A Case-Based Framework for Collaborative Semantic Search in Knowledge Sifter

Larry Kerschberg[1], Hanjo Jeong[1], Yong Uk Song[2], and Wooju Kim[3]

[1] E-Center for E-Business and Department of Computer Science
Volgenau School of Information Technology and Engineering
George Mason University, Fairfax, Virginia USA
{kersch,hjeong}@gmu.edu
http://eceb.gmu.edu/
[2] Department of MIS, Yonsei University Wonju Campus, Kangwon, Korea
yusong@yonsei.ac.kr
[3] Department of Information and Industrial Engineering
Yonsei University, Seoul, Korea
wkim@yonsei.ac.kr

Abstract. This paper addresses the role of case-based reasoning in semantic search, and in particular, as it applies to Knowledge Sifter, an agent-based ontology-driven search system based on Web services. The Knowledge Sifter architecture is extended to include a case-based methodology for collaborative semantic search, including case creation, indexing and retrieval services. A collaborative filtering methodology is presented that uses stored cases as a way to improve user query specification, refinement and processing.

Keywords: Agents, Semantic Search, Collaborative Filtering, Case-Based Frameworks, Knowledge Sifter.

1 Introduction

This paper addresses an important problem, that of assisting users in posing queries to multiple heterogeneous sources over the Internet and the World Wide Web. There is a *semantic mismatch* between how a person conceptualizes a query and how that query must be expressed using the limited keyword-based query interfaces of traditional search engines. This "semantic mismatch" has been addressed by WebSifter [1]; it performs a preprocessing step in which the user develops a semantic taxonomy tree of concepts – terms and their synonyms – which are then transformed into queries submitted to traditional search engines. The resulting best matches from the individual search engines are then rated by means of a multi-attribute decision model that associates weights to the syntactic, semantic, categorical and authoritative components of each page retrieved. The results are presented to the user who then has the opportunity to rate those URLs that best match his or her requirements. WebSifter served as the preferred embodiment for a recently-awarded patent [2].

R.O. Weber and M.M. Richter (Eds.): ICCBR 2007, LNAI 4626, pp. 16–30, 2007.

Knowledge Sifter [3] is the successor to WebSifter in that the lessons learned in designing and building WebSifter have been used to create an agent-based system that coordinates the search for knowledge in heterogeneous sources, such as the Web, semi-structured data, relational databases and the emerging Semantic Web.

This paper begins with an overview of the Knowledge Sifter (KS) agent-based architecture. The artifacts created by the agents during the formulation, refinement, processing and results ranking of a user query are captured and described in terms of a meta-schema. The artifacts can be indexed and stored in a repository as user-cases. A case-based framework is presented for specifying, storing, retrieving and recommending user-cases to assist in query formulation, recommendation and processing. The cases are represented in terms of an XML schema, are stored in a case repository and are managed by a case management agent. Finally, an algorithm is presented that uses a hybrid approach which combines both content-based and collaborative filtering techniques.

2 The Knowledge Sifter Agent-Based Architecture

The Knowledge Sifter project, underway at George Mason University, has as its primary goals: 1) to allow users to perform ontology-guided semantic searches for relevant information, both in-house and open-source; 2) to access heterogeneous data sources via agent-based knowledge services; and 3) to refine searches based on user feedback. Increasingly, users seek information from open sources such as the Web, XML-databases, relational databases and the emerging Semantic Web. The Knowledge Sifter project makes use of open standards for both ontology construction – the Web Ontology Language (OWL) – and for searching heterogeneous data sources – Web services. The Knowledge Sifter (KS) architecture, depicted in Fig. 1, may be considered a service-oriented architecture consisting of a community of cooperating agents. The rationale for using agents to implement intelligent search and retrieval systems is that agents can be viewed as autonomous and proactive.

The information domain we address is that of Image Analysis, but multiple ontologies and domains can be supported. The architecture has three layers: User Layer, Knowledge Management Layer and Data Sources Layer. Specialized agents reside at the various layers and perform well-defined functions. They support interactive query formulation and refinement, query decomposition, query processing, result ranking and presentation. The KS architecture is general and modular so that new ontologies[4] and new information resources can be incorporated easily, in almost a "plug-and-play" fashion. The various KS agents and services are presented below.

