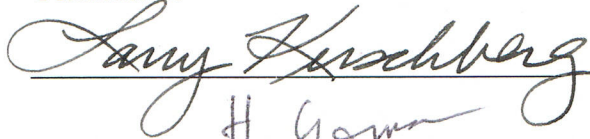


HYBRID FILTERING IN SEMANTIC QUERY PROCESSING

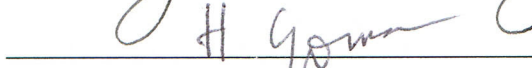
by

Hanjo Jeong
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

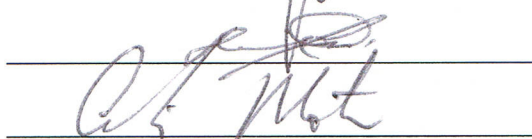
Committee:



Dr. Larry Kerschberg, Dissertation Director



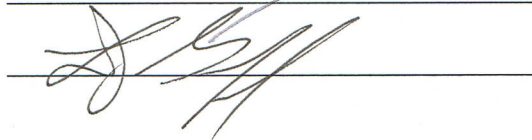
Dr. Hassan Gomaa, Committee Member



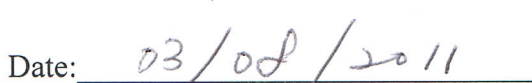
Dr. Daniel Menascé, Committee Member



Dr. Amihai Motro, Committee Member



Dr. Daniel Menascé, Senior Associate Dean



Dr. Lloyd J. Griffiths, Dean, The Volgenau
School of Information Technology and
Engineering

Date: 03/08/2011

Spring Semester 2011
George Mason University
Fairfax, VA

Hybrid filtering in Semantic Query Processing

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at George Mason University

By

Hanjo Jeong
Master of Science
George Mason University, 2003

Director: Larry Kerschberg, Professor
Department of Computer Science

Spring Semester, 2011
George Mason University
Fairfax, Virginia

Copyright 2011 Hanjo Jeong
All Rights Reserved

DEDICATION

This dissertation is dedicated to my mother Ok-Hui So and my two siblings Kwang-Jo Jeong and Young-Ok Jeong. I would also like to dedicate this dissertation in the memory of my father Chan-Young Jeong.

ACKNOWLEDGEMENTS

I would like to express my profound thanks and appreciation to my advisor Dr. Larry Kerschberg for his patient, thorough, and invaluable help for many years, from the development of the idea for this dissertation to its conclusion. I would also like to thank Dr. Hassan Gomaa, Dr. Daniel A. Menascé, and Dr. Amihai Motro for serving on my committee.

TABLE OF CONTENTS

	Page
LIST OF TABLES	viii
LIST OF FIGURES	ix
ABSTRACT	x
1 Introduction and Overview	1
1.1 Information Search on the Web and the Semantic Web	3
1.2 User Preferences in Information Search	5
1.3 Collaborative Information Search	7
1.4 Research Objectives	9
1.5 Research Hypothesis	10
1.6 Research Approach	11
1.7 Overview of Dissertation	11
2 Related Research	13
2.1 Semantic Search	13
2.1.1 Information indexing via Semantic Web	14
2.1.2 Role of the (Semantic) Web Services in Semantic Search	14
2.1.3 Role of Ontology in Semantic Search	16
2.2 Recommender Systems	17
2.2.1 Content-Based Filtering	18
2.2.2 Collaborative Filtering	19
2.2.3 Hybrid Filtering	23
3 Knowledge Sifter	24
3.1 KS Agent-Based Web Services Framework	24
3.1.1 User and Preferences Agents	26
3.1.2 Ontology Agent	27
3.1.3 Imagery Domain Model and Schema	27
3.1.4 Authoritative Name Services	29

3.1.5	Query Formulation Agent	30
3.1.6	Web Services Agent.....	30
3.1.7	Ranking Agent	31
3.1.8	Data Sources and Web Services	31
3.2	Knowledge Sifter End-to-End Scenario	32
3.3	KS Agent Interactions and Communications	40
3.4	Emergent Semantics in Knowledge Sifter	43
4	Case-Based Knowledge Sifter Framework	47
4.1	Case-Based Knowledge Sifter Architecture.....	47
4.1.1	Case Management Agent	49
4.1.2	Web Services-Based Wrapper Component Repository	51
4.1.3	Use Cases and Sequence Diagrams	52
4.2	Semantic Case Representation	57
4.2.1	Semantic Refinement of a User Query	61
4.3	Case Retrieval via Ontology-Based Indices.....	62
4.3.1	XML-based Representation of Ontology-Based Indices	63
4.3.2	Case Retrieval Algorithm via Ontology Index	64
5	Collaborative Query Refinement	69
5.1	Initial Query Refinement without User Feedback.....	73
5.2	Immediate Data-Item Recommendation from Neighbor Cases	74
5.3	Query Refinement via Query-to-Query Hybrid Filtering	77
6	Validation.....	82
6.1	Hypothesis.....	83
6.2	Experiments.....	84
6.2.1	Data Selection	85
6.2.2	Test Dataset Selection.....	87
6.2.3	Implementation	92
6.2.4	Experiments and Results.....	96
6.3	Conclusion of Validation	109
7	Conclusion	111
7.1	Contributions.....	111
7.2	Further Research	113

Appendix A: XML Schemas for Data Specification	115
A.1 XML Schema of User Query: Managed by Query Formulation Agent.....	115
A.2 XML Schema of WordNet Concept: Managed by Ontology Agent.....	115
A.3 XML Schema of Source-Specific Query for GNIS: Managed by Ontology Agent	116
A.4 XML Schema of Data Preference: Managed by Preference Agent	116
A.5 XML Schema of Search Result Retrieved from Various Sources: Managed by Web Services Agent.....	117
A.6 XML Schema of Source-Specific Query for Yahoo Image Search Engine: Managed by Web Services Agent.....	118
A.7 XML Schema of Result Data from Yahoo Image Search Engine: Managed by Web Services Agent.....	118
A.8 XML Schema of Source-Specific Query for TerraServer: Managed by Web Services Agent	118
A.9 XML Schema of Result Data from TerraServer: Managed by Web Services Agent.....	119
Appendix B: MySql Script File for KS Meta Schema.....	120
Appendix C: XML Schemas for Case-Based KS Framework.....	124
C.1 XML Schema of User Query Case.....	124
C.2 XML Schema of Ontology Index.....	125
REFERENCES	126

LIST OF TABLES

Table	Page
1. Overview of this Dissertation	12
2. A User Rating Prediction of Unseen Data Items	77
3. Example Feature Values	81
4. A Feature Weight Adjustment based on Data shown in Table 3	81
5. Sub-datasets of MLTest_d2000	90
6. Sub-datasets of MLTest_u1000	91
7. Experiment Metrics and Types	98
8. Comparison between Query-to-Query Hybrid Filtering (HF) and Content-Based Filtering (CBF) via the Spearman's rank correlation coefficient	99
9. Comparison between Query-to-Query Hybrid Filtering (HF) and Content-Based Filtering (CBF) via Precision and Recall	101
10. Comparison between Query-to-Query Hybrid Filtering (HF) and Collaborative Filtering (CF) via the Spearman's rank correlation coefficient	102
11. Comparison between Query-to-Query Hybrid Filtering (HF) and Collaborative Filtering (CF) via MAE	105
12. Performance of Query-to-Query Hybrid Filtering Using Multi-Features (sim_all) or Using Only One Feature	107
13. Clusters of Countries based on Social/Cultural Similarities	107
14. Performance of Query-to-Query Hybrid Filtering (HF) With/Without Semantics ..	108

LIST OF FIGURES

Figure	Page
1. Graphical Model Representations of the Latent Class Model Examples	22
2. Knowledge Sifter Agent-Based Web Services Architecture	26
3. Ontology Schema for the Image Domain Model.....	28
4. Registration Page	33
5. Main Page before User Logins	33
6. Sign-in Problems Page.....	34
7. Data Preference Pane	35
8. Main Page After a Search for User Query “Rushmore in SD”.....	37
9. GNIS Location Results Page	38
10. Google Earth with a place “Rushmore, Mount”	39
11. Image Results Page.....	40
12. A Flow Chart for KS Search Process via Agent Interaction.....	41
13. Knowledge Sifter Agent Communication Diagram.....	42
14. Knowledge Sifter Meta-Model Schema.....	44
15. Entity-Relationship Diagram of the Knowledge Sifter Meta Model	46
16. Knowledge Sifter Case-Based Architecture	48
17. Semantic Query Refinement Sequence Diagram.....	54
18. Collaborative Query Refinement Sequence Diagram.....	55
19. Data Retrieval Sequence Diagram	56
20. XML-based Semantic Representation of a User Query Case	59
21. An Example of Representing A User Query Using the User Query Case Representation	61
22. XML-Schema for Ontology Index.....	63
23. Case Retrieval Algorithm via Ontology Index	68
24. An Example of Collaborative Query Refinement for the Example in Figure 21	73
25. A database schema for user and item data with IMDB movie content information...	86
26. A database schema for a base and test dataset.....	87

ABSTRACT

HYBRID FILTERING IN SEMANTIC QUERY PROCESSING

Hanjo Jeong, Ph.D.

George Mason University, 2011

Dissertation Director: Dr. Larry Kerschberg

This dissertation presents a hybrid filtering method and a case-based reasoning framework for enhancing the effectiveness of Web search. Web search may not reflect user needs, intent, context, and preferences, because today's keyword-based search is lacking semantic information to capture the user's context and intent in posing the search query. Also, many users have difficulty in representing such intent and preferences in posing a semantic query due to lack of domain knowledge and different schemas used by data providers. This dissertation introduces a hybrid filtering method, *query-to-query hybrid filtering*, which combines semantic content-based filtering with collaborative filtering to refine user queries based not only on an active user's search history, but also on other users' search histories. Thus, previous search experience not only of an active user, but also of the other users is used to assist the active user in formulating a query. In addition, a case-based reasoning framework with Semantic Web technologies is introduced to systematically/semantically manage and reuse user search histories for

query refinement. Finally, ontologies are used for the hybrid filtering to mine preferable content patterns based on semantic match rather than just a keyword match. Validation of the query-to-query hybrid filtering method is performed on the GroupLens movie data sets.

1 Introduction and Overview

Today's search engines provide uniform search results for the entire user community with little regard for user intent, context and preferences. This may result in poor search performance in terms of both recall and precision; the search results do not take into account user intent and context. Also, this makes it difficult to provide users with information that is not only relevant to a user query, but also matches their preferences. Most search engines rely solely on keyword search and have difficulty in obtaining user feedback on the perceived relevance of the search results. A user may click on a few of the presented links, and this might be considered a form of user feedback. One of the goals of this thesis is to improve the effectiveness of search by incorporating user preferences and user feedback.

Another important challenge is to incorporate more meaning, or semantics, into a search query posed to a search engine. It is important to capture user context and intent, as a way to focus, sift and winnow the results to reflect a user's intent, whether it is to *dine* at a fine restaurant, *lease* an automobile, or *purchase* real estate. While the addition of semantics to query terms should improve the effectiveness of the search results – we call this semantic content-based filtering, another important component of the proposed approach is the use of collaborative filtering whereby other users' likes and dislikes in response to similar queries can be used to filter results presented to the current user. The

approach is called “query-to-query hybrid filtering” that combines the best of semantic content-based search with collaborative filtering, thereby mitigating some of the issue of user preferences, intent, context, and feedback mentioned above.

Finally, it is important to extend Web search to heterogeneous data sources such as XML-databases, multimedia, and the emerging Semantic Web, which relies on linked data, open standards, semantic ontologies expressed in RDF, and OWL, and Web services. This extends the reach of search to semi-structured data as well as Semantic Web data/knowledge bases.

This dissertation begins by providing an overview of Knowledge Sifter [37, 38]. Knowledge Sifter is an agent-based system that searches for information from heterogeneous data sources by incorporating semantic technology as well as user preferences, intent and context. In this research, the approach of using Semantic Web Services for materializing the Semantic Web [30] is used for the information search among heterogeneous data sources located both on the Web and the Semantic Web. Although Web Services are semantically well described in the Knowledge Sifter approach, there is still a problem of deciding which services to use due to the many services offerings. Clearly, the quality of search results depends on the quality of information contained in the data sources, and therefore, authoritative sources need to be identified and rated. The judicious selection of appropriate Web Services (data sources) will increase both the effectiveness and the efficiency of the searches.

Next, a hybrid filtering method - called “*query-to-query hybrid filtering*” - is introduced in order to refine a user search process based not only on the user’s feedback,

but also on other users' feedback. Based on the hybrid filtering approach, this algorithm allows Knowledge Sifter to select user-preferable/suitable data sources for a user query semi-automatically.

Finally, this thesis presents a case-based reasoning framework for systematically capturing, storing, retrieving and maintaining all of the artifacts produced during the user search process. These are used to suggest refinements of search terms for user queries, data sources to be accessed, and presentation of initial results based on previously stored cases. In the case-based reasoning framework, the artifacts created by the agents during the formulation, refinement, processing, and result-rating of a user query are captured automatically, described in terms of a meta-schema, and indexed and stored in a repository as user-cases. The cases are 1) represented in terms of an XML-based schema (Extensible Markup Language) [1], 2) stored in a case repository, and 3) managed by a case management agent, which is introduced as part of the case-based framework.

1.1 Information Search on the Web and the Semantic Web

With the emergence of the World Wide Web, or simply the Web, individuals and organizations are able to publish information freely on the Web. This open and distributed nature of the Web makes it difficult for Web search engines to find information related to user information needs due to the immense amount of information published on the Web. It is also challenging because data or information is represented by the data providers' own vocabulary, and many of the providers are unreliable due to their

lack of reputation. Lastly, today's search engines use keyword-based search rather than semantic-based technologies.

The semantic-based search using the emerging Semantic Web technologies such as XML, RDF/RDFS [50], and OWL [66] can help to mitigate the terminology ambiguity problems since the Semantic Web technologies are developed based on the idea of indexing information relying on semantics rather than just keywords. For example, a keyword "jaguar" can represent three different conceptual objects: "animal jaguar," "car jaguar," and "Mac OS Jaguar." The semantic-based search can use hierarchical generalizations and distinguishing characteristic information to distinguish the jaguar concept objects. For the hierarchical information, one can specify that the domain of the jaguar object be related to animal, automobile, or operating system. For the characteristic information, one can specify whether the jaguar object has legs or wheels to represent animal jaguar or car jaguar, respectively.

Even though the data is semantically well represented in the Semantic Web, assessing data quality may be problematic – data authenticity, data provenance, and data popularity – are not necessarily provided in the semantic markup of that data. The problem is that it is not practical to have one central authority assessing the aspects of the data in the distributed environment. By mining and interpreting the collections of user feedback (assessment) about the data, one can assess the relevance and quality of the data. The query-to-query hybrid filtering method introduced in this research semi-automatically determines quality attributes by using the meta-data patterns and user

feedback patterns that can be found from the collection of the artifacts captured during the user search process, including user feedback.

1.2 User Preferences in Information Search

Given a particular user query, many information retrieval/search systems employ user preference to retrieve and filter the query results. This is done by filtering and/or ordering the immense amount of total search results in the order of the results' similarity with the user preference. The user preferences are generally determined based on the entire user feedback by ignoring the topic or domain of the user queries for which the feedback has been made. However, the user preference in a general-purpose information search system is usually very dynamic and temporary, based on a user information need specified in the form of a query. In other words, only the user feedback obtained for an active query (i.e., a current query being posed by a user to specify the user's immediate information need) can be used to mine such short-term user preference. User feedback previously obtained for the user's other queries, which generally specify different information needs, cannot be used for the mining. Furthermore, most keyword-based information search/retrieval systems such as Google and Yahoo! obtain user feedback via click-throughs on links, for only a small number of data-items – typically less than 20. This is because the users only view the top N search results in general, where in N is small. Therefore, most information retrieval systems would have difficulty in providing the short-term user preference due to the lack of the user feedback.

On the other hand, information filtering systems use relatively static and long-term-based user preferences, which can be mined from the entire, or a fairly large subset (e.g., by selecting only recent searches) of, user feedback. The user interests for an information filtering system need not be dynamic and temporary, as compared to an information retrieval system, because most information filtering systems deal with only one specific domain of information such as news articles, email, movie, etc., [10, 13, 56]. Information is pushed to users according to a user's long-term information needs, while information is obtained from users in the retrieval systems via a query, which represents a user's short-term information requirements. As a result, user preferences in information filtering systems are generally formed with a static set of data-item features, and only the values and weights of the features need to be updated over time.

Some information retrieval/search systems also use such static and long-term user preference. In this case, such systems can obtain a sufficient amount of user feedback to mine user preference. However, since user preference should be matched with a topic or specific content of a data object for which a user query is posed, such static and long-term user preference often would not be valid for a general-purpose search system. For example, a user preference mined from a user search for locating restaurants would not be valid for a user search to find movies since the features of the data objects are totally different. A user's preferred feature weights and values from the restaurant search are of no use in predicting the user's preferable feature weights and values for the movie search. Therefore, the amount of user feedback that can be used to mine user preferences for a

user search would also be insufficient if there were just a few search histories for the same domain or topic of the user search.

An important issue is whether a user would have the patience to provide substantial feedback, given that they need a quick turnaround to their search query. There are circumstances, however, when users would provide such feedback. Consider a situation in which a user is involved in intellectual property creation, and where specialized searches, e.g., for relevant patents, are essential for determining the merit of a discovery, and its technical, economic, and legal viability. In such cases, users would gladly supply feedback in exchange for timely, high-quality, and pertinent results. Also, the accumulation and refinement of user preference features and feedback can be done in an incremental and iterative manner via the case-based framework proposed in this thesis.

1.3 Collaborative Information Search

As introduced in the previous section, many of information search systems suffer from the lack of user feedback to enable user preference mining. This problem can be mitigated by employing a collaborative approach that uses other users' feedback to mine the active user's preferences. This collaborative approach provides not only novelty, but also serendipity in information search. The novelty occurs when a user is given a recommendation that comes as a surprise because he was not aware of it; the serendipity arises when the information provided would likely not have been ascertained by the user [32, 62]. For example, if a system recommends new information provided by a user's favorite data sources, the information is novel, but not serendipitous. The information is

serendipitous if it were to come from a data source that was new to the user. Therefore, the novelty occurs if a system employs user preferences to predict the likelihood that the user would “like” certain information. However, the serendipity can only be obtained if a system mines a user’s preference not only from the user’s feedback, but also from neighbor users’ feedback. The neighbor users can be defined as other users who might have similar tastes.

The novelty and serendipity in information search is useful not only for recommending new data or information, but also for refining user queries themselves. Unlike most recommender systems, Knowledge Sifter is a query-based information retrieval system, i.e., the data/information provided to a user will be heavily dependent on how a user query is specified. Therefore, if the novel and serendipitous collaborative aspects are added to specifying and refining user queries, the provision of novel and serendipitous information to the users can result. In this research, a collaborative approach is used to automatically specify user queries via emergent semantics. The emergent semantics are obtained from content patterns among data items that are preferred for a user query not only by the active user, but also by the neighbor users. Finally, the user query is refined with the emergent semantics. This collaborative query refinement would also assist the users in specifying their information needs as a query using the specification recommendations created by the neighbor users and data providers’ vocabulary found on the data content patterns.

1.4 Research Objectives

According to the research motivations introduced in the previous sections, the main research objective of this thesis is to improve the effectiveness of search by 1) assisting users in formulating semantic queries that match the user's search intent and context, 2) using Semantic Web technology and Web Services to access heterogeneous semantic content, and 3) taking advantage of other user queries and their associated result sets to mine a user's short-term preference regarding the user's current query.

An information object, e.g., a search request, can be specified differently for distinct domains and views, specifically with regards to user intent and context. For example, a query "find a steakhouse near Washington Monument" can be formulated with two different search intents, "dining" and "starting a restaurant business". For the intent of dining, a semantic query can be specified with features, cuisine type, food cost, location, etc. For the intent of starting a restaurant business, the features will be business type, asking price, revenue, location, etc. Therefore, guiding users in formulating semantic queries based on their search intent and context should improve the search in terms of precision.

