Recommendation Techniques to enhance GUI of the Semantic Browser

Juan Esteban Maya Alvarez

Matriculation No: 20729239

maya.juan@googlemail.com

STS TU-HAMBURG

Supervised by: Prof. Dr. Ralf Möller

Contents

List of Figures

List of Tables

Abstract

The aim of this project is to research the utility of collaborative filtering techniques to improve the user interaction with the Semantic Browser. The practical part of the project describes the actual implementation of a recommender system engine, which recommends BOEMIE Concepts and/or Services (BOEMIE commands) based on the history of the user interaction. The collaborative filtering algorithm is built upon the assumption that the users with similar interaction will have a similar navigation taste in the future.

Chapter 1

Theorical Background

This chapter introduces the concepts of recommender systems, collaborative filtering and its utilization in the world wide web.

It is impossible to deny the boom that the Internet and the World Wide Web have had over the last decade, the amount of information extracted from Internet has increased in unimaginable proportions. Society has become a consumer of Internet as a source of information. However, searching through the vast amount of information has become a true challenge for everyone. There is too much information to make relevant decision or remain informed about a certain topic. Companies like Google and Yahoo, aware of this problematic, have developed solutions making use of Information Retrieval techniques. Information Retrieval, defined as the science of searching for information in documents, searching for documents themselves, searching for meta-data that describe documents, searching through databases, whether relational standalone databases or hypertext networked databases such as Intenet or Intranets, for text, sound, images or data, has become a billionaire business.

Through the years the research on Information Retrieval has developed several technology-based solutions, addressing information overload, like: Intelligent Agents, ranking algorithms, cluster analysis, web/data mining, personalization, recommender system and others. This project focuses specifically on recommender systems using collaborative filtering techniques as an application of Information Retrieval.

1.1 The Long Tail

The term Long Tail refers to the distribution and inventory costs of a business that have a significant profit by selling small volumes of items that are not popular, instead of selling large volumes of a reduced number of popular items. The Long Tail is identified by the group of people that buy a large amount of non-popular items.

Given a large enough availability of items and a large population of customers, the buying pattern of the population results in a Pareto Distribution as shown in the Figure 1.1. This suggests that a market with a high freedom of choice will create a certain degree of inequality by favoring the upper 20% of the items (head) against the other 80% (long tail).

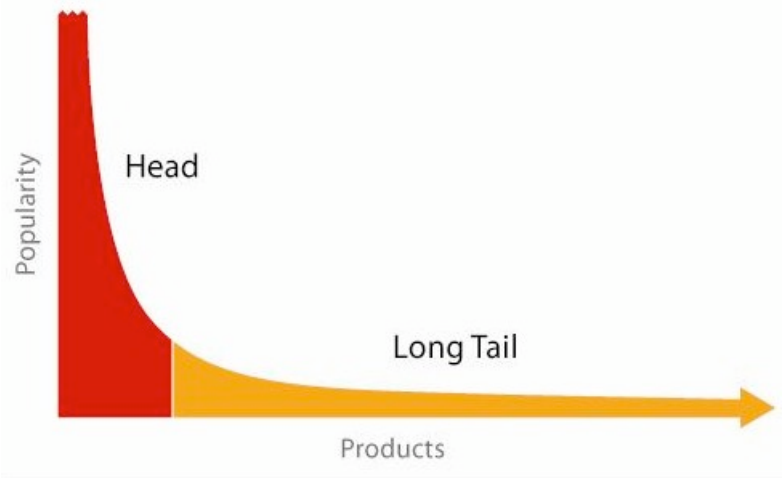

Figure 1.1: The Long Tail

The Long Tail has been used to create successful business models. Before a Long Tail works, only the most popular products are generally offered. When the cost of inventory storage and distribution fall, a wide range of products become available. This can have the effect of reducing demand for the most popular products. For example, Web Content businesses with broad coverage, such as Yahoo, may be threatened by the rise of smaller Web sites that focus on niches of content, and cover that content better than the larger sites. Some of the most successful Internet businesses have leveraged the Long Tail as part of their businesses. Examples include eBay, Yahoo, Google and Amazon among others.

Statistically the Long Tail is the name for a feature of some statistical distributions, also known as

heavy tails, power-law tails or Pareto tails. In "long-tailed" distributions a high-frequency population is followed by a low-frequency population which gradually tails off. The events at the far end of the tail have a very low probability of occurrence. As a rule of thumb, for such population distributions the majority of occurrences are accounted for by the first 20% of items in the distribution. What is unusual about a long-tailed distribution is that the most frequently-occurring 20% of items represent less than 50% of occurrences; or in other words, the least-frequently-occurring 80% of items are more important as a proportion of the total population.

In this project the Long Tail concept is not used as a business strategy but as a way to improve the user interaction with a system. As explained later in this chapter, a recommender system exploits the Long Tail to present relevant items to a user according to its profile. If instead, only the the most popular items were recommended, the items in the Long Tail would never be used, hiding some items that could be relevant for a particular user.

1.2 recommender systems

A recommender system is a type of Information filtering system whose objective is to remove redundant or unwanted information, reducing the information overload to the user and incrementing the semantic signal-to-noise-ratio. A recommender system attempts to predict information (items) that an user may be interested in. Typically, the predictions are done by taking into account information available on the user's profile.

In essence a recommendation engine is a matching engine that takes into account the context where the items are being shown and to whom they are being shown. Typically the following inputs could be taken into account to make a recommendation to an user: (Figure 1.2).

- **The user's profile -** Age, gender, geographical location, etc.
- **Information about the items available -** Name of the item, description, value, etc
- **Interactions of the user** - Ratings, tags, navigation, clicks, page views
- **The context where the items will be shown** - Subcategories of items to be considered.

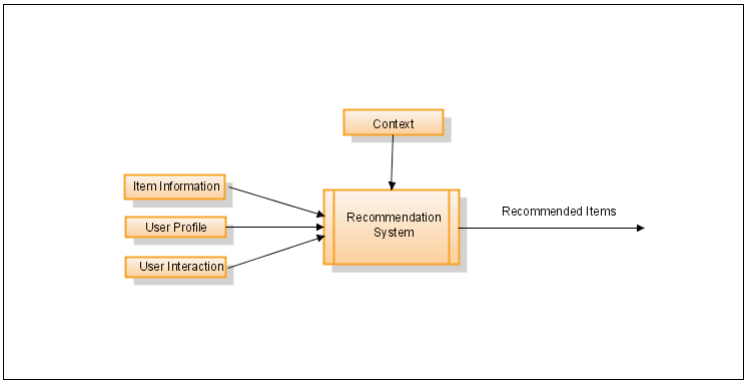

Figure 1.2: Recommendation process inputs

It's important to notice that in some cases an user could also behave as an item, therefore it could be recommended to other users. This is the case in the now popular social networking sites like Facebook, Myspace, Linkdl, Xing.

One of the easiest types of recommendations is to build a "Top item list", on which the items that have been viewed, clicked or bought the most in a period of time are shown to the user. Although showing this could be useful, the objective of a recommendation engine is to create a personalized list of recommendations to the user.

There are two main approaches for building recommender systems, based on whether the system searches for related items or related users. In item-based analysis, as shown in the Figure 1.3, the objective is to find the items related to an item. When an user likes a particular item, items related to that item are recommended to the user.

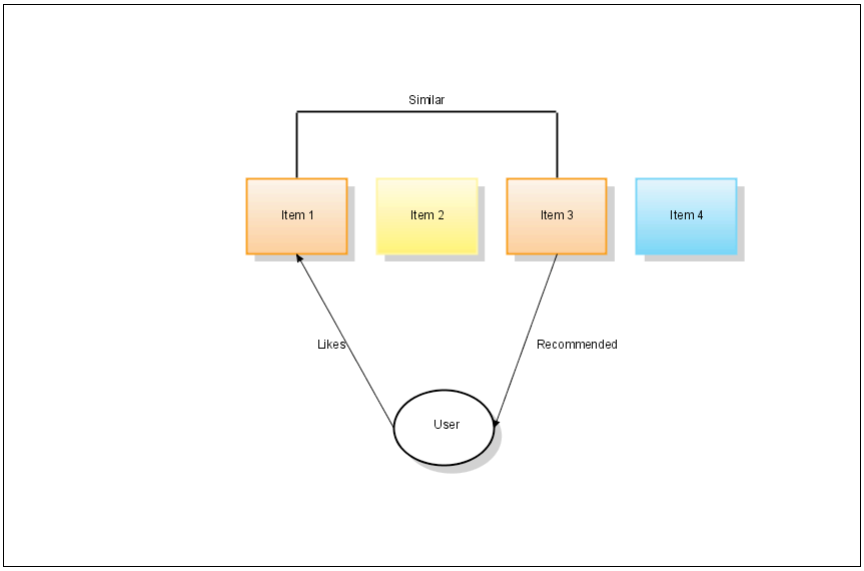

Figure 1.3: Item-Based Analysis

In user-based analysis, similar users are first clustered. As shown in Figure 1.4, if an user U likes an Item, then this Item can be recommended to users who are similar to user U. Similarity between users can be found using their profile information or using collaborative filtering techniques based on the interactions of the users with the system.

