------------	----------------------

XPATH API)	
N5, N6	http://www.owl-ontologies.com/opd.owl#productType
N7	http://www.owl-ontologies.com/opd.owl#hasProduct
N9	http://www.owl-ontologies.com/opd.owl#condition
N11	http://www.owl-ontologies.com/opd.owl#price

After translation, the following RDF graph will be created.

Figure 7: Amazon translation result

3.2.2 Algorithm

Jena framework was chosen to create an Ontology model and a Data model containing the instances of various classes of the ontology. The data model was made persistent in MySQL database using the persistent model support of Jena. Here's the recursive algorithm for processing a particular node of the XML tree:

```

processNode(node, parent class name, parent Jena object)
{
    if there's a mapping for this node in Class map
        create a child Jena object of the specified ontology class
    if there's a mapping for this node in Property map
        if the property type is an object property
            add the object property to child or parent Jena object
            (depending on domain/range)
        if the property type is a datatype property
            add the value of this node as the datatype property to parent Jena
object
        return without recursing;
    for all the child nodes of this node
        if a child object was created above
            call processNode(child node, child class name, child Jena object)
        else
            call processNode(child node, parent class name, parent Jena
object)
}

```

3.2.3 RSS to OWL translator

Buy.com doesn't provide XML web service but only an RSS feed. The data from RSS feed was read using Java ROME RSS library. Product brand and model was

extracted from the title and if the product is found in the Amazon catalog, a new object of Offer class was created. Thus, Buy.com's data was linked to Amazon catalog.

3.3 User interface design

3.3.1 Login interface

A user is allowed to login using his/her OpenID URL. OpenID is a single sign-on system with which one can login to several websites using his account with one particular website. It is secure because the account password is not shared with all the websites. Instead, when the user provides the OpenID URL, he is transferred to the OpenID provider's website where he gives permission to authenticate and then transferred back to the OpenID consumer website. Here's a screenshot of the login interface.

A MySQL Accounts table stores a record for every user that logged in using his openID URL as the primary key. This table is used to store other data such as SPARQL query, last login date/time etc.

3.3.2 Query/Filter interface

In order to personalize the shopping experience, every user is allowed to choose his filter criteria for laptops. An HTML form is presented to the user as follows where he can select the criteria such as Brand name, minimum hard disk capacity, minimum display size, max weight etc. The user can also specify upto 3 different sorting parameters and their order.

The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://localhost/SemWebMarket/form.jsp`. The page title is "Product Search (Laptops)". The form is titled "Please choose your filter settings:" and contains the following fields and controls:

- Brand: Sony (dropdown menu)
- Hard Disk Capacity (atleast): GB
- Memory Capacity (atleast): MB
- Display Size (atleast): inches
- Weight (atmost): lbs
- CPU Speed (atleast): MHz
- Price: From \$ To \$
- Sort: First by Date (Newest first) (dropdown menu)
- Then by None (dropdown menu)
- Then by None (dropdown menu)
- Submit and Reset buttons

The status bar at the bottom of the browser window shows "Done" and a green checkmark icon.

This interface is presented only the first time a particular user logs in. Once he selects his preferences, it is persistently stored in the Accounts table in the form of a SPARQL query. Next time he logs in, he doesn't have to provide his preferences again.

3.3.3 Query results interface

Query results are shown in multiple tabs, one tab that shows combined results from all merchants and one tab for each merchant. The offers that match the user criteria are listed in the order sorted by the parameters that user specified. Links to the offer pages of the merchants are also provided. Catalog data from Amazon for the product that is being offered is also displayed at the right.

The following controls are provided at the top:

Mark All as Viewed – Used as a marker so that the next user logs in, he is shown only the new/updated items

Modify Preferences – If user decides to change his filter settings, he could use this option.

Logout – To log out of the system

4.0 Implementation details

Figure 8: Data collection implementation

The figure above shows the files involved in the implementation of data collection. Two types of translators are employed:

`XML2OWLTranslator.java` – Generic XML to OWL translator as described in section 3.2.1. It uses `XPATH API` and `DOM API` to translate.

`RSS2OWLTranslator.java` – Translator for Buy.com RSS feed. It uses `ROME RSS API`.

XML2OWLTranslator is used by AmazonTranslator and YahooTranslator. These two files deal with the specifics of Amazon and Yahoo such as unit conversion of parameters such as price, weight etc (for ex. Amazon provides the price amount as flat number which should be converted to dollars and cents).

Figure 9: Shopping agent implementation

The figure above shows the implementation of shopping agent that utilizes the inter-linked database. At the backend, Jena framework is employed to access OWL ontology model and instances of various classes and their properties. MySQL is used for persistently storing this data. Filter.java implements the following:

- Generate a SPARQL query based on the filter criteria that user provided
- Use Jena classes and methods to execute the query on the database
- Store/retrieve the query (filter) in Accounts table in the record created for the user

It uses the following Jena classes:

- OntModel – Ontology Model
- ModelMaker – one that can be used to create/open persistent models
- IDBConnection – Jena’s connection to DB
- Query, QueryExecution – To form and execute SPARQL query
- ResultSet, QuerySolution – To get and process the query results

The frontend is served by Tomcat server through a set of JSP files. Two other Java Bean files are used to pass the filter data and get back results.

FilterBean.java – stores the filter criteria in 3 different hash maps:

- lowLimits – maps the parameter to its low limit (for ex. minimum hard disk size)
- highLimits – maps the parameter to its high limit (for ex. maximum weight)
- exactValues - maps the parameter to the exact value that is expected (for ex. brand name)

ProductBean.java is a bean used to package the query results from Jena so that it can be used in JSP files.

For providing OpenID support, open source package openID4java has been integrated. The library currently supports the OpenID specifications: OpenID Authentication 2.0 (latest version) and OpenID Attribute Exchange 1.0, draft 4.

The following JSP files are used for the front-end:

Index.jsp – Used to get login info

Form.jsp – Used to get filter settings from user

Laptops.jsp – Used to display the query results

Consumer_redirect.jsp – Used to redirect to OpenID provider

Consumer_returnurl.jsp – Used to process the response from OpenID provider

5.0 Usage Example

In this section, we will go through each step of using the system along with screenshots.

First, the user logs in with his/her openID URL <http://ramani.myopenid.com>.

Please login:

OpenID URL:

[Or login as Guest](#)

He is then forwarded to the openID provider “myopenid”.

Once the user verifies the consumer site, he's taken to the form interface.

The example search is for Sony laptop with at least 200GB hard drive with price less than \$1500 and results should be sorted by hard disk capacity first and then by price.

Product Search (Laptops)

Please choose your filter settings:

Brand: Sony

Hard Disk Capacity (atleast): 200 GB

Memory Capacity (atleast): MB

Display Size (atleast): inches

Weight (atmost): lbs

CPU Speed (atleast): GHz

Price: From \$ To \$1500

Sort: First by Hard disk capacity

Then by Price (Lowest first)

Then by None

Submit Reset

This will result in the following SPARQL query being generated.

```
PREFIX opd: <http://www.owl-ontologies.com/opd.owl#> PREFIX xsd:
<http://www.w3.org/2001/XMLSchema#> SELECT DISTINCT ?brand ?model
?title ?weight ?pageLink ?imageLink ?cpuSpeed ?memorySize ?hardDiskSize
?displaySize ?price ?offerLink ?merchant ?date WHERE { ?offer
opd:productType ?product .?product opd:brand ?brand . ?product
opd:model ?model . ?product opd:title ?title . ?product opd:weight
?weight . ?product opd:pageLink ?pageLink . ?product opd:imageLink
?imageLink . ?product opd:cpuSpeed ?cpuSpeed . ?product opd:memorySize
```

```
?memorySize . ?product opd:hardDiskSize ?hardDiskSize . ?product
opd:displaySize ?displaySize . ?offer opd:price ?price . ?offer
opd:offerLink ?offerLink . ?offer opd:merchant ?merchant . ?offer
opd:date ?date . FILTER (?brand == "Sony" && xsd:float(?hardDiskSize)
>=200 && xsd:float(?price) <= 1500)) } ORDER BY
DESC(xsd:float(?hardDiskSize)) xsd:float(?price) LIMIT 25
```

Filter criteria will be used to form the “FILTER” section of the query. Sort criteria will be used to form the ORDER BY list. The query results in the following output.