As the Web and the Semantic Web are open and distributed environments, the information objects (search requests) can be processed by accessing heterogeneous data sources, as there are many providers and they use their own vocabularies to specify the objects. The Semantic Web ontologies represented in XML, RDF and the Web Ontology Language (OWL) can help to mitigate the *semantic heterogeneity*, while Web Services provide *syntactic interoperability*. Especially, using OWL to specify relationships

between concepts in an ontology or in multiple ontologies will improve the effectiveness of search by mitigating semantic heterogeneity.

As addressed in Section 1.2, users provide relevance feedback implicitly or explicitly only for a few of the many items returned as a result of a search. Thus, most users can be regarded as new users because they provide feedback for a few search results and the feedback can be used only for an active (current) query. The proposed *query-to-query hybrid filtering* augments feedback information based on neighbor query cases, which have user queries similar to the active query, and then refine those queries based on the augmented feedback. This query refinement mitigates the new-user problem and improves the effectiveness of a search especially in the new-user situation that often occurs in a query-based search system. Also, the neighbor query cases can be found not only from the active user's search history, but also from other users' search history. Thus, the query refinement can be regarded as collaborative query refinement that uses opinions of a user community having similar search concepts and intent.

1.5 Research Hypothesis

Based on the research objectives, the main research hypothesis is “*The query-to-query hybrid filtering, which consists of semantic content-based search complemented with collaborative filtering, will improve the effectiveness of search, especially in the new-user situation.*”

1.6 Research Approach

A case-based reasoning framework is defined to assist users in formulating semantic queries based on user search concepts and intent including query-based case representation and ontology-index based case retrieval algorithm. The query-to-query hybrid filtering method is specified and developed for reusing previously-stored query cases in the collaborative query refinement process. I also validated and tested the query-to-query hybrid filtering algorithm by using well-known MovieLens dataset created by the GroupLens research group.

1.7 Overview of Dissertation

An outline of the remainder of this dissertation is shown in Table 1.

Table 1 Overview of this Dissertation

Chapter	Chapter Description
1	Introduction (this chapter)
2	Related Research: this chapter describes other past research related in common to the area of this research.
3	Knowledge Sifter: this chapter introduces Knowledge Sifter, an agent-based information search system which uses various Semantic Web technologies such as ontologies and Web Services to retrieve information from heterogeneous data sources.
4	Case-Based Knowledge Sifter Framework: this chapter presents a case-based framework for Knowledge Sifter which is designed to manage and reuse the previous user-query cases systematically and efficiently. An XML-based semantic representation of user-query cases, an ontology-based index structure, concept-based case-similarity calculation, a case retrieval algorithm, etc. are provided for the efficient retainment and retrieval of the cases.
5	Collaborative Query Refinement: this chapter provides a collaborated query refinement process which is based on the case-based Knowledge Sifter framework and a hybrid filtering method combining both collaborative filtering and content-based filtering. The hybrid filtering method is called query-to-query hybrid filtering and it is used to refine a user query by using user rating patterns and content patterns found not only from an active user's query case, but also from neighbor users' query cases.
6	Validation: details of various experiments for validating a primary method of this research, query-to-query hybrid filtering, such as experiment setup and results are described in this chapter.
7	Conclusions

2 Related Research

2.1 Semantic Search

Most of the traditional Web search engines such as Google and Yahoo! use keyword search, which indexes Web documents using a set of keywords and retrieves the documents based on keyword match rather than semantic match. The traditional Web search also uses semantics implicitly represented by keyword patterns found via using statistical mining technologies such as LSI (Latent Semantic Indexing) [18, 33]. However, the implicit semantics found from the term and document matrix can still be somewhat ambiguous because the synonymous relationships among the terms are found based on co-occurrences of terms in documents, not based on explicit synonymous relationships as defined in an explicit ontology. Furthermore, the traditional Web search is document-oriented while semantic search is object-oriented [26]. This characteristic of the traditional Web search also makes the statistically-mined synonym relationships from the term and document matrix ambiguous because a Web document can contain multiple objects and terms in a document can describe different objects. In addition, a synonym of a term is no longer synonymous to the term in different contexts (e.g., topic, temporal/spatial aspects, etc.) of a document.

Therefore, the basic idea of semantic search is to improve the effectiveness of information search on the Web using neither keyword/s nor the implicit semantics, but

the explicit semantics via the Semantic Web and Semantic Web Services which provide a semantic indexing scheme and semantic data transfer, respectively. The roles of the Semantic Web, Semantic Web Services, and ontologies for semantic search are described in the following subsections along with related research.

2.1.1 Information indexing via Semantic Web

Given the difficulty of natural language processing and the unsatisfactory performance of keyword/s-based search, one of the goals of the Semantic Web is to improve the effectiveness of the Web search by providing an information indexing scheme based on the explicit semantics. The traditional Web can be regarded as a Web of documents because it is originally created for human navigation. This repository nature of the traditional Web thus requires ad hoc machine-processible indices such as keyword/s or metadata-based document indices in order to perform the Web search. The emerging Semantic Web can be viewed as a Web of data (objects) [14] in which the data represent real-world objects using explicit semantics via machine-processible formats such as XML, RDF/RDFS and OWL. Thus, the Semantic Web itself also can be regarded as a huge (very dense) index which represents and identifies Web resources with semantics because of the machine-processibility of the Semantic Web data.

2.1.2 Role of the (Semantic) Web Services in Semantic Search

Data in the Semantic Web is basically specified by a graph-based framework, RDF (Resource Description Framework), which allows us to describe data in a machine-processible format while providing a linking framework between data [50]. The linkages

represent semantic relationships among data with a set of controlled vocabulary. One of the semantic search approaches is to find RDF data representing user concepts from static Semantic Web documents [19, 26]. However, most semantic search engines still rely on the simple keyword search while such semantic search requires sophisticated reasoning on the RDF graph. Most of the Semantic Web reasoning engines such as Jena [16] and Pellet [64] provide inference and reasoning on the RDF dataset, but the reasoning engines would not be well suited for the semantic search engines because of the distributed nature of the Web and the immense amount of Web RDF data. A RDF crawler would be required to gather the RDF data on the Web, and a RDF database would need to be maintained to store all of the gathered RDF data. It can be easily predicted that the size of the RDF (index) database will be unmanageable because all of the actual RDF data would need to be stored on the Semantic Web. Unlike the traditional Web search engines, which store only metadata (a set of keyword/s) of Web sites with the goal of finding Web sites, the goal of the semantic search is to find data objects (Web sites also can be regarded as one of the objects). Also, the huge amount of data would degrade the performance of the reasoning engines in terms of both their efficiency and effectiveness.

Another view of the Semantic Web considers it to be materialized by Semantic Web services. Many current Web sites use a back-end database, which allows users to access their data only from their customized Web application, because they do not want publish all of their data due to business and security issues [30]. Another reason is because their data is too massive to publish on the Web using static Web pages. For example, if Amazon.com were to use static Web pages to provide all their product data to

the customers and other vendors, the data would be unmanageable because of the immense number of the static Web pages required to present the data. Therefore, dynamic Web pages and Web Services access the back-end database to retrieve on-demand product information to display to the customers and other vendors, respectively. Furthermore, the current Web search usually finds only the Web sites which contain data related to the user query concept. Thus, the users further need to use the Web application of the retrieved sites to get the real data which they intended to search. For example, if a user wants to find a hotel in Paris whose price for a one night stay is under \$100, the current Web search engines cannot directly answer the user query. This is not only because they cannot process such a complex query due to the limitation of the term-to-document indexing, but also because there is no such data on the static Web. The search engines can find only some travel sites such as expedia.com and hotels.com, and the users are required to visit the Web sites in order to use a Web application provided by the travel sites to locate such hotels. This phenomenon supports using Semantic Web services approach to materialize the Semantic Web. For this research, the Semantic Web services approach has been chosen for materializing the Semantic Web for the semantic search.

2.1.3 Role of Ontology in Semantic Search

Ontology can be defined as “*a specification of a shared conceptualization of a domain*” [25]. OWL (Web Ontology Language) is a de facto standard language of representing ontologies for the emerging Semantic Web. OWL is built on top of RDF and extends vocabularies for describing concepts and their relationships for forming

ontology of Web resources [66], i.e., while RDF/RDFS represents only the taxonomical knowledge of Web resources, OWL is created to support ontological knowledge of Web resources. There are two main roles of ontology for semantic search: one role is to disambiguate user queries by linking a user term to a specific ontology concept representing actual user query concept [26, 29, 65], and the other role is to modify user queries based on concept-hierarchies defined in ontology [9, 15]. Two types of ontologies are mainly used for the query refinement: one type is a general upper ontology such as WordNet, and the other type is a domain ontology which defines domain-specific concepts for the domain of search data. In this research, both the query disambiguation and the query augmentation using WordNet and domain ontologies are used to refine user queries.

2.2 Recommender Systems

Recommender systems are software systems whose purpose is to provide more preferable data items to a user by predicting the user's preference of data items which the user has not yet seen, by analyzing and applying user rating patterns and data content patterns in user profiles. Recommender systems are based on three methodologies: content-based filtering, collaborative filtering, and hybrid filtering methods [12, 61]. Content-based filtering is a method of filtering unseen (technically, not yet rated by a user) items to an active user based on the similarity of the items to a query which is formulated by the content patterns mined from a set of items preferred only by the active user previously. On the other hand, collaborative filtering is a method of making

predictions of the active user's preference for the unseen items based not only on the active user's ratings, but also on neighbor users' ratings. The neighbor users are a group of users who have tastes (technically, rating patterns on commonly rated items) similar to the user. The bottom-line idea of the collaborative filtering is that people who agreed in the past are likely to agree on the future. Lastly, the hybrid methods combine content-based filtering and collaborative filtering to take advantages of their strength and mitigate their shortcomings. The following subsections describe these three methodologies in more detail.

2.2.1 Content-Based Filtering

Content-based filtering is a method for recommending the unseen items to a user based on the contents of items already in the user's profile. The content-based filtering is similar to a personalization method in terms of using the user preference constituted by eliciting user-preferred contents of items (objects) based on the user's relevance feedback. The main idea of this method is to find content patterns among the user-preferred items and to use those patterns to refine the user query so as to retrieve additional items relevant to the user-preferred items. This method is also similar to *association rule mining* [28, 67] in data mining because it discovers the features' association patterns from the user-preferred items in terms of a feature vector.

For example, the Multimedia Analysis and Retrieval System (MARS) [58, 63] incrementally learns a user's intention for query refinement from the user's profile. The query refinement method consists of query re-weighting and query modification techniques. These two techniques are based on relevance feedback in which a user

provides evaluations for result objects associated with the initial user query, in terms of multiple levels of relevance, e.g., *highly relevant*, *relevant*, *no opinion*, *not relevant*, or *highly not relevant*. The query re-weighting techniques adjust weights for each component (feature criterion) in multi-dimensional queries by calculating the weighted centroid of the result objects. The query modification techniques construct a multipoint query by clustering the result objects based on their points in the multi-dimensional feature space in order to expand the user query. The multipoint query is determined as a query which contains multiple values for each feature, i.e., it embraces multiple representations of the user query. The two refinement techniques are combined seamlessly and can be incrementally performed along with the updates of the relevance feedback.

2.2.2 Collaborative Filtering

Unlike content-based methods, collaborative filtering attempts to predict usefulness of as yet unseen items for an active user, who is currently using the system; the prediction is based on user rating values which have previously been given to the items not only by the active user, but also by the other users. Well-known recommender systems based on collaborative filtering are the GroupLens system [24, 31, 46, 59, 60] and the Amazon.com item-to-item collaborative filtering system [48]. The GroupLens system recommends items such as news articles and movies to the active user by filtering the items based on the predictions made for the active user regarding the user's likeness on the unseen items. The user's likeness of an unseen item is determined based on the user ratings of neighbor users on the unseen item. The neighbor users are selected if they

have similar rating patterns with the active users for the commonly-rated items in their profile. The Amazon.com item-to-item collaborative filtering finds the most popular co-purchased items within the group of users who have purchased the active item, which an active user is currently viewing. This is done by simply counting the number of co-occurrences on the group's profile. The main concern of the Amazon.com's methodology is to provide better performance and scalability enabling the system to perform real-time recommendations among the large collections of items.

In general, the collaborative filtering algorithms can be categorized into two classes: memory-based algorithms and model-based algorithms. The memory-based algorithms employ a user rating-history database in order to find neighbor users who have similar rating patterns with the active user. Recommended items based on the neighbor users' rating patterns using algorithms similar to those used in the GroupLens and Amazon.com recommender systems are then offered. The following equations are used to calculate a prediction for the active user's rating for an unseen item based on the *Pearson Correlation Coefficient* used in GroupLens:

$$p_{u^a, d^u} = \overline{r_{u^a}} + \frac{\sum_{u^n} (r_{u^n, d^u} - \overline{r_{u^n}}) * w_{u^a, u^n}}{\sum_{u^n} |w_{u^a, u^n}|} \quad (1)$$

$$w_{u^a, u^n} = \frac{\sum_{d^s} (r_{u^a, d^s} - \overline{r_{u^a}}) * (r_{u^n, d^s} - \overline{r_{u^n}})}{\sigma_{u^a} * \sigma_{u^n}} \quad (2)$$

where p_{u^a, d^u} represents a prediction for the active user u^a for an unseen (unrated) item d^u . u^n represents a neighbor user who has a rating record for the unseen data item for the active user. w_{u^a, u^n} is the similarity (correlation) weight between the active user and a neighbor user in terms of rating patterns as defined by the *Pearson Correlation Coefficient*. d^s is the commonly seen (rated) items between the active user and the neighbor. $\overline{r_{u^a}}$ and $\overline{r_{u^n}}$ represents arithmetic means for the ratings of data items obtained from the active user and the neighbor, respectively.

On the other hand, the model-based algorithms first cluster users into predefined classes based on user rating patterns. The usefulness (prediction of rating values) of items for the active user is evaluated by the overall ratings of one of the predefined classes in which the active user has been classified. Most algorithms in this approach are using a latent (hidden) class model based on *Bayesian (Belief) Networks*. Equation 3 is used to calculate the rating value for the latent class model (a) in which has strong assumptions that a user u and an item d are conditionally independent and a rating value r is also conditionally independent with the user and the item given a latent class z .

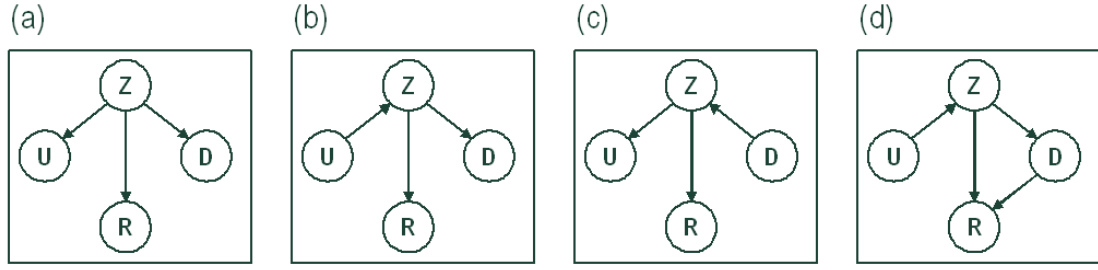


Figure 1 Graphical Model Representations of the Latent Class Model Examples

$$P(u, d, r) = \sum_{z \in Z} P(z) P(u | z) P(d | z) P(r | z) \quad (3)$$

where the latent class variable $z \in Z = \{z_1, z_2, \dots, z_k\}$ is associated with each observation (u, d, r) . $P(z)$ stands for class prior probability, $P(u|z)$ and $P(d|z)$ stand for class dependent distributions for users and items, respectively.

Generally, the memory-based algorithms are preferable to the model-based algorithms due to the limitation of clustering users and the lack of instances to cluster the users. However, the model-based algorithms can be more effective in terms of processing time because usefulness of items can be pre-calculated for the pre-defined classes so that the system only requires time to classify the active user. Hybrid algorithms that combine memory- and model-based algorithms are developed to use the best aspects of the two algorithms by clustering users while keeping the rating patterns in a database. This would make the recommendation process transparent, unlike the memory-based methods. Furthermore, the hybrid algorithms would allow a system to modify user classes over the entire user ratings unlike the model-based methods as in collaborative filtering by

personality diagnosis approach [57]. However, this approach would have too many clusters because users are clustered by their rating patterns. At worst, the number of clusters could be the same as the number of users, which would be identical to the memory-based algorithms.

2.2.3 Hybrid Filtering

Content-based recommender systems recommend items that are similar to items liked in the past by a given user. When a new user uses such a system, the system might not work properly because the user's profile does not contain a list of preferred items. This is called the *new user problem* [8, 59] which usually happens in the content-based recommender systems. Collaborative filtering alleviates the new user problem by allowing the system to refer to other users' profiles in the recommendation process. Nevertheless, the collaborative systems also have a similar problem which is called the *new item problem* [8]. When a new item arrives in the system, it can never be retrieved and rated because the collaborative filtering is based on ratings of items in user profiles without considering the contents of the items. Therefore, hybrid filtering methods that combine the content-based filtering and the collaborative filtering would compensate for the new user and new item problems by allowing a recommender system to learn the content patterns of items preferable to an active user based not only on the active user's own behaviors, but also on other users' behaviors.

3 Knowledge Sifter

The Knowledge Sifter (KS) project 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 [37, 38], and 3) to refine searches based on user feedback. Increasingly, users seek information outside of their own communities to open sources such as the Web, XML-databases, and the emerging Semantic Web. The Knowledge Sifter project also wishes to use open standards for both ontology construction and information search on heterogeneous data sources. For this reason OWL [21, 52] has been chosen to implement the specifications and data interchange, and Web Services [17] for communication among agents and information sources.

3.1 KS Agent-Based Web Services Framework

The rationale for using agents to implement intelligent search and retrieval systems is that agents can be viewed as autonomous and proactive. Each agent is endowed with certain responsibilities and communicates using an Agent Communication Language [23]. An agent architecture can be materialized as Web Services with ad hoc functionalities such as awareness of other agents in a meta-level, ontology reconciliation,

agent communication, and distributed coordination [34]. This is the approach taken to implement the agent community comprising Knowledge Sifter in this research. The family of agents presented here is a subset of those incorporated into the large vision for Knowledge Sifter. This work is motivated by earlier research into Knowledge Rovers [35, 36] performed at GMU. This research is also informed by a research on WebSifter [41, 42, 44], which is both a US patented invention [43] and a meta-search engine that gathers information from traditional search engines and ranks the results based on user-specified preferences and a multifaceted ranking criterion involving static, semantic, categorical, and popularity measures.

The Knowledge Sifter architecture [37, 38] may be considered a service-oriented architecture consisting of a collection of cooperating agents. The application domain is that of Image Analysis. The Knowledge Sifter conceptual architecture is depicted in Figure 2. The architecture has three layers: User Layer, Knowledge Management Layer, and Data Layer. Specialized agents reside at the various layers and perform well-defined functions. This collection of cooperating agents supports interactive query specification and refinement, query decomposition, query processing, ranking, as well as result ranking and presentation. The Knowledge Sifter architecture is general and modular so that new ontologies and new information resources can be easily incorporated [55]. The various agents and services are described below.