User and Preferences Agents. The User Agent interacts with the user to elicit user preferences that are managed by the Preferences Agent. These preferences include the relative importance attributed to terms used to pose queries, the user-perceived authoritativeness of Web search engine results, the biases a user has towards data sources, etc., used by the Ranking Agent. The Preferences

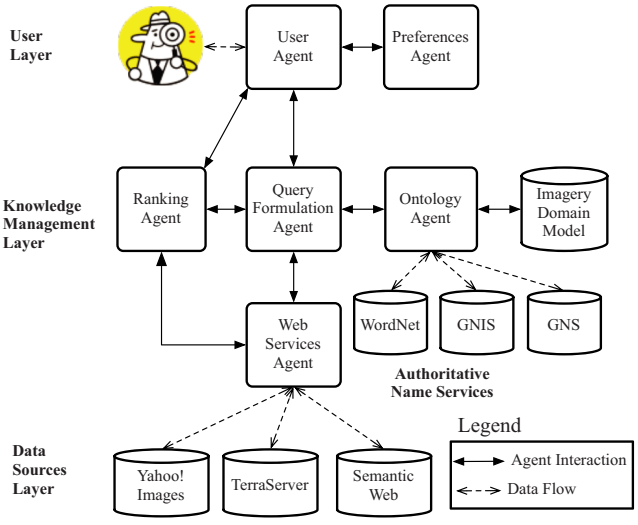

Fig. 1. The Knowledge Sifter Agent-Based Architecture

Agent can also learn the user's preference based on experience and feedback related to previous queries.

Ontology Agent. The Ontology Agent accesses an imagery domain model, specified in OWL, and depicted in Fig. 2. In addition, there are three authoritative name services: Princeton University's WordNet [5], the US Geological Survey's GNIS, and the National Geospatial-Intelligence Agency's GNS. They allow the Ontology Agent to use terms provided by the name services to suggest query refinements such as generalization, specialization and synonyms. For example, WordNet can provide a collection of synonyms for a term, while GNIS and GNS translate a physical place name – in the US and the World, respectively – into latitude and longitude coordinates that are required by a data source such as TerraServer. Other appropriate name and translation services can be added in a modular fashion, and the domain model can be updated to accommodate new concepts and relationships.

Authoritative Name Services. The three name services are WordNet, GNIS and GNS. When the initial query instance, specifying a person, place, or thing, is sent to the Ontology Agent, it then consults WordNet to retrieve synonyms. The synonyms are provided to the Query Formulation Agent to request that the user select one or more synonyms. The decision is communicated to the Ontology Agent which then updates the appropriate attribute in the instantiated version of the OWL schema. If the attribute value is the name of a class of type place then the Ontology Agent passes the instance to the both GNIS and GNS. These take the place name as input and provide the latitude-longitude coordinates as output. This information can then be communicated to the Query Formulation

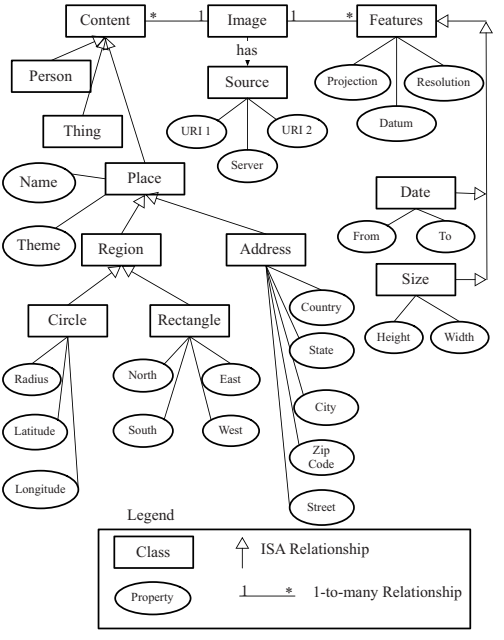

Fig. 2. Imagery Ontology Schema in Unified Modeling Language Notation

Agent which then forwards the information in the reformulated queries to the Web Services Agent for processing.

Query Formulation Agent. The User Agent poses an initial query to the Query Formulation Agent. This agent, in turn, consults the Ontology Agent to refine or generalize the query based on the semantic mediation provided by the available ontology services. Once a query has been specified by means of interactions among the User Agent and the Ontology Agent, the Query Formulation Agent decomposes the query into subqueries targeted for the appropriate data sources. This involves semantic mediation of terminology used in the domain model ontology and name services with those used by the local sources. Also, query translation is needed to retrieve data from the intended heterogeneous sources.

Web Services Agent. The main role of the Web Services Agent is to accept a user query that has been refined by consulting the Ontology Agent, and decomposed by the Query Formulation Agent. The Web Services Agent is responsible for the choreography and dispatch of subqueries to appropriate data sources, taking into consideration such facets as: user preference of sites; site authoritativeness and reputation; service-level agreements; size estimates of subquery responses; and quality-of-service measures of network traffic and dynamic site workload [6].

Ranking Agent. The Ranking Agent is responsible for compiling the sub-query results from the various sources, ranking them according to user preferences, as

supplied by the Preferences Agent, for such attributes as: 1) the authoritativeness of a source which is indicated by a weight – a number between 0 and 10 – assigned to that source, or 2) the weight associated with a term comprising a query.