Sony laptops with the highest hard disk capacity (320 GB) show up at the top and then lower capacity (300 GB/250 GB) laptops appear. Within the 320 GB laptops, ordering is based on price as requested. After clicking “Yahoo Shopping” tab, similar kind of results from Yahoo is displayed.

When the user logs in next time, updated results are displayed directly without going through the form interface.

5.1 Comparison with other shopping sites

In this section, we compare the shopping options provided by some other sites to our system. Laptops section of Officedepot.com provides the following options:

- Keyword search
- Narrow by Brand OR Price range OR Feature (but only one at a time)

It is clearly less flexible than our shopping agent which supports selecting both brand and price range at the same time and searching across multiple shopping sites.

Amazon has better search options:

- A browse option in the left, to browse by brand or by display size etc
- A deep search option to the right, to search by multiple parameters like brand, CPU speed, price range etc

But again, this works only on Amazon's product data and data from other sites are not inter-linked. On top of it, neither of these websites support

- saving user search preferences persistently
- hierarchical sorting by multiple parameters

6.0 Conclusion

Applicability of Semantic web technologies to shopping search has been explored and demonstrated in this project. OWL ontology is found to be a good way to represent the data in uniform manner and interlink it. Data was imported from Amazon, Yahoo shopping and Buy.com and interlinked using the ontology. User interfaces for login, query form and result display were developed and tested. The resulting system is unique because there's no other website where one can perform such complex search across the data from these major shopping sites and get results in a consistent interface.

7.0 Future work

This project can be enhanced and improved in the following ways:

1. Other types of product related data such as review data etc can be linked to catalog data.
2. Other product categories such as camera, camcorder etc can be added to expand the usefulness of this system.
3. More intelligence can be incorporated for filter criteria using fuzzy logic to allow for natural language search queries such as “Any lightweight laptop with large hard disk capacity but doesn’t cost more than \$1500”. This should result in finding the laptops that satisfy the price constraint and with weight and hard disk capacity parameters also meeting their constraints effectively, i.e. weight should be as low as possible and hard disk capacity should be as high as possible.

Appendix A

Ontology definition (OPD.owl):

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://www.owl-ontologies.com/opd.owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.owl-ontologies.com/opd.owl">
  <owl:Ontology rdf:about="" />
  <owl:Class rdf:ID="Review" />
  <owl:Class rdf:ID="Offer" />
  <owl:Class rdf:ID="Product" />
  <owl:Class rdf:ID="Seller" />
  <owl:ObjectProperty rdf:ID="hasProduct">
    <rdfs:domain rdf:resource="#Seller" />
    <rdfs:range rdf:resource="#Offer" />
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="isbn">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
    <rdfs:domain rdf:resource="#Product" />
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="displaySize">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
    <rdfs:domain rdf:resource="#Product" />
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="hardDiskSize">
    <rdfs:domain rdf:resource="#Product" />
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="memorySize">
    <rdfs:domain rdf:resource="#Product" />
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="cpuSpeed">
    <rdfs:domain rdf:resource="#Product" />
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="batteryLife">
    <rdfs:domain rdf:resource="#Product" />
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
  </owl:DatatypeProperty>
  <owl:FunctionalProperty rdf:ID="stockStatus">
    <rdfs:domain rdf:resource="#Offer" />
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty" />
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >how many product instances are available</rdfs:comment>
  </owl:FunctionalProperty>
```

```

    <owl:FunctionalProperty rdf:ID="model">
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      <rdfs:domain rdf:resource="#Product"/>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="length">
      <rdfs:domain rdf:resource="#Product"/>
      <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>length in inches</rdfs:comment>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="category">
      <rdfs:domain rdf:resource="#Product"/>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="upccode">
      <rdfs:domain rdf:resource="#Offer"/>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="title">
      <rdfs:domain>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#Product"/>
            <owl:Class rdf:about="#Offer"/>
          </owl:unionOf>
        </owl:Class>
      </rdfs:domain>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="type">
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:domain rdf:resource="#Product"/>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="height">
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:domain rdf:resource="#Product"/>

```

```

        <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >height in inches</rdfs:comment>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="brand">
        <rdfs:domain rdf:resource="#Product"/>
        <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="pageLink">
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <rdfs:domain rdf:resource="#Product"/>
        <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="imageLink">
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <rdfs:domain rdf:resource="#Product"/>
        <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="reviewDate">
        <rdfs:domain rdf:resource="#Review"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
        <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="email">
        <rdfs:domain rdf:resource="#Seller"/>
        <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="description">
        <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <rdfs:domain rdf:resource="#Product"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="price">
        <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
        <rdfs:domain rdf:resource="#Offer"/>
        <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >price in USD</rdfs:comment>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="offerLink">
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

```

```

        <rdfs:domain rdf:resource="#Offer"/>
        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        </owl:FunctionalProperty>
        <owl:FunctionalProperty rdf:ID="date">
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <rdfs:domain rdf:resource="#Offer"/>
        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        </owl:FunctionalProperty>
        <owl:FunctionalProperty rdf:ID="merchant">
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <rdfs:domain rdf:resource="#Offer"/>
        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        </owl:FunctionalProperty>
        <owl:FunctionalProperty rdf:ID="rating">
        <rdfs:range>
        <owl:DataRange>
        <owl:oneOf rdf:parseType="Resource">
        <rdf:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</rdf:first>
        <rdf:rest rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
        <rdf:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >5</rdf:first>
        <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-
rdf-syntax-ns#nil"/>
        </rdf:rest>
        <rdf:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >4</rdf:first>
        </rdf:rest>
        <rdf:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >3</rdf:first>
        </rdf:rest>
        <rdf:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >2</rdf:first>
        </rdf:rest>
        </owl:oneOf>
        </owl:DataRange>
        </rdfs:range>
        <rdfs:domain rdf:resource="#Review"/>
        <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >5-star rating</rdfs:comment>
        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        </owl:FunctionalProperty>

```

```

    <owl:FunctionalProperty rdf:ID="width">
      <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >width in inches</rdfs:comment>
      <rdfs:domain rdf:resource="#Product"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="condition">
      <rdfs:range>
        <owl:DataRange>
          <owl:oneOf rdf:parseType="Resource">
            <rdf:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >New</rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:rest rdf:parseType="Resource">
                <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-
syntax-ns#nil"/>
                <rdf:first
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >Refurbished</rdf:first>
              </rdf:rest>
            </rdf:rest>
          </owl:oneOf>
        </owl:DataRange>
      </rdfs:range>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:domain rdf:resource="#Offer"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="reviewSource">
      <rdfs:domain rdf:resource="#Review"/>
      <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >source website for the review</rdfs:comment>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="productType">
      <rdfs:domain rdf:resource="#Offer"/>
      <rdfs:range rdf:resource="#Product"/>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="color">
      <rdfs:domain rdf:resource="#Product"/>
      <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

```