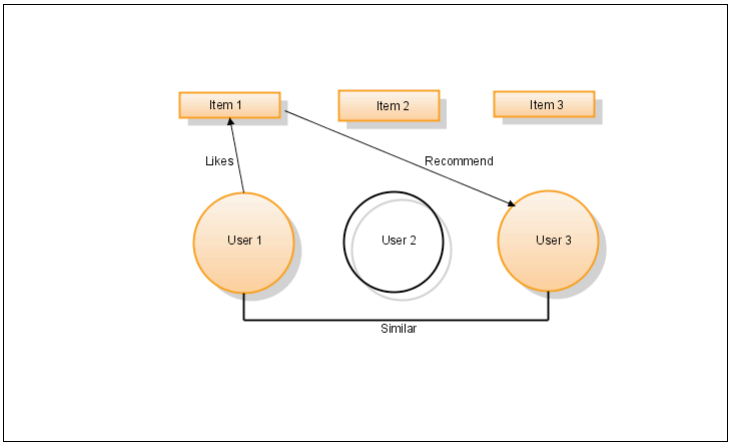

Figure 1.4: User-Based Analysis

1.3 Converting Interaction into Intelligence

In recent years, the increasing amount of Internet applications and its complexity have transformed the way users interact with applications. Users rate items, create a blog entries, tag items, create connections with other users, share interest items, etc. That information behind the users' interaction can be be collected, exploited and later used to understand the users' behavior. Typically this information is ignored by most of the applications. System owners in many cases are not aware that this information could help to create users' profiles that later could be used to improve the overall user experience.

1.3.1 Collaborative Filtering

The main idea of collaborative filtering is to select and recommend items of interest for a particular user based on the interaction of other users with the system. This technique relies on the assumption that a good way to find interesting content is to identify other users with similar interests and then select or recommend items that those similar people like. Someone browsing an e-commerce web site might not always have a specific request and so it is the web site's interest to provide compelling recommendations to get the user attention.

In the typical Collaborative filtering scenario, there is a list of m users: $U = \{u_1, u_2, ..., u_m\}$ and a list of n items: $I = \{i_1, i_2, ..., i_n\}$. Each user u_i has a list of items I_{u_i}, which the user has rated. It's important to notice that $I_{u_i} \subseteq I$ and it's possible for I_{u_i} to be a null set. The task of a collaborative filtering algorithm is to find the item likeliness for an active user a. The likeliness can be of two forms:

- **Prediction -** It's a numerical value $P_{a,j}$, that express the likeliness of item i_j. The predicted value is within the same scale as the ratings provided by the user.
- **Recommendation -** It's a list of N items, $I_r \subset I$, that the user a will like the most. This is known as the Top-N recommendations.

In collaborative-filtering, contrary to *Content-Based* Approaches, an item is considered a black box. The content of the item doesn't matter and the user interactions with the item are collected to build the intelligence needed to recommend an item of interest to the user. There are two main types of collaborative filtering algorithms: *Memory based* and *Model based*. In memory-based algorithms, the entire user-item database is used. The algorithm first finds a set of similar users and then generates a set of recommendations (Top N recommendations) by combining the preferences of the similar users. Model-based algorithms, used during the implementation of this project, try to model the user based on past ratings and then use the models to predict the ratings on items the user hasn't visited or rated.

1.3.2 Comparison of content-based and collaborative techniques

The following are some of the advantages and disadvantages of collaborative and content-based techniques.

- Collaborative-based techniques have the advantage that they treat an item as a black-box. They don't use any information about the content itself. Therefore, the same infrastructure is applicable across domains and languages.
- In content-based analysis, the algorithm has no notion of the item's quality. It's all based in the term vector. On the other hand, with collaborative-based approaches, there is usable quantitative information about the quality of an item.
- The results of a content-based approach don't change much over time; text associated with the item may not change much. Contrary to collaborative filtering that relies on user interaction.
- Collaborative-based systems rely on using the information provided by the interaction of users with the system. In the absence of an adequate amount of data, these systems can perform poorly in their predictions. For a user with little interaction history, there may not be enough information to find similar users using the user's interaction history. Typically to overcome this, user-profile information like age, gender, demographics is used.
- Collaborative-based system won't recommend new items added to the system unless they have been rated by a substantial amount of users.

Some recommender systems use a hybrid approach, combining content-based and collaborative analysis. The combination could be done by implementing the two methods separately and the combining the results.

1.4 BOEMIE

BOEMIE (Bootstrapping Ontology Evolution with Multimedia Information Extraction) is a project, partially funded by the European Commission, which has started in March,1 2006. The main BOEMIE objective is to automate the process of knowledge acquisition from multimedia content, by introducing the notion of evolving multimedia ontologies, which is used for the extraction of information from multimedia content in networked sources.

BOEMIE approach combines multimedia extraction and ontology evolution in a bootstrapping process involving, on the one hand, the continuous extraction of semantic information from multimedia content

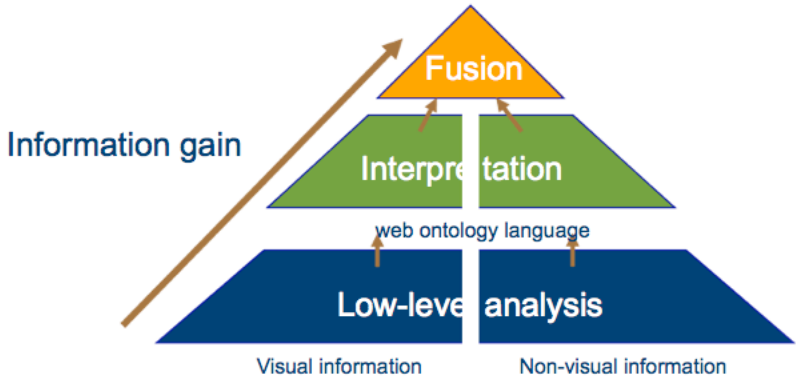

Figure 1.5: Boemie Semantic Browser

in order to populate and enrich the ontologies and the deployment of these ontologies to enhance the robustness of the extraction system.

1.4.1 Boemie Semantic Browser

The purposes of the Boemie Semantic Browser is to demonstrate the extra value that BOEMIE technologies provide for semantic web applications. It exploits different levels of automatic information extraction, media interpretation and multimedia fusion.

The Boemie Semantic browser was built to complain with the following uses cases:

- **Geographically-aware IR -** Demonstrates that semantic tagging of map information can be extended to cover multiple types of media, e.g. video, image, text. (Without the need of online communities)
- **Content activation -** Demonstrates the use of automatic information extraction to highlight relevant content of a specific domain on top of text or images to prepare the interface for further interaction possibilities.)
- **Dynamic suggestion of related information -** Exploits explicit and implicit information to provide for context related information.

The recommender system for this project improves the last two use cases. In the first case, due to the big amount of highlighted concepts the user interface is polluted by noise that at the end affects the user experience.

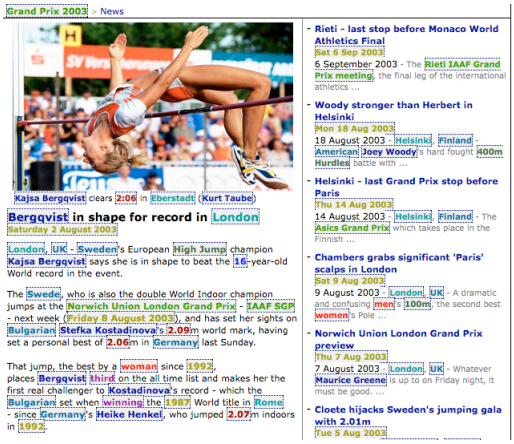

Figure 1.6: Concept highligthing in the Semantic Browser

The Image 1.4.1demonstrates the first problem. The Semantic Browser highlights too many concepts to the users. To solve this problem the recommender system will recommend concepts, highlighting only the top K concepts that are relevant based on the user's navigation history.