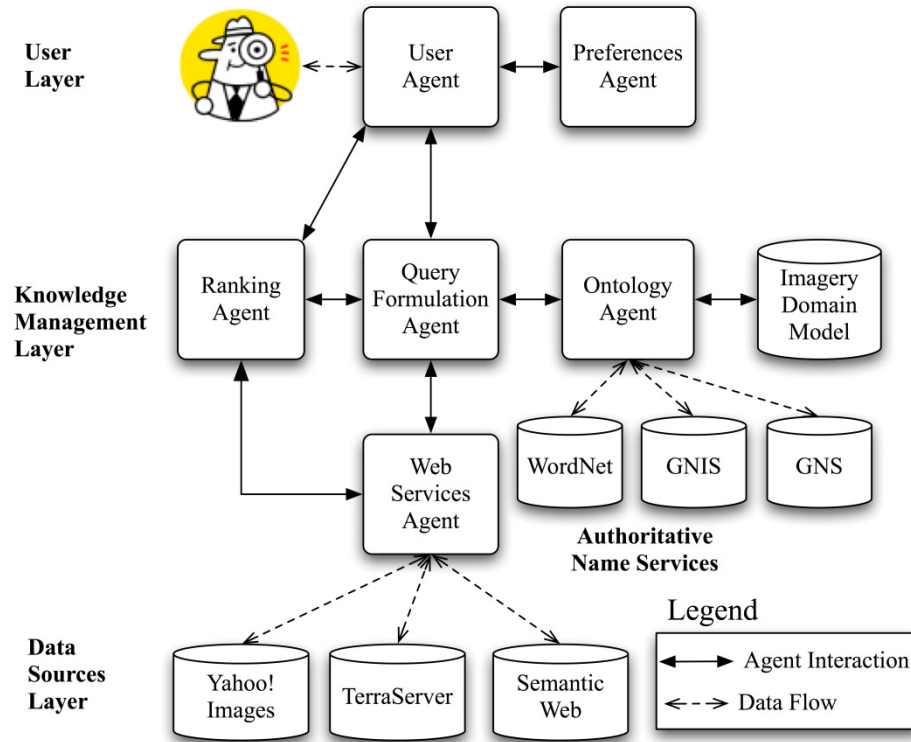


Figure 2 Knowledge Sifter Agent-Based Web Services Architecture

3.1.1 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 perceived authoritativeness of Web search engine results, and other preferences to be used by the Ranking Agent. The Preferences Agent can also learn the user's preference based on experience and feedback related to previous queries. The User Agent also takes responsibility for presenting results to the user in terms of an image visualization via its own visualization services and spatial visualization via external visualization services such as Google Maps API and Google Earth.

3.1.2 Ontology Agent

The Ontology Agent accesses an imagery domain model, which is specified in OWL and resides in the KS repository. In addition, there are three authoritative name services: Princeton University's WordNet [54], the U.S. Geological Survey's GNIS [2], and the GEOnet Names Server (GNS) [3]. They allow the Ontology Agent to use terms provided by the name services to suggest query enhancements such as generalization or specialization.

For example, WordNet can provide a collection of synonyms for a term, while GNIS and GNS translate a physical place in the US and the Earth, 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 would be updated to accommodate new concepts and relationships.

3.1.3 Imagery Domain Model and Schema

The principal ontology used by Knowledge Sifter is the Imagery Domain Model, specified using OWL. A Unified Modeling Language (UML)-like diagram of the ontology is provided in Figure 3. The class *Image* is defined as having *source*, *content*, and file-descriptive *features*. Subcategories of content are *person*, *thing*, and *place*. Since satellite and geographic images are primary data objects of interest, the class *place* has two general attributes, *name* and *theme*, together with the subclasses *region* and *address*. The *Region* is meant to uniquely identify the portion of the Earth's surface where the place is located, either by a *rectangle* or a *circle*. In the case of a rectangle two latitude values (*north* and *south*) and two longitude values (*east* and *west*) are needed, while the

latitude and *longitude* of its center point and a *radius* are needed to specify a circle. The *address* of our location is identified by *country*, *state*, *city*, *zip code*, and *street*. Each image belongs to a specific online source, the *server* and has *URI-1* as a unique identifier, together with a secondary *URI-2* for a thumbnail (if any). Some qualitative and quantitative attributes are also modeled as subclasses of the general class *features*, namely *resolution* (in square meters per pixel), *projection* and *datum* (for future GIS utilizations), a *date range*, and image size (with *height* and *width* expressed in pixels).

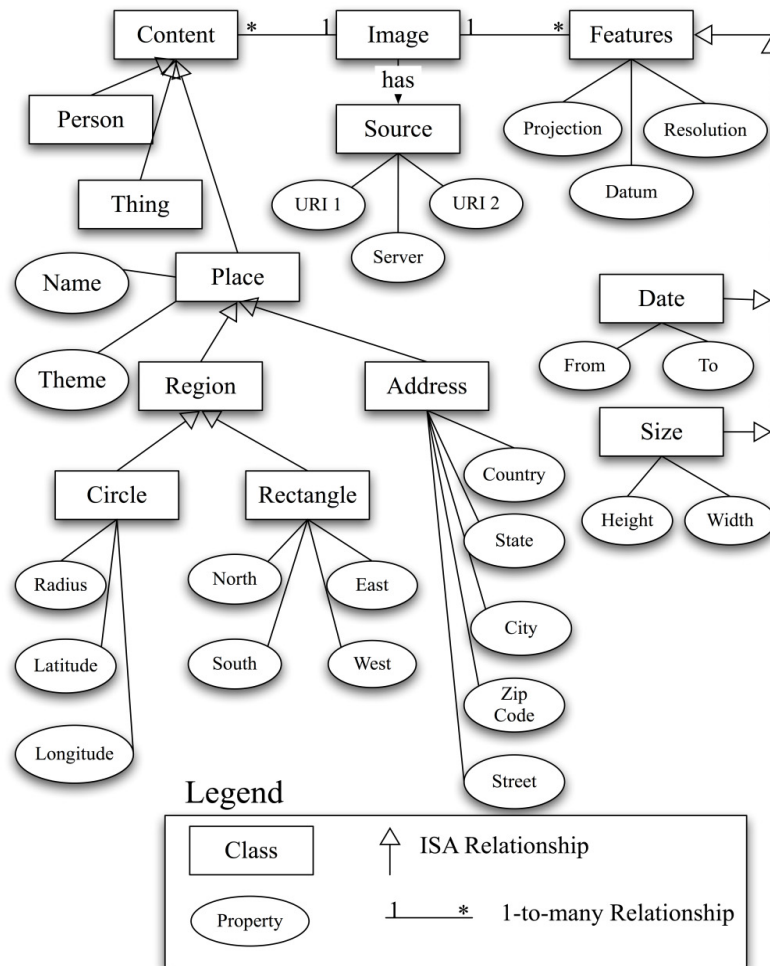


Figure 3 Ontology Schema for the Image Domain Model

3.1.4 Authoritative Name Services

The Ontology Agent accesses three authoritative name services. The first name service is WordNet, developed at Princeton University, which is a lexical database for the English language and provides *senses* (linguistic concepts) for a term and synonyms for a sense which allows KS to perform a broader search. The WordNet also provides hyponyms and hypernyms for a sense, which enhance queries in terms of specialization and generalization, respectively.

The second name service is the USGS Geographic Names Information System (GNIS) which is a database of geographic features within the United States and its territories. GNIS was developed by the USGS and the U.S. Board on Geographic Names (US BGN) to meet major national needs regarding geographic names and their standardization and dissemination. It is an integration of three separate databases, the National Geographic Names Data Base, the USGS Topographic Map Names Data Base, and the Reference Data Base. Records within the database contain feature name, state, county, geographic coordinates, USGS Geographic Map name, and others.

The last name service is the National Geospatial-Intelligence Agency (NGA) GEONet (GNS) which is also a geographic feature database for worldwide searches excluding the United States and Antarctica. GNS also integrates the NGA's geospatial information and the US BGN's database of foreign geographic feature names for the standardization and dissemination of foreign geographic feature names.

3.1.5 Query Formulation Agent

The user indicates 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.

3.1.6 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 Service 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 [53].

The Web Services Agent transforms the subqueries to XML Protocol (SOAP) requests to the respective local databases and open Web sources (TerraServer or Yahoo Images) that have Web Service published interfaces.

3.1.7 Ranking Agent

The Ranking Agent is responsible for compiling the subquery 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.

3.1.8 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) [22]-based 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 Image's metadata to filter images based on user preference. However, the metadata does not contain the creation time of images, which is a good measure of temporal aspect.

The TerraServer is a technology demonstration for Microsoft. There is a Web Service API for TerraServer. TerraServer is an online database of digital aerial photographs (DOQs – Digital Orthophoto Quadrangles) and topographic maps (DRGs – Digital Raster Graphics). Both data products are supplied by the U.S. Geological Survey (USGS). The images are supplied as small tiles, and these can be made into a larger image by creating a mosaic of tiles. The demonstrator at terraserver-usa.com uses a mosaic of 2x3 tiles.

The purpose of this approach is to take the ontology-enhanced query and generate specific sub-queries for the TerraServer metadata. The resulting image identifiers and their metadata are wrapped into an instance of the Knowledge Sifter image-domain ontology, and an array of these is returned to the Web Service Agent to compile with other results.

3.2 Knowledge Sifter End-to-End Scenario

In this section, end-to-end scenarios of each process created and implemented in Knowledge Sifter system are described with screen shots.

1. *Registration*: users are required to register first in order to use KS system because at present KS is implemented to support several users. The registration and login process allows KS to keep track of its users' context to customize its services based on the context.

Knowledge Sifter
Agent-Based Ontology-Driven Search Tool

Home
About
User Guide
New User

New User Registration:

All personal information will be kept strictly confidential and will not be given out under any circumstances.

First Name:

Last Name:

Industry:

Select an User ID:

Select a Password:

Confirm Password:

Your Security Question:

Your Answer:

Registration enables Knowledge Sifter to provide more accurate and reliable results though user identification and authentication via

Figure 4 Registration Page

2. *Login*: there are three options for setting a cookie as follows: 1) set no cookie (always asks for my ID and password), 2) set cookie for only user id (save my ID only), and 3) set cookie for user id and password (save my ID and password).

Knowledge Sifter
Agent-Based Ontology-Driven Search Tool

Home
About
User Guide
New User

User ID:

Password:

☒ Always asks for my ID and password ☐ Save my ID only ☐ Save my ID and password

Figure 5 Main Page before User Logins

3. *Sign-in Problems*: if users have forgotten their id and/or password, they can find their id or set new password for their account after they are authenticated by confirming their security answers which they set at registration time. Note that KS stores user passwords in the KS repository as a hash sum produced from a combination of multiple cryptographic hash functions. Therefore, users have no option except to create a new password if they forget their password.

Knowledge Sifter Sign-in Problems Page

If you have forgotten your userid and/or password to your account, please confirm your identity by filling out Item 1 below. First enter your first and last name and click on the button "Retrieve Your Security Question" Next, before answering the security question, please select your user id from the drop-down list of user ids. After confirming your security answer, you will be allowed to create a new password for your account. Note that your security answer should be identical to the one you provided during the registration process.

1. Confirm Your Identity

First Name:	<input type="text"/>
Last Name:	<input type="text"/>
<input type="button" value="Retrieve Your Security Question"/>	

2. Create Your New Password

Your User ID:	<input type="text"/>
Select a New Password:	<input type="text"/>
Confirm New Password:	<input type="text"/>
<input type="button" value="Create New Password"/>	

Figure 6 Sign-in Problems Page

4. *Preference Setup*: after a user registers with KS, the Preference Agent automatically creates default data preferences for the user. Users can modify weights and values for each criterion such as semantic, spatial, temporal, and other features of an image. For default, the Preference Agent sets the default weight for a user term (1.0) and for user-selected synonyms (0.8).

datapref - Microsoft Internet Explorer

[Data Preference Setup]

Semantic (Name&Desc): 10

Location: 10

Date: From: January 1 1900 (ex. 1999) To: February 23 2006 (ex. 1999) 10

Size: Width: Height: 10

Theme: Photograph 10 Topographic Map 10 10

Data Source: Terra Server 10 Yahoo 10 10

Update

Figure 7 Data Preference Pane

5. *Sentence Search*: KS can process simple natural language-like queries for spatial information. For example, if a user poses a text query on sentence search, “Rushmore in SD” to the User Agent, the User Agent passes the text query to the Ontology Agent. The Ontology Agent checks if the query has any words on a pre-specified list of prepositions for spatial information such as “in” and “within.” Based on the prepositions found in the query, the Ontology Agent parses user terms for search object and area of interest (AoI).

5A. *Object Search*: users specifically can type terms for search object and AoI by using the object search option. Note that at present KS does not support either point of interest (PoI) or line of interest (LoI) options for spatial query processing.

6. *Google Spell Checking*: after parsing user terms from the user query text in sentence search option or after getting user terms directly from object search option, the Ontology Agent requests spelling suggestions for the user terms using Google spell checking web

services. If there are any suggestions, KS shows them to users as an option to change their terms.

7. *WordNet*: the Ontology Agent requests synonym senses from WordNet for user terms and displays the senses with a default selection. KS allows users to select only one sense for each term. For the default selection on the AoI term, the Ontology Agent performs the automatic selection by checking a term “location” in hypernyms which are more general terms of the user AoI term in terms of the spatial sense. For example, a term “DC” has as two senses: 1) District of Columbia, D.C., DC (the district occupied entirely by the city of Washington; chosen by George Washington as the site of the nation's capital and created out of land ceded by Maryland and Virginia) and 2) direct current, DC (an electric current that flows in one direction steadily). KS initially selects the first sense because the sense has a term “location” in its hypernyms set while the second sense does not. Users can change the sense selection corresponding to their intended concept. After users select the senses, the Ontology Agent presents to users a list of the synonyms of the user term in terms of the user-selected sense.

8. *GNIS*: as a default, KS selects a state if any one of the user AoI terms and its synonyms is identical with a U.S. state name. Users can select multiple synonyms to find locations from GNIS, and a number of result locations for the term will be shown in brackets next to the term. The Ontology Agent poses refined queries to GNIS and shows the result by removing duplicated locations. Note that if there are more than 2000 locations for a term, GNIS returns no results. In this case, users need to try a more specialized query such as using more specific terms and/or states.

8A. *GNS*: KS also supports GNS locations. GNS provides location data in the coverage of world-wide, whereas GNIS is strictly for US locations. For example, let's suppose that a user types in the words ‘Lake Victoria’ and clicks the Sentence Search. KS will consult WordNet to get the synonym set: Lake Victoria and Victoria Nyanza. KS consults GNIS for the default term ‘Lake Victoria’ and finds locations in the US; GNS does not report any foreign locations for ‘Lake Victoria.’ However, if the user clicks the ‘Lake Nyanza’ button, KS re-consults both GNIS and GNS, and GNS presents two entries for ‘Lake

Nyanza’ in Uganda and Tanzania. This example shows the power of WordNet’s synonym sets.

8B. *Image Search Engine*: if a user wants to search only for images of a person or thing, users can skip the access to TerraServer for searching images of place, because the coordinates are required to retrieve the images from the TerraServer.

The screenshot shows the Knowledge Sifter web application interface. At the top, there's a navigation bar with links for Home, About, User Guide, and New User. Below this is a search bar containing the query "Rushmore in SD". To the right of the search bar are buttons for "Sentence Search" and "Advanced".

Below the search bar, the application displays "WordNet Senses for 'Rushmore'" and "WordNet Senses for 'SD'". The "Rushmore" section shows a sense: "Rushmore, Mount Rushmore, Mt. Rushmore (a mountain in the Black Hills of South Dakota; the likenesses of Washington and Jefferson and Lincoln and Roosevelt are carved on it)". The "SD" section shows a sense: "South Dakota, Coyote State, Mount Rushmore State, SD ((a state in north central United States))".

Below the WordNet senses, there's a section for "GNIS Results:". It includes a dropdown menu set to "South Dakota" and buttons for "Google Maps" and "Google Earth". Below this is a table with columns: Location, Type, County, State, Longitude, Latitude, GMap, GEarth, and Images.

Location	Type	County	State	Longitude	Latitude	GMap	GEarth	Images
Rushmore Airport	airport	Pennington	SD	-103.4167	43.9167	GMap	GEarth	Images
Rushmore Mall	locale	Pennington	SD	-103.2167	44.1069	GMap	GEarth	Images
Rushmore Plaza Civic Center	locale	Pennington	SD	-103.225	44.0881	GMap	GEarth	Images
Rushmore Shadows Resort	locale	Pennington	SD	-103.4592	43.9258	GMap	GEarth	Images
Rushmore, Mount	summit	Pennington	SD	-103.4583	43.8803	GMap	GEarth	Images

Below the table, there's another "GNIS Results:" section with "Google Maps" and "Google Earth" buttons. Below this is another table with columns: Location, Type, Country, Longitude, Latitude, GMap, GEarth, and Images.

Location	Type	Country	Longitude	Latitude	GMap	GEarth	Images
Rushmore	FRM	ZA	27.766666666666666	-15.8	GMap	GEarth	Images
Rushmore	EDM	AC	129.96666666666667	25.266666666666666	GMap	GEarth	Images

Figure 8 Main Page After a Search for User Query “Rushmore in SD”

9. *Google Map & Google Earth*: after KS receives the location information from GNIS/GNS, the Ontology Agent sends the location information to the User Agent; then the User Agent displays a location table with details of the location information via

Google Map API or Google Earth. For Google Earth, the Ontology Agent generates a geographic feature data set using Keyhole Markup Language (KML), which is an XML grammar and file format for modeling and storing geographic features such as points, lines, images, polygons, and models for display in Google Earth and Google Maps [4]. The User Agent then automatically invokes Google Earth on the user's local machine with that data set passed from the Ontology Agent. For the default zoom level, the User Agent automatically calculates the zoom level based on the distribution of locations. If a user clicks on a location on the map, a small blowup map with the closest zoom level will appear. At present KS uses three map servers, Google, NASA, and USGS. Note that the USGS map layer only covers the Washington DC area. The User Agent can access any Web Map Services (WMS) server via Google Map API if needed.

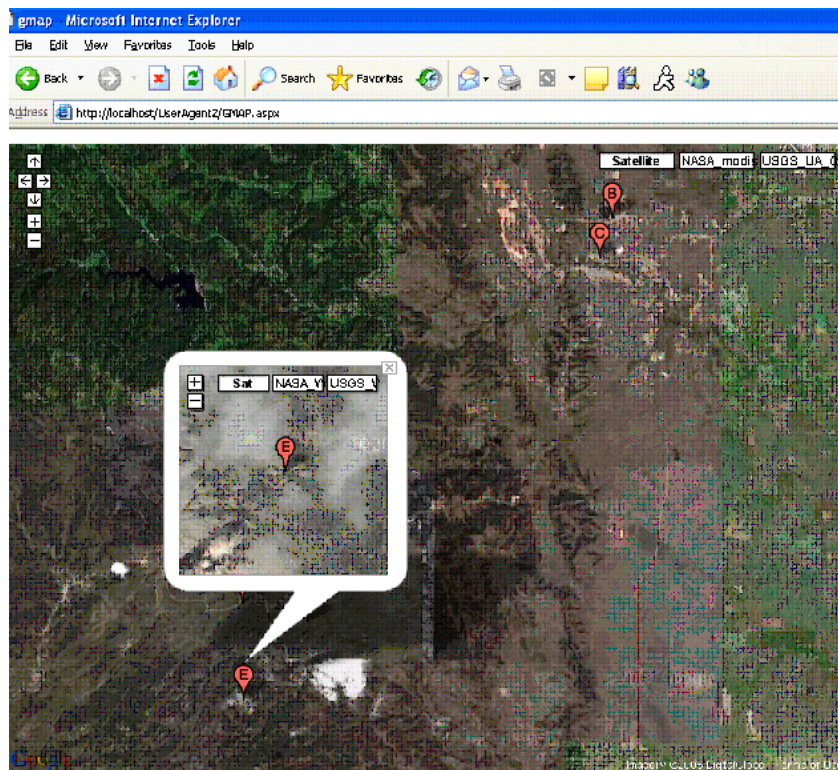


Figure 9 GNIS Location Results Page

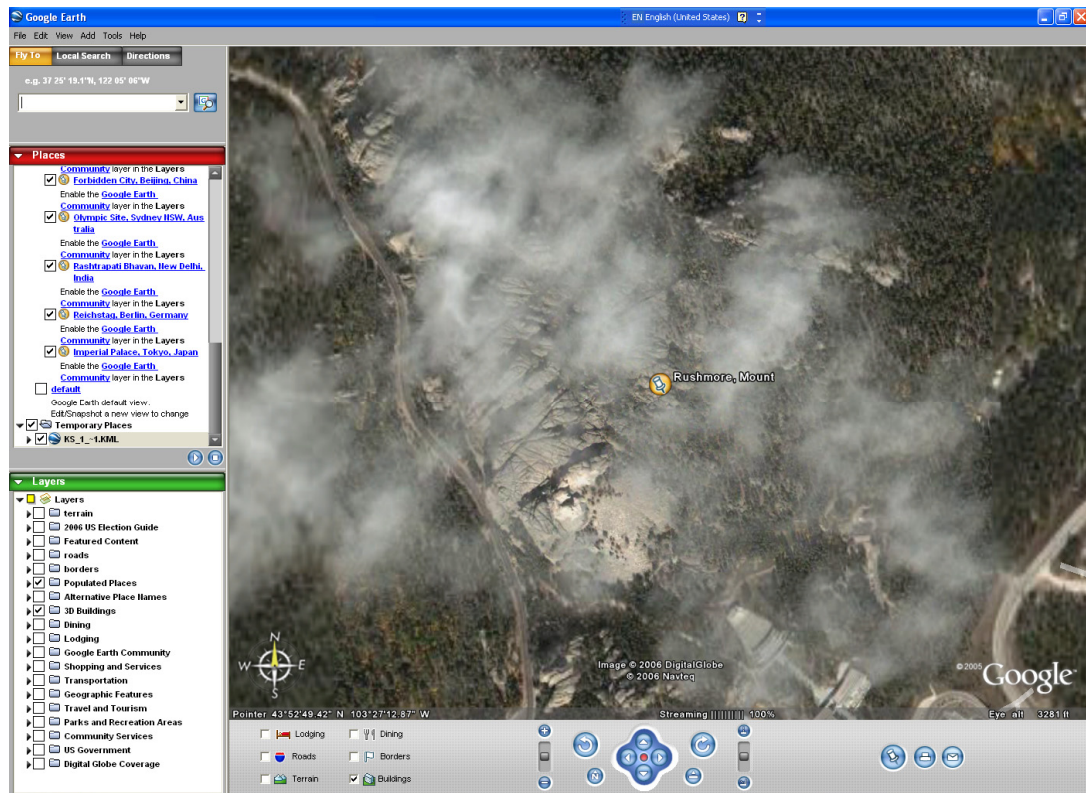



Figure 10 Google Earth with a place “Rushmore, Mount”

10. *TerraServer & Yahoo Images*: the User Agent passes user-selected location information and a synonym set to the Web Services agent. Then, the Web Services agent formulates refined queries for each source and requests images from the sources. After KS receives the image information, the User Agent requests an evaluation of the image results by passing the results together with the original user query and user preferences to the Ranking Agent. Lastly, the User Agent displays a results table initially sorted by total similarity received from the Ranking Agent. In the results table, the user can do the following: 1) sort results for each criterion by clicking column name, 2) see the original image or be directed to an original web page of the image source by clicking thumbnails in the results table, and 3) be directed to a request page of refined queries for TerraServer and Yahoo Images.

