RACOFI: A Rule-Applying Collaborative Filtering System

Michelle Anderson Marcel Ball Harold Boley Stephen Greene Nancy Howse
Daniel Lemire Sean McGrath

Internet Logic Group
Institute for Information Technology e-Business
National Research Council of Canada
Fredericton, NB E3B

Abstract

In this paper [1] *we give an overview of the RACOFI (Rule-Applying Collaborative Filtering) multidimensional rating system and its related technologies. This will be exemplified with RACOFI Music, an implemented collaboration agent that assists on-line users in the rating and recommendation of audio (Learning) Objects. It lets users rate contemporary Canadian music in the five dimensions of impression, lyrics, music, originality, and production. The collaborative filtering algorithms* $STI\,Pearson$, $STIN2$, *and the Per Item Average algorithms are then employed together with RuleML-based rules to recommend music objects that best match user queries. RACOFI has been on-line since August 2003 at [http://racofi.elg.ca].*

1. Introduction

Recent growth in the e-learning industry has prompted a demand for customized generation and filtered reuse of learning objects. The definition of a Learning Object is quite broad [6]: A learning object may be one of any number of items such as a map, a web page, an interactive application, an on-line video – any element that might be contained within a course.

There are presently countless Learning Objects available for corporate and academic use. Despite the advantages of having access to such ever-growing object libraries, e-learning now faces a more pressing challenge: how to find the most appropriate object for a given user/purpose? Common industry standards such as SCORM (Sharable Content Object Reference Model) and IMS (Instructional Management System) facilitate the location of learning objects from a repository by extended search capabilities. For example, the user can search by keyword, date, author, or any metadata field. However, since SCORM defines approximately 60 fields, average users are unlikely to completely specify their needs according to such a large number of attributes. As the granularity of the learning objects decreases and as the size of repositories increases, there will also be a need for much more fine-grained topic descriptions than either SCORM or IMS can provide. Even advanced searches can overwhelmingly return hundreds of thousands of results [7]. Still, this is an improvement from a simple query which could possibly return millions of results. Overall, the process may prove to be inadequate in a society that demands immediate, reliable results in order to meet the demands of their customers. We argue that software can work to alleviate such problems by trying to collaboratively "predict" what users will want rather than expect them to completely define their needs.

As part of the Sifter project, a joint project by NRC and KnowledgePool Canada, we have developed the rule-applying collaborative filtering system **RACOFI**. Its **RACOFI Music** implementation is a multi-dimensional recommender agent for Canadian music, on-line at [http://racofi.elg.ca]. This system proposes an alternative solution to the data overload problem. Its philosophy is quite simple: collaboratively share what users deem as valuable objects. It differentiates itself from other rating systems by being able to adapt dynamically to real-time rating streams, by its multidimensional structure of rating categories, and by its rule-applying querying capabilities. It works by collecting ratings from a number of users and then recommending items to each single user, based on correlations between ratings of the current user and other users in the database. By assuming that if several users rate the same way, they will most likely share an interest in various other objects as well. Upon request, the system returns a calculated recommendation of items that might be of interest to that user.

The system is designed around Canadian music, but

1 Proc. IEEE/WIC COLA'03, Halifax, Canada, October 2003. NRC 46507

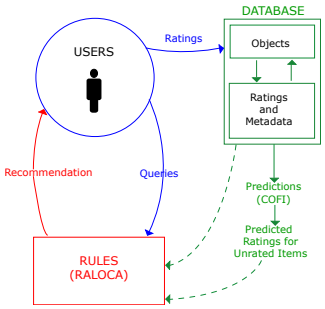

Figure 1. RACOFI General Architecture

not limited to it. A major advantage to RACOFI's flexible object-oriented framework is its capability to target the specific needs of a business. The system's music objects, for example, could just as easily be movie, tourism, or math objects. We achieve this high flexibility for two reasons: the collaborative filtering algorithms are content-neutral, only using ratings and none of the meta-data, whereas the content-based filtering is handled by RuleML through loading of domain-specific rules. For example, with music, it makes sense to attempt to determine whether a user likes a given artist or not or to take into account the genre.

The purpose of RACOFI's meta-data is to describe everything of relevance and interest about the object. So, there are mainly two types of meta-data: objective and subjective. Most of the meta-data is *objective*, which is often used to describes bibliographical information, for example: the date of the item, the author's name or the name of the album. *Subjective meta-data* depends mainly on users; how the users rated the object in attributes such as lyrics and originality. The recommendations must be based on both the objective and the subjective meta-data. We handle the objective meta-data using RuleML, whereas collaborative filtering process the subjective meta-data.

After the user has logged into the system, he / she has the option of rating existing objects, browsing the available items, or adding new ones. When the user chooses to rate an item, a music album in this case, he / she is given a list of various rating categories with a zero-to-ten scale. The following categories are available to rate a music object: impression, lyrics, music, originality, and production. The more ratings a user enters into the system, the broader the basis for the collaborative filtering algorithms, hence the more accurate his / her subsequent querying results (RACOFI's COFI subsystem). Another interesting feature is the ability to allocate weights to the ratings (RACOFI's RALOCA subsystem). In other words, the agent allows the possibility of specifying if some categories will carry more importance in the recommendation. Although this is also on a zero-to-ten scale, they both have very different functions. It is possible to enter into the agent certain rules for given circumstances that the user's information will be evaluated against. These are predefined entities and can easily be added or removed. We also use rules to support content-based collaborative filtering: if a user has rated highly one of Leonard Cohen's albums, then we favor other Leonard Cohen albums, and vice versa.