```

    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="family">
      <rdfs:domain rdf:resource="#Product"/>
      <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="name">
      <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:domain rdf:resource="#Seller"/>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="reviewer">
      <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      <rdfs:domain rdf:resource="#Review"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="reviewText">
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      <rdfs:domain rdf:resource="#Review"/>
      <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >text of the review</rdfs:comment>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="weight">
      <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:domain rdf:resource="#Product"/>
      <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >weight in lbs</rdfs:comment>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="subcategory">
      <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      <rdfs:domain rdf:resource="#Product"/>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:FunctionalProperty>
    <owl:FunctionalProperty rdf:ID="reviewOf">
      <rdfs:range rdf:resource="#Product"/>
      <rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
      <rdfs:domain rdf:resource="#Review"/>
    </owl:FunctionalProperty>
  </rdf:RDF>

```

```
<!-- Created with Protege (with OWL Plugin 3.3.1, Build 430)
http://protege.stanford.edu -->
```

XML2OWLTranslator.java:

```
import org.xml.sax.SAXException;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.xpath.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.util.FileManager;
import java.util.Hashtable;
import java.io.*;

public class XML2OWLTranslator {

    private Hashtable<Node, String> class_map = new
    Hashtable<Node,String>();
    private Hashtable<Node, String> property_map = new
    Hashtable<Node,String>();
    private OntModel ontmodel;
    private OntModel datamodel;

    XML2OWLTranslator(String[] owlfiles) {
        ontmodel = ModelFactory.createOntologyModel();

        InputStream in;
        for (int i = 0; i < owlfiles.length; i++) {
            in = FileManager.get().open(owlfiles[i]);
            if (in == null) {
                throw new IllegalArgumentException( "File:
not found");
            }

            // read the OWL file
            ontmodel.read(in, "");
        }

        ModelMaker maker = getModelMaker(false);
        Model m = maker.createModel("Laptops", false);
        datamodel = ModelFactory.createOntologyModel(getModelSpec(maker),
m);
    }

    private ModelMaker getModelMaker(boolean cleanDB) {
        // database connection parameters, with defaults
        String dbURL = "jdbc:mysql://localhost/test";
        //String dbURL =
"jdbc:mysql://bvibes.com/arunagmail1061com1085_shopfeedz";
        String dbUser = "rarunach";
        String dbPw = "gud_luk";
        String dbType = "MySQL";
        String dbDriver = "com.mysql.jdbc.Driver";
    }
}
```



```

        try {
            Class.forName( dbDriver );
            // Create database connection
            IDBConnection conn = new DBConnection( dbURL, dbUser,
dbPw, dbType );
            // do we need to clean the database?
            if (cleanDB) {
                conn.cleanDB();
            }

            // Create a model maker object
            return ModelFactory.createModelRDBMaker(conn);
        }
        catch (Exception e) {
            e.printStackTrace();
            System.exit( 1 );
        }

        return null;
    }

    private OntModelSpec getModelSpec(ModelMaker maker) {
        // create a spec for the new ont model that will use no
inference over models
        // made by the given maker (which is where we get the
persistent models from)
        OntModelSpec spec = new OntModelSpec( OntModelSpec.OWL_MEM );
        spec.setImportModelMaker( maker );

        return spec;
    }

    private void processNode(Node n, String parent_class, OntResource
parent) {
        boolean instantiated = false;
        OntResource child = null;
        String child_class = class_map.get(n);
        if (child_class != null) {
            OntClass cls = ontmodel.getOntClass(child_class);
            NamedNodeMap nnm = n.getAttributes();
            Node uriNode = nnm.getNamedItem("uri");
            if (uriNode != null) {
                String uri = uriNode.getNodeValue();
                if (datamodel.getOntResource(uri) != null) {
                    return;
                } else {
                    System.out.println("New product: " + uri);
                    child = datamodel.createIndividual(uri, cls);
                }
            } else child = datamodel.createIndividual(cls);
            instantiated = true;
        }
        String property = property_map.get(n);

        if (property != null) {
            String [] words = property.split(" ");

```

```

        if (words.length == 2) {
            String type = words[0].trim();
            String obj_property = words[1].trim();
            ObjectProperty objprop =
ontmodel.getObjectProperty(obj_property);

            if (type.equals("domain")) {
                if (child != null)
                    child.addProperty(objprop, parent);
            } else if (type.equals("range")) {
                if (parent != null)
                    parent.addProperty(objprop, child);
            }
        } else {
            DatatypeProperty dtprop =
ontmodel.getDatatypeProperty(property);
            parent.addProperty(dtprop,
n.getFirstChild().getNodeValue());
            return;
        }
    }
    NodeList nodes = n.getChildNodes();
    for (int i = 0; i < nodes.getLength(); i++) {
        if (instantiated)
            processNode(nodes.item(i), child_class, child);
        else
            processNode(nodes.item(i), parent_class, parent);
    }
}

private void parseMapFile(Document doc, String filename) {
    try {
        XPathFactory factory = XPathFactory.newInstance();
        XPath xpath = factory.newXPath();
        BufferedReader in = new BufferedReader(new
FileReader(filename));
        String line;

        while ((line = in.readLine()) != null) {
            String [] arr = {"", ""};
            boolean is_class = false;
            if (line.indexOf("->") != -1) {
                arr = line.split("->");
                is_class = true;
            } else if (line.indexOf("=") != -1) {
                arr = line.split("=");
                is_class = false;
            }
            XPathExpression expr = xpath.compile(arr[0].trim());
            NodeList nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
            for (int i = 0; i < nodes.getLength(); i++) {
                if (is_class) {
                    class_map.put(nodes.item(i), arr[1].trim());
                } else property_map.put(nodes.item(i),
arr[1].trim());
            }
        }
    }
}

```

```

        }
    }
    in.close();
} catch (Exception e) {
    e.printStackTrace();
}

}

public void translateFile(String xmlfile, String mapfile, String
toptag) {

    try {
        DocumentBuilderFactory domFactory =
DocumentBuilderFactory.newInstance();
        domFactory.setNamespaceAware(true);
        DocumentBuilder builder = domFactory.newDocumentBuilder();
        Document doc = builder.parse(xmlfile);

        parseMapFile(doc, mapfile);

        NodeList nodes = doc.getElementsByTagName(toptag);
        for (int i = 0; i < nodes.getLength(); i++) {
            processNode(nodes.item(i), "", null);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void translateDocument(Document doc, String mapfile, String
toptag) {

    try {
        parseMapFile(doc, mapfile);

        NodeList nodes = doc.getElementsByTagName(toptag);
        for (int i = 0; i < nodes.getLength(); i++) {
            processNode(nodes.item(i), "", null);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public OntModel getOntModel() {
    return ontmodel;
}

public OntModel getDataModel() {
    return datamodel;
}

public void write(String ns, String prefix) {
    datamodel.setNsPrefix(prefix, ns);
    datamodel.write(System.out, "RDF/XML-ABBREV");
}

```

```
}
```

AmazonTranslator.java:

```
import org.xml.sax.SAXException;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.xpath.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.query.*;
import java.util.Hashtable;
import java.util.Date;
import java.text.DateFormat;

public class AmazonTranslator {

    public static void divByHundred(NodeList nodes) {
        for (int i = 0; i < nodes.getLength(); i++) {
            Node n = nodes.item(i).getFirstChild();
            float amount = Float.parseFloat(n.getNodeValue());
            n.setNodeValue(String.valueOf(amount/100));
        }
    }

    public static void main(String[] args) {
        try {

            String[] owlfiles = {"OPD.owl"};
            XML2OWLTranslator translator = new XML2OWLTranslator(owlfiles);