The second problem comes with the services that the semantic browser presents for each concept. Each concepts has associated a list of services that can be executed by the user. When the list of services is too big, the user's experience is again affected by overwhelming amount of choices. In this case, the recommender system will present the services that the user will more likely like to use. The problem becomes clear in the Image 1.4.1.

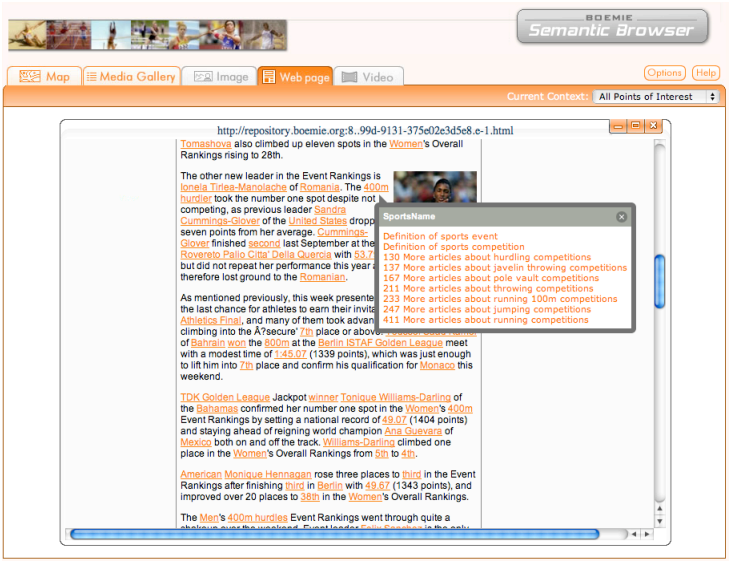

Figure 1.7: Services of a concept in the Semantic Browser

1.5 Outline

The rest of the project is organized in 6 chapters. Chapter 2 presents the basic algorithms used for collaborative filtering. Chapter 3 and 4 presents the System Architecture and implementation of the recommender system for the Boemie Semantic Browser. Chapter 5 presents the results of using collaborative filtering techniques to enhance the Semantic Browser GUI. Finally, chapter 6 summarizes the work done in this project.

Chapter 2

Algorithms

This chapter focuses on the main algorithms for collaborative filtering.

As mentioned before there exist two approaches for collaborative filtering:

- **Memory-Based Algorithms -** Operate online over the entire matrix to make predictions
- **Model-Based Algorithms** - Use matrix offline to estimate a model that is then used for predictions.

This chapter introduces the most popular collaborative filtering algorithms. First, the the main algorithms used for collaborative filtering are introduced: *User-Based Algorithm* and the *Item-Based Algorithm.* three algorithms to compute similarity are explained: *Cosine-Based Similarity*, *Correlation-Based Similarity* and *Adjusted-Cosine Based similarity.* At the end of the chapter *Singular Value Decomposition (SVD)* is introduced as a matrix dimension reduction technique.

2.1 User-Based Algorithm

The main idea of this approach is to predict the ratings of an user a; the user database is first searched for users with similar rating vectors to user a (users with similar taste). The rating similarity, $w(i, a)$, can reflect the similarity between user i and user a. The ratings of the n most similar neighbors are then used as predictions for the user.

The predicted ratings $P_{a,j}$ of the active user a for another user j, is assumed to be a weighted sum of the ratings for user j of those n most similar neighbors, as shown in the equation:

$$P_{a,j} = \overline{r_a} + \xi \sum \omega(a, n_i)(r_{n_i,j,} - \overline{r_{n_i}})$$

Where n_i is the i-th nearest neighbor with a non-zero similarity to the user and a non-empty rating r_{n_i} for user j; ξ represents a normalizing factor such that the absolute values of the similarities $\omega_{(a,n_i)}$ sum to unity. If R_u is a set of users which the user u has rated, then the mean rating of the user u is defined as:

$$\overline{r_u} = \frac{1}{|R_u|} \sum_{v \epsilon R_u} r_{u,v}$$

2.2 Item-Based Algorithm

The item-based algorithm, instead of utilizing the ratings matrix rows similarities, utilizes the rating matrix columns similarities. When predicting a $P_{a,j}$ of the user a, the user database is first searched for items with similar ratings vectors to the item j. The items' scores similarity $\varpi(i, j)$ is calculated in the same manner as for the User-Based algorithm. The rating of an user a for the most n most similar neighbors to j are then use to form the prediction $P_{a,j}$, as shown by the formula:

$$P_{a,j} = \overline{S_j} + \xi \sum_{i=1}^{k} \varpi(j, n_i)(r_{a,n_i} - \overline{S_{n_i}})$$

Where n_i, is the i-the nearest neighbor with a non-zero similarity to the target user j and a non-empty rating r_{a,n_i} from user a; ξ is a normalizing factor such that the absolute values of the similarities $\varpi(j, n_i)$ sum one, and S_u is the mean score for the user a.

2.3 Computing Similarity

Most collaborative filtering algorithms based on the nearest neighbor approach required the calculations of similarity. To find this similarity, the content associated with an user or item must be represented in a term vector.

In Content based analysis, recommending items simply amounts to finding items that have similar term vectors. In collaborative filtering each user is represented as an N-dimensional vectors of items, where N is the number of distinct items in the system. It is important to notice that the vector will be sparse for almost all users, where a typical user would have acted on only a few item of the N items.

The method used for computing the similarity does have an effect on the final recommendations presented to the user. There are 3 main algorithms used to find the similarity between 2 vectors. It is easier to explain how they work with an example:

The following User-Item Matrix will be used to find the similarity between the users.

	Item 1	Item 2	Item 3	Average
User 1	3	4	2	3
User 2	2	2	4	2.67
User 3	1	3	5	3
Average	2	3	3.67	8.67

Table 2.1: Example Data

2.3.1 Cosine-Based similarty Computation

Cosine-based similarity takes the dot product of two vectors as a measure of similarity. Formally, if R is the $n \times m$ user-item matrix, then the similarity between two items i and j is defined as the cosine of the n dimensional vectors corresponding to the *ith* and *jth* column of matrix R. The cosine between these vectors is given by

$$sim(i,j) = cos(\overrightarrow{i}, \overrightarrow{j}) = \frac{\overrightarrow{i} \cdot \overrightarrow{j}}{\left\|\overrightarrow{i}\right\|_2 * \left\|\overrightarrow{j}\right\|_2}$$

where "$\cdot$" denotes the dot-product operation.

One important feature of the cosine-based similarity is that takes into account the acting frequency of the different items (achieved by the denominator in the formula).

	User 1	User 2	User 3	Average
Item 1	3	2	1	2
Item 2	4	2	3	3
Item 3	2	4	5	3.67
Averate	3	2.67	3	8.67

Table 2.2: Dataset to describe Items

To learn about the items, the user-item matrix is transposed, so that a row corresponds to an item while the columns (users) to dimensions that describe the item as show in Table 2.2. Next, the values for each of the rows are normalized. This is done by dividing each of the cell entries by the square root of the sum of the squares of entries in a particular row.

For example, to get the normalized dataset for item 1 show in the Table 2.3, the following value is used:

$$\sqrt{3^2 + 2^2 + 1^2} = \sqrt{14} = 3.74$$

	User 1	User 3	User 3
Item 1	0.80	0.53	0.27
Item 2	0.74	0.37	0.56
Item 3	0.30	0.60	0.75

Table 2.3: Normalized vectors for each item

We can find the similarities between the items by taking the dot product of their vectors. For example, the similarity between *Item 1* and *Item 2* is computed as: $(0.80*0.74)+(0.53*0.37)+(0.27*0.56) = 0.94$

Repeating this operation, the Item-to-Item similarity matrix, shown in the Table 2.4, can be constructed. The values in the table allow to find the most related items for any given item. The closer to 1 a value in the similarity table is, the more similar the items are to each other. For example: *Item 1* and *Item 2* are very similar.