The main contribution of this paper is the joint use of rules and collaborative filtering algorithms, which allows the application of collaborative filtering in a multidimensional setting. The software is innovative because it offers dynamic on-line personalized recommendations based on multidimensional ratings with a pluggable architecture for both rules and collaborative filtering algorithms. The system is designed so we can easily add or remove rules and algorithms.

This paper is organized as follows. Section two introduces the background technologies; collaborative filtering, XML/XSLT technologies and RuleML. Section three describes the RACOFI system components; information retrieval, COFI and RALOCA. Section four gives an experience report.

2. Technologies

2.1. Introduction to Collaborative Filtering

Internet changed the way we see the world in many ways. We now have access to a large amount of data on most topics in matters of seconds rather than hours or days. Arguably, the value of information has thus gone down, but what is now more valuable than ever is our ability to retrieve quickly the most relevant information.

Information Retrieval offers very powerful tools exemplified by google, the famous search engine. Alas, these content-based tools make some assumptions that are not always satisfied in practice.

1. We assume that the semantic content can be processed by software. Often, this means means that content must be made of text and numerical values.
2. The user can express his query in terms of text or specific content-based criteria (e.g. size, date).

Other technologies attempt to address these two issues by using meta-data and rule-based engines. Even if the semantic content cannot be parsed directly, human intervention can help bridge the gap. However, meta-data is expensive and generally incomplete.

Imagine that an e-Learning developer is preparing a course on security at work. Using modern Internet tools, this developer can easily retrieve information about security in the form of text-based content. Making reasonable assumptions about what the client will prefer based on ex-

perience and some research, the developer can design a format for the course and submit it to the client.

However, in practice, it is likely that some of the time, the developer has had the ability to store in a database the preferences of the current client, if it is a returning client, and of possibly many other clients that may, or may not be related. Rightly so, the clients are expecting the developer to take into account their past feedback.

We argue that the software should support the developer in taking into account the clients' preferences. This can be very easily implemented as in the following example.

For the x^{th} time these past years, the developer has had to choose among a large database of background music or background images. As part of the feedback requested from clients, ratings on these objects has been stored. From these ratings, it is quite easy to quickly sort the possible choices from the most popular one, to the least popular. The developer will naturally be tempted to choose the object that have received the highest ratings.

However, this non-personalized approach is generally unsatisfying if the current client has already given us ratings and if our clients have sometimes different tastes. Imagine for example that this same client has consistently rated black and white images poorly, but that the most popular background image is a black and white one. Would the developer be wise to choose the black and white image?

It might look like all we need is to collect simple rules about user preferences. However, adapting to specific clients might not be simple in general. Imagine that we have no ratings on background images coming from this client, and we cannot expect clients to give us complete ratings all the time, but that the client has rated musical backgrounds several times. We know that the client has rated very poorly classical music, and we also have a number of clients rate poorly classical music **and** black and white backgrounds. Would the developer be wise to choose the black and white image?

In practice, of course, it is likely that the problem is even more complex with many conflicting ratings to the point where we cannot express the preferences of the current client in terms of clear rules.

Collaborative Filtering was introduced in the mid eighties as a way to cope with such problems [12]. One of the most common technique is to view the ratings of each user as an incomplete vector and to use as a similarity measure the Pearson correlation over their commonly rated items. Using the similarity measure, one can then compute a weighted average of all users in the database and present the result as a *prediction* to the current user. There are many ways to improve this generic scheme and it has been shown to consistently outperform [4]. in accuracy the naïve approach which consist in predicting that the current user will rate items as an *average user* (using per item average). The Pearson correlation scheme outperforms the naïve approach because it tries to leverage as much as possible the information known about the user instead of discarding it.

The accuracy of a collaborative filtering system can be easily measured. It suffices to *hide* one of the rating from each evaluation and see how well we can predict it by computing the absolute error. By averaging over many such test, we get an objective measure of the accuracy called *AllBut1 Mean Average Error*.

2.2. Algorithms

For the purpose of this paper, we distinguish between two types of schemes: lightweight or memory-based. Memory-based schemes, such as the Pearson scheme described above, require us to go through a large set of preferences each time a prediction is required. This task can quickly become expensive: doubling the number of users, roughly doubles the response time of the system. Ideally, one would want on-line constant time answers while using only a marginal amount of resources.

As a more scalable alternative, researchers have proposed using schemes such as the Bias From Mean algorithm [10]. Given u an incomplete vector of ratings, the Bias From Mean scheme can be described by the formula

$$P_{bias}(u)_i = \bar{u} + \frac{1}{card(S_i(\chi))} \sum_{w \in S_i(\chi)} w_i - \overline{w} \quad (1)$$

where $\bar{u}$ is the average of the incomplete vector and $S_i(\chi) = \{w \in \chi : i \in S(w)\}$ where $S(w)$ is the set of items rated in w and χ is the set of all incomplete vectors available (all users). It can be computed much faster, without accessing the full database, and is only about **10% less accurate**. It can be quickly updated when new ratings are entered and we only need to keep in fast storage a single vector of size proportional to the number of items in the database.

Other authors have proposed forcing the users to rate a standard set of items [9] before they can use the system. While these authors get good results, they haven't yet outperformed similar schemes while they add a constraint which might be problematic in practice. For example, e-Learning clients might not be interested in rating a set of items before they can do business with you. The reader can test their joke recommender system on-line [8] [http://shadow.ieor.berkeley.edu/humor/].