[Image Search results From TerraServer and Yahoo]

Thumbnail	Name	Theme	Location	Date&Time	Size	Resource	Total Similarity
		photo	-103.4583, 43.8803	1998-09-02T03:00:00	200*200	TERRA	0.57
		topo	-103.4583, 43.8803	1977-07-01T03:00:00	200*200	TERRA	0.57
	mt rushmore	photo			800*600	YAHOO	0.43
	mount rushmore bau 0625	photo			300*225	YAHOO	0.43
	07 Mt Rushmore	photo			1024* 768	YAHOO	0.43
	Mount Rushmore	photo			1170* 794	YAHOO	0.40

1 [2](#) [3](#) [4](#) [5](#) [6](#)

Figure 11 Image Results Page

3.3 KS Agent Interactions and Communications

As described in the previous sections, the KS system maintains various processes to achieve its goals, and KS Agents use Web Services to communicate with each other. The KS system also uses pre-specified workflows among agents based on scripts and protocols created for KS agent interchanges and communication. Figure 12 shows a flow chart that specifies a work flow of the user search process in terms of agent interaction, and it shows how each agent collaborates during the search process.

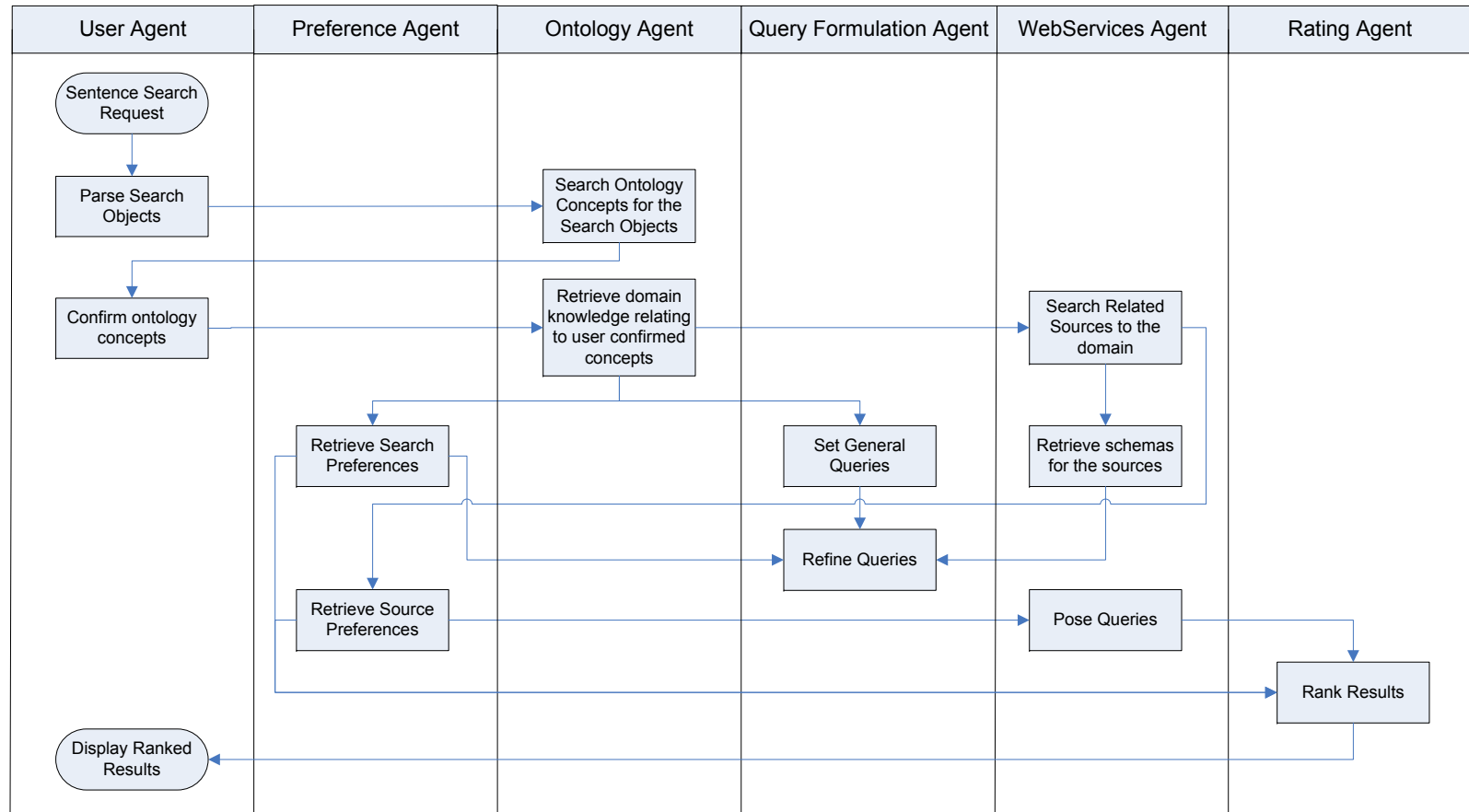


Figure 12 A Flow Chart for KS Search Process via Agent Interaction

Figure 13 represents KS agent communication scenarios using XML schemas through Web Services. Each agent maintains XML-based schemas of the specifications of request and response messages required for data exchange during agent interactions. Thus, any agent that wants to invoke Web Services from other agents is required to obtain the XML schema of the request from the service-provider agent to invoke the service. The requester agent also is required to obtain a result XML schema to parse the information from the resulting XML obtained from the service-provider agent. This mechanism prevents schema mismatching which might be caused by managing data schemas in more than one place. For example, let's assume a requester agent maintains an XML schema of data residing in a response agent. If the response agent changes the schema for some reason, the schema held by the request agent would no longer be valid. Therefore, the data schemas are better to be resident in only one agent that manages the data in order to remove the mismatch problem. Because of that, the schemas need to be requested by other agents every time when the data is requested to be exchanged. The XML schemas of representing data and its managing agents are provided in Appendix A.

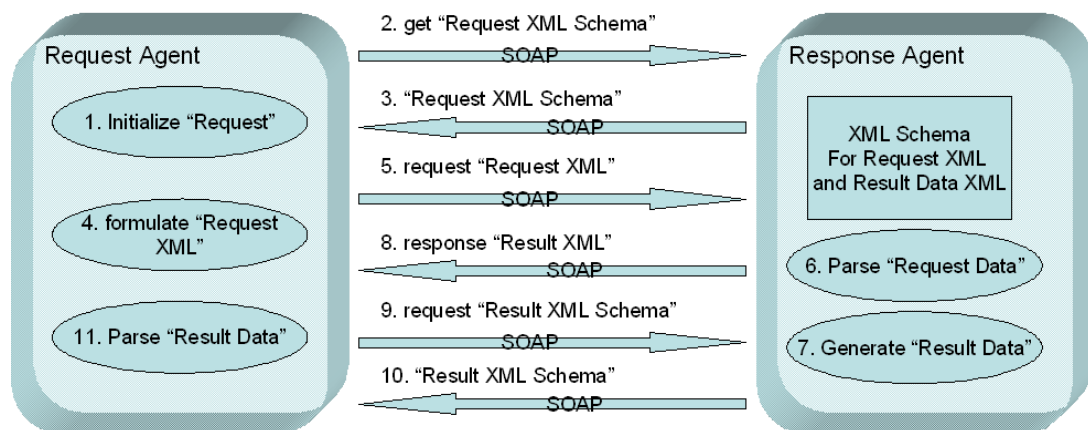


Figure 13 Knowledge Sifter Agent Communication Diagram

3.4 Emergent Semantics in Knowledge Sifter

The previous sections have described how the cooperative agents and Web Services support the search for relevant knowledge from both local and open-source data sources. The end-to-end scenario shows how the various agents and sources interact. 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 a reusable search platform [39, 40]. The 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). Figure 14 depicts the Static Model for the KSMM. What follows is a brief overview of the classes and relationships shown in Figure 15. 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 OntologyAgent 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.

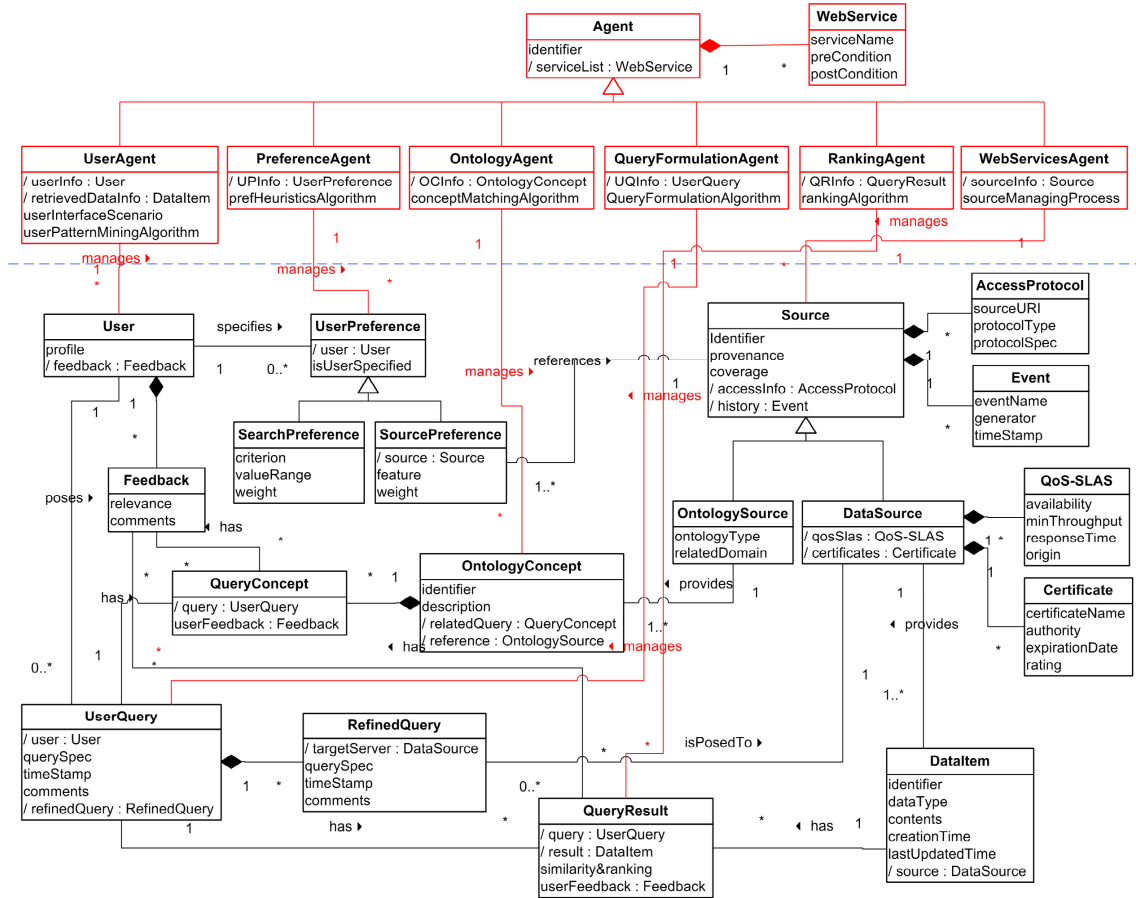


Figure 14 Knowledge Sifter Meta-Model Schema

A UserQuery consists of several RefinedQuery, each of which is posed to several DataSource. DataSource provides one or more DataItem as the QueryResult in response to a RefinedQuery. Based on the returned QueryResult, the User 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 have been 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 be used for the collaborative query refinement, which is introduced in Chapter 5, by providing the relationships among the artifacts such as which users provided which feedback on which items in terms of which queries, and which were the preferred data sources.

Figure 15 represents the Entity-Relationships (ER) model for the KSMM. The ER model is implemented by using the MySql server [54], and MySql script file for the relational version of this ER model is provided in Appendix B. An XML-based flexible structure is used to specify all of the data managing by the KSMM, e.g., specifications of data items are stored into specXML attribute in DataItem as an xml snippet instead of using a pre-specified attribute list such as title and resolution (for image data). The specifications of data can vary based on the type of data item and source and also the specifications can be changed over time. Therefore, using the flexible structure enables us to manage any types or domains of data retrieved from heterogeneous data sources. The XML-based data specifications, specXML would be governed by the KS Ontology Agent via using WordNet ontology and domain ontologies. The domain ontologies would have

information about restrictions for some attribute values, e.g., the values for type of data item would be determined as enumeration type having a set of pre-specified values such as image and web page.

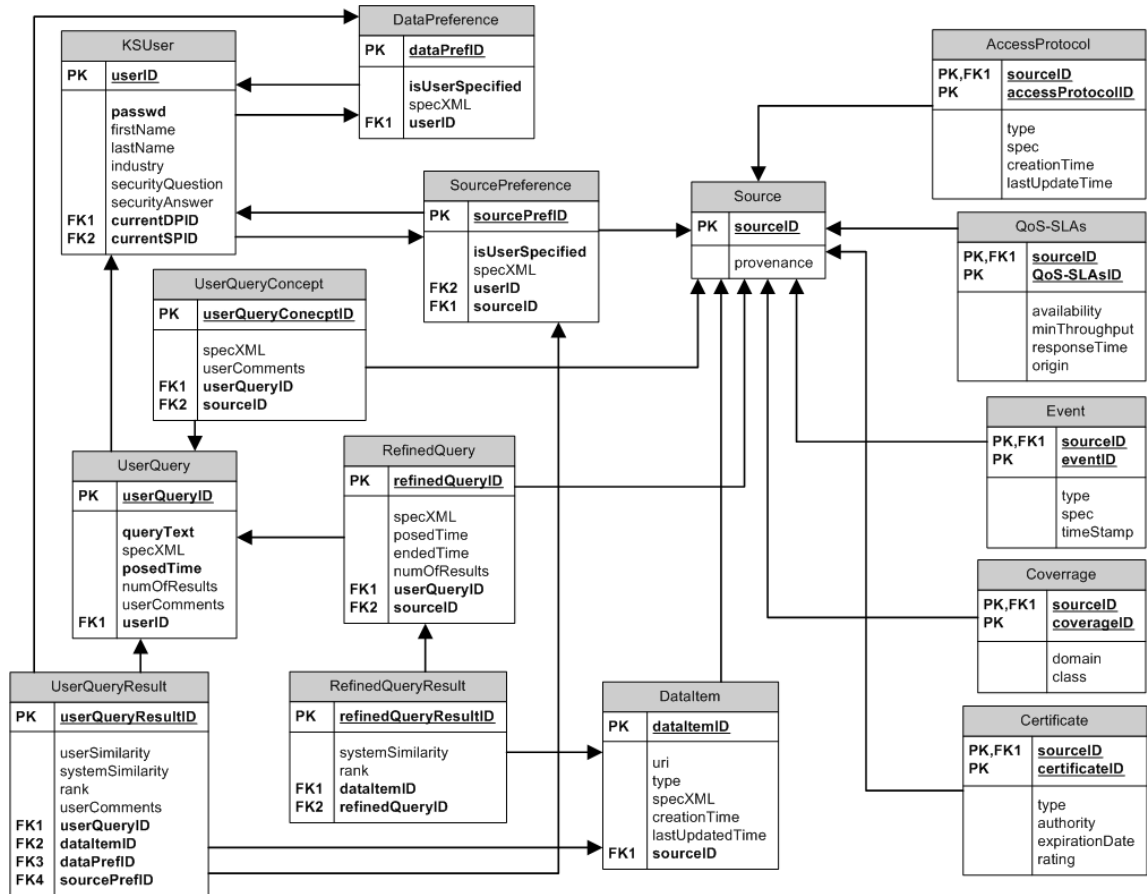


Figure 15 Entity-Relationship Diagram of the Knowledge Sifter Meta Model

4 Case-Based Knowledge Sifter Framework

The original Knowledge Sifter [37, 38] described in Chapter 3 maintains a repository of user queries and artifacts produced during the search process. The case-based reasoning methodology [7, 45] is adapted into the Knowledge Sifter framework to reuse knowledge contained in the repository of user queries and artifacts produced during the search process to improve its efficiency and effectiveness. In this chapter, a case-based reasoning framework is presented for Knowledge Sifter in order to retrieve and reuse the previously-stored user queries in a systematic way by specifying them as cases. The previously-stored user-query cases are represented in XML, retrieved per ontology-based concept indices, and reused to refine the specification of a user-query based on a hybrid filtering method combining collaborative filtering and content-based filtering. I call this process “*query-to-query hybrid filtering*.” The details of the query-to-query hybrid filtering are described in Chapter 5.

4.1 Case-Based Knowledge Sifter Architecture

Figure 16 shows an updated architecture representing the Case-Based Knowledge Sifter framework. A new agent, the Case Management Agent is introduced to manage processes and resources relating to the case-based reasoning methodologies used in the Case-Based Knowledge Sifter framework. The role of the Case Management Agent is to

communicate with the User Agent and retrieve those cases from the User Query Case Base that have the user relevance feedback. The Query Formulation Agent also communicates with the Case Management Agent in order to retrieve the cases of other user queries having concepts similar to the active user query (the query which is currently posed by the active user and targeted for the refinement). To efficiently and effectively retrieve such relevant cases, the Case Management Agent applies ontology-based indices to cases which index them based on one or more ontology concepts related to concepts of the user query cases. The details of the ontology-based indices are described in Section 4.3.

