

# REPORT ADAPTIVE TECHNIQUES EMPOWERING PERSONALIZED MARKETING FOR MOBILE DEVICES



## TABLE OF CONTENTS

1. Personalization and Recommender Systems.....	1
1.1. Introduction .....	1
1.2. Recommender Systems.....	1
1.3. Security.....	3
1.4. Live Recommenders .....	4
2. Our Approach .....	5
2.1. Initial Assumptions.....	5
2.2. Chosen Scenario.....	5
2.3. Techniques and Architecture Chosen .....	7
2.4. Future Extensions .....	8
2.5. How to utilize this work .....	8
3. Recommender Web Service Prototype .....	10
3.1. Introduction .....	10
3.2. Recommender techniques .....	10
3.3. Integration with a Master System .....	10
3.4. Usage Scenarios.....	12
3.5. Scenarios, C# sample code .....	14
3.6. Security.....	17
3.7. Error handling.....	17
4. Web Service Interface.....	19
4.1. AddUser .....	19
4.2. GetUser.....	19
4.3. AddItem.....	19
4.4. GetItem .....	20
4.5. GetKeywords.....	20
4.6. AddAction .....	20
4.7. AddActionEx.....	21
4.8. AddItemWasRecommended .....	21
4.9. IsItemRecommended .....	21
4.10. GetRecommendedItemsByUserItem .....	22
4.11. GetRecommendedItemsByUser.....	22
4.12. GetRecommendedItemsByItemEx .....	22
4.13. GetRecommendedItemsByUserEx .....	23
5. Message Types.....	24
5.1. AddUserRequest.....	24
5.2. AddUserReply .....	24
5.3. GetUserRequest .....	25
5.4. UserReply .....	25
5.5. AddItemRequest .....	26
5.6. AddItemReply .....	27
5.7. GetItemRequest .....	28
5.8. ItemReply .....	28
5.9. KeywordReply.....	29



5.10. AddActionRequest .....	30
5.11. AddActionReply.....	31
5.12. AddActionSlopeOneRequest.....	29
5.13. AddItemWasRecommendedRequest .....	30
5.14. IsItemRecommendedRequest .....	30
5.15. IsItemRecommendedReply .....	31
5.16. GetRecommendedItemsByUserItemRequest .....	31
5.17. GetRecommendedItemsReply .....	32
5.18. GetRecommendedItemsByUserRequest.....	33
5.19. GetRecommendedItemsByItemExRequest .....	33
5.20. GetRecommendedItemsByItemExReply.....	34
6. Appendix .....	35
6.1. Products Used.....	35
6.2. Data Model.....	35
7. References .....	37

## 1. PERSONALIZATION AND RECOMMENDER SYSTEMS

### 1.1. INTRODUCTION

The basic goal of any user-adaptive system is to provide users with what they need without their asking for it explicitly (*Mulvenna et al. 2000*). Automatic personalization, therefore, is a central technology used in such systems. On the Web, this typically involves the delivery of dynamic content, such as advertisements and product recommendations, etc., that are tailored to a user's needs.

Automatic personalization involves the creation and maintenance of user profiles with minimal user control. Recommender systems support such functionality. Their use is common in E-commerce Web sites, e.g, to provide customers with product recommendations.

Mobile devices are particularly interesting as targets for recommender systems. Most device screens are small and advertising space is hence very limited. The marketing value of showing the right message or product to the right customer must consequently be regarded as higher in this environment compared to full-sized browsers.

### 1.2. RECOMMENDER SYSTEMS

One of the most common problems users are faced with in today's information-rich world is that, in fact, there is too much information available. The vastly increased volume in information and products has led to the need for tools that can aid the user to find information/products of choice. Recommender systems have been developed to take on this task.

A comparison with marketing and decision support systems may be in place (*Schafer et al. 2001*).

Marketing systems support the marketer in making decisions about how to run marketing campaigns, usually targeted to groups of consumers. Supply-chain decision-support systems help marketers make decisions about how many products to manufacture and to which warehouses or retail stores to ship the products. In contrast, recommender systems answer questions about what an individual consumer is interested in buying at this moment.

Recommender systems are today important in many Web-based applications, particularly those involving e-commerce. The potential of using recommenders to achieve increased sales and keeping customers has been realized by businesses. Users have also learned to appreciate the help from such systems to find products and information of interest. Recommenders enhance E-commerce sales in several ways (*Schafer et al. 2001*):

**Converting Browsers into Buyers:** Visitors to a Web site often look over the site without purchasing anything. Recommender systems can help consumers find products they wish to purchase.

**Increasing Cross-Sell:** Recommender systems improve cross-sell by suggesting additional products for the customer to purchase.

**Building Loyalty:** Recommender systems improve loyalty by creating a value-added relationship between the site and the customer.

### 1.2.1. Types of Recommender Systems

This section describes common types of techniques employed in recommender systems. Recommenders may make use of more than one of these techniques (*Burke 2000, Mobasher 2007, Pazzani et al. 2007, Sarwar et al. 2001, Schafer et al. 2001*).

Each of these techniques is characterized by the specific type of data collected to construct user profiles, and by the algorithmic approach used to generate recommendations.

#### 1.2.1.1. Knowledge-based

Knowledge-based recommenders rely either on explicit domain knowledge about the items or knowledge about the users (such as demographic characteristics) to derive relevant recommendations. Many such systems rely on manually or automatically generated knowledge-based decision rules that are used to recommend items to users who satisfy constraints specified by the stored rules.

#### 1.2.1.2. Content-based filtering

In content-based filtering the system processes information from various sources and tries to extract useful elements about its content. This can involve, for example, keyword-based search or textual product descriptions.

Each user is assumed to operate independently, and the system requires a profile of the user's needs or preferences in order to be able to provide a recommendation. The profile content, specifying user interests, must be explicitly generated by the user. With the profile content as a basis, similar items are identified and returned as recommendations. Content-based filtering techniques do not depend on having other users in the system, and therefore they do not need a critical mass to be able to start providing recommendations. On the other hand, since the recommendations are based only content information, only items or topics similar to those already in the user's profile can be suggested.

#### 1.2.1.3. Collaborative filtering

Collaborative filtering recommender systems produce recommendations by identifying overlapping interests and opinions among users and computing the similarity between a user's preferences and those of other people. User opinions can be collected explicitly in the form of asking for product ratings or implicitly, derived from sources such as sales records. Predictions are calculated on how well suited products are to the user, and then the products with the highest prediction values become the recommended products. Such systems do not attempt to analyze or understand the content of the items being recommended. Rather than treating each user as an individual, they are treated as belonging to a group. The main advantage of such systems is that the pool from which recommendations originate is not restricted to items for which the active user has indicated a preference. It therefore becomes possible to discover and explore new items of interest simply because other people liked them.

Collaborative filtering algorithms can be divided into two main categories—memory-based (user-based) and model-based (item-based) algorithms (*Breese et al. 1998*). **Memory-based** systems learn online. Data is kept in memory, and prediction computations, generating recommendations, are performed online involving the whole user-item database. These systems may therefore encounter scalability issues. Furthermore, the huge user-item matrix, in which to find similar neighbors, is almost always very sparsely filled (even the most active user has interacted with only a very small portion of the items), potentially leading to the system being unable to generate any recommendations. **Model-based** systems learn

offline. Prediction models are generated offline, often constituting a lengthy and processing-intensive task. The online prediction computations, drawn from the generated model, are generally fast and not affected by user or item numbers. Memory-based systems continuously adapt to changes in user preferences while model-based systems must regenerate the model offline to take changed user behavior into account.

Slope One predictors have been suggested as another approach to implement collaborative filtering (*Lemire et al. 2005*). The authors claim that these schemes, in spite of being simple, updatable, computationally efficient, and scalable, are comparable in accuracy to schemes that forego some of the other advantages.

#### 1.2.1.4. Data Mining

Data mining techniques are typically applied in model-based recommender systems. Models are generated for prediction and recommendation purposes. The data mining cycle includes the steps of data collection, pre-processing, pattern discovery, and evaluation, all performed offline. The gathered knowledge is then implemented in a model that is accessed online to generate recommendations. Server log files and sales records are common data sources for data mining.