Data Sources and Web Services. At present, Knowledge Sifter consults two data sources: Yahoo Images and the TerraServer. Yahoo Images supports Representational State Transfer (REST)-based [7] web services which simply returns XML result data over HTTP. Yahoo Images supports the name and description for images; this allows the Ranking Agent to perform more precise evaluation for the semantic criteria. The Ranking Agent also uses the size of images contained in Yahoo Images metadata to filter images based on user preference, but the metadata does not contain the creation time of images which is a good measure of temporal aspect.

3 Emergent Semantics in Knowledge Sifter

This section presents some notions related to *emergent behavior and patterns* that arise from 1) the functioning of Knowledge Sifter, and 2) the use of composable Web services to create reusable search frameworks. This topic is discussed in detail in [3], so we present an overview here. Our approach to Emergent Semantics in Knowledge Sifter is to collect, index, organize and store significant artifacts created during the end-to-end workflow for KS. The KS workflow manages the entire search process, including, query specification, query reformulation, query decomposition, web service selection, data source selection, results ranking and recommendation presentation.

By stepping back and abstracting the agents, classes, their relationships and properties, one can construct the Knowledge Sifter Meta-Model (KSMM) [3]. Fig. 3 depicts the UML Static Model for the KSMM. What follows is a brief overview of the classes and relationships depicted in Fig. 3.

At the top is the Class Agent, which is specialized to those agents in the KS architecture, specifically the UserAgent, PreferencesAgent, OntologyAgent, QueryFormulationAgent, RankingAgent and WebServicesAgent. These agents manage their respective object classes, process specifications, and WebServices. For example, the UserAgent manages the User Class, the UserInterfaceScenario, the User PatternMiningAlgorithm, and the WebServices. The User specifies User Preferences that can be specialized to Search Preferences and Source Preferences. The User poses UserQuery that has several QueryConcept, which in turn relates to an OntologyConcept. The Ontology Agent manages both the UserQuery and the OntologyConcept that is provided by an OntologySource. Both OntologySource and DataSource are specializations of Source. Source is managed by the WebServicesAgent and has attributes such as provenance, coverage, access protocol and history. DataSource has attributes such as Quality-of-Service Service-Level-Agreements (QoS-SLAS) and Certificate.

A UserQuery consists of several RefinedQuery, each of which is posed to several DataSource. DataSource provides one-or-more DataItem in response to a RefinedQuery as the QueryResult. Based on the returned QueryResult, the User

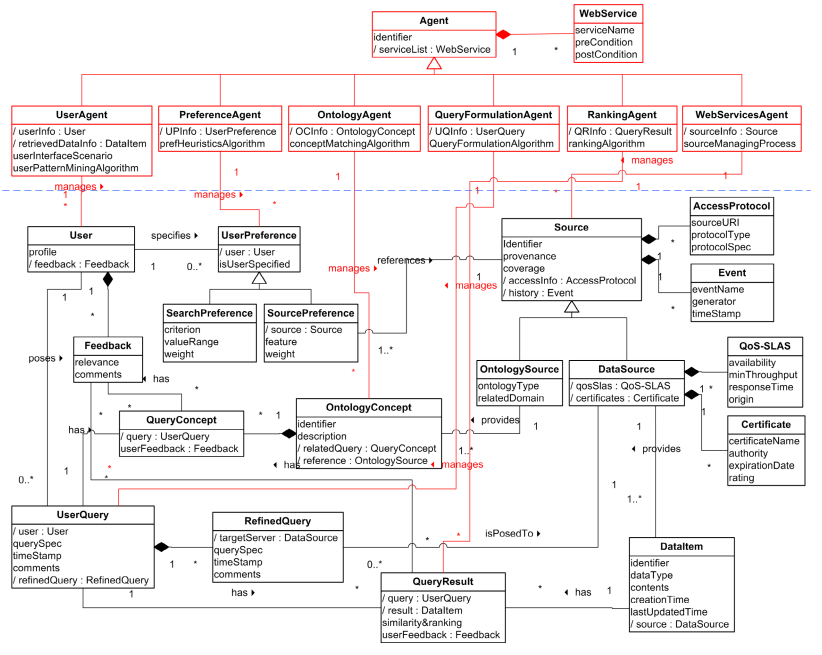

Fig. 3. Knowledge Sifter Meta-Model Schema in UML Notation

may provide Feedback as to the result relevance and other comments. These may impact the evolution of metadata associated with UserPreference, query formulation, data source usage and result ranking. The KSMM can be implemented as a relational database schema, which can be used to organize, store and interrelate the artifacts associated with a user query. The data can then be mined emergent properties related to the use of Knowledge Sifter resources.