            DocumentBuilderFactory domFactory =
DocumentBuilderFactory.newInstance();
            //domFactory.setNamespaceAware(true);
            DocumentBuilder builder = domFactory.newDocumentBuilder();

            for (int page = 1; page <= 100; page++) {
                Document doc =
builder.parse("http://ecs.amazonaws.com/onca/xml?Service=AWSECommerceSe
rvic&Operation=ItemSearch&AWSAccessKeyId=1KQ5QD8JBN22SGZ7XXR2&SearchIn
dex=Electronics&BrowseNode=565108&ResponseGroup=ItemAttributes,Images,O
ffers&ItemPage="+ page);

                // Set the URIs for all Items
                XPathFactory factory = XPathFactory.newInstance();
                XPath xpath = factory.newXPath();

                XPathExpression expr =
xpath.compile("//ItemSearchResponse");
                NodeList nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
                Element elem = (Element)nodes.item(0);
                elem.removeAttribute("xmlns");

                expr = xpath.compile("//Item");
                nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
```

```

        for (int i = 0; i < nodes.getLength(); i++) {
            Element itemelem = (Element)nodes.item(i);
            NodeList list =
itemelem.getElementsByTagName("Brand");
            if (list.item(0) == null) continue;
            String brand =
list.item(0).getFirstChild().getNodeValue();
            list = itemelem.getElementsByTagName("Model");
            if (list.item(0) == null) continue;
            String model =
list.item(0).getFirstChild().getNodeValue();
            itemelem.setAttribute("uri", "urn:product:" + brand +
":" + model);
        }
/*
*      expr = xpath.compile("//Offer/OfferListing");
*      nodes = (NodeList) expr.evaluate(doc, XPathConstants.NODESET);
*      for (int i = 0; i < nodes.getLength(); i++) {
*          Element itemelem = (Element)nodes.item(i);
*          NodeList list =
itemelem.getElementsByTagName("OfferListingId");
*          if (list.item(0) == null) continue;
*          String offerId =
list.item(0).getFirstChild().getNodeValue();
*          itemelem.setAttribute("uri", "urn:offer:" + offerId);
*      }
*/

// Convert all model names to uppercase
expr = xpath.compile("//Item/ItemAttributes/Model");
nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);

        for (int i = 0; i < nodes.getLength(); i++) {
            Node n = nodes.item(i).getFirstChild();
            String model = n.getNodeValue();
            n.setNodeValue(model.toUpperCase());
        }

// Convert the Offer amount from cents to dollars
expr = xpath.compile("//Offer/OfferListing/Price/Amount");
nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
divByHundred(nodes);

// Convert Itemdimensions to float values
expr =
xpath.compile("//Item/ItemAttributes/ItemDimensions/Height");
nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
divByHundred(nodes);
expr =
xpath.compile("//Item/ItemAttributes/ItemDimensions/Width");
nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
divByHundred(nodes);
expr =
xpath.compile("//Item/ItemAttributes/ItemDimensions/Length");

```

```

        nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
        divByHundred(nodes);
        expr =
xpath.compile("//Item/ItemAttributes/ItemDimensions/Weight");
        nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
        divByHundred(nodes);

        expr = xpath.compile("//Offer/Merchant/MerchantId");
        nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
        for (int i = 0; i < nodes.getLength(); i++) {
            Node n = nodes.item(i).getFirstChild();
            n.setNodeValue("Amazon");
        }

        Date date = new Date();
        expr = xpath.compile("//Offer/OfferListing/Availability");
        nodes = (NodeList) expr.evaluate(doc,
XPathConstants.NODESET);
        for (int i = 0; i < nodes.getLength(); i++) {
            Node n = nodes.item(i).getFirstChild();
            n.setNodeValue(Long.toString(new
Long(date.getTime())));
        }

        translator.translateDocument(doc, "amazonowl.txt",
"Items");
        System.out.println("Page " + page + "
=====");
        //translator.write("http://www.owl-
ontologies.com/opd.owl#", "opd");
    }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
}

```

RSS2OWLTranslator.java:

```

import com.sun.syndication.feed.synd.*;
import com.sun.syndication.io.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.util.FileManager;
import java.net.URL;
import java.util.List;
import java.util.Date;
import java.text.DateFormat;
import java.io.InputStream;

public class BuyReader {

```

```

private String url;
private OntModel ontmodel;
private OntModel datamodel;

BuyReader(String feedurl) {
    url = feedurl;

    try {
        ontmodel = ModelFactory.createOntologyModel();

        InputStream in;
        in = FileManager.get().open("OPD.owl");
        if (in == null) {
            throw new IllegalArgumentException( "File:  not
found");
        }

        // read the OWL file
        ontmodel.read(in, "");

        ModelMaker maker = getModelMaker();
        Model m = maker.openModel("Laptops", true);
        datamodel =
ModelFactory.createOntologyModel(getModelSpec(maker), m);
    } catch (Exception ex) {
        ex.printStackTrace();
        System.out.println("ERROR: "+ex.getMessage());
    }

}

private ModelMaker getModelMaker() {
    // database connection parameters, with defaults
    String dbURL = "jdbc:mysql://localhost/test";
    //String dbURL =
"jdbc:mysql://bvibes.com/arunagmail1061com1085_shopfeedz";
    String dbUser = "rarunach";
    String dbPw = "gud_luk";
    String dbType = "MySQL";
    String dbDriver = "com.mysql.jdbc.Driver";

    try {
        Class.forName( dbDriver );
        // Create database connection
        IDBConnection conn = new DBConnection( dbURL, dbUser,
dbPw, dbType );

        // Create a model maker object
        return ModelFactory.createModelRDBMaker(conn);
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    return null;
}

```

```

    private OntModelSpec getModelSpec(ModelMaker maker) {
        // create a spec for the new ont model that will use no
inference over models
        // made by the given maker (which is where we get the
persistent models from)
        OntModelSpec spec = new OntModelSpec( OntModelSpec.OWL_MEM );
        spec.setImportModelMaker( maker );

        return spec;
    }

    public void process() {
        try {
            URL feedUrl = new URL(url);
            SyndFeedInput input = new SyndFeedInput();

            SyndFeed feed = input.build(new XmlReader(feedUrl));

            List entries = feed.getEntries();
            for( int i = 0; i < entries.size(); i++ ){
                SyndEntry entry = (SyndEntry) entries.get(i);
                String title = entry.getTitle();
                String content =
entry.getDescription().getValue();
                String link = entry.getLink();
                Date date = entry.getPublishedDate();