### 1.3. SECURITY

Two different security aspects are identified. The first concern involves user privacy and the user's view of how a system gathers and handles user information. Users may be reluctant to give out personal information and participate in product ratings due to fear of abusive use of this information. Personalization and recommender systems that, to better serve the user, clearly state which type of information is collected and the purpose behind it, are probably easier to accept from a user perspective.

The second concern involves deliberate attacks on personalization and recommender systems. These attacks can be of several different types (*Mobasher et. a. 2007*) but generally have the purpose of altering the system's recommendation behavior. Collaborative recommender systems can be highly vulnerable to these attacks.

Security issues must be solved if users are to perceive these systems as objective, unbiased, and accurate. Research is underway to determine the best system architectures and defense mechanisms to counter attacks (*Mobasher et. a 2007*).

### 1.4. LIVE RECOMMENDERS

Recommender systems have existed for more than 10 years. A list of live recommender systems, employing collaborative filtering techniques, is presented below.

Movielens — Helping you find the right movie.

Movielens is a movie recommendation Web site that uses collaborative filtering techniques to generate movie recommendations. Users rate movies they have seen. Based on these ratings, Movielens generates personalized recommendations for other movies. The Web site is <http://www.movielens.org>.

inDiscover.net

inDiscover is a service designed to help bring independent music to new listeners. Users rate music and are in return offered recommended music playlists with the help of collaborative filtering techniques. The Web site is <http://indiscover.net>

#### Jester — Jokes for Your Sense of Humor

Jester uses a collaborative filtering algorithm called Eigentaste to recommend jokes to users based on their ratings of previous jokes. Web site: <http://eigentaste.berkeley.edu>.

#### Amazon.com

Amazon.com started out as an online bookstore but now offers a multitude of product lines like multimedia, furniture, toys, and more. Amazon.com employs item-to-item collaborative filtering. Instead of matching similar users, each of the user's purchased and rated items are matched to similar items and these similar items form the recommendations. Web site: <http://www.amazon.com>.

## 2. OUR APPROACH

This section describes our approach to the task of supporting personalized marketing both in mobile devices and in other customer interfaces with the help of a recommender platform.

### 2.1. INITIAL ASSUMPTIONS

Data is collected from identified users (logged in, cookies) visiting a Web site. The collected information is stored in a recommender system database to be used for both personalized and nonpersonalized recommendations.

- All user actions can potentially contribute to the user profile:
  - Keywords in a content/product search
  - Products/content viewed
  - Products purchased
  - Advertisement viewed (clicked on)
- Actions may be weighted differently
  - Purchase weighted higher than just viewing
- Applied to full-size- and mobile-browser sites
  - A “full-size-browser” site is the major source for data collecting.
  - A “full-size-browser” site can show several recommendations.
  - A “mobile-browser” site shows top-priority recommendations.

### 2.2. CHOSEN SCENARIO

We have chosen to apply the recommender platform the following scenario:

#### 2.2.1. High-level description

- An identified user performs a keyword-based search for products/services on a Web site.
- The Web site presents a ranked list of results delivered by the recommender engine. Ranked advertisements may also be presented on the same page.
- The user selects (clicks on) a result link or an advertisement with the purpose of
  - Viewing
  - Buying
  - Adding to wish list.

#### 2.2.2. Detailed description

##### *Step 1*

A user, logged in or identified by a cookie, enters a product search page. One or more search keywords are entered. The search request is submitted.

- The keyword entry process may be augmented with auto complete functionality based on the user's profile (previous actions of both current and similar users).

*Example: User enters "Spain" and "hotel."*

#### *Step 2*

The recommender engine receives input data (at least user identity and search keywords but may also include context, current browsing history, and other data).

- The acquired knowledge is added to the user profile (interest in "Spain" and "hotel").
- A ranked list of recommendations (product links and advertisements) is generated and returned to the Web application.
  - Previous recommendations may be taken into account:
    - ◆ Do not include previously recommended item.
    - ◆ Previously recommended items reappear in a round-robin manner.
  - The "full-size-browser" site shows n number of results/advertisements.
  - The "mobile-browser" site shows only the first item (highest ranking).

#### *Step 3*

The user selects (clicks on) a result list item or an advertisement.

- The acquired knowledge is added to the user profile ("Spain" and "hotel" weights are bumped up).
- The product/content database may contain several keywords/attributes per product (product profile). Such keywords, not specified by the user in the current search, may be added to the user profile in order to widen the base for future recommendations.
- The result list may offer the option to remember items in a "wish list." The recommender platform may add such choices to the user profile with a different weight compared to an actual product/advertisement selection.

### 2.3. TECHNIQUES AND ARCHITECTURE CHOSEN

#### 2.3.1. Algorithms

To have a working prototype in place within the limited time available, algorithms, reasonably simple to implement, have been chosen. These algorithms should, however, still be accepted ones in the recommender field and be able to provide useful functionality.

Two different collaborative filtering algorithms have been implemented in the recommender prototype:

- *Item-based collaborative filtering using cosine-based similarity to compute similarities (Sarwar et al. 2001).*

This algorithm looks into the items that users have rated and computes how similar they are to other rated items (in our implementation the number and types of user interactions with an item are translated to ratings for computing purposes).

Each item is thought of as a vector in the  $m$  dimensional user-space where  $m$  spans all users. Similarity between items  $i$  and  $j$ , denoted by  $\text{sim}(i, j)$  is given by

$$\text{sim}(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

where “ $\cdot$ ” denotes the dot-product of the two vectors.

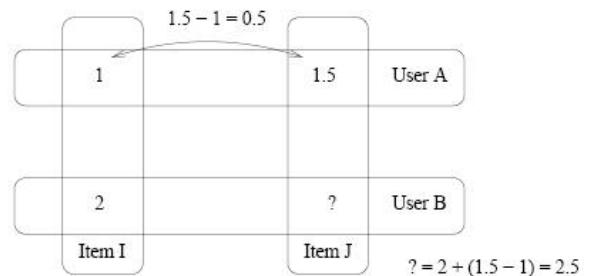
User interactions/ratings are collected in real time and recommendations are queried from precomputed similarity compilations. Precomputing is performed offline at regular intervals.

- *Weighted Slope One Scheme (Lemire et al. 2005).*

This algorithm considers “popularity” differences between items for users. These differences can be used to predict other users’ ratings of items.

(Figure from *Lemire et al. 2005*.)

User A’s ratings of two items and User B’s rating of a common item are used to predict User B’s unknown rating.



Both algorithms are relatively straightforward to realize in software and do not show any significant deviations in recommendation performance compared to other algorithms.

### 2.3.2. Architecture

The recommender algorithms are implemented with the help of tables and stored procedures in an Oracle relational database. A Web service interface makes the recommender functionality available to other systems. A detailed description of the recommender prototype and its use is found in Section 0.

### 2.3.3. Recommender Prototype

The Recommender Web service prototype is described in detail in Section 0. The prototype has so far been tested only on a desktop computer with randomly generated test data to verify basic functionality. Performance issues have not been detected, but the prototype needs to be evaluated in a real server environment with very large volumes of data. Integration with a master system, e.g., an e-commerce Web site, also remains to be tested.

## 2.4. FUTURE EXTENSIONS

Data mining techniques have not been employed in the recommender prototype. Several software packages, both commercial and open source, exist that implement various data mining methods. Our recommender platform could be extended with data mining capabilities, e.g., to learn from user actions (browsing history) and patterns in Web logs. Patterns discovered could be used for recommendations purposes.

User voting, rating, and other schemes in which the user actively contributes to the construction of his/her profile are not included in the prototype. Although the recommender database is able to store ratings, our current implementation of the algorithms does not take the ratings into consideration when recommendations are calculated. This is a subject for future work.

There is also the complicated field of trust. People often trust recommendations from others that are “like themselves.” The following scenario is then interesting from a recommender perspective: “*people who are like me and that viewed/bought this product/advertisement then/also viewed/bought X.*” The question arises how to measure and, in software, implement “are like me.” Since the trust factor can be important to people when evaluating products to buy, this field should be investigated further.

Related to the field of trust is the threat of attacks targeted at the recommender system. People might want to deliberately raise or lower the likelihood of certain products being recommended through bogus interactions with the system. A full-fledged commercial recommender implementation should include support for attack detection. This is also a subject for future work.