4 Case-Based Knowledge Sifter Framework

The original Knowledge Sifter [3] creates a repository of user queries and artifacts produced during the search process. In this section, a case-based framework is proposed for KS in order to recommend query specifications and refinements based on the previously-stored user-query cases. A user query case is generated only when a user provides relevance feedback for results returned for a query. The user feedback is the user's evaluation of the degree of relevance of a result to the refined query; e.g., highly relevant; relevant; highly not relevant, or unclear. This relevance feedback can also be regarded as a user rating of the result's information quality.

The role of the Case Management Agent in Fig. 4 is to communicate with the User Agent, and to obtain cases from the User Query Case Base that have user feedback annotations. The Query Formulation Agent communicates with the

Case Management Agent to retrieve cases according to a user query and user preferences. To efficiently retrieve cases, the Case Management Agent maintains ontology-based indices to cases as described in Sect.4.2. From the retrieved cases, a refined query with data source information will be selected using a collaborative filtering approach which is described in Sect.4.3. KS also maintains pre-compiled component repository for accessing data sources for each information domain such as places, music, movies, scholarly papers, etc. Based on the collaborative filtering approach, KS semi-automatically selects data sources and is dynamically configured with Web Services-based wrapper components for each selected data source.

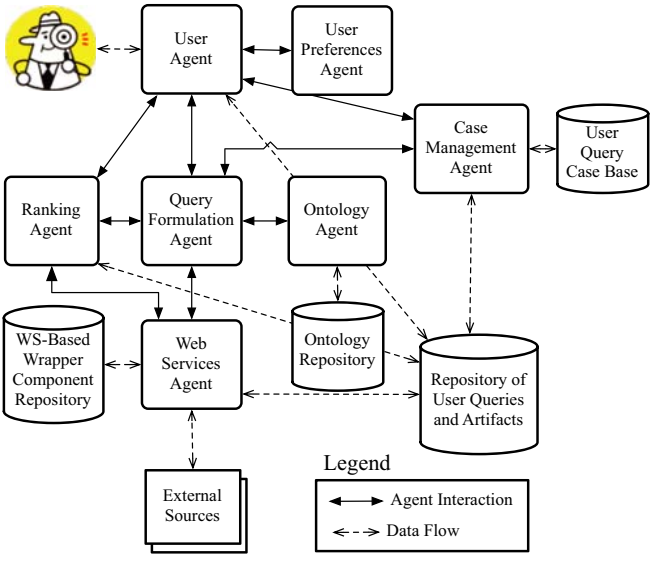

Fig. 4. Knowledge Sifter Case-Based Framework

4.1 Semantic Case Representation

Case-based Knowledge Sifter maintains cases representing a user query and its artifacts; these are required to recommend a refined query for each user-selected information domain. Fig. 5 shows an XML-based structure for the case representation. A case contains a username to identify its user, and this user identifier will be used to perform collaborative filtering and to retrieve the user's preferences. Also, each case has an associated user query and multiple refined queries, because KS generates a refined query for each information domain.

A user query can have multiple concepts which consist of a user term, multiple ontology references, and a weight. For example, suppose one wishes to visit the Washington Monument and then dine at a steakhouse in DC, then the keyword terms in a query might be "Washington monument" and "steakhouse". The ontology reference is a concept identifier in an ontology which contains the

concept. WordNet is employed as a general upper ontology and several domain-specific ontologies such as places, restaurants, and wine can be linked and used to represent user concepts. This referenced ontology concept serves as an index of the user query as described in Sect.4.2. The concept weight is a degree of importance the user assigns to a concept. A refined query has exactly one information domain for which the query is specified. The refined query is a weighted multi-dimensional/multi-valued query as represented in Fig. 5. The feature name is also a variable since the schema of a refined query will be determined by its information domain and the user-selected data source. The data source information is also a feature of the refined query and it can be represented as $FeatureName : data-source, FeatureValue : imdb.com$, where IMDB denotes the Internet Movie Data Base.

Thus, a feature can be not only content-based metadata, but also metadata created during on the information object's life-cycle[9]. The feature name may be standardized in the scope of KS to remove the ambiguity which can occur during the search and recommendation processes described in Sect.4.3. Some standardized metadata such as Dublin Core Metadata can be used to describe feature attributes.