                String [] words = title.split(" ");
                String brand, model;
                if (words[0].equals("HP")) {
                    brand = "Hewlett-Packard";
                } else if (words[0].equals("VAIO") ||
words[0].equals("Vaio")) {
                    brand = "Sony";
                } else if (words[0].equals("Satellite")) {
                    brand = "Toshiba";
                } else if (words[0].equals("Pavilion")) {
                    brand = "Hewlett-Packard";
                } else {
                    brand = words[0];
                }
                OntResource parent = null;
                String urn = null;
                for (int j = 1; j < 6; j++) {
                    model = words[j];
                    urn = "urn:product:" + brand + ":" +

model.toUpperCase();

                    parent = datamodel.getOntResource(urn);
                    if (parent != null)
                        break;
                }
                System.out.print(urn + " ");

                if (parent != null) {
                    System.out.print("FOUND, ");

```



```

        OntClass cls =
ontmodel.getOntClass("Offer");
        OntResource child =
datamodel.createIndividual(cls);
        DatatypeProperty priceprop =
ontmodel.getDatatypeProperty("http://www.owl-
ontologies.com/opd.owl#price");
        int start = content.indexOf("Our price:
$") + 12;
        int end = content.indexOf("</span>",
start+1);
        String price = content.substring(start,
end).trim();
        price = price.replace(",","");
        System.out.println(price);
        child.addProperty(priceprop, price);
        DatatypeProperty merchantprop =
ontmodel.getDatatypeProperty("http://www.owl-
ontologies.com/opd.owl#merchant");
        child.addProperty(merchantprop,
"Buy.com");
        DatatypeProperty linkprop =
ontmodel.getDatatypeProperty("http://www.owl-
ontologies.com/opd.owl#offerLink");
        child.addProperty(linkprop, link);
        DatatypeProperty dateprop =
ontmodel.getDatatypeProperty("http://www.owl-
ontologies.com/opd.owl#date");
        child.addProperty(dateprop,
Long.toString(new Long(date.getTime())));

        ObjectProperty objprop =
ontmodel.getObjectProperty("http://www.owl-
ontologies.com/opd.owl#productType");
        child.addProperty(objprop, parent);
    } else System.out.println("not found");
}
}
catch (Exception ex) {
    ex.printStackTrace();
    System.out.println("ERROR: "+ex.getMessage());
}
}

public static void main(String[] args) {
    BuyReader reader = new
BuyReader("http://www.buy.com/rss/feed.asp?loc=212&grp=2");
    reader.process();
}
}

```

Filter.java:

```
package SemWebMarket;

import java.util.*;
import java.util.logging.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.db.*;
import com.mysql.jdbc.exceptions.*;

public class Filter {

    private String filter = null;
    private String [] param_arr = {"brand", "model", "title", "weight",
    "pageLink", "imageLink", "cpuSpeed", "memorySize", "hardDiskSize",
    "displaySize", "price", "offerLink", "merchant", "date"};
    private String [] product_param_arr = {"brand", "model", "title",
    "weight", "pageLink", "imageLink", "cpuSpeed", "memorySize",
    "hardDiskSize", "displaySize" };
    private String [] offer_param_arr = {"price", "offerLink",
    "merchant", "date" };
    private Logger logger;

    public Filter() {
        try {

            // Add to the desired logger
            logger = Logger.getLogger("Filter");
            logger.setLevel(Level.INFO);

            //logger.info("Initialized");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private Connection getDBConnection() {
        String dbURL = "jdbc:mysql://localhost/test";
        String dbUser = "rarunach";
        String dbPw = "gud_luk";
        String dbDriver = "com.mysql.jdbc.Driver";
        Connection con = null;
        try {
            Class.forName(dbDriver).newInstance();
            con = DriverManager.getConnection(dbURL, dbUser, dbPw);
        } catch (Exception e) {
            logger.info("Unable to connect to DB");
            e.printStackTrace();
        }
        return con;
    }
}
```

```

    }

    private ModelMaker getModelMaker() {
        // database connection parameters, with defaults
        String dbURL = "jdbc:mysql://localhost/test";
        String dbUser = "rarunach";
        String dbPw = "gud_luk";
        String dbType = "MySQL";
        String dbDriver = "com.mysql.jdbc.Driver";

        try {
            Class.forName( dbDriver );
            // Create database connection
            IDBConnection conn = new DBConnection( dbURL, dbUser,
dbPw, dbType );

            // Create a model maker object
            return ModelFactory.createModelRDBMaker(conn);
        }
        catch (Exception e) {
            logger.info("Unable to get ModelMaker");
            e.printStackTrace();
        }
        return null;
    }

    public void set(String user_openid, Hashtable<String,String>
lowLimits,
                    Hashtable<String,String> highLimits,
                    Hashtable<String,String> exactValues,
                    String [] sortorder) {

        filter = "PREFIX opd: <http://www.owl-ontologies.com/opd.owl#> "
+
                    "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
                    "SELECT DISTINCT ";
        for (int i=0; i < param_arr.length; i++) {
            String param = param_arr[i];
            filter = filter + "?" + param + " ";
        }
        filter = filter + "WHERE { ?offer opd:productType ?product .";
        for (int i=0; i < product_param_arr.length; i++) {
            String param = product_param_arr[i];
            filter = filter + "?product opd:" + param + " ?" + param +
" . ";
        }
        for (int i=0; i < offer_param_arr.length; i++) {
            String param = offer_param_arr[i];
            filter = filter + "?offer opd:" + param + " ?" + param +
. ";
        }
        String filterops = "";
        boolean first = true;
        for (int i=0; i < param_arr.length; i++) {
            String param = param_arr[i];
            String value = exactValues.get(param);

```

```

        if (value == null) {
            String lowLimit = lowLimits.get(param);
            String highLimit = highLimits.get(param);
            if ((lowLimit != null) || (highLimit != null)) {
                if ((lowLimit != null) &&
(!lowLimit.equals(""))) {
                    if (!first)
                        filterops = filterops + " && ";
                    else first = false;
                    filterops = filterops + "xsd:float(?" +
param + ") >= " + lowLimit;
                }
                if ((highLimit != null) &&
(!highLimit.equals(""))) {
                    if (!first)
                        filterops = filterops + " && ";
                    else first = false;
                    filterops = filterops + "xsd:float(?" +
param + ") <= " + highLimit;
                }
            }
        } else {
            if (!first)
                filterops = filterops + " && ";
            else first = false;
            filterops = filterops + "?" + param + " = \"" + value
+ "\"";
        }
    }
    if (!filterops.equals(""))
        filter = filter + "FILTER (" + filterops + ")";
    filter = filter + " } ORDER BY ";

    for (int i = 0; i < 3; i++) {
        if (sortorder[i].equals("price"))
            filter = filter + "xsd:float(?price) ";
        else if (sortorder[i].equals("date"))
            filter = filter + "DESC(xsd:float(?date)) ";
        else if (sortorder[i].equals("brand"))
            filter = filter + "?brand ";
        else if (sortorder[i].equals("harddisk"))
            filter = filter + "DESC(xsd:float(?hardDiskSize)) ";
        else if (sortorder[i].equals("memory"))
            filter = filter + "DESC(xsd:float(?memorySize)) ";
        else if (sortorder[i].equals("display"))
            filter = filter + "DESC(xsd:float(?displaySize)) ";
        else if (sortorder[i].equals("cpu"))
            filter = filter + "DESC(xsd:float(?cpuSpeed)) ";
        else if (sortorder[i].equals("weight"))
            filter = filter + "xsd:float(?weight) ";
    }

    filter = filter + "LIMIT 25";