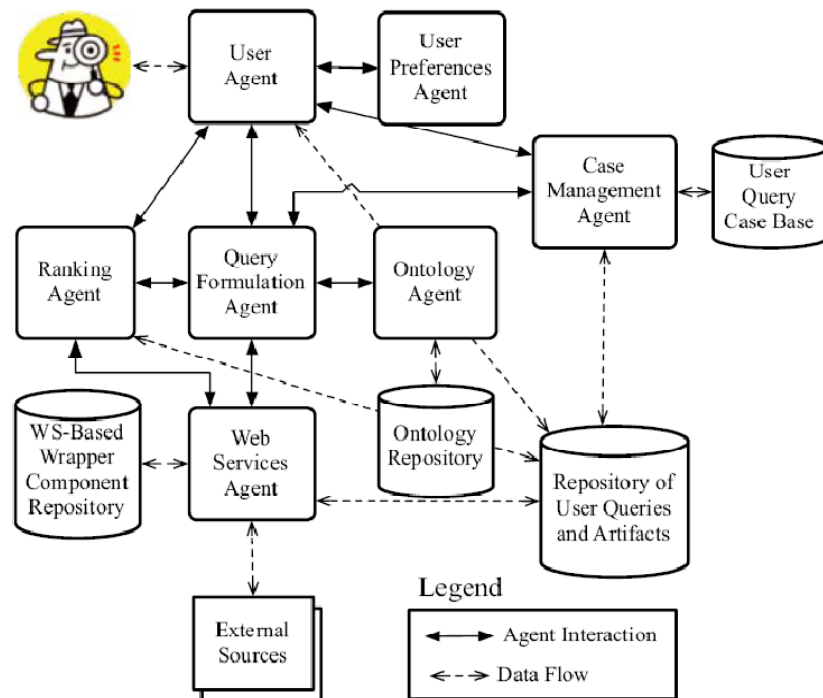


Figure 16 Knowledge Sifter Case-Based Architecture

An active user query is refined based on other user query cases that are similar to the active query using query-to-query hybrid filtering. Based on the specification of the refined query, Knowledge Sifter semi-automatically selects data sources and is dynamically configured with Web Services-based wrapper components for each selected data source. Knowledge Sifter also maintains a repository of the pre-compiled wrapper components for accessing data sources. The case-based Knowledge Sifter architecture inherits the general-purpose meta-model schema in Figure 14, and the XML-based flexible structure of the user query enables a system to manage many types of data retrieved from heterogeneous data sources.

4.1.1 Case Management Agent

The Case Management Agent, shown in Figure 16, has a role of managing the User Query Case Base by identifying user query cases from various agent interactions, storing to the User Query Case Base those cases having user relevance, and retrieving the cases based on ontology-based indices. The retrieved user query cases are sent to the Query Formulation agent upon its request. In other words, the Query Formulation Agent communicates with the Case Management Agent to retrieve cases according to user query and user preferences, and then uses the retrieved cases to refine user queries via the query-to-query hybrid filtering.

The user query cases are created and maintained in the User Query Case Base only if a user provides relevance feedback for a query result. Whenever a user provides relevance feedback for one or more query results and sends a request to save the relevance feedback into the Knowledge Sifter system, the User Agent sends the feedback

information with identifiers of a data source-specific query and its results to the Case Management Agent. All required information is then retrieved for creating a user query case from the repository of user queries and artifacts. The created user query cases are kept in the Case Management Agent cache until the Case Management Agent receives a notice of the end of the user query session from the User Agent. Upon receiving the session ending notice, the Case Management Agent permanently saves the user query cases to the User Query Case Base. The Case Management Agent uses the cache to avoid the duplicate retrieval of all the required information if the user sends another request of saving new relevance feedback or changing previous relevance feedback. This would be employed for a user query, data source-specific queries, and query results having the relevance feedback already in the user query session.

After the user gives relevance feedback, he is allowed to send a request to refine the query to the Knowledge Sifter system. When the user sends the request to refine the query, the Case Management Agent returns the cached user query case information, along with other user query cases in the User Query Case Base that are similar to the active user query case, to the Query Formulation Agent. The similar user query cases are selected based on the case retrieval algorithm that determines the similar query cases according to their similarity to the active user query case. This is accomplished in terms of the ontology-guided concept similarity as described in Section 4.3. When the user issues a new user query which is refined from the query-to-query hybrid filtering method and confirmed by the user, the Case Management Agent updates the user query in the cache. However, the Case Management Agent keeps the relevance feedback information

(identifiers of the active user and the query results) obtained from the previous versions of the user query because the relevance feedback would better represent the user's true search intent. The query results most highly rated by the active user can represent the user's search intent better than the user's query. This is because the users have difficulty in specifying their information needs as a query, and thus a user query often does not sufficiently represent a user's true information needs. All of the accumulated user feedbacks are used to refine an active user query incrementally. Lastly, the final version of the refined query and all the relevance feedback information in the user query session are stored in the User Query Case Base.

4.1.2 Web Services-Based Wrapper Component Repository

A wrapper component repository is created to maintain the pre-compiled Web Service-based wrappers, which are used to access heterogeneous data sources while providing a standard Application Programming Interface (API) to the Web Services Agent based on a Web Services standard. The wrappers provide a REST-based Web Services API, so that the Web Services Agent can retrieve data from the data sources through the wrappers with the simple and standardized syntax of data representation provided by the API. Therefore, using the Web Services API causes the Web Services Agent to consider only the semantic heterogeneity of the data representations created from heterogeneous data sources by removing the syntactic heterogeneity.

The domain ontologies available to Knowledge Sifter, such as the image domain ontology, are used to mitigate the semantic heterogeneity for the image search from heterogeneous data sources such as TerraServer and Yahoo! Images. The Web Services-

based wrappers are described by Web Services Definition Language (WSDL) with the domain ontology-guided semantic annotations on the WSDL elements, e.g., input and output elements of a Web Service. This approach of annotating WSDL elements by a domain-specific ontology is similar to the W3C approaches, SAWSDL [20, 47] and OWL-S [51]. In other words, each data source is endowed with a Semantic Web Service which annotates its data by concepts in a domain ontology shared among the agents in the Knowledge Sifter system. The domain ontology is also used to formulate user queries in the Knowledge Sifter system, so data sources can be dynamically bound to the Web Services Agent via a simple ontology-concept mapping. This is because the data schema of the data sources is semantically represented by using the shared domain ontology that are being used to formulate the user query by the Query Formulation Agent.

4.1.3 Use Cases and Sequence Diagrams

Knowledge Sifter has three main use cases: 1) semantic query refinement use case, 2) collaborative query refinement use case, and 3) data retrieval use case. The semantic query refinement use case represents a process which semi-automatically refines a keyword-based user query with its user-confirmed related ontology concepts. The collaborative query refinement use case represents a process which refines the user-confirmed domain-specific query by using the query-to-query hybrid filtering method. Lastly, the data retrieval use case represents a process which retrieves data from some external data sources related to the user-confirmed domain-specific query. A user search process can be materialized by combining either the semantic query refinement process or the collaborative query refinement with the data retrieval process.

Initially, if a user inputs a user query consisting of a set of keywords, the user query is refined with ontology concepts retrieved from external ontologies via the semantic query refinement process. A set of neighbor user query cases, which are semantically similar to the active user query, are found based on user-selected ontology concepts. Per the neighbor user query cases, a set of domain-specific queries are created and suggested to the active user. When the user selects a domain-specific query, results are retrieved from heterogeneous data sources by using data source-specific queries via the data retrieval process. After a set of sorted results is presented, the user can provide relevance feedback for top N results. KS then refines the user-selected domain-specific query via the collaborated query refinement process using the query-to-query hybrid filtering. Users are able to retrieve data again by modifying and posing the refined domain-specific query to the data retrieval process. Users can also repeat the query-to-query hybrid filtering-based search cycle by providing relevance feedback and posing the further-refined query based on the relevance feedback repeatedly via the collaborative query refinement and data retrieval processes. Sequence diagrams of the three main processes are shown in Figures 17-19.

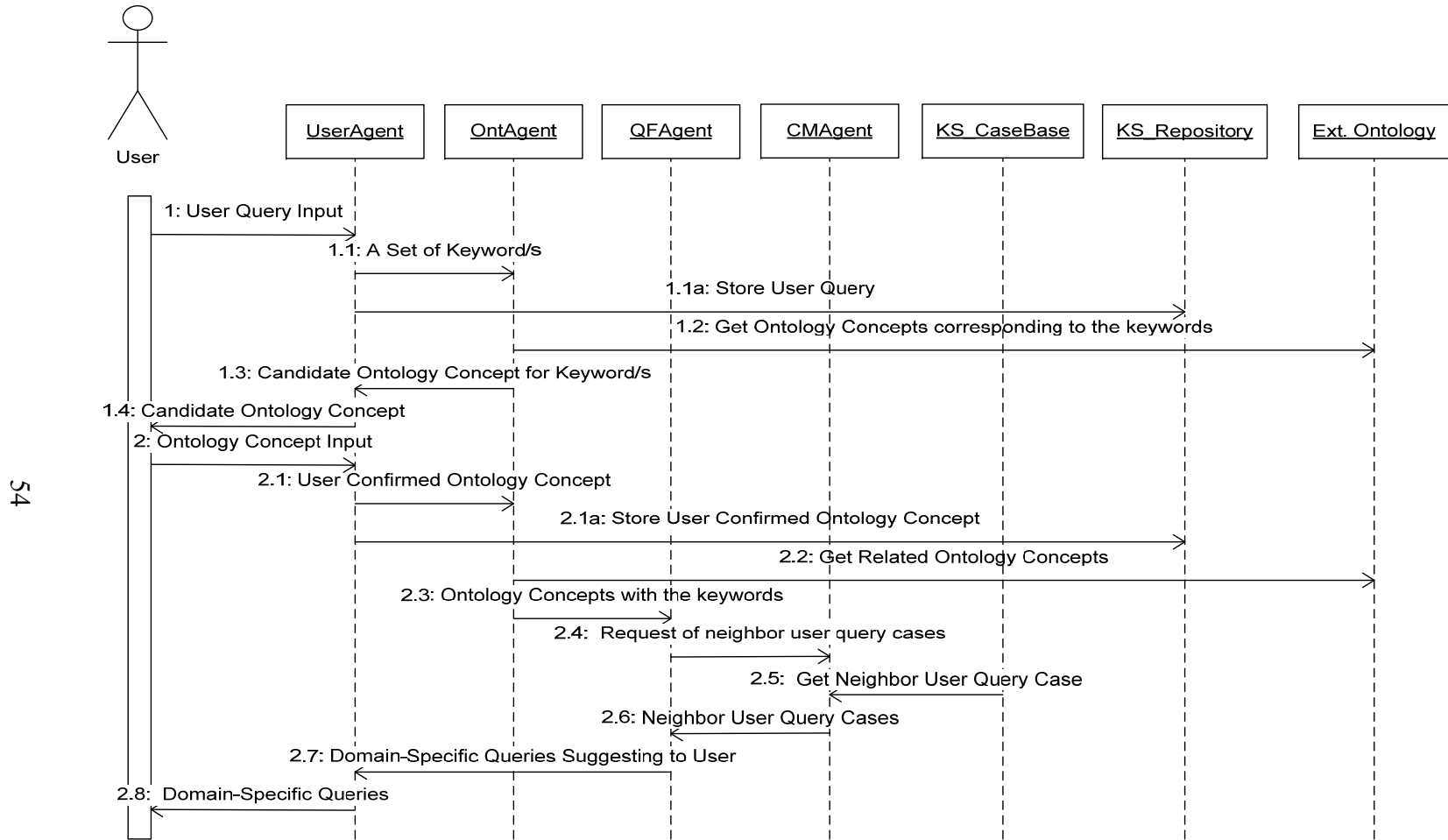


Figure 17 Semantic Query Refinement Sequence Diagram

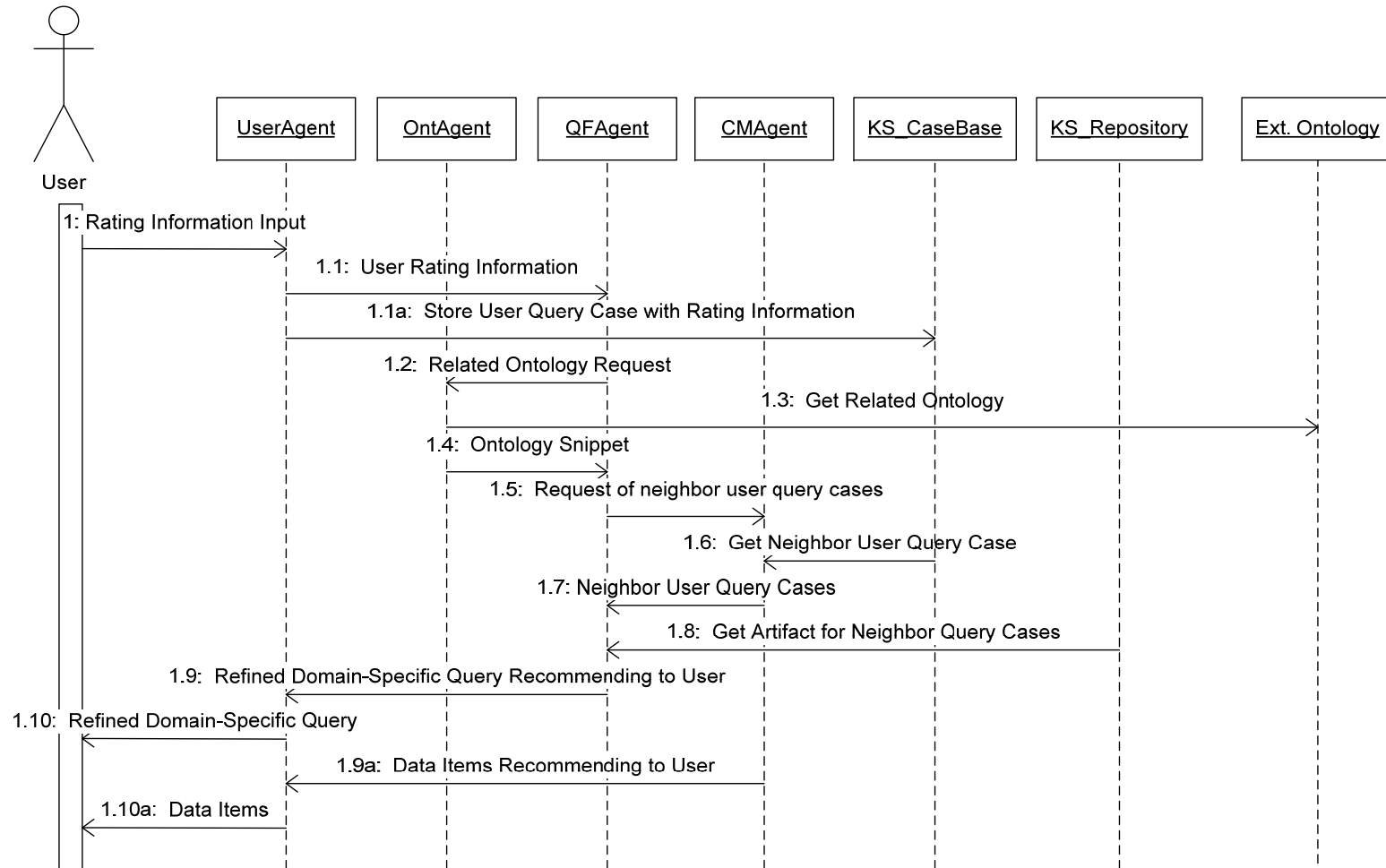


Figure 18 Collaborative Query Refinement Sequence Diagram

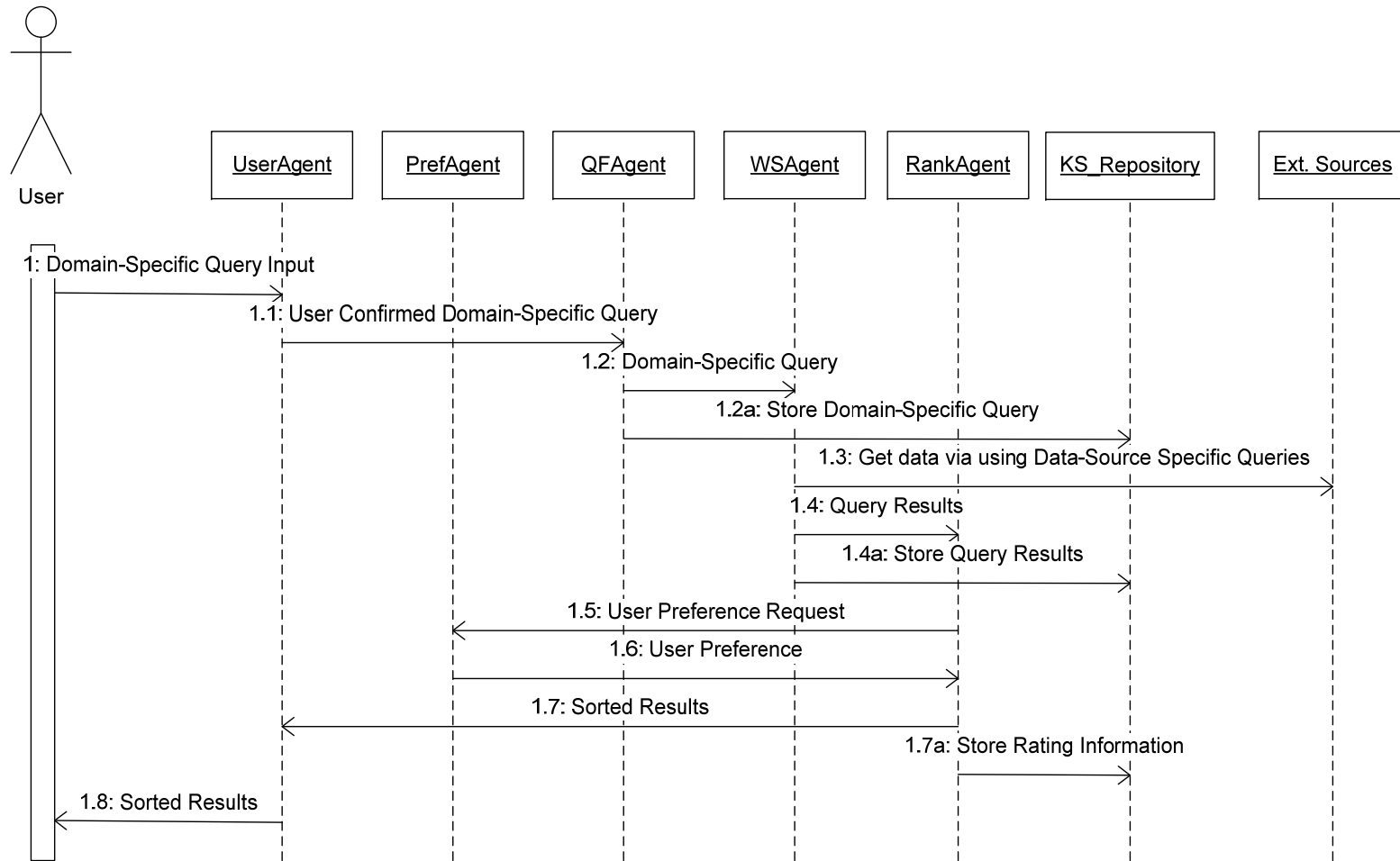


Figure 19 Data Retrieval Sequence Diagram

4.2 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 a user-selected information domain. Basically, KS uses two types of user query; one is conceptual query, and the other is domain-specific query. As a case consists of a problem and its solution in a case-based reasoning framework, the conceptual query is regarded as a problem and the domain-specific query is being regarded as the solution in the case-based Knowledge Sifter framework.

The conceptual query represents user information needs via using a set of ontology concepts. A conceptual query will be generated after a keyword/s-based user query is semantically refined by specifying WordNet concepts and domain ontology concepts representing user concepts. This conceptual query will be used to find a candidate set of neighbor user query cases that might be similar to the active user query case in terms of general concepts similarity. A set of domain-specific queries is created from a clustering of the candidate set of neighbor user query cases based on information domains of the cases. A domain-specific query consists of a set of content features (metadata) and their associated values and weights. Multiple domain-specific queries can be generated for a conceptual query since two conceptual queries consisting of a similar set of general concepts might represent different information domains of user interest.

Finally, the pre-filtering of the candidate set of neighbor user query cases by means of the conceptual query can reduce the significant number of user query cases

which are required to be navigated in performing query-to-query hybrid filtering. This process enhances the efficiency of the automatic query refinement overall because the hybrid filtering is expensive in terms of time and memory resources. Furthermore, using the domain-specific query enables KS to use hybrid filtering for query refinement. This is because queries of different information domains can have different lists of features for specifying the queries in which the hybrid filtering cannot be applied or would not be effective.