	Item 1	Item 2	Item 3
Item 1	1	0.94	0.76
Item 2	0.94	1	0.86
Item 3	0.76	0.86	1

Table 2.4: Item-to-Item Matrix

To determine similar users, the original data in the Table 2.1 needs to be considered. Here, associated with each user is a vector, where the rating associated with each item corresponds to a dimension in the vector. The analysis process is similar to the approach for calculating the Item-to-Item similarity table. First the vectors are normalized and the dot product between two normalized vectors is found to compute their similarities.The Table 2.9 contains the normalized vectors associated with each user. The process is similar to the approach take to compute Item-to-Item Matrix (Table 2.4) from Table 2.1. The normalizing factor for the vector of *User 1* is found like this:

$$\sqrt{3^2 + 4^2 + 2^2} = \sqrt{29} = 5.39$$

	Item 1	Item 2	Item 3
User 1	0.56	0.74	0.37
User 2	0.41	0.41	0.82
User 3	0.17	0.51	0.85

Table 2.5: Normalized vectors for each user

Next, a user-to-user similarity matrix can be computed as shown in the Table 2.6 by taking the dot product of the normalized vectors for two users.

	User 1	User 2	User 3
User 1	1	0.83	0.78
User 2	0.83	1	0.97
User 3	0.78	0.97	1

Table 2.6: User-to-User similarity Matrix

As shown in the Table 2.6 *User 1* and *User 2* are very similar. The preceding approach uses the raw values collected from the user. Another alternative is to focus on the deviations in the rating from the average values that user provides.

2.3.2 Correlation-Based Similarity Computation

Similar to the dot product or cosine of two vectors, the correlation between two items, also known as the Pearson-r correlation, can be used as a measure of the similarity. This correlation between two items is a number between -1 and 1, that tells the direction and magnitude of the association between two items or users. The higher the magnitude (closer to -1 or 1) the higher the association between the two items. The direction of the correlation tells how the variables vary. A negative number means one variable increases as the other decreases, or in this case, the interest in one service increases as the other one decreases.

To compute the correlation, it's needed to isolate the cases where the users co-rated items, in the example, it's the complete set of users, as all users have interacted with all the items. Let U be the set of users that have interacted with both item i and j. The formula looks like this:

$$Sim(i,j) = Corr(i,j) = \frac{\sum_{ueU}(R_{ui}-\overline{R_i})(R_{uj}-\overline{R_i})}{\sqrt{\sum_{ueU}(R_{ui}-\overline{R_i})^2}\sqrt{\sum_{ueU}(R_{uj}-\overline{R_j})^2}}$$

Where $R_{u,i}$ is the rating of user u for item i and Ri is the average rating of item i. the correlation computation looks for variances from the mean value for the items. For example to find the correlation between *Item 1* and *Item 2*:

$$Corr(1,2) = \frac{(3-2)(4-3)+(2+2)(2-3)+(1-2)(3-3)}{\sqrt{(3-2)^2+(2-2)^2+(2-2)^2}\sqrt{(4-3)^2+(2-3)^2+(3-3)^2}} = \frac{1}{2} = 0.5$$

It's useful, for the computation, to subtract the average value for a row as shown in the Table 2.11. Note that the sum of each row is zero.

	User 1	User 2	User 3
Item 1	1	0	-1
Item 2	1	-1	0
Item 3	-1.67	0.33	1.33

Table 2.7: Normalized Matrix for the correlation computation

The Table 2.8 show the correlation matrix between the items and allows to find the set of items related to an item. According to the table *Item 1* and *Item 3* are strongly negatively correlated.

	Item 1	Item 2	Item 3
Item 1	1	0.5	-0.98
Item 2	0.5	1	-0.65
Item 3	-0.98	-0.65	1

Table 2.8: Items Correlation Matrix

Similarly, the correlation matrix between the users is computed along the rows of the data shown in the Table 2.1. The Table 2.9 contains the normalized rating vectors for each user that will be used for computer correlation. Note that the sum of the values for each row is 0.

	Item 1	Item 2	Item 3
User 1	0	0.71	-0.71
User 2	-0.41	-0.41	0.81
User 3	-0.71	0	0.71

Table 2.9: Normalized vectors for each user

The resulting correlation matrix is shown in the Table 2.10 and allows to find the users that are similar to an user. Note that *User 2* and *User 3* are highly correlated. If one likes an item, chances are the other likes it too. *User 1* is negatively correlated; He dislikes what *User 2* and *User 3* like.

	User 1	User 2	User 3
User 1	1	-0.87	-0.5
User 2	-0.87	1	0.87
User 3	-0.5	0.87	1

Table 2.10: Users correlation matrix

2.3.3 Adjusted Cosine-Based Similarity

One drawback of computing the correlation between items is that it doesn't take into account the difference in the rating scale between users. For example, in the example data, the *User 3* is highly correlated with *User 2* but tends to give ratings toward the extremes. The adjusted cosine similarity offsets this drawback by subtracting the corresponding user average from each co-rated pair. Formally, the similarity between items i and j using this scheme is given by

$$similarity(i,j) = \frac{\sum_{ueU}(R_{u,i}-\overline{R}_u)(R_{u,j}-\overline{R}_u)}{\sqrt{\sum_{ueU}(R_{u,i}-\overline{R}_u)^2}\sqrt{\sum_{ueU}(R_{u,ij}-\overline{R}_u)^2}}$$

Where $\overline{R_u}$ is the average rating for user u. Here, instead of subtracting the average value for a row, the average value provided by an user is considered.

To compute this, it's again useful to normalize the dataset by removing the average rating from the column values. This leads to the data shown in the Table 2.11. Note that the sum of the entries for a column is equal to zero.

	User 1	User 2	User 3
Item 1	0	-0.67	-2
Item 2	1	-0.67	0
Item 3	-1	1.33	2

Table 2.11: Normalized Matrix for the adjusted cosine-based computation

The Table 2.12 shows the item-to-item similarity for the three items. Again, *Item 1* and *Item 3* are strongly negatively correlated, while *Item 2* and *Item 3* are similar.

	Item 1	Item 2	Item 3
Item 1	1	0.17	-0.80
Item 2	0.18	1	0.60
Item 3	-0.89	0.60	1

Table 2.12: Items similarity using correlation similarity

Similarly, to compute the similarity between users, the average rating associated with each item is subtracted. Table 2.13 shows the resulting table. Note that the sum of the values for a column is equal to zero.

	Item 1	Item 2	Item 3
User 1	1	1	-1.67
User 2	0	-1	0.33
User 3	-1	0	1.33

Table 2.13: Normalized vectors for each user

Again, normalizing each of the vectors to unit length leads to Table 2.14.

	Item 1	Item 2	Item 3
User 1	0.46	0.46	-0.76
User 2	0	-0.95	0.32
User 3	-0.6	0	0.8

Table 2.14: Normalized vectors to unit length

Finally the Table 2.15 contains the similarity matrix between the users by taking the dot product of the vectors.

	User 1	User 2	User 3
User 1	1	-0.67	-0.89
User 2	-0.67	1	-0.25
User 3	-0.88	-0.25	1

Table 2.15: Adjusted cosine similarity matrix for users

2.4 Dimension Reduction

One of the challenges of a recommender system is that most of the users will not interact with all the items of the application, therefore the user-item matrix will be a very sparse matrix. Adding to this, that if the users or/and items of the application grow constantly the user-item matrix could be too large from a computational point of view. For those reasons it may be needed to apply techniques to reduce the dimensionality of the matrix.

The motivation to perform dimension reduction in the user-item matrix are the following:

- If the user-matrix is too large from a computational point of view, an approximation may be more computational feasible.
- If the matrix is noisy, the transformed matrix may be a better approximation of the concepts.

- If the matrix is sparse, transforming the matrix may increase its density.