#### 2.5. HOW TO UTILIZE THIS WORK

The personalization and recommender domains have grown into large research fields. It is not viable to try to package the amount of knowledge available into a shrink-wrapped product at a fixed price. Instead, it is believed that the personalization and recommender knowledge should be regarded as the product. Knowledge, architecture, and software are offered to enhance customer Web sites with personalization and recommenders.

### 3. RECOMMENDER WEB SERVICE PROTOTYPE

#### 3.1. INTRODUCTION

The purpose of this Web service is to offer a prototype recommender system that can be used together with a master system of some sort. The master system could typically be an e-commerce Web site where identified customers search for products and services and view and buy these offerings. A “full-size-browser” site could be accompanied by a site for mobile devices, a “limited-size-browser” site.

The recommender takes as input records of how users have interacted with various items (products, advertisements, and the like). These users must be identified for the input data to have a meaning. As output the Recommender can deliver both personalized and nonpersonalized item recommendations. The recommendations can be related to a “current” item (product currently viewed) or not. Recommendations are delivered as lists (one or more items), with the “best” item first.

The master system’s “full-size-browser” site probably constitutes the primary source for collecting user actions. Although the mobile site very well can collect user actions, it is the main target for recommendations where one of the most highly predicted items is displayed as an advertisement/link.

#### 3.2. RECOMMENDER TECHNIQUES

The prototype Recommender implements two different collaborative filtering algorithms to compute recommendations:

- Item-based collaborative filtering using cosine-based similarity to compute similarities (*Sarwar et al. 2001*).
- Weighted Slope One Scheme (*Lemire et al. 2005*).

The first algorithm relies on regular offline batch processing to produce recommendation data. The second algorithm allows continuous online generation of recommendation data when user actions are registered.

These algorithms are implemented with the help of tables and stored procedures in an Oracle relational database. Offline cosine-based similarity calculations are carried out in a stored procedure. The essential parts of the data model are described in Section 0.

#### 3.3. INTEGRATION WITH A MASTER SYSTEM

The process of integrating the Recommender into a master application involves a number of steps.

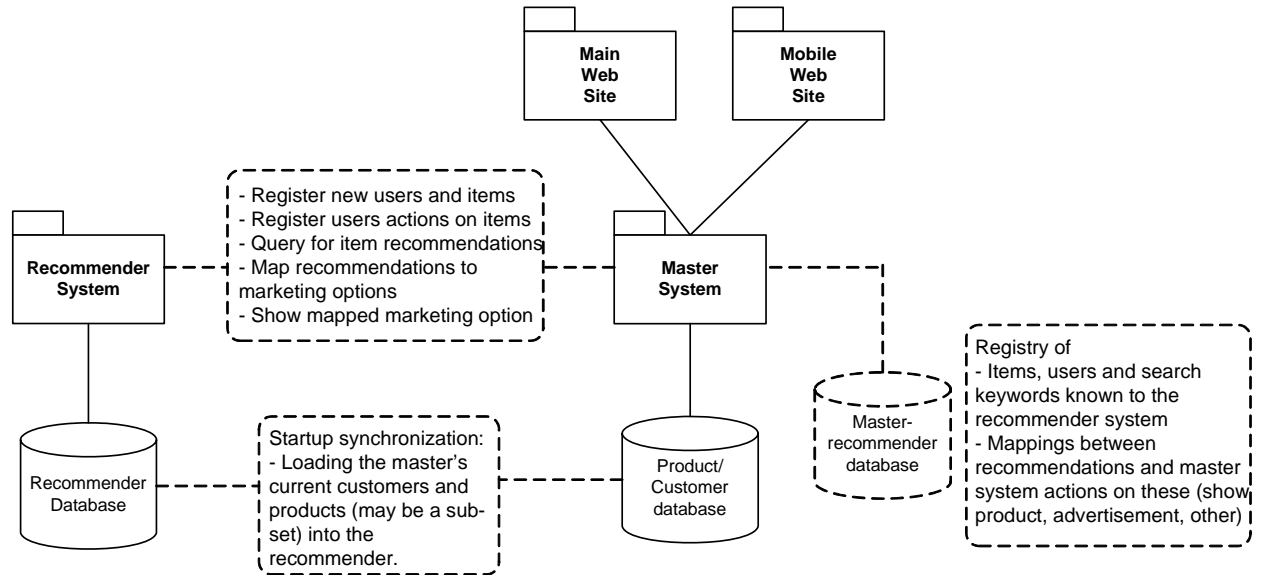
First of all, the following decisions have to be made:

- Which type of users should be included to interact with the recommender? All categories or selected groups, e.g., customer club members?
- Which products should be included? All products or certain types, selected by, e.g., price, sales volumes, etc.?
- If search keywords are to be employed, which keywords should be included in the recommender? Does the master allow only preselected keywords, or is a free text search possible? Should both these types be included in the recommender?

- Where in the navigational flow should the Recommender be interacted with?
- Which user actions should be recorded in the Recommender be interacted with? Information and product search, viewing, product purchasing, advertisement viewing, etc.?
- How should the recommendations be used? Product offerings, advertisements, new search suggestions?
- What to do if no recommendations are returned from the recommender?

The master system needs to be modified in order to take advantage of the recommender. The broken lines in the figure below identify required new functions and tasks.

- The master's customer and product databases (or selected parts thereof) must initially be copied to the Recommender to form its user and item registers. In addition to the master's user and item identities, the Recommender generates its own identities and these may optionally be transferred back to the master.
- After the initial user and item load, the master should continuously feed new users and items to the Recommender.
- The master system may want to keep track of which users and items, including search keywords that are known to the Recommender.
- The master system needs to feed user actions (customer A viewed product X) to the Recommender. Even if this may not affect Web page design, it requires additional code-behind.
- The master system needs to maintain a mapping scheme that determines how recommendations from the Recommender should be used: *Item X is recommended* → *show item X in a product list*. *Item Y is recommended* → *show advertisement Z in a banner*.
- The master system's Web sites need to include the recommendation requests and actions when Web pages are designed and rendered.



## Integration of Master and Recommender Systems

### 3.4. USAGE SCENARIOS

As explained above, the recommender system does not exist on its own but acts as a supporting tool for other systems that manage customer – product relations. The recommender could typically assist an e-commerce site of some sort. Consequently the usage scenarios described below involve a “master” system that utilizes the capabilities of the Recommender. A fictitious travel services application is chosen as master.

#### 3.4.1. Keeping common data synchronized

Both the master system and the Recommender contain customer/user records and product/item records. Initially a complete build of Recommender user and item records is performed, based on the master’s customer and product database. New customers and products should, however, be fed to the Recommender in real time.

Example:

- A new customer registers in the master system.
- The master generates a new unique customer identifier and calls the Recommender *AddUser* method supplying that new customer identifier.
- The Recommender creates a new user, identified both by a unique Recommender identifier and the supplied master customer identifier.
- The Recommender identifier is returned to the master. The master may choose to store this identifier or discard it.

New products and search keywords are handled similarly. The master may also check for the existence of customers/users and products/items in the Recommender.

### 3.4.2. Register a customer action

The Recommender draws its capabilities from previous user – item interactions. Hence, the master should feed all customer actions to the Recommender, continuously improving the quality of its recommendations.

Example:

- A customer uses the words “Spain” and “hotel” as search keywords looking for travel information.
- The master performs whatever search it implements.
- The master calls the recommender *AddActionEx* method to register that the customer has used these words as search keywords.

The master may also register that products were viewed and bought.

### 3.4.3. Obtain recommendations

Recommendations can be obtained in a number of ways with different prerequisites:

- Query recommendations based on a **“current” user and a current “item”**. The user is identified and is interacting with an identified item. The Recommender *GetRecommendedItemsByUserItem* method serves this type of recommendation (Algorithm A, Section 0).
- Query recommendations based on a **“current” user**. The user is identified. The Recommender *GetRecommendedItemsByUser* method serves this type of recommendation (Algorithm A, Section 0).
- Query recommendations based on a **“current” user**. The user is identified. The Recommender *GetRecommendedItemsByUserEx* method serves this type of recommendation (Algorithm B, Section 0).
- Query recommendations based on a **current “item”**. An item is identified, possibly interacted with by someone. The Recommender *GetRecommendedItemsByItemEx* method serves this type of recommendation (Algorithm B, section 0).