    //logger.info(filter);
    {
        boolean done = true;

```

```

        try {
            Connection con = getDBConnection();
            String cmd = "INSERT INTO accounts (openid,
laptop_filter) VALUES('" + user_openid + "', '" + filter + "')";
            Statement stmt = con.createStatement();
            stmt.executeUpdate(cmd);
            stmt.close();
            con.close();
        } catch (MySQLIntegrityConstraintViolationException
violation) {
            done = false;
        } catch (Exception ex) {
            logger.info("Set filter exception: " +
ex.toString());
            ex.printStackTrace();
        }
        if (!done) {
            try {
                Connection con = getDBConnection();
                String cmd = "UPDATE accounts SET laptop_filter = '"
+ filter + "' WHERE openid = '" + user_openid + "'";
                Statement stmt = con.createStatement();
                stmt.executeUpdate(cmd);
                stmt.close();
                con.close();
            } catch (Exception ex) {
                logger.info("Set filter exception: " +
ex.toString());
                ex.printStackTrace();
            }
        }
    }

    public String get(String user_openid) {
        try {
            Connection con = getDBConnection();
            String query = "SELECT laptop_filter FROM accounts
WHERE openid = '" + user_openid + "'";
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            if (rs.next()) {
                filter = rs.getString("laptop_filter");
            } else filter = null;
            stmt.close();
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return filter;
    }

    public List processForMerchant(String merchantName) {
        String newfilter;
        if (!merchantName.equals("All")) {

```

```

        newfilter = filter.replace("FILTER (", "FILTER (?merchant =
\"" +
    merchantName + "\" && ");
    } else newfilter = filter;
    System.out.println(newfilter);

    List<ProductBean> productList = new ArrayList<ProductBean>();

    try {
        ModelMaker maker = getModelMaker();
        Model m = maker.openModel("Laptops", true);
        Query query = QueryFactory.create(newfilter) ;
        QueryExecution qexec = QueryExecutionFactory.create(query,
m);

        com.hp.hpl.jena.query.ResultSet results =
qexec.execSelect();
        // Output query results
        // ResultSetFormatter.out(System.out, results, query);
        for ( ; results.hasNext() ; )
        {
            QuerySolution rb = results.nextSolution();
            String brand = rb.get("brand").toString();
            String model = rb.get("model").toString();
            String title = rb.get("title").toString();
            String price = rb.get("price").toString();
            String pagelink = rb.get("pageLink").toString();
            String imagelink = rb.get("imageLink").toString();
            String offerlink = rb.get("offerLink").toString();
            String merchant = rb.get("merchant").toString();
            String date = rb.get("date").toString();
            String attributes = "Hard Disk Capacity: " +
rb.get("hardDiskSize").toString() + " GB, Memory Capacity: " +
rb.get("memorySize").toString() + "MB, Display Size: " +
rb.get("displaySize").toString() + " inches, Weight: " +
rb.get("weight").toString() + " lbs, CPU speed: " +
rb.get("cpuSpeed").toString() + " GHz";

            ProductBean product = new ProductBean(brand, model,
title, price, pagelink, imagelink, offerlink, merchant, date,
attributes);

            productList.add(product);
        }
        qexec.close();
    } catch (Exception e) {
        logger.info("ProcessForMerchant exception: " +
e.toString());
        e.printStackTrace();
    }
    return productList;
}

public static String getLastViewed(String user_openid) {
    String last_viewed = null;
    try {
        Filter f = new Filter();

```

```

        Connection con = f.getDBConnection();
        Date date = new Date();
        String query = "SELECT last_viewed FROM accounts WHERE
openid = '" + user_openid + "'";
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        if (rs.next()) {
            last_viewed = rs.getString("last_viewed");
        }
        stmt.close();
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return last_viewed;
}

public static void setLastViewed(String user_openid) {
    try {
        Filter f = new Filter();
        Connection con = f.getDBConnection();
        Date date = new Date();
        String last = Long.toString(new Long(date.getTime()));
        String update = "UPDATE accounts SET last_viewed = '" +
last + "' WHERE openid = '" + user_openid + "'";
        //String update = "UPDATE accounts SET last_viewed = 'Wed
Mar 12 22:24:00 PDT 2008' WHERE openid = '" + user_openid + "'";
        Statement stmt = con.createStatement();
        stmt.executeUpdate(update);
        stmt.close();
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

FilterBean.java:

```

package SemWebMarket;

import java.util.*;
import SemWebMarket.Filter;

public class FilterBean {

    private Hashtable<String,String> lowLimits = null, highLimits =
null, exactValues = null;
    private String [] sortorder = new String[3];

    public FilterBean() {
        lowLimits = new Hashtable<String,String>();
        highLimits = new Hashtable<String,String>();
        exactValues = new Hashtable<String,String>();
    }
}

```

```

public String getPriceLow() {
    return lowLimits.get("price");
}
public String getPriceHigh() {
    return highLimits.get("price");
}
public String getBrand() {
    return exactValues.get("brand");
}
public String getWeightLow() {
    return lowLimits.get("weight");
}
public String getWeightHigh() {
    return highLimits.get("weight");
}

public void setPriceLow(String val) {
    lowLimits.put("price", val);
}
public void setPriceHigh(String val) {
    highLimits.put("price", val);
}
public void setBrand(String val) {
    if (!val.equals("Any"))
        exactValues.put("brand", val);
}
public void setWeightLow(String val) {
    lowLimits.put("weight", val);
}
public void setWeightHigh(String val) {
    highLimits.put("weight", val);
}
public void setHardDiskLow(String val) {
    lowLimits.put("hardDiskSize", val);
}
public void setMemoryLow(String val) {
    lowLimits.put("memorySize", val);
}
public void setDisplayLow(String val) {
    lowLimits.put("displaySize", val);
}
public void setCpuLow(String val) {
    lowLimits.put("cpuSpeed", val);
}

public void setFirst(String val) {
    sortorder[0] = val;
}
public void setSecond(String val) {
    sortorder[1] = val;
}
public void setThird(String val) {
    sortorder[2] = val;
}
public List processNew(String user_openid, String merchant) {
    Filter filter = new Filter();

```



```

        filter.set(user_openid, lowLimits, highLimits, exactValues,
sortorder);
        return filter.processForMerchant(merchant);
    }
    public List processOld(String user_openid, String merchant) {
        Filter filter = new Filter();
        filter.get(user_openid);
        return filter.processForMerchant(merchant);
    }
}

```

ProductBean.java:

```

package SemWebMarket;

public class ProductBean {
    private String brand, model, title, price, pagelink, imagelink,
offerlink, merchant, date, attributes;

    public ProductBean(String given_brand, String given_model, String
given_title,
                        String given_price, String given_pagelink,
                        String given_imagelink, String given_offerlink,
                        String given_merchant, String given_date,
                        String given_attributes) {
        brand = given_brand;
        model = given_model;
        title = given_title;
        price = given_price;
        pagelink = given_pagelink;
        imagelink = given_imagelink;
        offerlink = given_offerlink;
        merchant = given_merchant;
        date = given_date;
        attributes = given_attributes;
    }

    public String getBrand() {
        return brand;
    }
    public String getModel() {
        return model;
    }
    public String getTitle() {
        return title;
    }
    public String getPrice() {
        return price;
    }
    public String getPageLink() {
        return pagelink;
    }
    public String getImageLink() {
        return imagelink;
    }
}

```

```

        public String getOfferLink() {
            return offerlink;
        }
        public String getMerchant() {
            return merchant;
        }
        public String getDate() {
            return date;
        }
        public String getAttributes() {
            return attributes;
        }
    }
}

```

Index.jsp:

```

<%@ page session="true" %>
<html>
<head>
<style type="text/css">
body {
margin-top:200px;
}

#login {
width:300px;
color:#ffffff;
margin:0 auto 10px;
background: #00688B;
border: solid #ffffff 2px;
-moz-border-radius: 15px;
-webkit-border-radius: 15px;
}
#login a {color:#ffffff;}
</style>
</head>
<body>
<center>
<%
    if (request.getParameter("logout")!=null)
    {
        session.removeAttribute("openid");
        session.removeAttribute("openid-claimed");
    }
    if (session.getAttribute("openid")==null) {
%>
<div id="login">
<br/>
Please login: <br/><br/>
<form method="POST" action="consumer_redirect.jsp">
<strong>OpenID URL:</strong>
<input type="text" name="openid" width="30"/><br><br>
<input type="submit" value="Login"/>
</form>
<a href="form.jsp">Or login as Guest</a><br/><br/>
</div>