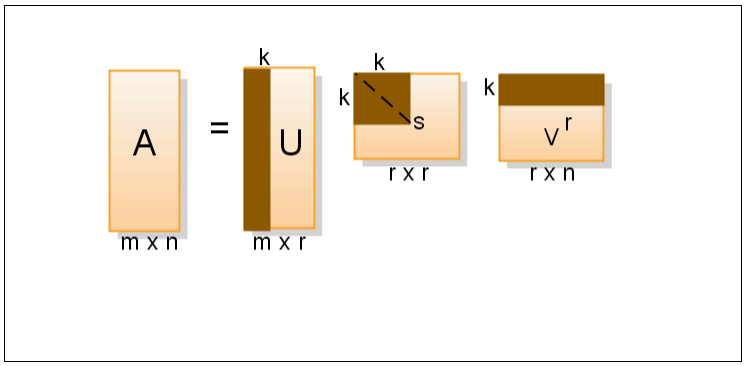

Figure 2.1: Singular Value Decomposition Matrices

2.4.1 Singular Value Decomposition

Singular Value Decomposition (SVD) is a common method to perform matrix dimension reduction. The way SVD transform a matrix is as follow:

An arbitrary matrix A of size m x n can be decomposed into three matrices using SVD. Let r be the rank of A. As show in the Figure 2.1, the matrices are:

- U, an orthogonal square matrix of size m x r.
- S, a diagonal matrix of size r x r, with each diagonal value being the eigen value for the matrix. All values of S are positive and stored top to bottom in decreasing order of magnitude.
- V is an orthogonal square matrix of size n x r.

SVD can be used to reduce the r x r dimensionality of the matrix S to use only the top k singular values. As show in in the Figure 2.1, the matrices U and Vt are reduced in dimensionality for this approximation of A. The lower dimensionality allows to approximate using k singular values.

Chapter 3

Architecture

This chapter introduces the architecture and design of the recommender system for the semantic browser.

The system architecture of a recommender system must be flexible enough so users or developers of the system can focus on the collaborative filtering techniques instead of the system complexity. At the same time, the system must scale well under high traffic. The General view of the architecture of the recommender system for the semantic browser is shown on Image 3.1

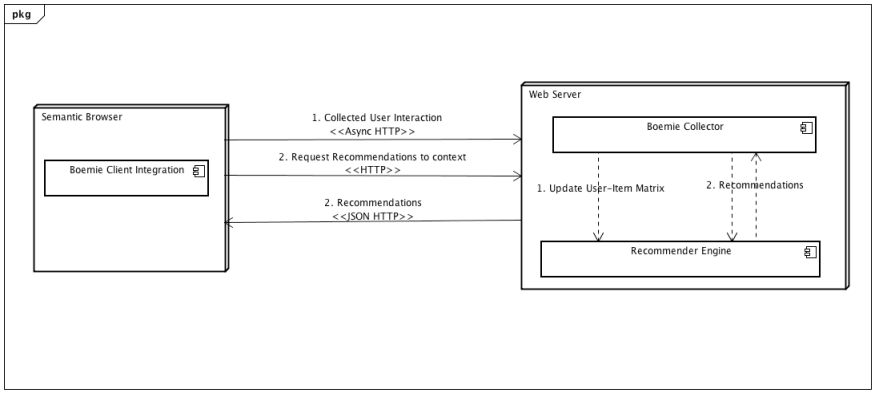

Figure 3.1: Architecture Overiew

The system architecture contains 3 main components: *Recommender Engine, Boemie Collector* and *Boemie Client Integration.*

3.1 Recommender Engine

The Recommender Engine is a generic recommender system that allows to create an Item similarity matrix and provide Top-N recommendations to its users. The component abstracts the type of item that is being recommended. Therefore, It could be used to recommend any kind of item, not only Boemie Concepts or Actions. To increase the scalability, and overall performance of the system, the engine keeps a cache of the recommended items for each user and recreates periodically the Item similarity matrix.

The recommender engine component can run embedded as a library in a web application or as a standalone component in an Application server. The later would warranty better performance and scalability of the system.

3.2 Boemie Collector

The Boemie Collector is the module that allows the semantic browser to interact with the generic recommender engine. It runs as a web application listening for HTTP Requests. Its three main purposes are:

- **Keep track of boemie users -** Identifies the user interacting with the Boemie Browser to keep track of its interaction.
- **Collect User Interaction -** Collects the boemie concepts and services (commands) viewed by the users in the semantic browser.
- **Update User-Item Matrix -** The collector modules updates the User-Item matrix, in the recommender engine, every time a concept or service from the semantic browser is collected.
- **Load Recommendations -** Based in the user and the current context, the component requests recommendations to the recommender engine that are then delivered to the semantic browser via a JSON String over an HTTP connection.

3.3 Boemie Client Integration

The Integration client component provides recommendations functionality to the semantic browser without drastically changing its inner code. Its main purposes are:

- **Collect the user interaction -** This is done by intercepting clicks made to Concepts and Services. The clicked item is sent to the Collector module via an asynchronous call to the *Boemie Concept Collector*. The asynchronous calls warranty that the original user experience of the user is not affected.
- **Request and display recommended items -** Every time the user loads a page, the client component asks the Collector component for recommendations in a particular page. Similarly, when a user wants to access the available actions of a concept, the client component requests recommended items for a specific concept (Context) that are later display to the user.

Chapter 4

Design and Implementation

This chapter presents the details of the design and implementation of the recommender system for the semantic browser.

4.1 Recommender Engine

As explained in the last chapter, the recommender engine is a generic component that allows to create an Item similarity matrix without being aware of the type of item being processed.

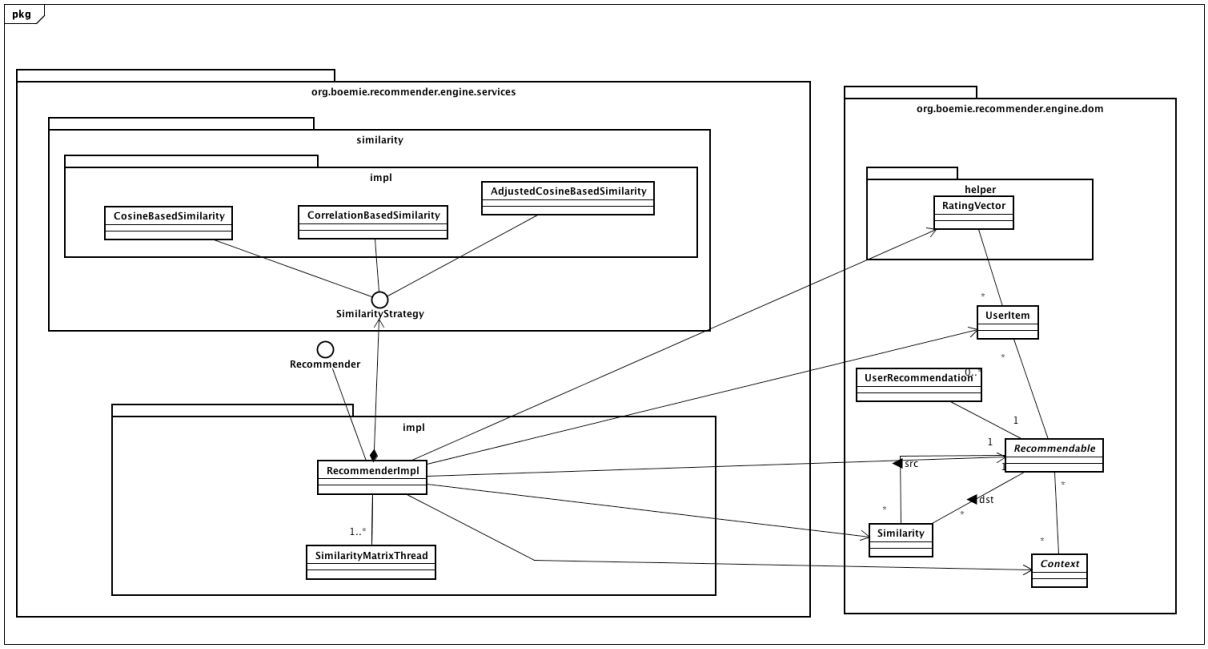

Figure 4.1: Recommender Engine Class Diagram

The class diagram in Figure 4.1 shows the design of the recommender Engine. Its main elements are:

- **Class Recommendable -** It's the base class used by the classes that represent the items that would be recommended by the engine. The component has no knowledge about the items as such.
- **Class UserItem -** Represents an entry in the User-Item Matrix. It holds the rating given by an user to a specific item.
- **Class Context -** Base Class that represents the Context where a recommendable item is been shown.
- **Class Similarity -** Contains the similarity value between two items.
- **Class UserRecommendable -** It works as a cache of an item previously recommended to an user. It helps to improve the scalability and performance of the system by keeping a cache of the recommended items for an user in a specific context.
- **Interface SimilarityStrategy -** Represents the strategy used to find the similarity between 2 items. The recommender system supports the 3 main algorithms to find the similarity between

vectors, each of them is implemented in the classes: *CosineBasedSimilarity, CorrelationBased-Similarity* and *AdjustedCosineBasedSimilarity*. New strategies could be easily added to the system by implementing this interface.