The master may want to keep track of which items actually were recommended to users. For this purpose the Recommender supplies the methods:

- *IsItemRecommended* to query for the recommendation status of item (last time and number of times recommended)
- *AddItemWasRecommended* to enable the master to register that items were actually recommended.

### 3.5. SCENARIOS, C# SAMPLE CODE

#### 3.5.1. Register a new Recommender user

```
// Basic definitions
recsvc.Recommender mySvc = new recsvc.Recommender();
string myRecommenderUserId;

// Create SOAP Header for passing of user credentials
recsvc.ClientAuthenticationHeader myHeader = new recsvc.ClientAuthenticationHeader();
myHeader.UserName = "master_user";
myHeader.UserPassword = "secret";
mySvc.ClientAuthenticationHeaderValue = myHeader;

// New user
recsvc.AddUserRequest newUser = new recsvc.AddUserRequest();
recsvc.AddUserRequestItem[] newUserList = new recsvc.AddUserRequestItem[1];
newUserList[0] = new recsvc.AddUserRequestItem();
newUserList[0].ClientUser = "new_master_userid";
newUser.ItemList = newUserList;

// Perform Web service request
recsvc.AddUserReply myReply = mySvc.AddUser(newUser);

// Handle the reply
if (myReply.AuthReply.Authenticated == true && myReply.Status.Status == 0)
{
    foreach (recsvc.AddUserReplyItem item in myReply.ItemList)
    {
        // Pick up the Recommender's new userid
        myRecommenderUserId = item.UserIdRecommender.ToString();
        // Do something with this new id
    }
}
```

#### 3.5.2. Register a customer action

```
// Basic definitions
recsvc.Recommender mySvc = new recsvc.Recommender();

// Create SOAP Header for passing of user credentials
recsvc.ClientAuthenticationHeader myHeader = new recsvc.ClientAuthenticationHeader();
myHeader.UserName = "master_user";
myHeader.UserPassword = "secret";
mySvc.ClientAuthenticationHeaderValue = myHeader;

// New user action (a product was viewed)
recsvc.AddActionSlopeOneRequest newAction = new recsvc.
AddActionSlopeOneRequest();
recsvc.AddActionSlopeOneRequestItem[] newActionList = new recsvc.
AddActionSlopeOneRequestItem[1];
newActionList[0] = new recsvc.AddActionSlopeOneRequestItem();
```

```

newActionList[0].UserItem = new recsvc.UserItemRequest();
newActionList[0].UserItem.User = new recsvc.UserRequestItem();
// Customerid in master system
newActionList[0].UserItem.User.UserIdType =
recsvc.UserIdTypes.CLIENT_SYSTEM_USERID;
newActionList[0].UserItem.User.UserId = "customerid1";
newActionList[0].UserItem.Item = new recsvc.ItemRequestItem();
// Productid in master system
newActionList[0].UserItem.Item.ItemIdType =
recsvc.ItemIdTypes.CLIENT_SYSTEM_ITEMID;
newActionList[0].UserItem.Item.ItemId = "product1";
newActionList[0].UserInteract = new recsvc.UserInteractions();
// Actions
newActionList[0].UserInteract.SearchKeyword = 0;
newActionList[0].UserInteract.Viewed = 1;
newActionList[0].UserInteract.Purchased = 0;
newActionList[0].UserInteract.Rating = 0;
// Action weights
newActionList[0].Weight = new recsvc.Weights();
newActionList[0].Weight.WeightKeyword = 1;
newActionList[0].Weight.WeightViewed = 1;
newActionList[0].Weight.WeightPurchased = 1;

newAction.ItemList = newActionList;

// Perform web service request
recsvc.AddActionReply myReply = mySvc.AddActionEx(newAction);

// Handle the reply
if (myReply.AuthReply.Authenticated == true && myReply.Status.Status == 0)
{
// OK
}

3.5.3. Obtain recommendations for a customer
// Basic definitions
recsvc.Recommender mySvc = new recsvc.Recommender();
uint recommenderItemId;
string searchKeyword;
string masterItemId;
float simValue;
uint numTimesKeyword, numTimesViewed, numTimesPurchased,
numTimesRecommended, rating;
DateTime lastRecommended;

// Create SOAP Header for passing of user credentials
recsvc.ClientAuthenticationHeader myHeader = new recsvc.ClientAuthenticationHeader();
myHeader.UserName = "master_user";
myHeader.UserPassword = "secret";
mySvc.ClientAuthenticationHeaderValue = myHeader;

```

```

// New user action (a product was viewed)
recsvc.GetRecommendedItemsByUserRequest myRequest = new
recsvc.GetRecommendedItemsByUserRequest();
myRequest.User = new recsvc.UserRequestItem();
// Customerid in master system
myRequest.User.UserIdType = recsvc.UserIdTypes.CLIENT_SYSTEM_USERID;
myRequest.User.UserId = "customer1";
// Similarity threshold for returned recommendations
myRequest.SimThreshold = 0.001;
// Max number of recommended items to return
myRequest.MaxNumItems = 5;

// Perform Web service request
recsvc.GetRecommendedItemsReply myReply =
mySvc.GetRecommendedItemsByUser(myRequest);

// Handle the reply
if (myReply.AuthReply.Authenticated == true && myReply.Status.Status == 0)
{
    foreach (recsvc.GetRecommendedItemsReplyItem item in myReply.ItemList)
    {
        // Item identifiers (Recommender's itemid, search keyword, master's itemid)
        recommenderItemId = item.ItemId;
        searchKeyword = item.SearchKeyword;
        masterItemId = item.ClientItemId;
        // Similarity measure
        simValue = item.SimilarityValue;
        // Customer's previous interactions with this item
        numTimesKeyword = item.NumTimesSearchKeyword;
        numTimesViewed = item.NumTimesViewed;
        numTimesPurchased = item.NumTimesPurchased;
        numTimesRecommended = item.NumTimesRecommended;
        lastRecommended = item.LastRecommended;
        rating = item.Rating;
        // Do something with the results
    }
}

```

### 3.6. SECURITY

#### 3.6.1. Encryption

A deployment may opt to protect Recommender communications using SSL.

#### 3.6.2. Authentication

The Recommender Web service requires clients to authenticate their requests. Credentials for authentication purposes are passed in SOAP headers and validated against a user database of choice.

### 3.7. ERROR HANDLING

- Authentication outcome, success and failure, is reported in the *AuthReply* element of all reply messages.
- Request status is reported in the *Status* element of all reply messages. Failure to find information for users, items, and recommendations is indicated by a non-zero value in element *Status.Status*.
- All other failures are reported as SOAP faults.

A consumer must be prepared to handle all these types of replies.

#### 3.7.1. Reply message

In addition to recommendation-related data, all reply messages contain authentication and execution status information.

The *AuthReply* element, of type *AuthenticationReply*, contains authentication status:

```
<s:complexType name="AuthenticationReply">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="Authenticated" type="s:boolean"
/>
    <s:element minOccurs="0" maxOccurs="1" name="AuthenticationMessage"
type="s:string" />
  </s:sequence>
</s:complexType>
```

*Authenticated* is true on success, otherwise false.

*AuthenticationMessage* contains a text description when the authentication attempt failed.

The *Status* element, of type *StatusReply*, contains execution status.

```
<s:complexType name="StatusReply">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="Status" type="s:int" />
    <s:element minOccurs="0" maxOccurs="1" name="StatusMessage" type="s:string"
/>
  </s:sequence>
</s:complexType>
```

*Status* is set to 0 (zero) when a request was successful and data was found or added. Otherwise it is set to a non-zero value.

*StatusMessage* contains a text description, typically “No data found” when users, items, or recommendations were not found.

## 4. WEB SERVICE INTERFACE

This section contains a description of all methods published by the Recommender Web service.

### 4.1. ADDUSER

Method: **AddUser**

Description: Create new users in the Recommender system.

Takes as input a list of master system user identities and creates corresponding Recommender users. The created user identities are returned as reply.

Parameters: **AddUserRequest** type message containing master system user identities; see Section 0.