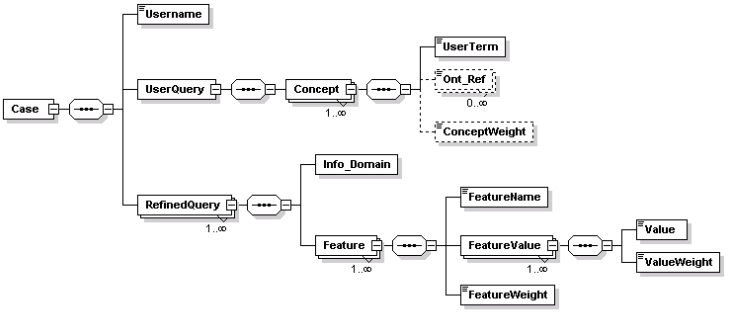

Fig. 5. XML-Based Semantic Representation of a User Case

4.2 Case Retrieval Via Ontology-Based Indices

The Case Management Agent maintains ontology-based indices for entire cases. As represented in Fig. 5, each user term of the query concept can have referenced ontology concepts. For each ontology concept, case identifiers referencing the ontology concept can be stored as the indices. Fig. 6 represents a simple index structure for an ontology which has an ontology identifier, several concept indices consisting of ontology concept identifiers of that ontology, and case identifiers for each of the ontology concepts. This approach allows for efficient retrieval of similar cases because it explores related ontology concepts first, rather than navigating a large number of the user query cases. Fig. 7 represents an algorithm for retrieving cases similar to the user query via ontology-based indices. First, the algorithm generates expanded queries of every possible combinations of

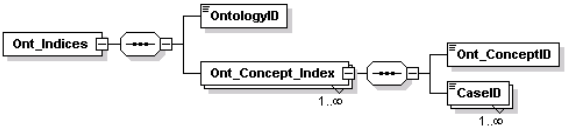

Fig. 6. XML-Schema for Ontology Index

concepts, including their equivalent and generalized concepts. For example, a user query {Washington Monument, steakhouse} can be expanded via ontology navigation as: {Washington Monument, chophouse}{Washington Monument, restaurant}{DC, steakhouse}, etc. The DC concept is obtained from WordNet through the "Part Holonym" relationship of the "Washington Monument" concept to the "DC" concept, and this can be regarded as a *spatial generalization*.

The algorithm then retrieves cases which are indexed by all the concepts of an expanded query, but limiting the number of the cases to a prespecified maximum. For efficiency purposes, whether the required number of cases are retrieved or not will be checked before expanding one element query of powerset of the user query because the expanded queries cannot be more similar to the user query than the original element query. The weighted sum of each query can be calculated from (1). The $sim(C_a, C_i)$ in the algorithm is a similarity between the expanded user query of the active case and the user query of the retrieved cases using cosine correlation which is widely used for the vector model in IR [8] as defined in (2). This similarity will be used in Sect. 4.3 as the similarity between the active case and its similar cases in terms of the similarity of their user queries. Note that the original user query of the active case is also one of the expanded user queries.

$$w(uq_i) = \sum_j tw(c_{ij}) * uw(c_{ij}) \tag{1}$$

$$tw(c_{ij}) = \begin{cases} 1.0 & \text{if } c_{ij} \text{ is a user concept} \\ syw & \text{if } c_{ij} \text{ is an equivalent concept of the user concept} \\ hyw & \text{if } c_{ij} \text{ is a generalized concept of the user concept} \\ syw * hyw & \text{if } c_{ij} \text{ is a generalized concept of the equivalent concept} \end{cases}$$

where uq_i represents a user query for case i and $tw(c_{ij})$ represents a predefined weight for the type of j^{th} concept in uq_i. The terms syw and hyw denote the predefined weight for an equivalent (synonym) concept and a generalized (hypernym) concept, respectively. The term $uw(c_{ij})$ is a user defined weight for a concept c_{ij}.

$$sim(C_a, C_i) = \frac{\sum_{j \in EQ_a} cw_{aj} \cdot cw_{ij}}{\sqrt{\sum_{j \in EQ_a} cw_{aj}^2} \cdot \sqrt{\sum_{j \in EQ_a} cw_{ij}^2}} \tag{2}$$

where cw_{aj} and cw_{ij} represent the weights of j^{th} concept in the expanded user query of the active case EQ_a and the user query of the retrieved case respectively.

```
Input: the active user query uq_a
Output: a number (maxnc) of cases similar to the user query

maxnc ← 10
A set of retrieved cases RCS ← the Empty Set
A set of expanded queries EQS ← the Empty Set
CQS ← the powerset of uq_a except the empty set
Sort elements of CQS in descending order of their weights
FOREACH cq in CQS
 IF #RCS < maxnc THEN
  FOREACH concept in cq
   EC ← a set containing the concept and its equivalent concepts
  END
  CPECS ← Cartesian product of EC sets
  FOREACH cpec in CPECS
   FOREACH concept in cpec
    HC ← a set containing the concept and its generalized concepts
   END
   CPHCS ← Cartesian product of HC sets
   FOREACH cphc in CPHCS
    WeightOfeq ← a weighted sum of concept weights in cphc
    Add cphc to EQS
   END
  END
  Sort elements of EQS in descending order of WeightOfeq
  FOREACH eq in EQS
   If WeightOfeq > WeightOfNextcq THEN
    Remove eq from EQS
    CASES ← a set of cases indexed by every concept in eq
    IF #RCS < maxnc THEN
     FOREACH case in CASES
      sim(C_a,C_i) ← a cosine similarity of eq and case's user query
     END
     Sort elements of CASES in descending order of sim(C_a,C_i)
     FOREACH case in CASES
      IF #RCS < maxnc THEN
       Add case to RCS
     END
  END
END
```