- **SimilarityBuilderThread -** The biggest bottleneck of the application is to build the similarity matrix. The *SimilarityBuilderThread* class allows to split the computation of the matrix in different threads. Each of them takes a set of item vectors to find its similarity. In a production system the threads could be running each of them on different machines giving a bust on performance and scalability, achieving almost real time processing funcionality.
- **Interface Recommender -** This interface exposes the 2 main services of the component to its clients: *addUserItem* and *getRecommendations*. *addUserItem* allows to add new entries to the user-item matrix*; getRecommendations* finds the top-k recommendations for an user in a particular context.

4.1.1 Build Similarity Matrix

The similarity matrix is built periodically by a Job that can be configured to run at any specific time. This is specially useful when the recommender engine is running in the same server as the collector component, because it allows to control at what time the system consumes more resources. The process of building the similarity matrix is done by a number of configurable threads that take a set of items' vectors and find its similarity. The downside of this approach is that using a job doesn't provide real time processing. To achieve such funcionality, an event driven architecture is recommended, where the Similarity Builder Job listens for new entries in the User-Item matrix and finds the similarity in a distributed manner.

Once the Item similarity matrix has been created, the cached recommendations (*UserRecommendable* objects) are erased allowing the recommendation process to discover new recommended items for the users.

4.1.2 Select recommendations

To make recommendations the system needs the following inputs: the user for whom the items are being recommended, the context where the items are been recommended and the total number of items to recommend. The process to find user recommendations is the following:

- **Obtain Recommendation from cache -** The system checks if the client has already asked for recommendations in the context. If found, the recommendations are retrieved from cache.

- **Obtain User Items -** If there are no cached items the system obtains the items that the user has rated in the context.
- **Finds similar items -** Based on the user items, the system selects the top-k similar items in the current context, based on the Item similarity matrix.
- **Save recommendations in cache -** Once the top-k recommended items have been found, they are saved in the user recommendations cache to speed up the recommendation process in later requests.

4.2 Boemie Collector Component

As explained before, the Boemie collector works as a bridge between the recommender engine and the Semantic Browser. The Figure 4.2 show its main classes.

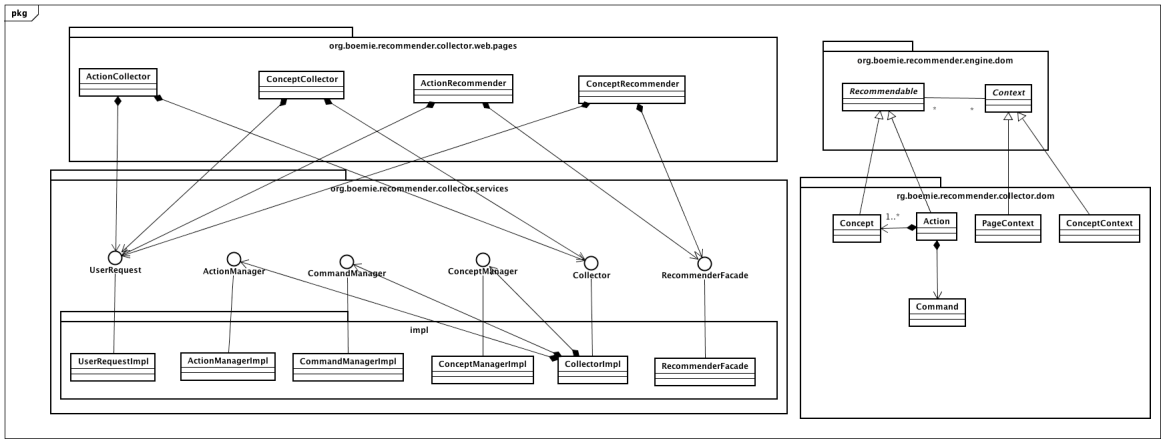

Figure 4.2: Boemie Collector Class Diagram

The main elements of the components are:

- **Collector Actions -** The collector component has 2 controllers that are constantly listening for requests. *ActionCollector* and *ConceptCollector* collect, respectively, Services and Concepts. Both require the item as such (Concept or Service), the user viewing the item and the context where the item was accessed. The context is represented by the classes *PageContext* and *ConceptContext*.

- **Collector Interface -** The collector interface exposes the services to the collector controllers. The responsability of this class is to make the necesarry calls to save the user interaction history, this includes calling the *addUserItem* Service of the *Recommender* Interface defined in the Recommender Engine.
- **RecommenderFacade Interface -** It works as a facade between the collector component and the recommender engine. It simplifies the calls made to the recommender engine.
- **Concept Class -** Represents a Boemie Concept that has been seen by an user and could be recommended. Notice that it extends the base *Recommendable* class.
- **Action Class -** Represents a Boemie Action that has been seen by an user and could be recommended. Notice that it extends the base *Recommendable* class.
- **Manager Classes -** The Manager classes provide CRUD operations to persist information in the database.

4.2.1 Create User Profile

One of the purposes of the collector component is to create an user profile based on the user's interaction history. Currently, the Semantic Browser doesn't have the concept of users. Therefore, the first step to build an user profile is to support users. To add users to the system, the componet uses a cookie based identifier that is created the first time the Semantic Browser makes a request to the boemie-collector server. The user key is created randomly and it's unique for each user. As long as the visitor uses the same browser, the user would be identified by the previously defined key.

Once an identifier has been assigned to the user, the system starts to collect Boemie Concepts and Services. This is done by the controllers *ActionCollector* and *ConceptCollector*. Both controllers receive the name of the Concept or Service and the context where the item was accessed. The collector component creates an instance of the item and adds it to the User-Item matrix via the Recommender service of the Recommender Engine Component. If an User-Item entry already exists the count for the user is increased by one.

It's important to notice that the User-Item vector contains Boemie Concepts and Service; for the recommender system they are just different items that are shown in different Contexts: Components are shown inside a Page and Actions are shown inside a Concept.

4.2.2 Obtain Recommendations

The collector component also allows the Semantic Browser to obtain recommendations. To provide recommendations the user and the context where there recommended items will be shown,are required. With this information the controllers (*ConceptRecommender* and *ActionRecommender*), listening for requests, can make calls to the RecommenderFacade service that asks the recommender engine for recommended items in the given context. The list of recommended items is returned to the server using a JSON String of the form: *{"recommended item 1", "recommended item 2",...,"recommended item N"}*

4.3 Boemie Integration Component

The boemie integration client runs in the user browser and its purpose is to track the clicks to Boemie Concepts and Services. One of the objectives during the design of the Integreation Component was to add the new functionality without modifying the existing code. The only change required is to add a line, on each page, that points to the library with the integration component.

4.3.1 Cross Domain Requests

The pages shown by the Semantic Browser require to make asyncronous calls, that collect and obtain recommendations, to the Boemie Collector component installed in a different domain. Those cross domain calls break the Cross Domain security enforced by the web browsers that specifies that all the Ajax calls made in page must be made to the same domain where the page is hosted. A page on *http://www.boemie.org/* could only make Ajax calls to pages on the domain *www.boemie.org*. To overcome this problem, a library that uses a Cross Domain policy file, originally used by flash objects, on the target server is been used.

The cross domain policy is a file that contains the list of domains that are allowed to make calls to the domain where the file is present. The format looks like this:

<?xml version=\"1.0\"?>

<!DOCTYPE cross-domain-policy SYSTEM \"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd\">

<cross-domain-policy>

<allow-access-from domain="www.boemie.org" />

<allow-http-request-headers-from domain="www.boemie.org" headers="" />*

</cross-domain-policy>

The Cross Domain Policy File of the Boemie Collector component is created dynamically by the *CrossDomainPolicy* Controller. Every time a page is loaded, the server checks if the origin domain is a valid Cross Domain Server and grants or denies the privilege to make Ajax requests to the domain.