In [11], we proposed to modify both the Bias From Mean and Pearson scheme, two of the most state-of-the-art algorithms, to make them *Scale and Translation Invariant* (STI). Doing so, we showed we could consistently outperform the original schemes in accuracy by **at least 3%** which is significant in this context. In effect, our lightweight schemes become nearly as accurate as Pearson (**within 4%**). The main idea behind these schemes is that we must factor out

the amplitude and the mean of the ratings of all users. Indeed, some users will tend to rate consistently high whereas others will tend to rate consistently lower and we must ignore such differences since we assume that only the relative rating matters. Also, some users are more *shy*, rarely giving a very bad or a very good rating, whereas other users are more extreme in their ratings. If we don't factor out this amplitude, more extreme raters are likely to count more in the prediction while they may not be the best experts. In effect, normalization is necessary so that the system can be somewhat **democratic**. Taken in this sense, **democracy** improves the accuracy of our prediction. It is still possible for a user to count more than others however: the more items one rate, the more one's opinion will count in the system. We found that if we penalized frequent raters, the accuracy was generally diminished.

One of our most accurate lightweight scheme is $STIN2$ (STI Non-Personalized 2 steps). The term Non-Personalized refers to the fact that the scheme matches all users against a fixed set of vectors as we shall see. As the name suggest, it can be defined in as a two-step process. We begin by defining the array $\mathbf{v}^{(1)}$ as

$$\mathbf{v}_i^{(1)} = \frac{1}{card(S_i(\chi))} \sum_{u \in S_i(\chi)} \frac{u - \bar{u}}{\|u - \bar{u}\|_{l_2}}$$

where we define the normalized Lebesgue norm l_2 as

$$\|u - \bar{u}\|_{l_2} = \sqrt{\sum_{i \in S(u)} \frac{(u_i - \bar{u})^2}{card(S(u))}}.$$

Then, given any evaluation u, we can match it against $\mathbf{v}^{(1)}$ so as to minimize the residual energy $\|u - \alpha - \beta \mathbf{v}^{(1)}\|_{l_2}$ where α and β can be solved for by regression. Once we have solved for α and β, the vector $\alpha + \beta \mathbf{v}^{(1)}$ constitute a first order prediction noted $STIN1$. Given any evaluation u, note the remainder $u_{STIN1} = \alpha + \beta \mathbf{v}^{(1)}$ where α and β depend on u. We found out empirically the we could further use these remainders to improve the accuracy slightly. We use these u_{STIN1} to define the second-order vector $\mathbf{v}^{(2)}$ as

$$\mathbf{v}_i^{(2)} = \frac{1}{card(S_i(\chi))} \sum_{u \in S_i(\chi)} \frac{u - u_{STIN1}}{\|u - u_{STIN1}\|_{l_\infty}}$$

where we have

$$\|u - u_{STIN1}\|_{l_\infty} = \max_{i \in S(u)} |u_i - u_{STIN1,i}|$$

with the convention that $u_{STIN1,i} = \alpha + \beta \mathbf{v}_i^{(1)}$. Thus, given any evaluation u, the $STIN2$ scheme is defined by matching u against both $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ using the formula

$$u - \alpha - \beta \mathbf{v}^{(1)} - \gamma \mathbf{v}^{(2)}$$

where α, β, and γ can be determined by regression. The most expensive component of this algorithm is the computation of $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$, but this can be done offline. Once the system is running, we can show that the vectors can be updated in constant time as new ratings are entered in the system and we've implemented such fast updates in RACOFI.

Our most accurate memory-based is $STI\,Pearson$. Given a database of evaluations χ and a current evaluation w, we compute a *similarity measure* between w and all evaluations $u \in \chi$ using the scalar product

$$s_{u,w} = \left\langle \frac{u - \bar{u}}{\|u - \bar{u}\|_{l_2}}, \frac{w - \bar{w}}{\|w - \bar{w}\|_{l_2}} \right\rangle$$

where the scalar product is defined as a sum over the product of common ratings $\langle u, w \rangle = \sum_{i \in S(u) \cap S(w)} u_i w_i$. Clearly, it can be expensive to compute $s(u, w)$ if the database χ is large because the evaluation w is specific to each user and thus, these results cannot be buffered. On the other hand, it is possible to precompute the values $\bar{u}$ and $\|u - \bar{u}\|_{l_2}$ for all $u \in \chi$ and this is what we chose to implement in RACOFI. In order to ensure that the recommendations can always be done on-line, it is wise to bound the size of the set χ as we have done. We found empirically that raising the similarity measures $s_{u,w}$ to the power 2.5 like so $s_{u,w}^{2.5} = s_{u,w} \times |s_{u,w}|^{1.5}$, improved the results since it favors higher similarity values. This has been found true of other scheme and is referred to as *case amplification* [4]. Once the $s_{u,w}$ have been computed and raised to a power, we compute the vector $\mathbf{v}$ as

$$\mathbf{v}_i = \frac{1}{card(S_i(\chi))} \sum_{u \in S_i(\chi)} s_(u, w)^{2.5} \frac{u - \bar{u}}{\|u - \bar{u}\|_{l_2}}$$

where $\mathbf{v}$ depends on w as well as on the set of evaluations χ. Finally, the prediction is given by $\alpha + \beta \mathbf{v}$ where α and β are found by regression so as to minimize the remainder $u - \alpha - \beta \mathbf{v}$.