Return value: **AddUserReply** type message containing a list of new Recommender users with master system identities and Recommender system identities; see Section 0.

Pseudo Syntax: `public AddUserReply AddUser(AddUserRequest addUser)`

### 4.2. GETUSER

Method: **GetUser**

Description: Retrieve users from the Recommender system.

Takes as input a list of user identities, master or Recommender identities and return a list of found users.

Parameters: **GetUserRequest** type message containing user identities, see section 0.

Return value: **UserReply** type message containing a list of users found in the Recommender system users, see section 0.

Pseudo Syntax: `public UserReply GetUser(GetUserRequest userSpec)`

### 4.3. ADDITEM

Method: **AddItem**

Description: Create new items in the Recommender system.

Takes as input a list of master system item identities and creates corresponding Recommender items. The items, as specified by the master, are either of type Product or type Search Keyword. The created item identities are returned as reply.

Parameters: **AddItemRequest** type message containing master system item identities, see section 0.

Return value: **AddItemReply** type message containing a list of new Recommender items with master system identities and Recommender system identities; see Section 0.

Pseudo Syntax:   public **AddItemReply** **AddItem**(**AddItemRequest** addItem)

#### 4.4. GETITEM

Method: **GetItem**

Description:       Retrieve items from the Recommender system.

Takes as input a list of item identities, master or Recommender identities and return a list of found items.

Parameters:       **GetItemRequest** type message containing item identities; see Section 0.

Return value:     **ItemReply** type message containing a list of items found in the Recommender system users; see Section 0.

Pseudo Syntax:   public **ItemReply** **GetItem**(**GetItemRequest** itemSpec)

#### 4.5. GETKEYWORDS

Method: **GetKeywords**

Description:       Retrieve search keyword items from the Recommender system.

Returns a list of found keywords.

Parameters:       **None**

Return value:     **KeywordReply** type message containing a list of keyword items found in the Recommender system user;, see Section 0.

Pseudo Syntax:   public **KeywordReply** **GetKeywords**()

#### 4.6. ADDACTION

Method: **AddAction**

Description:       Store user – item interactions in the Recommender system.

Takes as input a list of user – item interactions and stores this information in the Recommender.

Parameters:       **AddActionRequest** type message containing user – item interaction data, see section 0.

Return value:     **AddActionReply** type message containing the result of the request, see section 0.

Pseudo Syntax:   public **AddActionReply** **AddAction**(**AddActionRequest** addAction)

#### 4.7. ADDACTIONEX

Method: **AddActionEx**

Description:       Store user – item interactions in the Recommender system and update Slope One algorithm data.

Takes as input a list of user – item interactions and stores this information in the Recommender.

Parameters: **AddActionSlopeOneRequest** type message containing user – item interaction data, see section 0.

Return value: **AddActionReply** type message containing the result of the request; see Section 0.

Pseudo Syntax: public **AddActionReply AddActionEx(AddActionSlopeOneRequest addAction)**

#### 4.8. ADDITEMWASRECOMMENDED

Method **AddItemWasRecommended**

Description: Store that an item was actually recommended to a user in the Recommender system (the master system recommended this item in some form to the user).

Takes as input a list of user – item and stores the information in the Recommender.

Parameters: **AddItemWasRecommendedRequest** type message containing user – item data; see Section 0.

Return value: **AddActionReply** type message containing the result of the request; see Section 0.

Pseudo Syntax: public **AddActionReply AddItemWasRecommended(AddItemWasRecommendedRequest addAction)**

#### 4.9. ISITEMRECOMMENDED

Method: **IsItemRecommended**

Description: Retrieve information from the Recommender system about the number of times and the last time an item was actually recommended to a user (the master system recommended this item in some form to the user).

Takes as input a list of users – items.

Parameters: **IsItemRecommendedRequest** type message containing user – item data, see Section 0.

Return value: **IsItemRecommendedReply** type message containing the result of the request; see Section 0.

Pseudo Syntax: public **IsItemRecommendedReply IsItemRecommended(IsItemRecommendedRequest isRecommendedRequest)**

#### 4.10. GETRECOMMENDEDITEMSBYUSERITEM

Method: **GetRecommendedItemsByUserItem**

Description: Obtain cosine-based similarity item recommendations from the Recommender system for the item specified (a situation where a user, identified or not, just interacted with an item).

Takes as input a user – item combination.

User is made use of only to return information about previous interactions with the recommended items and can be given any value.

Parameters: **GetRecommendedItemsByUserItemRequest** type message containing user – item data; see Section 0.

Return value: **GetRecommendedItemsReply** type message containing a list of recommended items; see Section 0.

Pseudo Syntax: public **GetRecommendedItemsReply**  
**GetRecommendedItemsByUserItem(GetRecommendedItemsByUserItemRequest getRecommendations)**

#### 4.11. GETRECOMMENDEDTITEMSBYUSER

Method: **GetRecommendedItemsByUser**

Description: Obtain cosine-based similarity item recommendations from the Recommender system for this user (a situation where the user is known but no item exists to compare with).

Takes a user as input.

Parameters: **GetRecommendedItemsByUserRequest** type message containing user, see section 0.

Return value: **GetRecommendedItemsReply** type message containing a list of recommended items; see Section 0.

Pseudo Syntax: public **GetRecommendedItemsReply**  
**GetRecommendedItemsByUser(GetRecommendedItemsByUserRequest getRecommendations)**

#### 4.12. GETRECOMMENDEDTITEMSBYITEMEX

Method: **GetRecommendedItemsByItemEx**

Description: Obtain Slope One-based similarity recommendations from the Recommender system for this item (a situation where an item was interacted with by someone).

Takes an item as input.

Parameters: **GetRecommendedItemsByItemExRequest** type message containing item; see Section 0.

Return value: **GetRecommendedItemsByItemExReply** type message containing a list of recommended items; see Section 0.

Pseudo Syntax: public **GetRecommendedItemsByItemExReply**  
**GetRecommendedItemsByItemEx(GetRecommendedItemsByItemExRequest getRecommendations)**

## 4.13. GETRECOMMENDEITEMSBYUSEREX

Method: **GetRecommendedItemsByUserEx**

Description: Obtain Slope One-based similarity recommendations from the Recommender system for this user (a situation where the user is identified).

Takes a user as input.

Parameters: **GetRecommendedItemsByUserRequest** type message containing user; see Section 0.

Return value: **GetRecommendedItemsReply** type message containing a list of recommended items; see Section 0.

Pseudo Syntax: **public GetRecommendedItemsReply  
GetRecommendedItemsByUserEx(GetRecommendedItemsByUser  
Request getRecommendations)**

## 5. MESSAGE TYPES

This section describes the XML data types exchanged in the SOAP messages.

### 5.1. ADDUSERREQUEST

Specifies master system users for which Recommender system users should be created.

```
<s:complexType name="AddUserRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfAddUserRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfAddUserRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="AddUserRequestItem"
nillable="true"
type="tns:AddUserRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="AddUserRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ClientUser" type="s:string" />
  </s:sequence>
</s:complexType>
```

*ClientUser* is the master system's user identity.

### 5.2. ADDUSERREPLY

Reply after Recommender system users were created.

```
<s:complexType name="AddUserReply">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfAddUserReplyItem" />
    <s:element minOccurs="0" maxOccurs="1" name="Status" type="tns:StatusReply" />
    <s:element minOccurs="0" maxOccurs="1" name="AuthReply"
type="tns:AuthenticationReply" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfAddUserReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="AddUserReplyItem"
nillable="true"
type="tns:AddUserReplyItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="AddUserReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ClientUserId" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="UserIdRecommender"
```

```

type="s:unsignedInt" />
</s:sequence>
</s:complexType>

```

*ClientUserId* is the master system's user identity.

*UserIdRecommender* is the Recommender system's user identity.

### 5.3. GETUSERREQUEST

Specifies users to search for in the Recommender system.

```

<s:complexType name="GetUserRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfUserRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfUserRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="UserRequestItem"
nillable="true"
      type="tns:UserRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="UserRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="UserId" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="UserIdType"
type="tns:UserIdTypes" />
  </s:sequence>
</s:complexType>
<s:simpleType name="UserIdTypes">
  <s:restriction base="s:string">
    <s:enumeration value="RECOMMENDER_SYSTEM_USERID" />
    <s:enumeration value="CLIENT_SYSTEM_USERID" />
  </s:restriction>
</s:simpleType>

```

*UserIdType* specifies the type of user identifier, the Recommender system's or the master's.