4.3.2 Collect User Interaction

The integration component intercepts the clicks made by the users to Boemie Concepts and Services. This is done by registering observers that listen for clicks in any of the page's links. The following is the step by step process to collect the user interaction history:

- **Register link observer -** The first time a page loads, the integration component registers an observer that listens for clicks in the page's links.
- **Identify user action -** Every time a link is clicked, the observer process the user action identifying if the element clicked was a Concept or a Service.
- **Send Collect Notification -** If the element clicked was successfully identified as a Concept or Service, an Ajax notification is sent to the Boemie Collector component to process the item. The notification is asynchronous via Ajax, therefore it doesn't affect the user ineraction with the semantic browser.

4.3.3 Show Recommendations

The process to recommend concepts is as follow:

- **Request recommendations -** The first time a page loads the integration component requests for concept recommendations to the *ConceptRecommender* controller.
- **Process response -** The *ConceptRecommender* controller returns a JSON String with the list of recommended concepts for the current page and user. The integration component processes the response and identifies the html elements in the page that contain the recommended concepts.
- **Show recommendations -** If concepts to recommend were found, the integration component disables the highlighting of those concepts that are not recommended. The user can still toggle the view to display all the components, by clicking a link on the bottom of the page or by pressing *alt + t*. If there were not recommended concepts found the component shows all the originally highlighted concepts.

The process to recommend actions works similar to the concept recommendation process:

- **Request recommendations -** The integration component request for Services recommendations to the *ActionRecommender* every time a concept inside a page is clicked.
- **Process Response -** The *ActionRecommender* retuns a JSON String with the list of recommended services. The integration component process the response to identify the recommended actions.
- **Display Recommendation -** If there are recommended Services, the integration component displays a menu that contains only those recommended items to the user. the original menu is shown if there are no recommended items. The user can always toggle the menu to see the list of all the available Services.

4.4 Technologies

The server side components, *Recommender Engine* and *Boemie Collector*, are written using Java 6 standard edition. The use of java warranties portability between multiple application servers. For development purposes, *jetty*, a lightweight Servlet container was used. *Derby* was chosen as relational database because of its easy deployment process, however the server side components can be used in top of any database without making any changes to the code. The integration component was written with JavaScript and runs in the users' browser.

4.4.1 Frameworks and Libraries

During the development of the project some libraries and frameworks were used to connect the components between them.

Hibernate - Hibernate was used as Object Relational Mapping technology because it warranties the interoperability between databases and leverage the complexities to persist java objects in a database.

Apache Tapestry 5 (MVC) - Tapestry 5 is a robust MVC framework built upon the standard Java Servlet API. It was used to build the controllers on the Boemie Collector component that listen for Concepts and Services, as well as the controllers that recommend Items to the Semantic browser.

Apache Tapestry 5 (IOC) - The IoC (Inversion of Control) module of Tapestry 5 allowed to design the system from many small, easily testable pieces. By breaking the system into small pieces, it became

easier to modify and extend the system.

Javascript Prototype - Prototype is a JavaScript framework that aims to ease development of dynamic web applications. It features a toolkit for class-driven development and Ajax development. Prototype is vastly used by the Semantic Browser so it doesn't add more code base to its code.

Javascript flXHR (flex-er) - Flex-er is a client-based cross-browser, XHR-compatible tool for cross-domain Ajax (Flash) communication. It utilizes an invisible flXHR.swf instance that acts as a client-side proxy for requests, combined with a Javascript object/module wrapper that exposes an identical interface to the native XMLHttpRequest (XHR) browser object. Flexer brings cross-domain Ajax and API-consistency to any browser with Javascript and Flash Player plug-in v9+ support.

Apache Maven - Maven is a software tool for Java project management and build automation. Maven dynamically downloads the project dependencies from Internet helping to distribute the application and install dependencies during development.

4.5 Deployment and Configuration

4.5.1 Requirements

The only requirements to deploy the server side components are:

- Java 1.6 `http://java.sun.com`
- Apache Maven `http://maven.apache.org`
- Derby or any relational database. `http://db.apache.org/derby/`

4.5.2 Deployment

The server side components can be deployed in any java complaint Servlet container or Application server. There are 2 different ways to deploy the application:

- **Development and testing mode -** In the root folder of the recommender engine, *<workspace>/boemie-recommender/recommender-engine,* run the maven command*: mvn install.* The first time the command is executed, maven will download all the required dependencies of the project. This

may take a few minutes but simplifies the libraries distribution process. Once the process finishes, the embedded jetty Servlet container can be executed. To run the embedder jetty server go to: *<workspace>/boemie-recommender/boemie-collector/* and run the command: *mvn jetty:run*. Again, maven will download the required libraries to execute jetty. Once the process finishes the application will be available at *http://localhost:8080/boemie-collector*.

- **Production mode -** The first step is to create a *Web Application Archive* (war) that contains the complete application. To do this, run the command: *mvn install* in the folder: *<workspace>/boemie-recommender*. Maven will download the required dependecies and generate the war file in*: <workspace>/boemie-recommender/boemie-collector/target/boemie-collector.war*. The installation of the war in a Servlet container or Application server differs depending of the server of choice. The process normally consist in just dropping the war file in a specific directory.

The deployment process requires the successful configuration of a persistence database. Out of the box, the server side components are configured to use a Derby Database in network mode, running in the same machine as the servlet container. The file *hibernate.cfg.xml* allows to modify the database to be used. The database structure required by the system is created, automatically, the first time the server starts.

4.5.3 Configuration

Both server side components are configured using java properties files. The behaviour of the recommender engine can be modified via the *recommender.properties* file. The following properties can be configured:

- **max-matrix-threads -** Specifies the number of threads used to generate the Item-to-Item similarity matrix. If not specified, the default number of threads is 5.
- **matrix-interval -** Specifies a Cron expression that specifies how often the Item-to-Item similarity matrix is recreated. If not specified, the matrix is created every day at 1 am.
- **similarity-strategy -** Specifies the similarity strategy used to create the Item-to-Item similarity matrix. The possible values are: COSINE, ADJUSTED_COSINE or PEARSON-R. If not specified, the default value is COSINE.

The boemie-collector component can be configured in the *collector.properties* file. The following are the available configuration properties:

- **number-recommended-concepts -** Specifies the number of recommended concepts that will be presented to the users. If not specified, the default value is 10.

- **number-recommended-actions -** Specifies the number of recommend actions that will be presented to the user. If not specified, the default value is 2.

Chapter 5

Analysis of the Semantic Browser GUI

This chapter analysis the Semantic browser UI and how a recommender engine could improve the overall user experiece.

The usability and utility, and no the visual design, are the factors that determine the success or failure of a web-site. The visitor of a page is at the end the one that takes the final decision of how he wants to use the application.

5.1 How do users think?

Users' habits on the Web can be comparable to the customers' habits in a store. Visitors look at each page, scan some of the text and click on the link that catches their interest or vaguely resembles their taste. Typical users don't read, they scan a document. Users search for fixed points which would guide them through the content of the page. The Image 5.1 shows the typical scan process. Notice, how the eyes of the user abruptly stop in the middle of sentences.

Figure 5.1: Typical user scanning process

Visitors are impatient and expect instant gratification. The higher the cognitive load and the less intuitive the navigation, the more difficult is to keep the users in the web-site. Typically, users don't make optimal choices, they search for the quickest way to find the information they are looking for.

5.2 Principles of effective web design

With the boom of web applications a set of principles, heuristics and approaches for successful web design have appear through the years which used properly can lead to more sophisticated design decisions simplifying the process of presenting information. The following list summarize the main principles for a successful web design.

- **Don't make users think -** A web-page should be obvious and self-explanatory. The content must be well-understood and visitors should feel comfortable with the way the interact with the

system.

- **Don't make users lose their patient -** Keep the user requirements minimal. The less action is required from the users the better. Letting the user see clearly what options are available is a fundamental principle of successful user interface design.
- **Manage to obtain users' attention -** Focusing users' attention to specific areas of the site with moderate use of visual elements can help visitors to navigate through the page without making the user think. The less doubts the users have, the better sense of orientation and better user experience which is the aim of usability in the first place.
- **Search for simplicity -** Users are rarely on a site to enjoy the design. In most cases they are looking for the information despite the design.