2.3. Implementations

Thus, we used $STIN2$ as a lightweight recommender and $STI\,Pearson$ as a memory-based scheme (See Tab. 1) An implementation of these algorithms in Java is available from the authors by correspondence. Further mathematical details and motivation are given in in [11]. On two standard data sets (MovieLens and Jester), we show that they can outperform competitive collaborative filtering systems (Bias From Mean and Pearson). Tab. 1 also shows that recommendations where we don't leverage our knowledge of the current user into account (using the Per Item Average scheme) are less accurate.

In the current system, if each object has many ratings on different attribute, they are considered separately. For exam-

(EachMovie)	AllBut1 MAE	query cost
Per Item Average	0.223	$O(1)$
Bias From Mean	0.203	$O(1)$
$STIN2$	0.194	$O(1)$
Pearson	0.187	$O(m)$
$STI\,Pearson$	0.166	$O(m)$
(Jester)	AllBut1 MAE	query cost
Per Item Average	4.06	$O(1)$
Bias From Mean	3.42	$O(1)$
$STIN2$	3.32	$O(1)$
Pearson	3.24	$O(m)$
$STI\,Pearson$	3.05	$O(m)$

Table 1. AllBut1 Mean Absolute Error (MAE) of different recommender schemes the EachMovie and Jester data sets. The complexity of the queries relative to the number of users m is given. For EachMovie, ratings ranged from 0 to 1 in increments of 0.2 whereas for Jester, the range of values is given to be -10.0 to 10.0. Average where computed over 6 trials of at least 100,000 predictions each with training sets including at least 50,000 ratings. Lesser values are better. The Pearson and Bias From Mean scheme are competitive algorithms. Our implementation of the memory-based Pearson used case amplification with a power of 2.5 as with $STI\,Pearson$.

ple, when rating music albums, we treat separately the ratings for lyrics from the ratings for originality. We haven't yet designed a truly multidimensional collaborative filtering system nor do we know of one in the literature, and this is a subject for future research. We believe a multidimensional scheme could outperform our current one-dimensional approach. However, one benefit of our orthogonal approach is that new dimensions can be added easily and we can cope automatically with dimensions that would not apply to the current object or with users who do not wish to rate a given dimension. These benefits are not fully leveraged in RACOFI since the web site doesn't allow users not to rate a given dimension, but this limitation is not algorithmic.

By design, our approach to collaborative filtering doesn't leverage all meta-data and implicit ratings such as how often or how long a user stayed on a page. For example, we do not currently profile users to determine whether they are professional musicians or at least somewhat knowledgeable of Canadian Music. For that matter, we do not attempt to track users to prevent abuses. We also did not consider many of the psychological factors that will bias the results such as how the various dimensions can be interpreted differently by some users despite our documentation.

2.4. XML Technologies (XSLT)

The extensibility of XML has given rise to its use not only in data structuring, management, and processing, but also as a metadata tagging language (e.g., RDF/XML). RACOFI uses XML and related technologies in the storing and retrieval of user ratings as well as in their processing into recommendations.

By providing an infinitly expandable syntactic framework through the use of definable tags as well as the ability to declare a finite subset of tags through Data Type Declarations (DTDs), XML has allowed developers to define new languages as XML 'applications'. XML has thus become a "structural breeding ground" for new web based languages such as RuleML (see section 2.5), as used in RACOFI.

XML also provides an associated stylesheet language, XSLT (Exstensible Stylesheet Language for Transformations), which allows the transformation of XML into other ASCII based languages. This allows XML documents to be written to address a specific domain, and then be transformed to another XML subset or to an HTML rendering.

Within RACOFI, XSLT has been used to transform RuleML rules from the newer object-oriented form, to their earlier positional form, often still used for processing. This allows developers to take advantage of object-oriented concepts to support development while maintaining runtime advantages of processing a predefined positional rulebase.

2.5. RuleML

Semantic Web [http://www.w3.org/2001/sw] *ontologies* can be composed from the taxonomy-implementing description logic of OWL [5] and the rules-implementing Horn logic of RuleML [3]. While the current RACOFI system has a focus on rules, taxonomies will also become important in the future, e.g. for music "genre" classification.

Object-Oriented RuleML (OO RuleML) [2] [http://www.ruleml.org/indoo] is an extension of the Rule Markup Language that provides not only positional facts and rules but also clauses with unordered argument collections keyed on *roles*, as also found in description logic and F-logic.

RuleML's XML/RDF-integrating system syntax augments 'type tags' by 'role tags' that represent the 'slots', 'features', or "attributes" of type-tagged objects. OO RuleML makes such role tags available as a user syntax in atomic formulas and complex terms: the atom and cterm elements – after their positional argument children – can now also contain non-positional argument chil-

dren "_r" (for the metarole 'role') with a required CDATA attribute 'n' (for the user-defined role 'name'). This also allows for mixed positional and object-oriented representations. OO RuleML has been extended by role weights specifying the relative importance of slots [1].

OO RuleML has been first implemented via an XSLT translator to the positional jDREW [13], and then directly in the form of OO jDREW, the latter being used in the RACOFI system.

3. RACOFI

The RACOFI system is the result of two integrated systems: COFI (collaborative filtering) and RALOCA (Rule Applying Learning Object Comparison Object). The diagram below shows the interaction between the two components. As the diagram show, they are fully integrated: all software runs in the same Java Virtual Machine (JVM), using the same Java objects through the Singleton construct. At the same time, RALOCA focuses on the objective meta-data whereas COFI processes the subjective meta-data. In order to provide the final recommendations, RALOCA combines both the predictions generated by COFI and its own content-based processing.