Fig. 7. Case Retrieval Algorithm via Ontology Index

4.3 Collaborative Incremental Query Specification

Content-based filtering is a method for recommending unseen items to a user based on the contents of items they have already seen and are stored in their profile. It can assist users in refining a query based on the artifacts of their past queries which are similar to the active query. However, similar queries may not yet exist in the active user's profile, or the acceptable number of the user-preferred data items cannot be easily obtained because of insufficient feedback data provided thus-far.

This situation is ameliorated by using collaborative filtering, which attempts to predict usefulness of as yet unseen items for an active user, by proposing items based on those previously rated by other users. The basic idea is to recommend a set of unseen items that are preferred by other users who have tastes similar to the active user. Thus, the drawbacks of content-based filtering can be addressed with a higher level of confidence.

However, collaborative filtering cannot be applied directly to our case-based KS framework because more than one user-query case, stored in the case repository, may be similar to the active user query. A better approach is to recommend a single *aggregated* refined query from the cases having a certain level of user-query similarity. Therefore, a hybrid filtering approach which combines both collaborative filtering and content-based filtering can be used effectively in this architecture. However, if there is no previously-stored user query posed by the active user in the selected similar cases, the collaborative filtering cannot be directly used for the active refined query because the recommendation of the query specification should be made before retrieving results from data sources, i.e., no user feedback on results of the query which is required for the collaborative filtering exists on the recommendation time. To address this problem, an aggregated refined query from the refined queries of the selected cases can be recommended.

The case-based KS recommends the refined query and the user confirms that this is to now be the active refined query. During this confirmation step, the user can fine-tune the query parameters, e.g., for the data source feature, the user might add or remove data sources and adjust the weights for each data source. Then, KS retrieves results from the data sources in the user-confirmed refined query by dynamically translating it to one or more queries according to each data source's schema/ontology. The active user can provide feedback on some results and can request another recommendation of the specification. At this time, collaborative filtering can be used because the artifacts of active refined query will have been stored in the case base as a new case, it can then be selected as a similar case because the case's user query would be identical to the current specification of the active user query.

Data Item Recommendation via Query-to-Query Collaborative Filtering. With the user rating values for result data items of the active refined query, the active user's rating value of unseen data items can be predicted from the results and their rating values for the active refined query and neighbor refined queries which can be found from the KS repository. The prediction can be calculated from (3) and (4) which are derived from the well-known collaborative filtering approach used in GroupLens [10].

This refined query-based collaborative filtering allows KS to show the unseen data items immediately because the data items are found in a neighbor's search history in the repository. The mismatch problem between user queries and refined queries can be alleviated by using a threshold for the similarity between the active refined query and neighbor refined query, i.e., only the neighbor refined

query having a certain high similarity value will be selected for this prediction process.

$$p_{rq_{ad},dt_u} = \overline{r_{rq_{ad}}} + \frac{\sum_{i \in NC} (r_{rq_{id},dt_u} - \overline{r_{rq_{id}}}) \cdot sim(rq_{ad}, rq_{id}) \cdot sim(C_a, C_i)}{\sum_{i \in NC} |sim(rq_{ad}, rq_{id})| \cdot sim(C_a, C_i)} \quad (3)$$

$$sim(rq_{ad}, rq_{id}) = \frac{\sum_{s \in SD} (r_{rq_{ad},dt_s} - \overline{r_{rq_{ad}}}) \cdot (r_{rq_{id},dt_s} - \overline{r_{rq_{id}}})}{\sigma_{rq_{ad}} \cdot \sigma_{rq_{id}}} \quad (4)$$

where p_{rq_{ad},dt_u} represents a prediction for an unseen (unrated) data item dt_u for the active refined query rq_{ad}. rq_{ad} represents a refined query for the active user case a for the domain d. $sim(rq_{ad}, rq_{id})$ is the correlation weight for user rating patterns of the refined queries rq_{ad} and rq_{id} as defined by the Pearson Correlation Coefficient shown in (4). $sim(C_a, C_i)$ represents the similarity between the active case C_a and a neighbor case C_i as defined in (2). NC is a set of neighbor cases selected as similar to the active case. SD is a set of common seen (rated) items between rq_{ad} and rq_{id}. $\overline{r_{rq_{ad}}}$ and $\overline{r_{rq_{id}}}$ represent mathematical means for the ratings of the result data items of the queries rq_{ad} and rq_{id}, respectively.