Figure 20 shows an XML-based structure for the case representation, and the source code of its XML schema can be found in Appendix C.1. A case has its own identifier, `caseID`. A case also contains a `UserName` to identify its user, and this user identifier will be employed to perform the hybrid filtering and retrieve the user's preferences. Each case includes an associated conceptual query and a domain-specific query. A conceptual query can have multiple concepts which consist of a user term, zero or more WordNet senses, zero or more domain ontology concepts, and a weight. The `DomainConceptID` is an identifier of a domain concept which has been found from the semi-automatic semantic refinement of user query. WordNet sense identifiers chosen by a user for specifying the user's actual concepts of a search term are also included. WordNet is employed as a general upper ontology, and the referenced WordNet concepts and the referenced domain concepts serve as an index of the user query, as described in Section 4.3. The concept weight is the degree of importance the user assigns to a concept.

A domain-specific query has exactly one information domain for which the query is specified. The domain-specific query is a weighted multi-dimensional/multi-valued

query. The feature name is also a variable since the schema of a domain-specific query will be determined by its information domain and the user-selected data sources. The data source information is also a feature of the domain-specific query and it 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 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 because KS retrieves data from multiple heterogeneous data sources. Some standardized metadata such as Dublin Core Metadata [6] can be used to describe feature attributes.

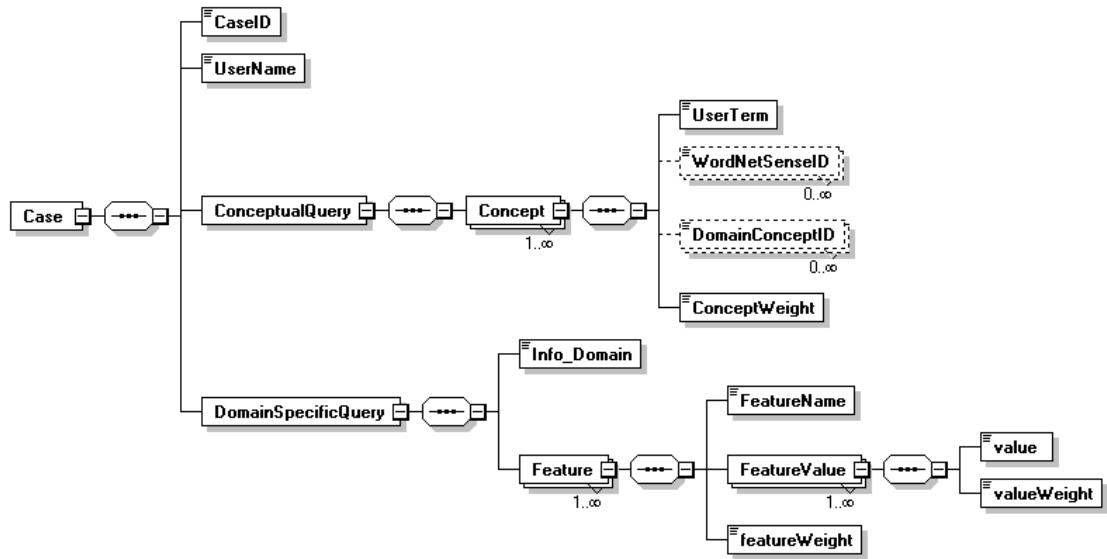


Figure 20 XML-based Semantic Representation of a User Query Case

Figure 21 represents an example that shows how a user query can be specified through using the semantic representation of user query cases. Let's suppose one wishes to visit the Washington Monument and then dine at a steakhouse located near the Washington Monument. The keyword terms in a query might then be "steakhouse" and "Washington monument." Ontology concepts defining user concepts can be semi-automatically found from referenced ontologies by using the keyword terms. The ontology concepts, "WN:steakhouse_1" and "WN:washington_monument_1" denote WordNet concepts representing user concepts for the "steakhouse" and "Washington monument" keywords respectively. The "GNIS:Washington_monument" concept can be found by a direct search on the geographic-domain ontology, or GNIS using the "Washington monument" term, or by ontology reasoning using ontological relationships such as "OWL:equivalentClassOf" between the WordNet concepts and the GNIS concepts.

An active user selects ontology concepts that match to the user's intended concepts among the automatically retrieved concepts. Then, the candidate domain-specific queries are created through an automatic domain-based accumulation of domain-specific queries found from the other user query cases, having ontology concepts related to the ontology concepts chosen by the active user. Two candidate domain-specific queries can be generated for the example query as shown in Figure 21. One is for the food and restaurant domain, and the other is for the real estate domain. If the active user were to add either to concept "dining" or the concept "starting a restaurant business",

then a candidate domain-specific query either for the food and restaurant domain or for the real estate domain would be generated, respectively.

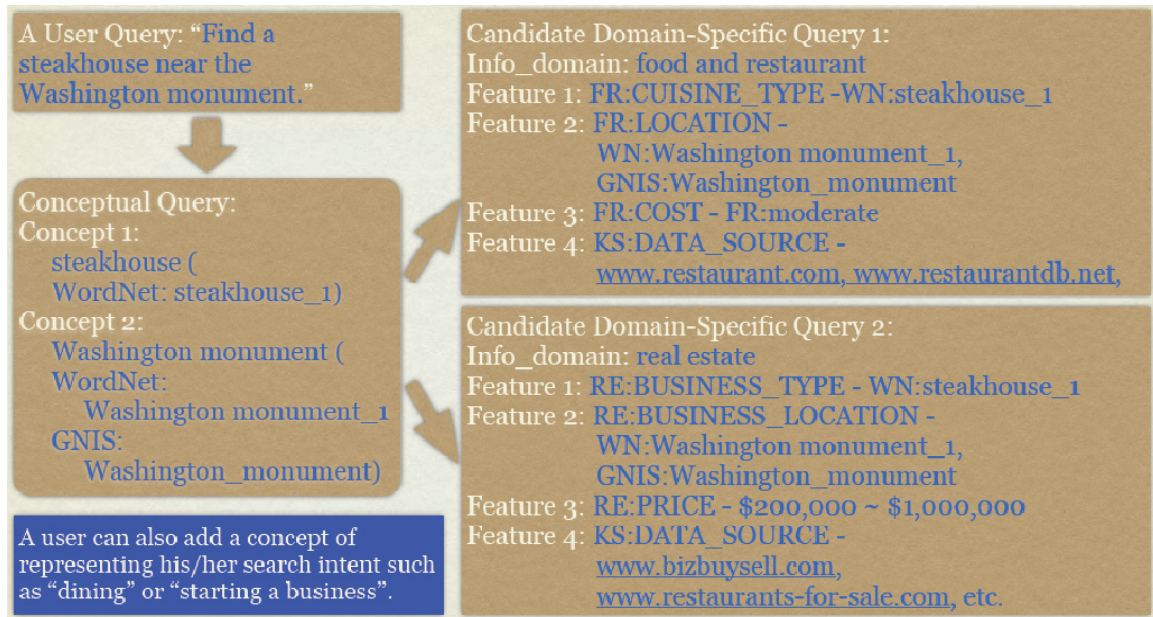


Figure 21 An Example of Representing A User Query Using the User Query Case Representation

4.2.1 Semantic Refinement of a User Query

The WordNet sense for a user term is created to relate domain ontology concepts to the user term, thereby enabling Knowledge Sifter to perform the semantic search rather than just the keyword/s-based search. WordNet is employed in order to avoid the semantic ambiguity of linguistic terms. WordNet is also regarded as a general upper ontology in Knowledge Sifter and several domain-specific ontologies such as places,

restaurants, and wines can be linked to the WordNet ontology. The linkages between WordNet concepts and domain ontology concepts are used to translate a user query into domain-specific queries and/or data source-specific queries.

As described in the sequence diagram shown in Figure 17, a user is required to select one of the WordNet senses for the user-entered search term. Then, KS stores to the KS repository the user's selected WordNet sense with the search term. The WordNet senses are used to enlarge the user's vocabulary. Therefore, domain concepts related to the user search concepts are obtained from domain ontologies by using the enlarged vocabulary such as synonyms of the user terms found in WordNet. These obtained WordNet senses and domain concepts will be used to create domain-specific queries and be maintained as a user query case in the KS case base for a later use of refining other user's queries.

4.3 Case Retrieval via Ontology-Based Indices

The Case Management Agent maintains ontology-based indices for entire cases. As represented in Figure 20, each user concept in a conceptual query can have zero or more ontology concepts related to the user concept. For each ontology concept, case identifiers referencing the ontology concept are stored as the indices. This ontology-based index approach allows for efficient retrieval of cases having similar search concepts because it explores related ontology concepts first, rather than navigating the large number of the user query cases. WordNet senses related to the user concept are also included and the senses are used as bridges between two different domain concepts which

are similar in semantics, as described in the previous section. Thus, this ontology index structure also assists KS to effectively find other similar user query cases in terms of using the virtual upper ontology approach.

4.3.1 XML-based Representation of Ontology-Based Indices

In this section, I describe the XML schema for the ontology-based index structure. Figure 22 represents the index structure which has domain concept-based indices consisting of a domain concept identifier, its related WordNet sense identifiers, and its corresponding case identifiers. The source code of XML-schema of the index structure can be found in Appendix C.2.

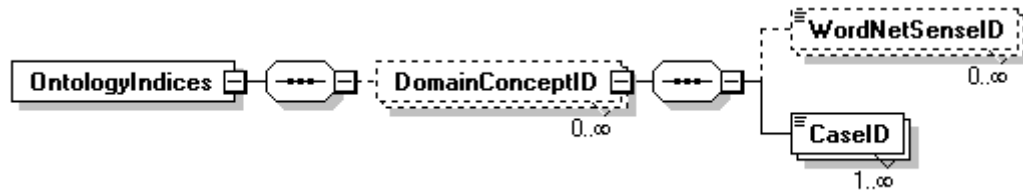


Figure 22 XML-Schema for Ontology Index

As represented in Figure 22, the root element, OntologyIndices can have zero or more DomainConceptIDs which represent identifiers of domain ontology concepts. The identifiers are RDF-based concept identifiers, which also have the identifier information of their owner ontologies specified in OWL. Each domain concept has zero or more related WordNet senses, so the domain concepts and their related cases can be found by

the referenced WordNet senses. Lastly, each domain concept has one or more of its corresponding case identifiers. The minimum cardinality of the case identifier is set to one because only the domain concepts having at least one related user query case are stored to the index structure.

4.3.2 Case Retrieval Algorithm via Ontology Index

As described in the previous section, WordNet concepts are used as the main ontology indices, and other user query cases consisting of concepts similar to the concepts of an active case can be retrieved by using the ontology indices. However, a user query case contains multiple WordNet concepts in general. For example, a user query {Washington Monument, steakhouse} has two concepts, “Washington Monument” and “steakhouse.” This approach is to create the ontology indices based on each atomic concept of a user query case. Thus, the case identifier of the user query case example will be stored into each ontology index for concepts, “Washington Monument” and “steakhouse.” This user query case can be retrieved as similar cases for user queries {Washington Monument}, {steakhouse}, and {Washington Monument, steakhouse}. This approach is computationally more efficient than creating an ontology index based on a set of all concepts in a user query case. It is preferable because if a conceptual query consists of n concepts, and indices are created for all of the concept combinations (the powerset of the set containing the all n concepts except the empty set), thus $2^n - 1$ indices would need to be visited instead of n indices. Furthermore, if we assume that overall conceptual queries consist of n concepts, the size of the indices would be 2^n times bigger than the size of the indices being created in this approach.

Figure 23 presents an algorithm for retrieving cases similar to the user query via ontology-based indices. First, the algorithm generates expanded queries of every possible combination of concepts, including their equivalent and generalized concept using the function, *expandedQueries*. The function uses Cartesian product to find the every possible combination. 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 limits the number of the cases retrieved to a pre-specified maximum. The maximum is bounded by the parameter *maxnc* in algorithm specification. For efficiency purposes, whether the pre-specified maximum number of cases is retrieved or not will be checked before expanding one element query of powerset of the user query. This is 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 Equation (4). Note that the original user query of the active case is also one of the expanded user queries.

The $\text{sim}(C_a, C_i)$ in the algorithm is a similarity between the expanded user query of the active case and the user query of the retrieved cases. The similarities are calculated by using the cosine correlation which is widely used for the vector model in the information retrieval area [11] as defined in Equation (5). This measure will be used in

Chapter 5 as the similarity measure between the active case and the retrieved cases in terms of the semantic relatedness among their conceptual user queries.

$$w_{eq_a} = \sum_{c \in EC} tw_c \cdot uw_c$$

$$tw_c = \begin{cases} 1.0 & \text{if } c \text{ is an original user concept} \\ syw & \text{if } c \text{ is an equivalent concept of the user concept} \\ hyw & \text{if } c \text{ is a generalized concept of the user concept} \\ syw \times hyw & \text{if } c \text{ is a generalized concept of the equivalent concept} \end{cases} \quad (4)$$

where eq_a represents an extended query of the active user query, and tw_c represents a pre-defined weight for a concept c in eq_a as defined above. The terms syw and hyw denote the pre-defined weight for an equivalent (synonym) concept and a generalized (hypernym) concept of an original concept in the active user query, respectively. The term uw_c is a user-defined weight for an original user concept of concept c . EC is the entire set of concepts consisting of a query eq_a .

$$sim(C_a, C_i) = \frac{\sum_{c \in CC} w_{c,eq_a} \cdot w_{c,uq_i}}{\sqrt{\sum_{c \in CC} w_{c,eq_a}^2} \cdot \sqrt{\sum_{c \in CC} w_{c,uq_i}^2}} \quad (5)$$

where w_{c,eq_a} and w_{c,uq_i} represent the weight for a concept c in an expanded user query of the active case eq_a and the user query of the other retrieved case uq_i , respectively. CC is a set of common concepts shared by eq_a and uq_i .

The similarities between user query cases are also used to terminate the case retrieval algorithm by using a threshold, `simThreshold`. Finding every possible expanded queries using the powerset and Cartesian product can be double exponential. This can be a burden to a system even though users use only a few concepts for creating a query in general. Thus, the `maxnc` and `simThreshold` bases are used to terminate the retrieval algorithm if the sufficient number of neighbor query cases are retrieved and expanding queries further does not preserve a certain level of similarity with the active user query. This will improve the efficiency of the case retrieval by not sacrificing its effectiveness, i.e., by finding expanded queries having the greatest similarity with the active user query at a given time.

```

SET maxnc TO 100
SET threshold TO 0.7
SET RCS TO the empty set
SET EQS TO the empty set
SET PCS TO the powerset of uqa excluding the empty set

ADD all elements of PCS TO EQS
SORT EQS in descending order of a weighted sum of concept weights in eq
FOREACH eq in EQS
  IF eq is in PCS
    ADD expandedQueries(eq) TO EQS
    SORT EQS in descending order of a weighted sum of concept weights in eq
  ENDIF
  SET NCS TO a set of other query cases indexed by every concept in eq
  FOREACH Ci in NCS
    IF sim(Ca, Ci) > threshold
      ADD Ci TO RCS
      IF COUNT(RCS) > maxnc
        RETURN(RCS)
      ENDIF
    ELSE
      RETURN(RCS)
    ENDIF
  ENDFOR
  REMOVE eq FROM EQS
ENDFOR

FUNCTION expandQueries(query)
  SET temp_EQS TO the empty set
  SET ECS TO the empty set
  FOREACH concept in query
    ADD a set containing the concept and its equivalent concepts TO ECS
  END FOR
  SET CPECS TO Cartesian product of all sets in ECS
  FOREACH cpec in CPECS
    SET HCS TO the empty set
    FOREACH concept in cpec
      ADD a set containing the concept and its generalized concepts TO HCS
    ENDFOR
    SET CPHCS TO Cartesian product of all sets in HCS
    FOREACH cpec in CPECS
      ADD cphc TO temp_EQS
    ENDFOR
  ENDFOR
  RETURN(temp_EQS)

```

Figure 23 Case Retrieval Algorithm via Ontology Index

5 Collaborative Query Refinement

In the previous chapter, a Case-based Knowledge Sifter framework and detailed process sequences are introduced to reuse knowledge, which is obtained during user search processes. In this chapter, a hybrid filtering-based method, *query-to-query hybrid filtering*, is introduced. This method shows how the retrieved search cases can be collaboratively reused to refine user queries based on the artifacts captured during the user search processes, including user feedback. Technically, query-to-query hybrid filtering combines content-based filtering with collaborative filtering to use the search history obtained not only from an active user, but also from other users, for mining the active user's preference and refining the active user's query. Also, the query-to-query hybrid filtering approach is based on the ternary relationships among users, user queries, and data items. The relationships are maintained and provided from the KS repository via using KS meta-model schema shown in Figure 14.

Content-based filtering is a method of recommending unseen (unrated) data items to a user based on the content patterns of data items preferred only by the active user. It can assist the user in refining a query based on the artifacts of their past queries that are similar to the active query. However, for a new user, the similar queries may not yet exist in the active user's profile, or an acceptable number of data items preferred by the user are not available because most users do not take the time to provide feedback. This lack

of feedback is ameliorated by using collaborative filtering, which attempts to predict the usefulness of as yet unseen items for an active user, by proposing items based on those previously rated by other users. The basic idea of collaborative filtering is to recommend a set of unseen items that are preferred by other users who have tastes similar to the active user. Thus, by combining collaborative filtering with content-based filtering, the short-comings of each technique, when used separately, can be mitigated.

Nevertheless, collaborative filtering cannot be applied directly to the case-based Knowledge Sifter framework because more than one user-query case per user, stored in the case repository, may be similar to the active user query. Most collaborative filtering-based recommendation systems [46, 48, 60] are not query-based retrieval systems and they simply recommend data items which are predicted to be of interest to users without considering topics or subjects of user information needs. That is, they consider only the binary relationships between items and users while the ternary relationships among items, users, and user queries are considered in this research.

A better approach is to recommend a *single-aggregated domain-specific* query from the search cases of other user queries that are similar to the active user query. Therefore, the query-to-query hybrid filtering that refines the active domain-specific query, based both on the active user query case and the neighbor user query cases, can be performed effectively. This is because the lists of the content features (metadata) for a data item are determined by the domain of the data, e.g., the price, cuisine, and location features can be used for restaurant search, and the genre, director, and actor features can be used for movie search. Thus, determining the domain of search prior to the hybrid

filtering-based query refinement helps our system to find metadata patterns preferable to the active user using the hybrid filtering.

However, if there are no previously-stored user-query cases posed by the active user in the selected similar cases, the collaborative filtering cannot be directly used for refining the active user query. This situation occurs because no active user's feedback exists for performing the collaborative filtering. To address this problem, a set of domain-specific queries that can be simply aggregated from the domain-specific queries of the selected cases similar to the active query case can be recommended. The active user chooses one of the domain-specific queries having a domain of the user's interest; then, the query becomes the active domain-specific 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 domain-specific query by dynamically translating it to one or more queries according to each data source's schema/ontology, as shown in the previous chapter.

Finally, the active user can provide more feedback on some other results and request another recommendation of the query specification. At this time, the amount of user feedback for performing the hybrid filtering is incrementally enlarged with the new user feedback. The query refinement via the query-to-query hybrid filtering uses the entire user feedback obtained from all of the search processes during the multiple times of the query refinement requested by the active user. As described in the previous chapter, the final refined query will be stored in the case base as a new case with the entire user

feedback results provided by the active user during a session, which ends if the active user poses a new query representing different information needs or logs out from the KS system.

Figure 24 shows an example representing how the query-to-query hybrid filtering works using the example shown in Figure 21. After the active user selects and poses an initial domain-specific query and provides relevance feedback for the first few results, collaborative filtering can be performed to obtain more data items from the neighbor user query cases. The combined resulting data items can be used to refine the active domain-specific query by using semantic content-based filtering. As shown in the example, the active query is refined with new values such as “american” and “www.restaurantrow.com” for the cuisine type and data source features, respectively. In addition, the active query can be refined even with a new criterion, “feature” and its value.

The following subsections describe in greater detail how to achieve the abovementioned recommendations. Section 5.1 describes how a domain-specific query is recommended to an active user without using any of the active user’s feedback. Section 5.2 describes how previously found data items can be recommended to the active user from the neighbor users’ search cases based on a query-to-query hybrid filtering created primarily by using collaborative filtering. Section 5.3 introduces another query-to-query hybrid filtering method that refines the active user’s domain-specific query primarily based on content-based filtering.