5.3 Improving the Semantic Browser User Interface

5.3.1 Concept Highlighting

Some of the principles presented in the last section can be applied to enhance the Semantic browser's user interface and improve the overall user experience. The semantic browser could be seen as a sophisticated Document viewer that improves the experience by highlighting concepts and providing useful actions over those concepts. However, the users could feel lost when the semantic browser highlights too many concepts per page, affecting the overall user experience. Figure 5.2 shows how the highlighting can become difficult for the users to read. The big number of highlighted concepts in a very small space, doesn't manage to get the user attention on the items that he might be interested in.

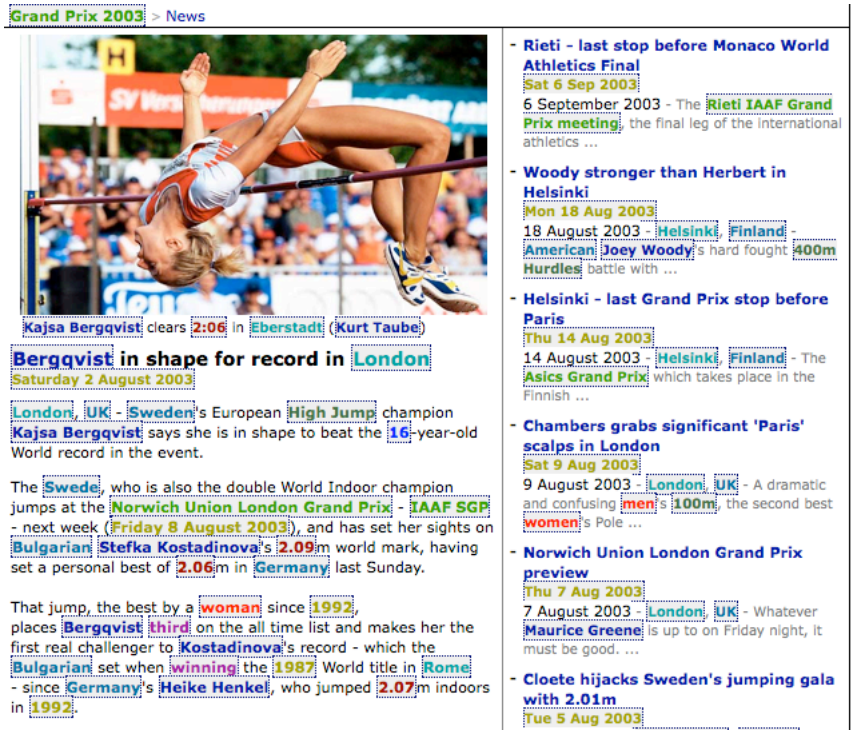

Figure 5.2: Semantic Browser - Concept Hightlighting

The recommender engine can improve, drastically, the way documents are shown to the user. By highlighting only those concepts that the user may be interested, the interface will successfully manage to obtain the users' attention and keep the focus in the most important part of the document: The information. Image 5.3 shows the same content that Image 5.2, but this time only the recommended items for the user are highlighted. If the user wants to know more about a Concept, there would be a high chance that it would be one of the recommended ones. The functionality of the semantic browser is still available but with an improved user interface.

Figure 5.3: Semantic Browser - Concept Recommendation

5.3.2 Service Selection

Every highlighted concept in a page has a list of associated items that could be accessed by any user that has a particular interest in a concept. Once the highlighted concept is selected a menu with the list available services is displayed to the user.

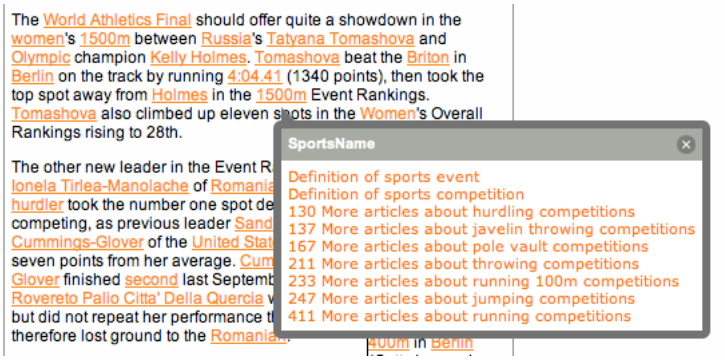

Figure 5.4: Boemie Concept - Services Menu

As shown in the Figure 5.4, the list of available services for each concept could be long and overwhelming for the user. The recommender engine proposed in this projects displays only the set of recommended services of interest for the user. By reducing the number of items, the user would not lose time searching through the list of available actions, making the interaction with the Semantic Browser much simpler. Currently, most of the concepts have a small number of actions and the use of a recommender system could be an overkill for the Semantic Browser. However, it could be very useful to use recommendations when the number of actions per concept starts to grow.

Chapter 6

Conclusion

This chapter summarizes the contributions of the project and proposes possible directions for future work.

6.1 Contribution

The main goals and structure of the work taken during this project were:

- General recommender system, collaborative filtering and similarity algorithms research.
- Application of collaborative filtering concepts for the enhancement of the GUI of the semantic browser.
- Implementation of a generic recommender system engine.
- Implementation of the integration between the semantic browser and the generic recommender system component.

All the goals were successfully accomplished during the past 3 months of work. The general collaborative filtering research is summarized in Chapter 1 and Chapter 2, where the concepts of collaborative filterings are described together with the main filtering algorithms. Chapter 3 and Chapter 4 deal with the architecture and implementation of the components required to recommend items to the semantic browser. And finally, the effects of using a recommender system to improve user experience of the semantic browser are presented in Chapter 5.

The research gathered during this project, proved that collaborative filtering is a promising approach that could simplify and improve the user experience with the Semantic Browser and in general with any complex web site. And even though collaborative filtering algorithms are quite computationally expensive, the recommendation process does not necessarily have to be real-time, as offline recommendation computations are sufficient.

6.2 Future Work

The viability to enhance the user interface of the semantic browser was the main objective of the project. However, several issues still may be resolved to make a better use of recommendation techniques for this purpose:

- **Scalability -** The presented collaborative filtering algorithms are too general and therefore unoptimized and computationally expensive. More advanced algorithms should be tested or developed to answer the problem of growing number of users and huge data sets.
- **Distributed solution -** Another answer to the scalability issue could be a distributed recommender system.
- **Hybrid algorithms** – There is an extensive area of so called hybrid collaborative filtering algorithms. These algorithms combine the collaborative filtering together with a content-based approach.

Bibliography

[1] The Long Tail, Chris Anderson; Wired, Oct. 2004

[2] Qualitative analysis of user-based and item-based prediction algorithms for recommendation agents by Manos Papagelis, Dimitris Plexousakis, University of Crete & FORTH-ICS, Greece, 2005.

[3] Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion by Jun Wang, Arjen P. de Vries, Marcel J.T. Reinders; Delft University of Technology, 2006.

[4] Using Mixture Models for Collaborative Filtering by Jon Kleinberg, Mark Sandler; Cornell University, 2004.

[5] Analysis of Recommendation Algorithms for E-Commerce by Badrul M. Sarwar, George Karypis, Joseph A. Konstan, John T. Riedl; GroupLens Research Group Army HPC Research Center Department of Computer Science and Engineering University of Minnesota, 2000.

[6] Evaluation of Item-Based Top-N Recommendation Algorithms y George Karypis; University of Minnesota, Department of Computer Science / Army HPC Research Center, 2000.

[7] A Java-Based Approach to Active Collaborative Filtering by Christopher Lueg and Christoph Landolt; AI-Lab, Department of Computer Science, University of Zurich, 1998.

[8] Implicit Interest Indicators by Mark Claypool, Phong Le, Makoto Waseda and David Brown; Worcester Polytechnic Institute, 2000.

[9] Item-based Collaborative Filtering Recommendation Algorithms by Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl; GroupLens Research Group/Army HPC Research Center Department of Computer Science and Engineering University of Minnesota, 2001.

[10] Event-Driven Applications: Costs, Benefits and Design Approaches by K. Mani Chandy; California Institute of Technology, 2006.

[11] Principles of beautiful web design by Jason Beaird; Sitepoint Magazine, April 2007.

[12] The Principles of Design by Joshua David McClurg-Genevese; Digital Web Magazine, June 2005.