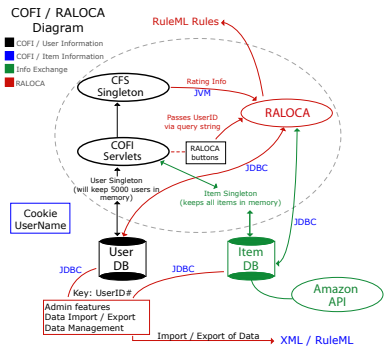

Figure 2. RACOFI, System Diagram

3.1. Information Retrieval

The first step in the information retrieval process was to collect an extensive list of Canadian artists, whose commercial recordings we wanted to have available to be rated by users. This involved the parsing of source code from two Canadian music sites which listed popular Canadian artists. Artist names were extracted from the source and stored into a text file, making sure that no duplicate entries were made.

Once the artist names were in place, it came time to harvest a list of ASIN numbers (a unique number used by Amazon.com to identify a product in their store) from Amazon.com using their external web API. A python script was used to retrieve a complete list of albums from each artist stored in the text file. During this phase any imported albums and CD singles were removed to prevent any duplicate entries.

Another check was also performed to ensure that the albums returned from the amazon search were in fact from the artist that was searched for, and not from an artist with a similar name.

3.2. COFI (Collaborative Filtering)

COFI deals with the subjective aspect of the metadata. On the front end, its major function is to gather user ratings in each of the five dimensions via a servlet-based web interface. To facilitate this process, COFI retains in memory metadata for all items, as well as that for the five thousand most recently logged in users - both subjective and objective. COFI's back end manages the interaction between the web interface, user and item metadata, and the collaborative filtering algorithms used to predict user ratings for unrated items. It is important to note that within the context of COFI, the five dimensions to be rated are entirely independent. It is only when the ratings are introduced to RALOCA that the five dimensions are merged, thus producing even more personalized and meaningful results.

The back end of COFI can be divided into three sections: the singleton which manages the CollaborativeFilteringSystem Objects, the singletons which manage objective user and item metadata, and the servlets which process information collected from the html forms which comprise COFI's front end. The singleton which manages the CollaborativeFilteringSystem algorithms is known as CFSSingleton. The CFSSingleton, along with the CFS class, permits COFI to maintain the collaborative filtering information in memory. A CFS object consists of a CollaborativeFilteringSystem object for each of the algorithms in use. The CFSSingleton keeps in memory a CFS object for each of the five rating dimensions (impression, lyrics, music, originality, and production,) allowing COFI to update predictions in memory rather than recalculating them every time a new rating is entered. This results in improved speed and memory usage.

The user metadata is read initially from the MySQL database upon start up, and upheld by the UserSingleton. When COFI first connects with the database, the 5000 most recently logged in users (as listed by timestamp) are uploaded into memory. Thereafter, the 5000 user maximum is sustained. Should a new user be created, the eldest user in memory is removed and the new user added. An existing user who logs in and can be found in the database but not in memory is treated in a similar manner. Should a user already in memory log in, they are removed from their posi-

tion in the queue, their timestamp is updated in the database, and they are re-added to the list as the most recent user. This queue of users is maintained using a LinkedHashMap data structure. The Hash Map properties of this data structure allow for quick access of information by keyed reference, while the linked list aspect tracks the order by which the users were added to the list and allows for easy removal of the eldest element.

Moving toward the front end are the java servlets which process information from the html forms. The servlets also serve as the bulk of the front end, dynamically creating html pages. The servlets act to create new users, process logins, and add new ratings. They also display lists of artists in the database, albums already rated, top recommendations, and the current top ten albums as rated by all users.

3.3. RALOCA (Rule Applying Learning Object Comparison Agent)

RALOCA is a rule based tool for the selection and comparison of objects (e.g., products) that are rated in multiple dimensions. This rule system – using OO jDREW and Object-Oriented RuleML [http://www.ruleml.org/indoo] – allows for high flexibility when selecting which objects to rank, and how to rank them. RALOCA lets us select objects based upon search criteria that the objects' metadata must meet, and permits rules that operate upon data (facts) about the objects and the user of the system.

Rules can also be added to the system to perform dynamic incremental modification of the product data, based upon the product information and information about the user. This functionality is currently used in two ways within the RACOFI system. The first is to add the appropriate taxes for a user's location to the cost of the objects; the second is to adjust the predicted ratings to better match the preferences of the user.

A weaknesses of collaborative filtering systems when used by themselves is that all objects are treated as though they were independant. One advantage of using a rule based system like RALOCA on top of a collaborative filtering system is that adjustments can be made to the predictions based upon correlations between the objects, such as a common artist/author or genre.

In general, "Users like and prefer to buy previously familiar recommendations" [14]. Therefore, it is necessary to tailor the results given by the collaborative filtering system to the user, by giving preference to objects the user would have familiarity with, reinforcing their trust in the system.

In our system this is done by adjusting the predicted ratings from the collaborative filtering system via rules. If the user rates an album by a particular artist highly, the predicted ratings for other albums by this artist are adjusted to be higher. Similarily, if a user rates an album low, other albums by this artist are adjusted to have a lower predicted rating. Currently, the amount of the adjustments to the predicted rating is up to 1 rating unit. Further studies would need to be done before trying higher adjustment values.