*UserId* is the user identifier.

### 5.4. USERREPLY

Reply to a GetUser request.

```

<s:complexType name="UserReply">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfUserReplyItem" />
    <s:element minOccurs="0" maxOccurs="1" name="Status" type="tns:StatusReply" />
    <s:element minOccurs="0" maxOccurs="1" name="AuthReply"
type="tns:AuthenticationReply" />
  </s:sequence>
</s:complexType>

```

```

</s:sequence>
</s:complexType>
<s:complexType name="ArrayOfUserReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="UserReplyItem"
nillable="true"
type="tns:UserReplyItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="UserReplyItem">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="UserIdRecommender"
type="s:unsignedInt" />
    <s:element minOccurs="0" maxOccurs="1" name="ClientUserId" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="CreateDate" type="s:dateTime" />
  </s:sequence>
</s:complexType>

```

*UserIdRecommender* is the Recommender system's user identity.

*ClientUserId* is the master system's user identity.

*CreateDate* is the time when user was created in the Recommender system.

#### 5.5. ADDITEMREQUEST

Specifies master system items (products, keywords) for which Recommender system items should be created.

```

<s:complexType name="AddItemRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfAddItemRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfAddItemRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="AddItemRequestItem"
nillable="true"
type="tns:AddItemRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="AddItemRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemId" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="ItemIdType" type="tns:ItemIdTypes"
/>
  </s:sequence>
</s:complexType>
<s:simpleType name="ItemIdTypes">
  <s:restriction base="s:string">
    <s:enumeration value="RECOMMENDER_SYSTEM_ITEMID" />
  </s:restriction>
</s:simpleType>

```

```

    <s:enumeration value="CLIENT_SYSTEM_ITEMID" />
    <s:enumeration value="SEARCH_KEYWORD" />
  </s:restriction>
</s:simpleType>

```

*ItemIdType* specifies the type of item identifier, the Recommender system's or the master's or a common search keyword. An item is either a product (type **CLIENT\_SYSTEM\_ITEMID**) or a search keyword (type **SEARCH\_KEYWORD**).

*ItemId* is the item identifier.

## 5.6. ADDITEMREPLY

Reply to an AddItem request.

```

<s:complexType name="AddItemReply">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfAddItemReplyItem" />
    <s:element minOccurs="0" maxOccurs="1" name="Status" type="tns:StatusReply" />
    <s:element minOccurs="0" maxOccurs="1" name="AuthReply"
type="tns:AuthenticationReply" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfAddItemReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="AddItemReplyItem"
nillable="true"
    type="tns:AddItemReplyItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="AddItemReplyItem">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="ItemId" type="s:unsignedInt" />
    <s:element minOccurs="0" maxOccurs="1" name="SearchKeyword" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="ClientItemId" type="s:string" />
  </s:sequence>
</s:complexType>

```

*ItemId* is the Recommender system's item identity.

*SearchKeyword* is the common keyword if this is a keyword type item.

*ClientItemId* is the master system's item identity if this a product type item.

## 5.7. GETITEMREQUEST

Specifies items to search for in the Recommender system.

```

<s:complexType name="GetItemRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfItemRequestItem" />
  </s:sequence>

```

```

</s:complexType>
<s:complexType name="ArrayOfItemRequestItem">
<s:sequence>
  <s:element minOccurs="0" maxOccurs="unbounded" name="ItemRequestItem"
nillable="true"
  type="tns:ItemRequestItem" />
</s:sequence>
</s:complexType>
<s:complexType name="ItemRequestItem">
<s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="ItemId" type="s:string" />
  <s:element minOccurs="1" maxOccurs="1" name="ItemIdType" type="tns:ItemIdTypes"
/>
</s:sequence>
</s:complexType>

```

*ItemIdType* specifies the type of item identifier, the Recommender system's or the master's or a common search keyword. An item is either a product (type **CLIENT\_SYSTEM\_ITEMID**) or a search keyword (type **SEARCH\_KEYWORD**).

*ItemId* is the item identifier.

## 5.8. ITEMREPLY

Reply to a GetItem request.

```

<s:complexType name="ItemReply">
<s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfItemReplyItem" />
  <s:element minOccurs="0" maxOccurs="1" name="Status" type="tns:StatusReply" />
  <s:element minOccurs="0" maxOccurs="1" name="AuthReply"
type="tns:AuthenticationReply" />
</s:sequence>
</s:complexType>
<s:complexType name="ArrayOfItemReplyItem">
<s:sequence>
  <s:element minOccurs="0" maxOccurs="unbounded" name="ItemReplyItem"
nillable="true"
  type="tns:ItemReplyItem" />
</s:sequence>
</s:complexType>
<s:complexType name="ItemReplyItem">
<s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="ItemId" type="s:unsignedInt" />
  <s:element minOccurs="0" maxOccurs="1" name="SearchKeyword" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="ClientItemId" type="s:string" />
  <s:element minOccurs="1" maxOccurs="1" name="CreateDate" type="s:dateTime" />
</s:sequence>
</s:complexType>

```

*ItemId* is the Recommender system's item identity.

*SearchKeyword* is the search keyword if this item is of type **SEARCH\_KEYWORD**.

*ClientItemId* is the master system's item identity if this item is of type **CLIENT\_SYSTEM\_ITEMID**.

*CreateDate* is the time when item was created in the Recommender system.

### 5.9. KEYWORDREPLY

Reply to a GetKeywords request.

```
<s:complexType name="KeywordReply">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfKeywordReplyItem" />
    <s:element minOccurs="0" maxOccurs="1" name="Status" type="tns:StatusReply" />
    <s:element minOccurs="0" maxOccurs="1" name="AuthReply"
type="tns:AuthenticationReply" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfKeywordReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="KeywordReplyItem"
nillable="true"
type="tns:KeywordReplyItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="KeywordReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="SearchKeyword" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="ItemId" type="s:unsignedInt" />
  </s:sequence>
</s:complexType>
```

*ItemId* is the Recommender system's item identity of this keyword.

*SearchKeyword* is the keyword.

### 5.10. ADDACTIONREQUEST

Specifies user – item actions to store in the Recommender system.

```
<s:complexType name="AddActionRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfAddActionRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfAddActionRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="AddActionRequestItem"
nillable="true"
type="tns:AddActionRequestItem" />
  </s:sequence>
```

```

</s:complexType>
<s:complexType name="AddActionRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="UserItem"
type="tns:UserItemRequest" />
    <s:element minOccurs="0" maxOccurs="1" name="UserInteract"
type="tns:UserInteractions" />
  </s:sequence>
</s:complexType>
<s:complexType name="UserItemRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="User" type="tns:UserRequestItem"
/>
    <s:element minOccurs="0" maxOccurs="1" name="Item" type="tns:ItemRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="UserInteractions">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="SearchKeyword"
type="s:unsignedInt" />
    <s:element minOccurs="1" maxOccurs="1" name="Viewed" type="s:unsignedInt" />
    <s:element minOccurs="1" maxOccurs="1" name="Purchased" type="s:unsignedInt" />
    <s:element minOccurs="1" maxOccurs="1" name="Rating" type="s:unsignedInt" />
  </s:sequence>
</s:complexType>

```

*User* is specified as in *GetUserRequest*; see Section 0.

*Item* is specified as in *GetItemRequest*; see Section 0.

*SearchKeyword* is the number of interactions with this item as search keyword to store in the Recommender.

*Viewed* is the number of viewing interactions with this item to store in the Recommender.

*Purchased* is the number of purchasing interactions with this item to store in the Recommender.

*Rating* is the user's rating of this item.