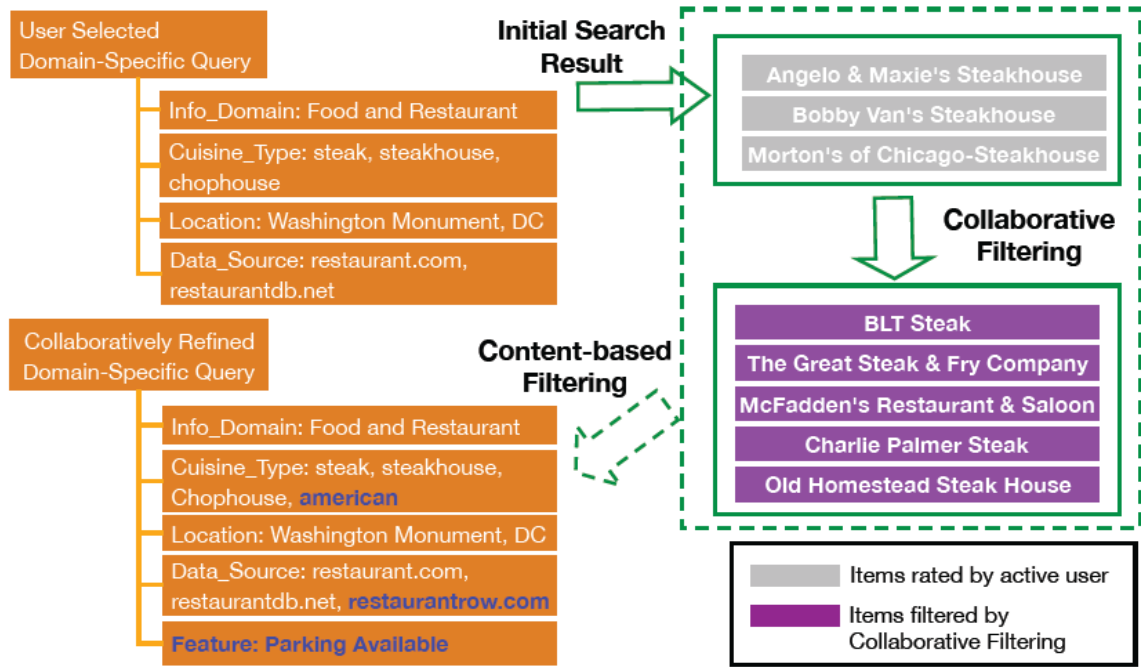


Figure 24 An Example of Collaborative Query Refinement for the Example in Figure 21

5.1 Initial Query Refinement without User Feedback

As discussed above, query-to-query hybrid filtering cannot be applied without the user feedback (the active user's ratings on items). In this case, a query specification can be generated by statistically aggregating the neighbor-refined queries, which can be found from the retrieved similar cases using the ontology-based indices. The aggregated feature weight can be found by a simple statistical mean determined in Equation (6). The value weight can also be determined in a similar way. However, the recommended specification can be meaningless if there are a number of significantly different domain-specific queries in the set of similar cases. This can happen because a user's conceptual

queries, which index the cases, are not specific enough. The problem can be alleviated by the query-to-query filtering approach described in the following sections. Further, this approach can provide most popular features and their values according to the user query, e.g., popular (meaning that data sources are reliable) data sources can be recommended based on their associated weights from the data source feature.

$$w_{f,q_a} = \frac{\sum_{q_i \in NC} w_{f,q_i} \cdot \text{sim}(C_a, C_i)}{\sum_{q_i \in NC} \text{sim}(C_a, C_i)} \quad (6)$$

where w_{f,q_a} represents a weight for a feature f in the active domain-specific query q_a , and w_{f,q_i} is a weight for a feature f in a neighbor domain-specific query q_i found in a set of neighbor cases, NC . $\text{sim}(C_a, C_i)$ represents the similarity between the active case and a neighbor case in terms of their conceptual user query and it can be calculated using Equation (5).

5.2 Immediate Data-Item Recommendation from Neighbor Cases

After an active user provides some ratings on resulting data items retrieved for a domain-specific query, the active user's rating values on unseen data items can be predicted from a query-to-query hybrid filtering based on the rating patterns of the active user and neighbor users for domain-specific queries. The prediction can be calculated

from Equations (7) and (8), which are derived from the well-known collaborative filtering approach used in GroupLens [10].

This query-based hybrid filtering allows Knowledge Sifter to show the unseen data items immediately because the data items can be found in a neighbor's search history in the KS repository. The mismatch problem between users' conceptual queries and domain-specific queries can be alleviated by using a threshold for the similarity between the active user's conceptual query and a neighbor user's conceptual query. Only the domain-specific queries obtained from neighbor cases having a certain high similarity value in terms of the conceptual specification will be selected for this prediction process. It is a hybrid filtering-based technique that combines collaborative filtering and content-based filtering and includes a content-based filtering-based similarity measure for the case similarities.

$$p_{q_a, d_u} = \bar{r}_{q_a} + \frac{\sum_{q_i \in NC} (r_{q_i, d_u} - \bar{r}_{q_i}) \cdot \text{sim}(q_a, q_i) \cdot \text{sim}(C_a, C_i)}{\sum_{q_i \in NC} |\text{sim}(q_a, q_i)| \cdot \text{sim}(C_a, C_i)} \quad (7)$$

$$\text{sim}(q_a, q_i) = \frac{\sum_{d_s \in SD} (r_{q_a, d_s} - \bar{r}_{q_a}) \cdot (r_{q_i, d_s} - \bar{r}_{q_i})}{\sigma_{q_a} \cdot \sigma_{q_i}} \quad (8)$$

where p_{q_a, d_u} represents a prediction for an unseen (unrated) data item d_u for the active domain-specific query q_a . $\text{sim}(q_a, q_i)$ is the correlation weight which shows the similarity

between the user rating patterns of the domain-specific queries q_a and q_i as defined by the Pearson Correlation Coefficient shown in Equation (7). $sim(C_a, C_i)$ represents the similarity between the active case C_a and a neighbor case C_i in terms of the conceptual query as defined in Equation (5). NC is a set of neighbor cases selected as similar to the active case. SD is a set of common seen (rated) data items between q_a and q_i . $\overline{r_{q_a}}$ and $\overline{r_{q_i}}$ represent mathematical means for the ratings of the result data items of the domain-specific queries q_a and q_i , respectively.

Table 2 represents a user rating prediction of unseen data items using the query-to-query hybrid filtering shown in Equations (7) and (8). The table depicts five seen items d_{si} , which the active user rated for the active domain-specific query q_a . There are five unseen items d_{ui} , which do not have a rating value for the active domain-specific query from the active user, but have some rating values for the neighbor domain-specific queries q_i not only by the active user, but also by other users. The unseen items are found via the case retrieval algorithm introduced in Section 4.3. The rating values of data items for each domain-specific query in the examples shown in Table 2 have only binary rating values, but the equations can also work using various other rating scales. The rating value 1 represents that a user liked a data item and/or regarded the data item as relevant for a query, and 0 represents the dislike and/or the irrelevance of the data item. The $sim(q_a, q_i)$ values are calculated using the user rating values as shown in Equation (8). The $sim(C_a, C_i)$ values are given as shown in the table. Finally, the predicted rating values of the active domain-specific query q_a for the unseen data items d_{ui} can be calculated using Equation (7). As can be seen from Table 2, neighbor domain-specific queries having

higher similarities with / higher similarity values with the active domain-specific query can have a greater influence in determining the prediction of the rating values. The absence of rating values for an unseen data item, (e.g., d_{u5}) show that the formulas can also work well with sparse data.

Table 2 A User Rating Prediction of Unseen Data Items

	q_1	q_2	q_3	q_4	q_a
d_{s1}	1	1	1	0	1
d_{s2}	0	0	1	1	0
d_{s3}	1	1	0	0	1
d_{s4}	1	1	1	1	1
d_{s5}	0	1	1	0	0
d_{u1}	1	1	0	0	0.87
d_{u2}	1	1	1	1	0.78
d_{u3}	0	0	0	0	0.33
d_{u4}	0	0	1	1	0.24
d_{u5}	1	N/A	N/A	0	0.94
$sim(q_a, q_i)$	5.00	3.06	-2.04	-0.83	
$sim(C_a, C_j)$	1	1	0.6	0.7	

5.3 Query Refinement via Query-to-Query Hybrid Filtering

This section describes how the active user's domain-specific query can be refined by a query-to-query hybrid filtering technique which also combines a content-based filtering technique and a collaborative filtering technique simultaneously. The main idea of this technique is to find the content patterns preferable to an active user by refining the active user's domain-specific query in a collaborative manner; the list of features and their weights of representing the content patterns are determined by the hybrid filtering

technique based on the data items preferred not only by the active user, but also by the neighbor users who have rating patterns similar to the active user.

First, the active user's preferred set of data items are augmented by adding the neighbor users' preferred set of data items with their rating values predicted from Equations (7) and (8) in the previous section for the active user. The active user's domain-specific query is refined by content patterns of the augmented data items by using Equations (9) to (12). If the active user confirms and poses the refined query to KS, a new result set can be retrieved from a new data source set according to the newly refined query. Then, more data items unseen by the active user can be found from the collaborative query refinement with the new search artifacts. Thus, the domain-specific query can be incrementally refined by aggregating the rated/predicted data items from the several iterations of the search and refinement processes.

Equations (9) and (10) determine the value weight for each feature of the active domain-specific query simply based on the number of occurrences of values in the augmented data item set. The feature weight can be determined by Equations (11) and (12) 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 in determining his preference for the data. Therefore, this approach also takes into account the negative examples, which have a negative feedback from users whereas most 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 (see Eq. (11)) because the negative correlation would not necessarily mean that the user rated a data item as a relevant one since it is dissimilar to his/her query in the dimension of the feature, or vice versa.

$$w_{v_{ij}, q_a} = \frac{\overline{r_{v_{ij}, q_a}}}{\sum_j r_{v_{ij}, q_a}} \quad (9)$$

$$\overline{r_{v_{ij}, q_a}} = \frac{\sum_{d_m \in MD} r_{q_a, d_m} \cdot Occur(v_{ij}, d_m)}{\sum_{d_m \in MD} Occur(v_{ij}, d_m)} \quad (10)$$

$$w_{f_i, q_a} = \frac{n(sim(f_i, q_a))}{\sum_i n(sim(f_i, q_a))}; \quad n(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

$$sim(f_i, q_a) = \frac{\sum_{d_m \in MD} (sim(f_i, d_m) - \overline{sim(f_i)}) \cdot (r_{q_a, d_m} - \overline{r_{q_a}})}{\sigma_{f_i} \cdot \sigma_{q_a}} \quad (12)$$

where w_{v_{ij}, q_a} represents the weight of a value j for a feature i in the query q_a . 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. $\overline{r_{v_{ij}, q_a}}$ represents an average rating value for data items in the set MD having a value j for a feature i . $Occur(v_{ij}, d_m)$ is a binary variable which represents

whether the data item d_m has the value v_{ij} , and if yes, its value is 1, otherwise it is 0. w_{f_i, q_a} represents the weight of a feature f_i for the query q_a . $sim(f_i, q_a)$ represents the correlation weight between the criterion (feature) similarity and the original/predicted user ratings for the query q_a . $sim(f_i, d_m)$ represents the similarity value between the values of the query q_a and the data item d_m in terms of the dimension of the feature f_i .

Table 3 represents the feature vectors of an example query and data items. Table 4 represents a feature weight adjustment based on the example data shown in Table 3. The feature weight adjustment uses the weighted multi-valued query, which can be generated only from the positive examples via using Equations (7) and (8) and increased user feedback information via the query-to-query hybrid filtering. The domain-specific queries and data items in the example have only binary values for each feature, but the equations would also work using the real-number values. Table 4 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 it is 0. From the values given in Table 3, it is desirable that the feature f_i 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 using a multi-dimensional/multi-valued query and heterogeneous types of values in each criterion thereby requiring different metrics for evaluating the values.

The incrementally-specified query seems to degrade the prediction ratio and efficiency of the search process because it aggregates contents of multiple data items. However, it can clearly have a 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 problem can occur if a refined domain-specific query has more values because the number of data sources can be increased. Also, some data sources do not provide multi-valued queries so that the domain-specific query must be translated to a number of data source-specific queries. To address this problem, the translated queries having higher weight values can be posed in advance to a data source with a certain degree of parallel processing, and the partial results can be shown to the users.

Table 3 Example Feature Values

	f_1	f_2	f_3	f_4
d_1	1	0	1	0
d_2	0	0	0	1
d_3	1	1	1	0
d_4	0	1	1	0
q_a	1	0	1	0

Table 4 A Feature Weight Adjustment based on Data shown in Table 3

	d_1	d_2	d_3	d_4	$sim(f_i, q_a)$	w_{f_i, q_a}
$sim(f_1, d_i)$	1	0	1	0	1.00	0.46
$sim(f_2, d_i)$	1	1	0	0	0.00	0.00
$sim(f_3, d_i)$	1	0	1	1	0.58	0.27
$sim(f_4, d_i)$	1	0	1	1	0.58	0.27
r_{q_a, d_i}	1	0	1	0		

6 Validation

First of all, this research provides a case-based reasoning framework for the Knowledge Sifter system and collaborative query refinement based on the framework. To validate the entire framework and the collaborative query refinement, at least hundreds or thousands of users might be required to use the system, and a number of queries and feedback on result data items would also be required from each user as a recommender system using collaborative filtering generally requires a number of user rating data.

Due to the limitations described above, I have focused on evaluating the new hybrid filtering method, called *query-to-query hybrid filtering* — one of main contributions of this research — by using a publically available user-rating dataset for academic research. The Movielens dataset, which has been created by the GroupLens research group in the Department of Computer Science and Engineering at the University of Minnesota, has been chosen for this validation. As the query-to-query hybrid filtering method combines both content-based and collaborative filtering, this validation has focused on the effect of hybrid filtering approach on: 1) the new-user problem and 2) the new-item problem of content-based filtering and collaborative filtering, respectively. The query-to-query hybrid filtering method mitigates the new-user problem because, unlike a simple content-based filtering method, it refines a query based not only on an active user's feedback, but also on the neighbor users' feedback. It also mitigates the new-item

problem because it recommends a data item based not only on user ratings of the item, but also on the similarity of the item to the user preference.

To show the advantages of the query-to-query hybrid filtering approach for the new-user problem, I have compared the performance of the query-to-query hybrid filtering method to a general content-based filtering method based on the nearest neighbor algorithm for the new-user situation. For the new-item problem, I have performed another experiment which compares the performance of a general collaborative filtering algorithm and the query-to-query hybrid filtering for the new-item situation. Finally, two additional experiments have been performed to show the advantages of the query-to-query hybrid filtering method over the other hybrid filtering methods introduced in the literature. The query-to-query hybrid filtering method is assumed to be better than the other hybrid filtering methods since it uses multiple features, collaborative feature-weight distribution, and semantics of the data-item contents.

6.1 Hypothesis

The goal of this experiment is to test the following six hypotheses related to research questions:

Hypothesis 1: *The query-to-query hybrid filtering method performs better than a pure content-based filtering method overall in terms of precision and recall.*

Hypothesis 2: *The query-to-query hybrid filtering method performs better than a pure content-based filtering method for the new-user problem in terms of precision and recall.*

Hypothesis 3: *The query-to-query hybrid filtering method performs better than a pure collaborative filtering method overall in terms of accuracy of predictions.*

Hypothesis 4: *The query-to-query hybrid filtering method performs better than a pure collaborative filtering method for the new-item problem in terms of accuracy of predictions.*

Hypothesis 5: *Semantically-enhanced search by using ontology performs better than the keyword-based search in terms of precision and recall.*

Hypothesis 6: *The multi-feature-based query-to-query hybrid filtering method performs better than other hybrid filtering methods in terms of precision and recall.*

6.2 Experiments

The query-to-query hybrid filtering approach was originally developed to improve the effectiveness of an active user's search based not only on an active user's search history, but also on other users' search history. As Knowledge Sifter is a general-purpose search system, the query-to-query hybrid filtering can be applied to a search system for any data domain or application domain such as Web sites, movies, music, images, etc., as long as the content of the data can be explicitly represented by a set of metadata (features). The MovieLens dataset is selected for the experiments because there is no other data publicly available with the scale of explicit user feedback data.

6.2.1 Data Selection

The Movielens dataset, widely used in academia for validating a recommender system using a collaborative filtering or a hybrid filtering technique, is used to perform this experiment. The Movielens dataset containing *1 million ratings for 3900 movies by 6040 users* has been chosen for this experiment, because it has a sufficient number of data items for this experiment. A relational database has been created to store the Movielens data by using the MySQL database system with a database schema represented in Figures Figure 25 and Figure 26. The movie content data, except the genre data, is parsed from IMDB [5] because the Movielens datasets only have the genre information of movie data items.

As represented in the database schema, a table is created for each feature of the movie data because a movie data item may have multiple values for each feature. The query-to-query hybrid filtering can use the multiple feature information of data items dynamically, i.e., the XML-based multi-dimensional query structure provides the flexibility of creating a query with a set of features found in real time during query refinement. The dynamic query refinement with the flexible query structure is useful for a general-purpose information search system, since the content features of data items are heterogeneous in the data domain and sources. Also, user preference on the content features may differ from user to user, e.g., a user chooses a movie based on genre feature (i.e., the user likes a movie because its genre is one of his or her favorites), and another user might choose a movie based on the director or actor feature. The variability of user

preference with respect to the content features requires that a system should not pre-assume which features are important or unimportant.

```
CREATE TABLE User (  
    userid INTEGER,  
    age INTEGER,  
    gender CHAR(1),  
    occupation INTEGER,  
    zipcode CHAR(10),  
    numOfRatings INTEGER,  
    PRIMARY KEY (userid));  
  
CREATE TABLE Item (  
    itemid INTEGER,  
    title CHAR(100),  
    imdb_id CHAR(9),  
    numOfRatings INTEGER,  
    PRIMARY KEY (itemid));  
  
CREATE TABLE ItemGenre (  
    itemid INTEGER,  
    genre CHAR(11),  
    PRIMARY KEY (itemid, genre),  
    FOREIGN KEY (itemid) REFERENCES Item);  
  
CREATE TABLE ItemDirector (  
    itemid INTEGER,  
    director CHAR(50),  
    PRIMARY KEY (itemid, director),  
    FOREIGN KEY (itemid) REFERENCES Item);  
  
CREATE TABLE ItemActor (  
    itemid INTEGER,  
    actor CHAR(50),  
    PRIMARY KEY (itemid, actor),  
    FOREIGN KEY (itemid) REFERENCES Item);  
  
CREATE TABLE ItemKeyword (  
    itemid INTEGER,  
    keyword CHAR(50),  
    PRIMARY KEY (itemid, keyword),  
    FOREIGN KEY (itemid) REFERENCES Item);  
  
CREATE TABLE ItemCountry (  
    itemid INTEGER,  
    country CHAR(21),  
    PRIMARY KEY (itemid, country),  
    FOREIGN KEY (itemid) REFERENCES Item);
```

Figure 25 A database schema for user and item data with IMDB movie content information

```

CREATE TABLE MLBase (
    userid INTEGER,
    itemid INTEGER,
    rating INTEGER,
    PRIMARY KEY (userid, itemid),
    FOREIGN KEY (userid) REFERENCES User,
    FOREIGN KEY (itemid) REFERENCES Item);

CREATE TABLE MLTest (
    userid INTEGER,
    itemid INTEGER,
    rating INTEGER,
    cf_prediction DOUBLE,
    cbf_sim_all DOUBLE,
    kshf_sim_all DOUBLE,
    kshf_sim_genre DOUBLE,
    kshf_sim_keyword DOUBLE,
    kshf_sim_country DOUBLE,
    kshf_sim_director DOUBLE,
    kshf_sim_actor DOUBLE,
    PRIMARY KEY (userid, itemid),
    FOREIGN KEY (userid) REFERENCES User,
    FOREIGN KEY (itemid) REFERENCES Item);

```

Figure 26 A database schema for a base and test dataset

6.2.2 Test Dataset Selection

As shown in Figure 26, MLBase and MLTest tables are created to store the base dataset and the test dataset, respectively. Initially, the MLBase table has one million rating values with corresponding pairs of ids of user and data item data. The MLTest table has all of the same million rating data copied from MLBase. Additionally, it has ad-hoc attributes to manage prediction values and similarity values which are calculated from the query-to-query hybrid filtering approach, as well as other content-based filtering, or collaborative filtering techniques used for the comparisons.