For example, in our system

> **If** a user rates 9 the originality of an album by artist X **then** the predicted originality rating, for this user, of all other albums by artist X is increased by a value of 0.5.

The system could be expanded so that similar correlations and adjustments are made based upon the genre of the albums. The entire adjustment rule set currently used in RACOFI is included as appendix A.1.

RALOCA is able to exclude objects from the RACOFI result listings using rules that process information known about the objects present in the database, and any information that is known about the RACOFI user. This functionality can be used to exclude objects from appearing to certain users. For example, a set of rules can be defined so that certain objects are only offered to those within certain provinces or countries; or, so that objects that have prerequisite requirements are offered only to those users who meet all these requirements for the objects.

The scoring system combines into a single value (predicted) rankings in multiple dimensions that are provided by COFI with weights representing the users' preferences. This single value can then be used to sort the results into a total order. The scoring system can be called in two modes, computing a normalized weighted sum either via a set of rules or via a hardcoded Java program. The 'rule' mode permits the scoring system to be easily customized at runtime, without having to modify and recompile the Java code. The 'hardcoded' mode permits to use a more efficient Java implementation if flexibility is not required.

The rule sets for RACOFI are included as appendix A.

4. Experience Report

During an informal opening on August 12th, 2003, the RACOFI system was presented and demoed to a selected subset of the New-Brunswick e-learning community. The feedback led us to improvements in the user interface, information submission form, and the dimension definitions.

As stated earlier, the more ratings a user enters (and the more accounts created), the recommendation algorithms become more effective. Up to date, the system has been live for two weeks. The database has already grown to approximately a thousand ratings, with over a hundred users.

At the algorithm level, experience has shown us that it is necessary to verify whether a user has enough ratings to attempt a personalized recommendation. Empirically, we found that setting a threshold of 3 rated items before we use

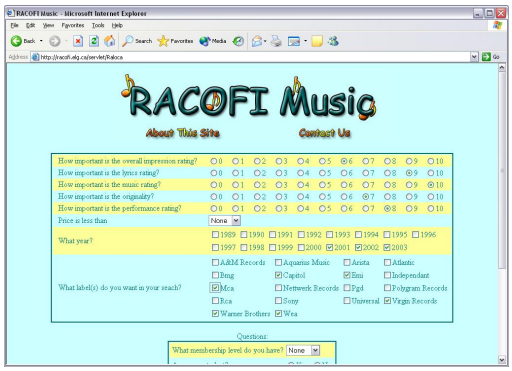

Figure 3. The RACOFI interface: users are given the option to give weights to the different dimensions based on the level of importance. Users can also select various fields from which to select specific data and select from three different algorithms: $STIN2$ (the one selected in this case), the $STI\,Pearson$, and the Per Item Average. The results and computational complexity varies depending on the algorithm.

our algorithms worked well, and otherwise we fall back on the Per Item Average. We currently do not have enough data to measure the accuracy of our algorithms on our own data set, but we expect the simpler algorithm ($STIN2$) to currently outperform $STI\,Pearson$ due the relative small size of our data set.

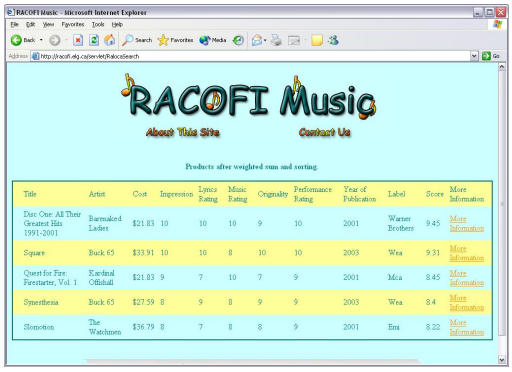

Figure 4. Results from the query in Fig.3. These personalized recommendations are optimized for the current user.

5. Conclusion

As part of the completion of Phase II of the Sifter NRC project, we are planning to transform RACOFI into a fully generic rating system beyond Canadian music. In theory, RACOFI can be molded to represent any given scenario, if the need for a recommender system arises even though we expect challenges.

Among the interesting research questions is the ability of RACOFI to scale as we plan to apply the system to databases of over 30,000 learning objects with a large number of possible attributes. It is likely that we will need to leverage the hierarchical nature of such databases in order to keep the problem tractable.

A. RACOFI Rule Listing

Rules of the form

If $condition_1$ & ... & $condition_n$ **then** $action$

are represented in our PR syntax[2] as

$action$:- $condition_1, \ldots, condition_n$.

where $action$ and $condition_i$ use "->"-infixed role->filler slots and "?"-prefixed ?variables. Internally, this becomes XML based OO RuleML (see section 2.5). The sample rule of section 3.3 is the first `modify` rule in appendix A.1.1.