Incremental Refined Query Specification. The active refined query can be incrementally specified based not only on the data items rated by the active user, but also on the data items whose rating value predicted from (3) and (4). That is, the refined query can be specified by content patterns of the rated data items and a new result set can be retrieved from a new data source set. More unseen data items can be found from above collaborative filtering with the new search artifacts. Thus, the refined query can be incrementally specified by aggregating the rated and predicted data items.

At first, the value weight for each feature of the active refined query can be found from (5) and (6). Then, the feature weight can be determined by (7) and (8) which also uses the Pearson Correlation Coefficient. This is based on an idea that if the similarity value patterns for a criterion (feature) and the user rating patterns are similar, the feature would be an important factor (feature) for the user to determine his likeness on the data. Therefore, this approach also takes into account the negative examples which have a negative feedback from users whereas content-based filtering systems [11][12] consider only the positive examples to refine queries in terms of weight adjustments. Furthermore, the negative correlation weight will become zero via the $n(x)$ function because the negative correlation would not necessarily mean that the user rated a data item as an relvant one since it is dissimilar to his query in the dimension of the feature or vice versa.

$$vw_{adfv} = \frac{\overline{r_{vv_{adfv}}}}{\sum_{l=1}^{m_{adf}} \overline{r_{vv_{adfl}}}} \quad (5)$$

$$\overline{r_{vv_{adfl}}} = \frac{\sum_{m \in MD} r_{rq_{ad},dt_m} \cdot O(vv_{adfl}, dt_m)}{\sum_{m \in MD} O(vv_{adfl}, dt_m)} \tag{6}$$

$$fw_{adf} = \frac{n(sim(f_{adf}, rq_{ad}))}{\sum_{k=1}^{m_{ad}} n(sim(f_{adk}, rq_{ad}))}; \quad n(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases} \tag{7}$$

$$sim(f_{adk}, rq_{ad}) = \frac{\sum_{m \in MD} (sim(f_{adk}, dt_m) - \overline{sim(f_{adk})}) \cdot (r_{rq_{ad},dt_m} - \overline{r_{rq_{ad}}})}{\sigma_{f_{adk}} \cdot \sigma_{rq_{ad}}} \tag{8}$$

where vw_{adfv} represents the weight of the value vv_{adfv} for a feature f_{adf} of the query rq_{ad}. MD is a set of data items representing the union of the set of the seen data items and the set of predicted unseen data items.

$\overline{r_{vv_{adfv}}}$ represents an average rating value for data items in the set MD having a value vv_{adfv}. $O(vv_{adfl}, dt_m)$ is a binary variable which represents whether the data item dt_m has the value vv_{adfl}, and if yes, its value is 1, otherwise 0. $sim(f_{adk}, rq_{ad})$ represents the correlation weight between the criterion (feature) similarity and the original and predicted user ratings for the query rq_{ad}. $sim(f_{adk}, dt_m)$ represents the similarity value between the values of the query rq_{ad} and the data item dt_m in terms of the dimension of the feature f_{adk}.

Fig. 8 represents an example of the feature weight adjustment using the multiple weighted-valued query generated only from the positive examples via (3) and (4) and increased user feedback information via the query-to-query collaborative filtering. For the explanation purpose, the queries and data items in the example have only binary values for each features, but the equations surely work for the real values. The left table represents the feature vectors of the query and data items. The right table represents similarity values of the query and data items for each feature and rating values of the data items for the query. In this example, the similarity value of the query and a data item for a feature is 1 if they have same value, otherwise 0. Intuitively, the feature f_{ad1} would be regarded as an important criterion for which the user determines the relevance of the data items; therefore, it would be beneficial to have a higher weight on the feature for the efficiency of the system's automatic rating/search process. This approach would be advantageous for adjusting criterion weights for the systems of using the weighted/multi-valued query-based and heterogeneous types of values in each criterion thereby requiring different metrics for evaluating the values.