#### 5.11. ADDACTIONREPLY

Reply to AddAction request.

```

<s:complexType name="AddActionReply">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="Status" type="tns:StatusReply" />
    <s:element minOccurs="0" maxOccurs="1" name="AuthReply"
type="tns:AuthenticationReply" />
  </s:sequence>
</s:complexType>

```

## 5.12. ADDACTIONSLOPEONEREQUEST

Specifies user – item actions to store in the Recommender system. Contains additional data for the Slope One algorithm.

```

<s:complexType name="AddActionSlopeOneRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfAddActionSlopeOneRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfAddActionSlopeOneRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
name="AddActionSlopeOneRequestItem" nillable="true"
type="tns:AddActionSlopeOneRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="AddActionSlopeOneRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="UserItem"
type="tns:UserItemRequest" />
    <s:element minOccurs="0" maxOccurs="1" name="UserInteract"
type="tns:UserInteractions" />
    <s:element minOccurs="0" maxOccurs="1" name="Weight" type="tns:Weights" />
  </s:sequence>
</s:complexType>
<s:complexType name="Weights">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="WeightKeyword"
type="s:unsignedInt" />
    <s:element minOccurs="1" maxOccurs="1" name="WeightViewed"
type="s:unsignedInt" />
    <s:element minOccurs="1" maxOccurs="1" name="WeightPurchased"
type="s:unsignedInt" />
  </s:sequence>
</s:complexType>

```

User is specified as in *GetUserRequest*; see Section 0.

Item is specified as in *GetItemRequest*;;see Section 0.

Interaction is specified as in *AddActionRequest*; see Section 0.

*WeightKeyword* specifies how to weight this interaction if item is a search keyword (should be set = 1).

*WeightViewed* specifies how to weight this interaction if item was viewed (should be set = 1).

*WeightPurchased* specifies how to weight this interaction if item was purchased (might be set higher than 1).

## 5.13. ADDITEMWASRECOMMENDEDCREQUEST

Specifies item that was recommended to a user.

```
<s:complexType name="AddItemWasRecommendedRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfAddItemWasRecommendedRequestItem"/>
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfAddItemWasRecommendedRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
name="AddItemWasRecommendedRequestItem" nillable="true"
type="tns:AddItemWasRecommendedRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="AddItemWasRecommendedRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="UserItem"
type="tns:UserItemRequest" />
    <s:element minOccurs="1" maxOccurs="1" name="LastRecommended" nillable="true"
type="s:dateTime" />
  </s:sequence>
</s:complexType>
```

User is specified as in *GetUserRequest*, see section 0.

Item is specified as in *GetItemRequest*, see section 0.

*LastRecommended* specifies time when item was recommended (current time is default):

## 5.14. ISITEMRECOMMENDEDCREQUEST

Specifies user – item to query recommendation status for.

```
<s:complexType name="IsItemRecommendedRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfIsItemRecommendedRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfIsItemRecommendedRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
name="IsItemRecommendedRequestItem" nillable="true"
type="tns:IsItemRecommendedRequestItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="IsItemRecommendedRequestItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="UserItem"
type="tns:UserItemRequest" />
```

```
</s:sequence>
</s:complexType>
```

*User* is specified as in *GetUserRequest*, see section 0.

*Item* is specified as in *GetItemRequest*, see section 0.

### 5.15. ISITEMRECOMMENDEDREPLY

Reply to *IsItemRecommended* request.

```
<s:complexType name="IsItemRecommendedReply">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfIsItemRecommendedReplyItem" />
    <s:element minOccurs="0" maxOccurs="1" name="Status" type="tns:StatusReply" />
    <s:element minOccurs="0" maxOccurs="1" name="AuthReply"
type="tns:AuthenticationReply" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfIsItemRecommendedReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
name="IsItemRecommendedReplyItem" nillable="true"
type="tns:IsItemRecommendedReplyItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="IsItemRecommendedReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="User" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="Item" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="NumRecommendations" type="s:int"
/>
    <s:element minOccurs="1" maxOccurs="1" name="LastRecommended" nillable="true"
type="s:dateTime" />
  </s:sequence>
</s:complexType>
```

*User* is the user as it was specified in the request.

*Item* is the item as it was specified in the request.

*NumRecommendations* is number of times this item has been recommended to this user.

*LastRecommended* is the last time this item has been recommended to this user.

### 5.16. GETRECOMMENDEDITEMSBYUSERITEMREQUEST

Specifies an item to obtain cosine-based similarity recommendations for.

```
<s:complexType name="GetRecommendedItemsByUserItemRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="UserItem"
type="tns:UserItemRequest" />
```

```

    <s:element minOccurs="1" maxOccurs="1" name="SimThreshold" nillable="true"
type="s:float" />
    <s:element minOccurs="1" maxOccurs="1" name="MaxNumItems" nillable="true"
type="s:unsignedInt" />
  </s:sequence>
</s:complexType>

```

User is specified as in *GetUserRequest*; see Section 0.

**Note:** User is only made use of to return information about previous interactions with the recommended items and can be given any value. Set to 0 (zero) to disregard user.

Item is specified as in *GetItemRequest*; see Section 0.

*SimThreshold* specifies the minimum similarity value for items to return.

*MaxNumItems* specifies the maximum number of items to return.

### 5.17. GETRECOMMENDEDITEMSREPLY

Reply containing items recommended by the Recommender system.

```

<s:complexType name="GetRecommendedItemsReply">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfGetRecommendedItemsReplyItem" />
    <s:element minOccurs="0" maxOccurs="1" name="Status" type="tns:StatusReply" />
    <s:element minOccurs="0" maxOccurs="1" name="AuthReply"
type="tns:AuthenticationReply" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfGetRecommendedItemsReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
name="GetRecommendedItemsReplyItem" nillable="true"
type="tns:GetRecommendedItemsReplyItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="GetRecommendedItemsReplyItem">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="ItemId" type="s:unsignedInt" />
    <s:element minOccurs="0" maxOccurs="1" name="SearchKeyword" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="ClientItemId" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="SimilarityValue" type="s:double" />
    <s:element minOccurs="1" maxOccurs="1" name="NumTimesSearchKeyword"
type="s:unsignedInt" />
    <s:element minOccurs="1" maxOccurs="1" name="NumTimesViewed"
type="s:unsignedInt" />
    <s:element minOccurs="1" maxOccurs="1" name="NumTimesPurchased"
type="s:unsignedInt" />
    <s:element minOccurs="1" maxOccurs="1" name="NumTimesRecommended"
type="s:unsignedInt" />
  </s:sequence>
</s:complexType>

```

```

    <s:element minOccurs="1" maxOccurs="1" name="LastRecommended" nillable="true"
type="s:dateTime" />
    <s:element minOccurs="1" maxOccurs="1" name="Rating" type="s:unsignedInt" />
</s:sequence>
</s:complexType>

```

*ItemId* is the Recommender system's item identity.

*SearchKeyword* is the search keyword if this item is of type **SEARCH\_KEYWORD**.

*ClientItemId* is the master system's item identity if this item is of type **CLIENT\_SYSTEM\_ITEMID**.

*SimilarityValue* is the cosine-based similarity value for this item.

*NumTimesSearchKeyword* is the number of times this item was previously interacted with as a search keyword by this user.

*NumTimesViewed* is the number of times this item was previously viewed by this user.

*NumTimesPurchased* is the number of times this item was previously purchased by this user.

*NumTimesRecommended* is the number of times this item was previously actually recommended to this user.

*LastRecommended* is the last time this item was actually recommended to this user.

*Rating* is the user's rating of this item.

#### 5.18. GETRECOMMENDEDTITEMSBYUSERREQUEST

Specifies a user to obtain Cosine-based similarity recommendations for.

```

<s:complexType name="GetRecommendedItemsByUserRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="User" type="tns:UserRequestItem"
/>
    <s:element minOccurs="1" maxOccurs="1" name="SimThreshold" nillable="true"
type="s:float" />
    <s:element minOccurs="1" maxOccurs="1" name="MaxNumItems" nillable="true"
type="s:unsignedInt" />
  </s:sequence>
</s:complexType>

```

User is specified as in *GetUserRequest*; see Section 0.

*SimThreshold* specifies the minimum similarity value for items to return.

*MaxNumItems* specifies the maximum number of items to return.