A.1. Modification Rules

A.1.1. Adjustment rules

```
modify(amount->"0.5";
       comment->"Adjusting originality rating (by 0.5)
                 for high ratings of other albums
                 by this artist.";
       variable->originality;
       product->?item)
   :-
     rating(itemID->?item2;originality->"9.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"1";
       comment->"Adjusting originality rating (by 1)
                 for high ratings of other albums
                 by this artist.";
       variable->originality;
       product->?item)
   :-
     rating(itemID->?item2;originality->"10.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-0.5";
       comment->"Adjusting originality rating (by -0.5)
                 for low ratings of other albums
                 by this artist.";
       variable->originality;
       product->?item)
   :-
```

2 http://www.ruleml.org/submission/ruleml-shortation.html

```
     rating(itemID->?item2;originality->"1.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-1";
       comment->"Adjusting originality rating (by -1)
                 for low ratings of other albums
                 by this artist.";
       variable->originality;
       product->?item)
   :-
     rating(itemID->?item2;originality->"0.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"0.5";
       comment->"Adjusting impression rating (by 0.5)
                 for high ratings of other albums
                 by this artist.";
       variable->impression;
       product->?item)
   :-
     rating(itemID->?item2;impression->"9.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"1";
       comment->"Adjusting impression rating (by 1)
                 for high ratings of other albums
                 by this artist.";
       variable->impression;
       product->?item)
  :-
     rating(itemID->?item2;impression->"10.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-0.5";
       comment->"Adjusting impression rating (by -0.5)
                 for low ratings of other albums by
                 this artist.";
       variable->impression;
       product->?item)
   :-
     rating(itemID->?item2;impression->"1.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-1";
       comment->"Adjusting impression rating (by -1)
                 for low ratings of other albums
                 by this artist.";
       variable->impression;
       product->?item)
   :-
     rating(itemID->?item2;impression->"0.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"0.5";
       comment->"Adjusting lyrics rating (by 0.5)
                 for high ratings of other albums
                 by this artist.";
       variable->lyrics;
       product->?item)
   :-
     rating(itemID->?item2;lyrics->"9.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"1";
       comment->"Adjusting lyrics rating (by 1)
                 for high ratings of other albums
                 by this artist.";
       variable->lyrics;
       product->?item)
   :-
     rating(itemID->?item2;lyrics->"10.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-0.5";
       comment->"Adjusting lyrics rating (by -0.5)
                 for low ratings of other albums
                 by this artist.";
       variable->lyrics;
       product->?item)
   :-
     rating(itemID->?item2;lyrics->"1.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-1";
       comment->"Adjusting lyrics rating (by -1)
                 for low ratings of other albums
                 by this artist.";
       variable->lyrics;
       product->?item)
   :-
     rating(itemID->?item2;lyrics->"0.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"0.5";
       comment->"Adjusting music rating (by 0.5)
                 for high ratings of other albums
                 by this artist.";
       variable->music;
       product->?item)
   :-
     rating(itemID->?item2;music->"9.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"1";
       comment->"Adjusting music rating (by 1)
                 for high ratings of other albums
                 by this artist.";
       variable->music;
       product->?item)
   :-
     rating(itemID->?item2;music->"10.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-0.5";
       comment->"Adjusting music rating (by -0.5)
                 for low ratings of other albums
                 by this artist.";
       variable->music;
       product->?item)
   :-
     rating(itemID->?item2;music->"1.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-1";
       comment->"Adjusting music rating (by -1)
                 for low ratings of other albums
                 by this artist.";
       variable->music;
       product->?item)
   :-
     rating(itemID->?item2;music->"0.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"0.5";
       comment->"Adjusting performance rating (by 0.5)
                 for high ratings of other albums
                 by this artist.";
       variable->performance;
       product->?item)
   :-
     rating(itemID->?item2;performance->"9.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).
```

```
modify(amount->"1";
       comment->"Adjusting performance rating (by 1)
                 for high ratings of other albums
                 by this artist.";
       variable->performance;
       product->?item)
   :-
     rating(itemID->?item2;performance->"10.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-0.5";
       comment->"Adjusting performance rating (by -0.5)
                 for low ratings of other albums
                 by this artist.";
       variable->performance;
       product->?item)
   :-
     rating(itemID->?item2;performance->"1.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).

modify(amount->"-1";
       comment->"Adjusting performance rating (by -1)
                 for low ratings of other albums
                 by this artist.";
       variable->performance;
       product->?item)
   :-
     rating(itemID->?item2;performance->"0.0"!?REST0),
     product(itemID->?item2;artist->?artist!?REST1),
     product(itemID->?item;artist->?artist!?REST2).
```

A.1.2. Tax Rules

```
tax(amount->"%15";
    comment->"15 percent HST")
   :-
     location(nb).
tax(amount->"%15";
    comment->"15 percent HST")
   :-
     location(ns).
tax(amount->"%15";
    comment->"15 percent HST")
   :-
     location(nl).
tax(amount->"%7";
    comment->"7 percent GST")
   :-
     location(pe).
tax(amount->"%10";
    comment->"10 percent PST")
   :-
     location(pe).
tax(amount->"%0";
    comment->"Unknown taxes for people
              located outside Canada.")
   :-
     location(notCanada).
tax(amount->"%7";
    comment->"7 percent GST")
   :-
     location(al).
tax(amount->"%7";
    comment->"7 percent GST")
   :-
     location(bc).
tax(amount->"%7";
    comment->"7 percent PST")
   :-
     location(bc).
tax(amount->"%7";
    comment->"7 percent GST")
   :-
     location(ma).
tax(amount->"%7";
    comment->"7 percent PST")
   :-
     location(ma).
tax(amount->"%7";
    comment->"7 percent GST")
   :-
     location(nw).
tax(amount->"%7";
    comment->"7 percent GST")
   :-
     location(nv).
tax(amount->"%7";
    comment->"7 percent GST")
   :-
     location(on).
tax(amount->"%8";
    comment->"8 percent PST")
   :-
     location(on).
tax(amount->"%7";
    comment->"7 percent TPS")
   :-
     location(pq).
tax(amount->"%7.5";
    comment->"7.5 percent TVQ")
   :-
     location(pq).
tax(amount->"%7";
    comment->"7 percent GST")
   :-
     location(sk).
tax(amount->"%7";
    comment->"7 percent PST")
   :-
     location(sk).
tax(amount->"%7";
    comment->"7 percent GST")
   :-
     location(yk).
modify(amount->?Amt;
       comment->?Cmt;
  variable->price;
  product->ANY)
   :-
     tax(amount->?Amt;
    comment->?Cmt).
```