```

```

<%
    } else {

%>
Logged in as <%= session.getAttribute("openid") %><p>
<a href="?logout=true">Log out</a>

<% } %>
</body>
</html>

```

Consumer_redirect.jsp:

```

<%@ page session="true" %>
<%@
                                                                    page
import="java.util.Map,java.util.Iterator,org.openid4java.discovery.Iden
tifier,
                                                                    org.openid4java.discovery.DiscoveryInformation,
org.openid4java.message.ax.FetchRequest,
org.openid4java.message.ax.FetchResponse,
org.openid4java.message.ax.AxMessage,
                                                                    org.openid4java.message.*,
org.openid4java.OpenIDException, java.util.List, java.io.IOException,
javax.servlet.http.HttpSession, javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse,
org.openid4java.consumer.ConsumerManager,
org.openid4java.consumer.InMemoryConsumerAssociationStore,
org.openid4java.consumer.InMemoryNonceVerifier" %>

<%
    // README:
    // Set the returnUrl string to the appropriate value for this JSP
    // Since you may be deployed behind apache, etc, the jsp has no
    real idea what the
    // absolute URI is to get back here.

    Object o = pageContext.getAttribute("consumermanager",
PageContext.APPLICATION_SCOPE);
    if (o == null)
    {
        ConsumerManager newmgr=new ConsumerManager();
        newmgr.setAssociations(new InMemoryConsumerAssociationStore());
        newmgr.setNonceVerifier(new InMemoryNonceVerifier(5000));
        pageContext.setAttribute("consumermanager", newmgr,
PageContext.APPLICATION_SCOPE);
    }
    ConsumerManager manager=(ConsumerManager)
pageContext.getAttribute("consumermanager",
PageContext.APPLICATION_SCOPE);
    String openid=request.getParameter("openid");

    try
    {
        // determine a return_to URL where your application will
receive
        // the authentication responses from the OpenID provider
        String returnUrl =
"http://localhost/SemWebMarket/consumer_returnurl.jsp";

```

```

        // perform discovery on the user-supplied identifier
        List discoveries = manager.discover(openid);

        // attempt to associate with an OpenID provider
        // and retrieve one service endpoint for authentication
        DiscoveryInformation discovered =
manager.associate(discoveries);

        // store the discovery information in the user's session
        session.setAttribute("openid-disco", discovered);

        // obtain a AuthRequest message to be sent to the OpenID
provider
        AuthRequest authReq = manager.authenticate(discovered,
returnToUrl);

        // redirect to the OpenID provider
        response.sendRedirect(authReq.getDestinationUrl(true));

    }
    catch (OpenIDException e) {
        // present error to the user
        response.sendError(500);
    }
}
%>

```

Consumer_returnurl.jsp:

```

<%@ page session="true" %>
<%@ page import="org.openid4java.discovery.Identifier,
org.openid4java.discovery.DiscoveryInformation,
org.openid4java.message.ax.FetchRequest,
org.openid4java.message.ax.FetchResponse,
org.openid4java.message.ax.AxMessage, org.openid4java.message.*,
org.openid4java.OpenIDException, java.util.List, java.io.IOException,
javax.servlet.http.HttpSession, javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse,
org.openid4java.consumer.ConsumerManager,
org.openid4java.consumer.InMemoryConsumerAssociationStore,
org.openid4java.consumer.VerificationResult" %>
<%@ page import="SemWebMarket.Filter" %>
<%

        ConsumerManager manager=(ConsumerManager)
pageContext.getAttribute("consumermanager",
PageContext.APPLICATION_SCOPE);
try
    {
        // --- processing the authentication response

        // extract the parameters from the authentication response
        // (which comes in as a HTTP request from the OpenID
provider)
        ParameterList responselist =
            new ParameterList(request.getParameterMap());

        // retrieve the previously stored discovery information

```

```

        DiscoveryInformation discovered =
            (DiscoveryInformation)
session.getAttribute("openid-disco");

        // extract the receiving URL from the HTTP request
StringBuffer receivingURL = request.getRequestURL();
String queryString = request.getQueryString();
if (queryString != null && queryString.length() > 0)

receivingURL.append("?").append(request.getQueryString());

        // verify the response; ConsumerManager needs to be the
same
        // (static) instance used to place the authentication
request
        VerificationResult verification = manager.verify(
            receivingURL.toString(),
            responselist, discovered);

        // examine the verification result and extract the verified
identifier
        Identifier verified = verification.getVerifiedId();
        if (verified != null)
        {
            AuthSuccess authSuccess =
                (AuthSuccess) verification.getAuthResponse();

            session.setAttribute("openid",
authSuccess.getIdentity());
            session.setAttribute("openid-claimed",
authSuccess.getClaimed());
            Filter filter = new Filter();
            if (filter.get(authSuccess.getIdentity()) == null)
                response.sendRedirect("form.jsp");
            else
                response.sendRedirect("laptops.jsp?merchant=all");
        }
        else
        {
%>
            Failed to login!
<%
        }
    }
    catch (OpenIDException e)
    {
%>
        Login error!
<%
        // present error to the user
    }

%>

Form.jsp:

<html>

```

```

<head>
<style type="text/css">
body {
    font-size:90%;
}
#formdiv {
width:600px;
margin:0 auto 10px;
background: #f1f1f1;
border: solid #00688B 2px;
-moz-border-radius: 15px;
-webkit-border-radius: 15px;
}
</style>
<script type="text/javascript">
function validateSubmit() {
    if      ((document.myform.priceLow.value      <      0)      ||
(document.myform.priceHigh.value < 0))
        alert("Please provide non-negative values for price
range");
    else      if      (document.myform.priceLow.value      >
document.myform.priceHigh.value)
        alert("Please check the price range");
    else document.myform.submit();
}
</script>
</head>
<body>
<center>
<h2>Product Search (Laptops)</h2>
<div id="formdiv">
<br/>
Please choose your filter settings:<br/><br/>
<form      name="myform"      action="laptops.jsp?new=true&merchant=all"
onsubmit="javascript: validateSubmit();" method="post">

Brand:
<select name="brand">
<option value="Any">Any</option>
<option value="Apple">Apple</option>
<option value="Hewlett-Packard">HP</option>
<option value="Sony" selected>Sony</option>
<option value="Asus">Asus</option>
<option value="Toshiba">Toshiba</option>
</select>
<br/><br/>
Hard Disk Capacity (atleast): <input type="text" name="hardDiskLow"
size=5 maxlength=5> GB<br/><br/>
Memory Capacity (atleast): <input type="text" name="memoryLow" size=5
maxlength=5> MB<br/><br/>
Display Size (atleast): <input type="text" name="displayLow" size=2
maxlength=2> inches<br/><br/>
Weight (atmost): <input type="text" name="weightHigh" size=2
maxlength=2> lbs<br/><br/>
CPU Speed (atleast): <input type="text" name="cpuLow" size=5
maxlength=5> GHz<br/><br/>