#### 5.19. GETRECOMMENDEDTITEMSBYITEMEXREQUEST

Specifies an item to obtain Slope One-based similar items for.

```

<s:complexType name="GetRecommendedItemsByItemExRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="Item" type="tns:ItemRequestItem" />
    <s:element minOccurs="1" maxOccurs="1" name="SimThreshold" nillable="true"
type="s:unsignedInt" />
    <s:element minOccurs="1" maxOccurs="1" name="MaxNumItems" nillable="true"
type="s:unsignedInt" />
  </s:sequence>
</s:complexType>

```

*Item* is specified as in *GetItemRequest*; see Section 0.

*SimThreshold* specifies the minimum similarity value for items to return.

*MaxNumItems* specifies the maximum number of items to return.

## 5.20. GETRECOMMENDEDITEMSBYITEMEXREPLY

Reply to an item-Slope One-based similarity request.

```

<s:complexType name="GetRecommendedItemsByItemExReply">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="ItemList"
type="tns:ArrayOfGetRecommendedItemsByItemExReplyItem" />
    <s:element minOccurs="0" maxOccurs="1" name="Status" type="tns:StatusReply" />
    <s:element minOccurs="0" maxOccurs="1" name="AuthReply"
type="tns:AuthenticationReply" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfGetRecommendedItemsByItemExReplyItem">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
name="GetRecommendedItemsByItemExReplyItem" nillable="true"
type="tns:GetRecommendedItemsByItemExReplyItem" />
  </s:sequence>
</s:complexType>
<s:complexType name="GetRecommendedItemsByItemExReplyItem">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="ItemId" type="s:unsignedInt" />
    <s:element minOccurs="0" maxOccurs="1" name="SearchKeyword" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="ClientItemId" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="SimilarityValue" type="s:double" />
  </s:sequence>
</s:complexType>

```

*ItemId* is the Recommender system's item identity.

*SearchKeyword* is the search keyword if this item is of type **SEARCH\_KEYWORD**.

*ClientItemId* is the master system's item identity if this item is of type **CLIENT\_SYSTEM\_ITEMID**.

*SimilarityValue* is the Slope One-based similarity value for this item.

## 6. APPENDIX

### 6.1. PRODUCTS USED

The following products are used in the prototype Recommender system:

Database

- Oracle 9.2 on Windows

Web service

Microsoft .NET C#

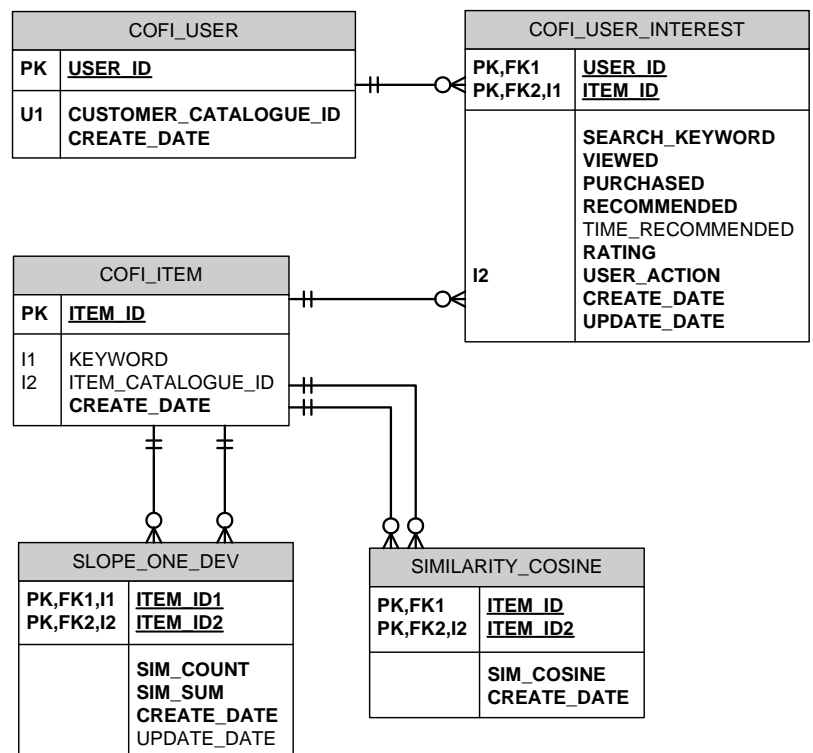
Microsoft .NET Framework 2.0

Microsoft .NET Framework Data Provider for Oracle

Oracle Database Client 9.2

### 6.2. DATA MODEL

The important database tables are described below. The full data model, including scripts to create tables and stored procedures, is available on request.



#### 6.2.1. COFI\_USER

The **COFI\_USER** table contains one record per user/customer that interacts with the Recommender system. The **CUSTOMER\_CATALOGUE\_ID** column links to the master system's ordinary customer database.

#### 6.2.2. COFI\_ITEM

This table contains one record per item (search keyword, product, advertisement, etc.) that is included in the Recommender system. The column **KEYWORD** should link to keywords

used as search criteria in the master system. The column **ITEM\_CATALOGUE\_ID** should link to the master system's product database.

An item should be regarded as either a keyword or a product. Consequently, either **KEYWORD** or **ITEM\_CATALOGUE\_ID** should have a value.

### 6.2.3. COFI\_USER\_INTEREST

This table contains one record per user – item interaction pair.

The first time user X interacts with item Y a record X-Y is created. Each following X interaction with Y updates the existing record (counters are incremented).

When the master system has recommended item Y to user X the RECOMMENDED counter should be incremented and TIME\_RECOMMENDED set.

### 6.2.4. SIMILARITY\_COSINE

This table contains item-to-item similarities measured as the cosine of the angle between item vectors in Table COFI\_USER\_INTEREST (each item is thought of as a vector in the m dimensional user-space where m spans all users).

Similarity between items i and j, denoted by  $\text{sim}(i, j)$  is given by

$$\text{sim}(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

where “•” denotes the dot-product of the two vectors.

The content is regularly regenerated offline.

### 6.2.5. SLOPE\_ONE\_DEV

This table contains popularity differentials between items, calculated according to the Weighted Slope One algorithm.

Each time a COFI\_USER\_INTEREST record is created or updated the rating\* differentials are calculated between that item and all other items, previously acted upon by that user. The differentials are applied to the corresponding SLOPE\_ONE\_DEV records.

\* Rating is here calculated as the sum of the (SEARCH\_KEYWORD + VIEWED + PURCHASED) COFI\_USER\_INTEREST-column values when creating a new SLOPE\_ONE\_DEV record.

When the corresponding SLOPE\_ONE\_DEV record already exists, the sum of the (SEARCH\_KEYWORD + VIEWED + PURCHASED) increment is instead applied and the SIM\_COUNT counter is not incremented (not a new rating, just a changed one).

## 7. REFERENCES

- Breese, J. S., Heckerman, D., and Kadie, C.: Empirical Analysis of Predictive Algorithms for Collaborative Filtering. in Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, pp. 43-52 (1998).
- Burke R.: Knowledge-based recommender systems. In A. Kent, editor, Encyclopedia of Library and Information Systems, volume 69. Marcel Dekker, New York (2000).
- Lemire D., Maclachlan A.: Slope One Predictors for Online Rating-Based Collaborative Filtering. In SIAM Data Mining (SDM'05), Newport Beach, California, April 21-23 (2005).
- Mobasher B.: Recommender Systems. Kunstliche Intelligenz, Special Issue on Web Mining. No. 3, pp. 41-43 (2007). BotcherIT Verlag, Bremen, Germany.
- Mobasher B., Burke R., Bhaumik R., Sandvig J.J.: Attacks and Remedies in Collaborative Recommendation. IEEE Intelligent Systems. Vol. 22, No. 3, pp. 56-63, May/June, (2007).
- Mulvenna, M., Anand, S.S., Büchner, A.G.: Personalization on the net using web mining. Communication of ACM 43(8) 122–125 (2000).
- Pazzani M.J., Billsus D.: Content-Based Recommendation Systems In The Adaptive Web, pp. 325-341, Springer Verlag Berlin Heidelberg (2007).
- Sarwar, B. M., Karypis, G., Konstan, J. A., & Riedl, J. T.: Item-based collaborative filtering recommendation algorithms. Proceedings of the 10th International World Wide Web Conference (pp. 285–295) (2001).
- Schafer, J. B., Konstan, J. A., and Riedl, J.: E-commerce recommendation applications. Data Mining and Knowledge Discovery, 5(1/2):115–153 (2001).