The incrementally specified query can seem to degrade the prediction ratio and efficiency of the search process because it aggregates contents of multiple data items. However, clearly it can have better recall ratio. The prediction ratio can be alleviated by using the weights so that the results can be automatically rated and sorted by a similarity measure based on the weights. The efficiency

	f_{ad1}	f_{ad2}	f_{ad3}	f_{ad4}
dt_1	1	0	1	0
dt_2	0	1	0	1
dt_3	1	1	1	0
dt_4	0	1	1	0
rq_{ad}	1	0	1	0

	dt_1	dt_2	dt_3	dt_4	$sim(f_{adk},rq_{ad})$	fw_{adk}
$sim(f_{ad1},dt_i)$	1	0	1	0	4	0.63
$sim(f_{ad2},dt_i)$	0	1	0	1	-4	0
$sim(f_{ad3},dt_i)$	1	1	1	0	2.31	0.37
$sim(f_{ad4},dt_i)$	0	1	1	0	0	0
$rating(rq_{ad},dt_i)$	1	0	1	0		

Fig. 8. An Example of Feature Weight Adjustment

problem can be caused if the refined query has more values because the number of data sources can be increased and some data sources do not provide multi-valued queries so that the refined query can be translated to a number of data source-specific queries. To address this problem, the translated queries having higher weight values can be priorly posed to a data source with a certain degree of parallel processing and the partial results can be shown to the users.

5 Conclusions

The Case-Based Knowledge Sifter framework expands on the original KS architecture by incorporating a novel XML-based index together with an indexing scheme for the efficient storage and retrieval of user-query cases. A methodology is presented for specifying, refining and processing user queries, based on a hybrid filtering approach that combines the best aspects of both content-based and collaborative filtering techniques.

The XML-based indexing scheme uses ontology-based concepts to index user-query cases. This leads to efficient algorithms for associative retrieval of relevant related cases, thereby avoiding a sequential search of the case base, as is the case in other case-based collaborative filtering systems [13][14][15].

Acknowledgments. This work was sponsored in part by a NURI from the National Geospatial-Intelligence Agency (NGA). This work was also sponsored in part by MIC & IITA through IT Leading R&D Support Project.

References

1. Kim, W., Kerschberg, L., Scime, A.: Learning for Automatic Personalization in a Semantic Taxonomy-Based Meta-Search Agent, Electronic Commerce Research and Applications (ECRA) 1, 2 (2002)
2. Kerschberg, L., Kim, W., Scime, A.: Personalizable semantic taxonomy-based search agent. USA: Patent Number 7,117,207, George Mason Intellectual Properties, Inc (Fairfax, VA) (October 3, 2006)
3. Kerschberg, L., Jeong, H., Kim, W.: Emergent Semantics in Knowledge Sifter: An Evolutionary Search Agent based on Semantic Web Services. In: Spaccapietra, S., Aberer, K., Cudré-Mauroux, P. (eds.) Journal on Data Semantics VI. LNCS, vol. 4090, pp. 187–209. Springer, Heidelberg (2006)

4. Morikawa, R., Kerschberg, L.: MAKO-PM: Just-in-Time Process Model. In: Althoff, K.-D., Dengel, A., Bergmann, R., Nick, M., Roth-Berghofer, T.R. (eds.) WM 2005. LNCS (LNAI), vol. 3782, pp. 688–698. Springer, Heidelberg (2005)
5. Miller, G.A.: WordNet a Lexical Database for English. Communications of the ACM 38(11), 39–41 (1995)
6. Menascé, D.A.: QoS Issues in Web Services. IEEE Internet Computing 72–75 (November/December 2002)
7. Fielding, R.: Architectural styles and the design of network-based software architectures. Ph. D. Dissertation, University of California at Irvine (2000)
8. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press, New York (1999)
9. Smith, J.R., Schirling, P.: Metadata Standards Roundup. IEEE MultiMedia 13(2), 84–88 (2006)
10. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An Algorithmic Framework for Performing Collaborative Filtering. SIGIR 230–237 (1999)
11. Porkaew, K., Chakrabarti, K.: Query refinement for multimedia similarity retrieval in MARS. ACM Multimedia (1), 235–238 (1999)
12. Wu, L., Faloutsos, C., Sycara, K., Payne, T.: Falcon: Feedback adaptive loop for content-based retrieval. In: Proceedings VLDB Conference, pp. 297–306 (2000)
13. Bradley, K., Smyth, B.: An Architecture for Case-Based Personalised Search. In: Funk, P., González Calero, P.A. (eds.) ECCBR 2004. LNCS (LNAI), vol. 3155, pp. 518–532. Springer, Heidelberg (2004)
14. Coyle, L., Doyle, D., Cunningham, P.: Representing Similarity for CBR in XML European Conference on Advances in Case-Based Reasoning, Spain (2004)
15. McCarthy, K., McGinty, L., Smyth, B., Salamo, M.: The Needs of the Many: A Case-Based Group Recommender System. In: Roth-Berghofer, T.R., Göker, M.H., Güvenir, H.A. (eds.) ECCBR 2006. LNCS (LNAI), vol. 4106, pp. 196–210. Springer, Heidelberg (2006)