```



```
<%@ page import="SemWebMarket.FilterBean" %>
<%@ page import="SemWebMarket.ProductBean" %>
<jsp:useBean          id="filterBean"           class="SemWebMarket.FilterBean"
scope="request">
<jsp:setProperty name="filterBean" property="*" />
</jsp:useBean>
<html>
<head>
<style type='text/css'><!--
body {width:80%; margin-left:10%}
#toplinks {font-size:80%; }
#frame {background-color:#caelff; padding: 10px 10px 10px 10px;}
#old-content    {background-color:#f1f1f1;   border-bottom:1px      solid
#DCD7B9;padding: 5px 10px 5px 10px;}
#new-content    {background-color:#ffffff;   border-bottom:1px      solid
#DCD7B9;padding: 5px 10px 5px 10px;}
#title {font:16px Arial bold;}
#catalog {font:13px Arial; border: 1px solid #DCD7B9; width:500px;
margin: 10px 10px 10px 10px; padding: 10px 10px 10px 10px;}
#offerbox {padding: 10px 10px 10px 10px;}
.price {font:14px Tahoma;}
.date {font:11px Lucida Grande,Tahoma;}
#tabs {clear:both; height:30px; border-bottom:1px solid #DCD7B9;
margin:0 0 0 0;padding: 0 0 0 20px;text-decoration:none,list-style-
type:none;}
.selected { color:#00688B; background: #caelff; font:11px Lucida
Grande,Tahoma bold; margin: 0 5px 0 5px; padding:9px 10px; float:left;
text-decoration:none;font-weight: bold}
.unselected { color:#00688B; background: #f1f1f1; font:11px Lucida
Grande,Tahoma bold; margin: 0 5px 0 5px; padding:9px 10px; float:left;
text-decoration:none;font-weight: bold}
--></style>
</head>
<body>
<div id="toplinks">
Welcome back!&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
<a href="laptops.jsp?markviewed=true&merchant=amazon">Mark all as Viewed</a>&nbsp;&nbsp;&nbsp;&nbsp;&~
<a href="form.jsp">Modify Preferences</a>&nbsp;&nbsp;&nbsp;&~
<a href="index.jsp?logout=true">Logout</a>
</div>
<br>
<div id="tabs">
<%
if (request.getParameter("merchant").equals("all")) {
%>
<a class="selected" href="laptops.jsp?merchant=all">All</a>
<a class="unselected" href="laptops.jsp?merchant=amazon">Amazon</a>
<a class="unselected" href="laptops.jsp?merchant=yahoo">Yahoo Shopping</a>
<a class="unselected" href="laptops.jsp?merchant=buy">Buy.com</a>
<%
        } else if (request.getParameter("merchant").equals("amazon")) {
%>
<a class="unselected" href="laptops.jsp?merchant=all">All</a>
<a class="selected" href="laptops.jsp?merchant=amazon">Amazon</a>
```



```

<a      class="unselected"      href="laptops.jsp?merchant=yahoo">Yahoo
Shopping</a>
<a class="unselected" href="laptops.jsp?merchant=buy">Buy.com</a>
<%
    } else if (request.getParameter("merchant").equals("yahoo")) {
%>
<a class="unselected" href="laptops.jsp?merchant=all">All</a>
<a class="unselected" href="laptops.jsp?merchant=amazon">Amazon</a>
<a      class="selected"      href="laptops.jsp?merchant=yahoo">Yahoo
Shopping</a>
<a class="unselected" href="laptops.jsp?merchant=buy">Buy.com</a>
<%
    } else {
%>
<a class="unselected" href="laptops.jsp?merchant=all">All</a>
<a class="unselected" href="laptops.jsp?merchant=amazon">Amazon</a>
<a      class="unselected"      href="laptops.jsp?merchant=yahoo">Yahoo
Shopping</a>
<a class="selected" href="laptops.jsp?merchant=buy">Buy.com</a>
<%
    }
%>
</div>

<div id="frame">
<%
String openId = (String) session.getAttribute("openid");
if (openId == null) openId = "Guest";
if (request.getParameter("markviewed") != null) {
    Filter.setLastViewed(openId);
}

String merchant;
if (request.getParameter("merchant").equals("all"))
    merchant = "All";
else if (request.getParameter("merchant").equals("amazon"))
    merchant = "Amazon";
else if (request.getParameter("merchant").equals("yahoo"))
    merchant = "Yahoo";
else merchant = "Buy.com";
List<ProductBean> productList;
if (request.getParameter("new") != null) {
    productList = filterBean.processNew(openId, merchant);
} else {
    productList = filterBean.processOld(openId, merchant);
}
for (Iterator itr = productList.iterator(); itr.hasNext(); ) {
    ProductBean product = (ProductBean)itr.next();
    String publishedDate = product.getDate();
    String lastViewDate = Filter.getLastViewed(openId);
    if (lastViewDate.equals("")) {
%>
<div id="new-content">
<%
        } else {
            if (publishedDate.compareTo(lastViewDate) > 0) {
%>

```

```

<div id="new-content">
<%
                } else {
%>
<div id="old-content">
<%
                }
%>
<table>
<tr>

<td width=30%>
<div id="title"><%= product.getBrand() %>&nbsp; <%= product.getModel()
%></div>
<div id="offerbox">
<span class="price">
Offer:
<%
if (product.getMerchant().equals("Amazon")) {
%>
    <a href="<%= product.getPageLink() %>">$<%= product.getPrice()
%></a><br/>
<%
} else {
%>
    <a href="<%= product.getOfferLink() %>">$<%= product.getPrice()
%></a><br/>
<%
}
if (request.getParameter("merchant").equals("all")) {
%>
Merchant:&nbsp;<%= product.getMerchant() %><br/>
<%
    }
long dateInLong = Long.parseLong(product.getDate());
Date date = new Date(dateInLong);
DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
String displayDate = df.format(date);
%>
Date:&nbsp;<%= displayDate %></span>
</div>
</td>
<td>
<div id="catalog">
<b>Catalog data from Amazon:</b><br/><br/>
<%= product.getAttributes() %>
</div>
</div>
</td>
</tr></table>
</div>

<%
}
%>
</div>

```

```
</body>  
</html>
```

REFERENCES

1. Brunner J., Ma L., et al, Explorations in the Use of Semantic Web Technologies for Product Information Management, Proceedings of the 16th international conference on World Wide Web WWW '07, May 2007.
2. Guo J., Sun C., Context representation of product data, ACM SIGEcom Exchanges, Vol. 4, No. 1, May 23, 2003.
3. Karacapilidis N., Moraitis P., Proceedings of the fifth international conference on Autonomous agents, May 2001
4. Rinard M. C., Lam M. S., The design, implementation and evaluation of Jade, ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 20 Issue 3, May 1998.
5. Rumelhart, D.E. and Zipser, D. Feature discovery by competitive learning. Cognitive Science, 9. 75-112.
6. Singh R., Iyer L., Salam A.F., The Semantic E-business vision, Communications of the ACM, December 2005.
7. Sloman, S.A. Two systems of Reasoning. in Gilovich, T., Griffin, D. and Kahneman, D. eds. Heuristics and Biases: The Psychology of Intuitive Judgment, Cambridge University Press, Cambridge, UK, 2002, 379-396.
8. SPARQL Query language for RDF, <http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050419>
9. Wan Y., Menon S., Ramaprasad A., A classification of product comparison agents, Proceedings of the 5th international conference on Electronic commerce, September 2003.
10. Web Ontology Language overview, <http://www.w3.org/TR/owl-features/>
11. Wilkinson K., Sayers C., Kuno H., Reynolds D., Efficient RDF storage and retrieval in Jena2, HP Laboratories
12. Zeng Z., Meng B., A Multi-Agent Based Intelligent System for Internet Shopping, Wuhan University, China