A.2. Elimination Rules

```
NotOffered(itemID->?itemID)
   :-
     student(yes),
     product(price->?price;itemID->?itemID!?REST),
     $lt(?price,22,false).

NotOffered(itemID->?itemID)
   :-
     userLevel(beginner),
     product(itemID->?itemID;impression->?IMP!?REST),
     $lt(?IMP,7,true).

NotOffered(itemID->?itemID)
   :-
     userLevel(beginner),
     product(itemID->?itemID;music->?IMP!?REST),
     $lt(?IMP,7,true).

NotOffered(itemID->?itemID)
   :-
     userLevel(beginner),
     product(itemID->?itemID;lyrics->?IMP!?REST),
     $lt(?IMP,7,true).

NotOffered(itemID->?itemID)
   :-
     userLevel(beginner),
     product(itemID->?itemID;performance->?IMP!?REST),
     $lt(?IMP,7,true).
```

```
NotOffered(itemID->?itemID)
   :-
     userLevel(beginner),
     product(itemID->?itemID;originality->?IMP!?REST),
     $lt(?IMP,7,true).

NotOffered(itemID->?itemID)
   :-
     userLevel(intermediate),
     product(itemID->?itemID;impression->?IMP!?REST),
     $lt(?IMP,5,true).

NotOffered(itemID->?itemID)
   :-
     userLevel(intermediate),
     product(itemID->?itemID;music->?IMP!?REST),
     $lt(?IMP,5,true).

NotOffered(itemID->?itemID)
   :-
     userLevel(intermediate),
     product(itemID->?itemID;lyrics->?IMP!?REST),
     $lt(?IMP,5,true).

NotOffered(itemID->?itemID)
   :-
     userLevel(intermediate),
     product(itemID->?itemID;performance->?IMP!?REST),
     $lt(?IMP,5,true).

NotOffered(itemID->?itemID)
   :-
     userLevel(intermediate),
     product(itemID->?itemID;originality->?IMP!?REST),
     $lt(?IMP,5,true).

NotOffered(itemID->?itemID)
   :-
     product(itemID->?itemID;
             impression->"0.0";
     music->"0.0";
     lyrics->"0.0";
     performance->"0.0";
     originality->"0.0"
     !?REST).
```

Acknowledgments. Bruce Spencer has kindly supported the development of RACOFI as part of the Sifter project. We thank Jo Lumsden for her help with the web interface review, as part of the Human-Computer Interaction team of the National Research Council. Special thanks to Rodrigue Savoie, Stephen Downes, Hélène Fournier, and Chaouki Regoui, as well as to Luc Belliveau for his technical assistance with server set-up and maintenance. Our discussions with Terry Matthews from KnowledgePool Canada helped us jump-starting RACOFI. The authors would like to thank Compaq Research and professor Ken Goldberg for access to the MovieLens and Jester data sets respectively.

References

[1] V. C. Bhavsar, H. Boley, and L. Yang. A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in e-Business Environments. In *Proc. Business Agents and the Semantic Web (BASeWEB) Workshop*, pages 53–72. NRC 45836, June 2003.

[2] H. Boley. Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses, and Order-Sorted Terms. In *Proc. Rules and Rule Markup Languages for the Semantic Web (RuleML-2003)*. Sanibel Island, Florida, LNCS 2876, Springer-Verlag, Oct. 2003.

[3] H. Boley, S. Tabet, and G. Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proc. Semantic Web Working Symposium (SWWS'01)*, pages 381–401. Stanford University, July/August 2001.

[4] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. Technical report, Microsoft Research, 1998.

[5] M. Dean and G. Schreiber. OWL Web Ontology Language – Reference. W3C Candidate Recommendation, W3C, August 2003.

[6] S. Downes. Learning objects: Resources for distance education worldwide. International Review of Research in Open and Distance Learning, 2001.

[7] S. Downes. Topic representation and learning object metadata. http://www.downes.ca/cgi-bin/website/view.cgi?dbs=Article&key=1011985735, last checked on 6/8/2003, 2002.

[8] K. Goldberg et al. Jester 2.0 - jokes for your sense of humor. http://shadow.ieor.berkeley.edu/humor/, last checked on 1/8/2003, 2001.

[9] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[10] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. of Research and Development in Information Retrieval*, 1999.

[11] D. Lemire. Scale and translation invariant collaborative filtering systems. To appear in Information Retrieval, 2003.

[12] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proc. ACM Computer Supported Cooperative Work*, pages 175–186, 1994.

[13] B. Spencer. The Design of j-Drew: A Deductive Reasoning Engine for the Web. In *Joint CoLogNet Workshop on Component-based Software Development and Implementation Technology for Computational Logic Systems of LOPSTR '02*. Technical University of Madrid, Spain, September 2002.

[14] K. Swearingen and R. Sinha. Interaction design for recommender systems. In *Designing Interactive Systems 2002*, 2002.