6.2.2.1 Test Dataset for the overall performance

A rating dataset having users and items both with more than 1000 ratings is created from the MLTest table in order to test the overall performance of the query-to-query hybrid filtering compared to pure content-based filtering and collaborative filtering as follows:

```
CREATE TABLE MLTest_1000 LIKE MLTest;
INSERT INTO MLTest_1000 SELECT * FROM MLTest
WHERE userid IN (SELECT userid FROM MLBase GROUP BY userid HAVING
COUNT(*)>1000) AND itemid IN (SELECT itemid FROM MLBase GROUP BY itemid
HAVING COUNT(*)>1000);
```

6.2.2.2 Test Dataset for the new-user problem

The new-user problem is one of the main problems addressed in this research, since the goal of the query-to-query hybrid filtering method is to improve the effectiveness of information search and personalization by refining user queries via using content patterns and preference patterns found not only from an active user's profile, but also from other users' profiles collaboratively. Most information search systems using content-based filtering have difficulty in mining user preference due to the lack of user feedback. This is the case of the new-user problem which often occurs in information filtering systems. Thus, the test datasets are split into sub-datasets having different sets of users selected per the number of their ratings provided. Initially, the test dataset for this experiment is conditionally-selected based on the number of ratings on data items to remove un-expected side effects regarding the results of the experiments, such as those

arising from the new-item problem. The test data is filtered to obtain data items having at least 2000 ratings as follows:

```
CREATE TABLE MLTest_d2000 LIKE MLTest;  
INSERT INTO MLTest_d2000 SELECT * FROM MLTest  
WHERE itemid IN (SELECT itemid FROM MLBase GROUP BY itemid HAVING  
COUNT(*)>2000);
```

As specified in the above sql statements, the MLTest_d2000 has a test dataset only for data items having more than 2000 ratings on the base dataset. The MLTest_d2000 dataset also has a sufficient number of test data, which is 75,996 pairs of users and items. Finally, the MLTest_d2000 dataset is split into several sub-datasets having a set of users based on the number of their ratings as shown in Table 5. Thus, the sub-datasets are created to show how the query-to-query hybrid filtering works better than a pure content-based filtering mainly when an active user gives a small amount of feedback.

Table 5 Sub-datasets of MLTest_d2000

Name of Sub-Table	Sql Statements of Creating Sub-Tables
MLTest_d2000_u20	CREATE TABLE MLTest_d2000_u20 LIKE MLTest_d2000; INSERT INTO MLTest_d2000_u20 SELECT * FROM MLTest_d2000 WHERE userid IN (SELECT userid FROM MLBase GROUP BY userid HAVING COUNT(*)<=20);
MLTest_d2000_u50	CREATE TABLE MLTest_d2000_u50 LIKE MLTest_d2000; INSERT INTO MLTest_d2000_u50 SELECT * FROM MLTest_d2000 WHERE userid IN (SELECT userid FROM MLBase GROUP BY userid HAVING COUNT(*)>20 AND COUNT(*)<=50);
MLTest_d2000_u100	CREATE TABLE MLTest_d2000_u100 LIKE MLTest_d2000; INSERT INTO MLTest_d2000_u100 SELECT * FROM MLTest_d2000 WHERE userid IN (SELECT userid FROM MLBase GROUP BY userid HAVING COUNT(*)>50 AND COUNT(*)<=100);
MLTest_d2000_u500	CREATE TABLE MLTest_d2000_u500 LIKE MLTest_d2000; INSERT INTO MLTest_d2000_u500 SELECT * FROM MLTest_d2000 WHERE userid IN (SELECT userid FROM MLBase GROUP BY userid HAVING COUNT(*)>100 AND COUNT(*)<=500);
MLTest_d2000_u1000	CREATE TABLE MLTest_d2000_u1000 LIKE MLTest_d2000; INSERT INTO MLTest_d2000_u1000 SELECT * FROM MLTest_d2000 WHERE userid IN (SELECT userid FROM MLBase GROUP BY userid HAVING COUNT(*)>500 AND COUNT(*)<=1000);
MLTest_d2000_u1001	CREATE TABLE MLTest_d2000_u1001 LIKE MLTest_d2000; INSERT INTO MLTest_d2000_u1001 SELECT * FROM MLTest_d2000 WHERE userid IN (SELECT userid FROM MLBase GROUP BY userid HAVING COUNT(*)>1000);

6.2.2.3 Test Dataset for the new-item problem

As opposed to the pre-selection of the test dataset for the new-user problem experiments, only the test data with users providing more than a sufficient number of ratings are selected for this experiment as follows:

```
CREATE TABLE MLTest_u1000 LIKE MLTest;
INSERT INTO MLTest_u1000 SELECT * FROM MLTest
WHERE userid IN (SELECT userid FROM MLBase GROUP BY userid HAVING
COUNT(*)>1000);
```

A partial dataset with users providing more than 1000 ratings instead of users providing more than 2000 ratings are selected as a test dataset since there is only one user providing more than 2000 ratings. There are 40 distinct users providing more than 1000 ratings, and the MLTest_u1000 has 49893 rating data. The following sql statements represent the test dataset split from the MLTest_u1000 dataset with items having different numbers of ratings on them:

Table 6 Sub-datasets of MLTest_u1000

Name of Sub-Table	Sql Statements of Creating Sub-Tables
MLTest_u1000_d20	CREATE TABLE MLTest_u1000_d20 LIKE MLTest_u1000; INSERT INTO MLTest_u1000_d20 SELECT * FROM MLTest_u1000 WHERE itemid IN (SELECT itemid FROM MLBase GROUP BY itemid HAVING COUNT(*)<=20);
MLTest_u1000_d50	CREATE TABLE MLTest_u1000_d50 LIKE MLTest_u1000; INSERT INTO MLTest_u1000_d50 SELECT * FROM MLTest_u1000 WHERE itemid IN (SELECT itemid FROM MLBase GROUP BY itemid HAVING COUNT(*)>20 AND COUNT(*)<=50);
MLTest_u1000_d100	CREATE TABLE MLTest_u1000_d100 LIKE MLTest_u1000; INSERT INTO MLTest_u1000_d100 SELECT * FROM MLTest_u1000 WHERE itemid IN (SELECT itemid FROM MLBase GROUP BY itemid HAVING COUNT(*)>50 AND COUNT(*)<=100);
MLTest_u1000_d500	CREATE TABLE MLTest_u1000_d500 LIKE MLTest_u1000; INSERT INTO MLTest_u1000_d500 SELECT * FROM MLTest_u1000 WHERE itemid IN (SELECT itemid FROM MLBase GROUP BY itemid HAVING COUNT(*)>100 AND COUNT(*)<=500);
MLTest_u1000_d1000	CREATE TABLE MLTest_u1000_d1000 LIKE MLTest_u1000; INSERT INTO MLTest_u1000_d1000 SELECT * FROM MLTest_u1000 WHERE itemid IN (SELECT itemid FROM MLBase GROUP BY itemid HAVING COUNT(*)>500 AND COUNT(*)<=1000);
MLTest_u1000_d1001	CREATE TABLE MLTest_u1000_d1001 LIKE MLTest_u1000; INSERT INTO MLTest_u1000_d1001 SELECT * FROM MLTest_u1000 WHERE itemid IN (SELECT itemid FROM MLBase GROUP BY itemid HAVING COUNT(*)>1000);

6.2.3 Implementation

To use the Movielens datasets for validating the query recommendation framework using query-to-query hybrid filtering, a user profile (a user's all-time preference) is regarded as a user query because the datasets have user ratings on data items only by users, not by user queries. That is, it is assumed that each user has posed only one query representing the user's long-term information need. I will refer to that user query as *user-profile query* from now on. As the goal of the query-to-query hybrid filtering is to refine user queries, an initial user-profile query needs to be generated prior to the refinement. Then, the initial query is used to calculate similarities of data items rated by neighbor queries with the initial query for each dimension. The neighbor queries are, in fact, the neighbor users as it is assumed that a user profile is a user query in this experiment. Thus, the neighbor queries (users) are selected if a user has rating patterns similar to the active user's rating patterns in this experiment. Finally, the initial query is refined based on the content patterns of data items preferred not only by the active user, but also by the neighbor users. All of the methods used in this experiment including the query-to-query hybrid filtering tailored to the Movielens dataset as described as above are implemented by using JAVA, and MySQL database is used to maintain the Movielens data and experiment results.

6.2.3.1 Initial User Profile Query Generation

As described as above, the initial profile query must be generated to use the Movielens dataset for validating the query-to-query hybrid filtering. To generate the initial query, a set of data items preferred by an active user having a rating value greater

than 4 (out of 5) is selected from the set of data items rated by the active user. Then, the entire list of values with a number of their occurrences is found from the preferred-item set for each feature. The weights of each value are determined by using the value list with the number of occurrences as in Equation 13. The weights of each feature of the initial query are evenly distributed with a value $1/n(f)$, where in $n(f)$ is the number of features.

$$w_{v_{ij}} = \frac{\sum_{d \in P} Occur(v_{ij}, d)}{n(d)} \quad (13)$$

where $w_{v_{ij}}$ represents the weight for a value j of a feature i , and P is the set of data items preferred by an active user. $Occur(v_{ij}, d)$ is a binary variable which represents whether the data item d has the value v_{ij} , and if yes, its value is 1, otherwise 0. $n(d)$ is the total number of data items in the preferred-item set.

6.2.3.2 Candidate User-Query Case Selection via Ontology-Based Indices

In the case-based Knowledge Sifter framework, only the user query cases having a query consisting of topics and concepts similar to the active query are pre-selected by using the ontology-based indices. This pre-filtering helps us to reduce the number of user query cases that are employed to calculate similarity of their rating patterns with the active user's rating patterns for selecting the neighbor user query cases. In this experiment, this pre-filtering is ignored because the Movielens dataset has ratings by users, not by user queries. Therefore, a user's entire rating history is added to the

candidate user-query (user) cases if the user commonly rated at least a data item with the active user, for this experiment.

6.2.3.3 Neighbor Query (User) Selection

As I described in the previous sections, for this experiment, the neighbor users are employed instead of the neighbor queries to find the preferred data items. This neighbor user selection is based on the idea of collaborative filtering; that is, the neighbor users will be selected if the user's rating patterns are similar to the active user's rating patterns using the Pearson Correlation Coefficient (PMCC). Two parameter values are employed to find the neighbor users: one is a threshold of the minimum number of data items commonly rated by an active user; the other is a threshold of the PMCC value, which represents how user rating values of the active user and a neighbor user for the commonly-rated data items are related each other. The two thresholds have a significant effect on the performance of the query-to-query hybrid filtering, and a set of ideal thresholds is found based on statistics and the appropriate size of test dataset satisfying the thresholds - 20 and 0.7 which are the thresholds of the number of neighbor users and the PMCC value, respectively. The threshold 20 is not a small number of sample data for calculating a PMCC value, and the PMCC threshold 0.7 also indicates that there is a high correlation between the two datasets. The threshold pair is also tested with the two-tailed paired T-test with $n-2$ degrees of freedom and it turns out to be statistically significant at conservative significance level of 0.01.

The query-to-query hybrid filtering actually pre-filters the other user queries based on the similarity of topics of user interest before the neighbor user selection

process using PMCC is performed. That is, an active user's initial query is primarily compared to the other user's query in terms of the similarity of topics contained in each query as described in Section 4.3.2. The ontology-based pre-filtering dramatically reduces the number of other query histories subjecting for the rating pattern comparison since there are an uncountable number of query histories in a search system (Google receives 400 million search queries per a day in US). However, it is assumed that a user is a candidate neighbor user if the user commonly-rated an item with the active user for these experiments using the Movielens dataset.

6.2.3.4 Refined Query Generation based on Query-to-Query Hybrid Filtering

After the neighbor users have been determined, the neighbor users' rated data items having a prediction value higher than a threshold, 4 (out of 5) for the active user, are found by using Equations 7 and 8. The found data items are added to the active user's preferred item set. Then, a domain-specific query is initially formulated with a set of content values and their weights for each criterion found by using Equations 9 and 10 on the extended preferred item set for the active user. Finally, the initial domain-specific query is refined by determining the weight of each feature using Equations 11 and 12. A feature weight is determined by a correlation value between the value similarity of the initial domain-specific query to a data item in terms of the feature and the actual or predicted rating values for the data item by the active user. Therefore, if the user rating values and the similarity values in terms of a criterion have similar patterns, the weight of the criterion would be higher. Otherwise, the weight would be lower. That is, each feature

weight actually represents that how a data item feature (dimension) is important to the users' selectiveness of the data item.

6.2.4 Experiments and Results

6.2.4.1 Experiment Metrics and Types

As shown in Table 7, I have performed various experiments for the validation. A Spearman's rank correlation coefficient (Spearman's Rho)-based metric was used for all of the experiments because the query-to-query hybrid filtering produces the similarity values of test data items for the refined domain-specific query, not the prediction values of user ratings on the items. That is because the goal of the query-to-query hybrid filtering method is to refine user queries, not to predict actual user rating values for the items. The Spearman's rank correlation coefficient metric measures the extent to which two different rankings agree independent of the actual values of the variables, but it does not handle weak (partial) orderings well [32]. Weak orderings occur in ranking items based on the actual user rating values since there would be many items in a rank due to the small variability of the user rating values (only five different values, 1 to 5 out of 5). On the other hand, the system may return a complete ordering of items because they are ranked by their similarity to a refined query. In this case, the Spearman's rank correlation metric will be penalized for every pair of items that the user has rated the same, but the system ranks at different levels. Therefore, the ranks obtained from the query-to-query hybrid filtering are normalized based on the actual user rating values.

In detail, data items are grouped by their actual user rating values, and the number of items in each group is counted. Data items are ordered by similarity values calculated

from the query-to-query hybrid filtering. The ranks of the data items for the query-to-query hybrid filtering are normalized based on the actual ranks given by the similarity values and the number of items in the groups created by the actual user rating values. For example, if a data item is ranked 40 based on the similarity value, and the total numbers of items in the groups for user rating values 5 and 4 (out of 5) are 20 and 100, respectively, the data item is given a normalized rank, 4 for ranking based on the similarity values. That is, the ranks of data items for the similarity values are re-scaled corresponding to the scale of the user rating values to avoid the penalty which might be given to using the Spearman's rank correlation metric on the weak ordering data. Finally, the correlation coefficient is calculated based on the actual user rating values and the re-scaled rank values to represent the performance of the query-to-query hybrid filtering. Then, the Spearman's rank correlation-based metrics for measuring the performance of the comparing methods are also calculated in a similar way to evenly quantify their performance, including the methods which have a same range of prediction values with the actual user rating values. Furthermore, the two-tailed paired T-test with $n-2$ degrees of freedom was performed to test the hypothesis for the correlation coefficient, which is that the similarity values and the actual user rating values are correlated each other, at the various statistical significance levels such as the 0.10, 0.05, and 0.01 level.

Table 7 Experiment Metrics and Types

Exp#	Algorithm	Comparing Algorithm	Metrics
1	Query-to-Query Hybrid Filtering	Content-based Filtering based on a Nearest Neighbor algorithm	Spearman's Rho, Precision/Recall
2	Query-to-Query Hybrid Filtering	GroupLens Collaborative Filtering	Spearman's Rho, MAE
3	Query-to-Query Hybrid Filtering using Multiple Features	Query-to-Query Hybrid using only One Feature	Spearman's Rho, Precision/Recall
4	Query-to-Query Hybrid Filtering with concept generalization/specialization using ontology	Query-to-Query Hybrid Filtering without concept generalization/specialization	Spearman's Rho, Precision/Recall

6.2.4.2 Comparison between Query-to-Query Hybrid Filtering and Content-Based Filtering

As described in the previous section, a Spearman's rank correlation-based metric was basically used to compare the query-to-query hybrid filtering method and a pure content-based filtering method. Table 8 shows the results of the comparisons between the query-to-query hybrid filtering and a content-based filtering using the subsets of the million rating data. As shown in the results, the query-to-query hybrid filtering outperforms the content-based filtering technique using Euclidean distance overall. Furthermore, the query-to-query hybrid filtering does not seem to have the new-user problem while the content-based filtering performs poorly for users who provided a few ratings. All of the Spearman's rank correlation coefficient (Spearman's rho) values of the subsets obtained for testing the performance of the query-to-query hybrid filtering also turned out to be statistically significant at a conservative significance level of 0.01 from

the two-tailed paired T-test with $n-2$ degrees of freedom, i.e., it cannot be said that the similarity values found by the query-to-query hybrid filtering and the actual user rating values are not correlated with each other.

Table 8 Comparison between Query-to-Query Hybrid Filtering (HF) and Content-Based Filtering (CBF) via the Spearman's rank correlation coefficient

	d2000_ u20	d2000_ u50	d2000_ u100	d2000_ u500	d2000_ u1000	d2000_ u1001	d2000_ Overall
Spearman's Rho_HF	0.638	0.640	0.625	0.606	0.592	0.614	0.609
Spearman's Rho_CBF	0.505	0.523	0.571	0.563	0.543	0.558	0.555
Improvement	26.49%	22.29%	9.46%	7.62%	9.02%	10.16%	9.70%

For this comparison, the precision and recall metrics are also used, which have been widely employed in *Information Retrieval* [11], to measure the performance of movie-item retrieval done by both methods. To calculate the precision and recall metrics for the query-to-query hybrid filtering method, two sorted item-sets are created: one is ordered by actual user rating values and the other is ordered by the similarity values obtained from the query-to-query hybrid filtering. Items with the actual user rating value equal or higher than 4 (out of 5) were regarded as relevant, and other items having the actual user rating value lower than 4 were classified as non-relevant. Then, the number of the relevant items is found and use the number as a bound rank to decide whether a movie item was classified by the query-to-query hybrid filtering as relevant or not from

the ordered item set. Therefore, the precision and recall values can be calculated from Equations 13 and 14, respectively since the test datasets have a fixed number of data items [49].

As shown in Table 9, the results are similar to the results of the comparisons using the Spearman's rank correlation-based metric, i.e., the query-to-query hybrid filtering performs better than a pure content-based filtering both for overall and for the new-user situation in terms of the precision and recall (for both measures, higher is better). Furthermore, the one-tailed paired T-test was performed to test the hypothesis. The null hypothesis, which is that there is no improvement in using query-to-query hybrid filtering, can be rejected in 95% confidence level for all of the test cases since the p-values are less than 0.05. Especially for the new-user situation, the null hypothesis can be rejected in 99% confidence level.

$$precision = \frac{tp}{tp + fp} \quad (13)$$

$$recall = \frac{tp}{tp + fn} \quad (14)$$

where tp (*true positives*) represents a number of relevant movie items also classified as relevant by the query-to-query hybrid filtering. fp (*false positives*) represents a number of non-relevant movie items classified as relevant by the query-to-query hybrid filtering. fn

(*false negatives*) represents a number of relevant movie items classified as non-relevant by the query-to-query hybrid filtering.

Table 9 Comparison between Query-to-Query Hybrid Filtering (HF) and Content-Based Filtering (CBF) via Precision and Recall

	Precision				Recall			
	HF	CBF	Difference	One-tailed T-test	HF	CBF	Difference	One-tailed T-test
d2000_u20	0.908	0.821	10.62%	< 0.005	0.908	0.821	10.62%	< 0.005
d2000_u50	0.905	0.842	7.53%	< 0.01	0.893	0.838	6.57%	0.016
d2000_u100	0.890	0.850	4.68%	0.039	0.882	0.849	3.85%	0.044
d2000_u500	0.883	0.838	5.38%	0.034	0.868	0.837	3.71%	0.043
d2000_u1000	0.879	0.832	5.62%	0.032	0.871	0.830	4.90%	0.039
d2000_u1001	0.879	0.831	5.74%	0.029	0.879	0.829	5.99%	0.027
d2000_overall	0.886	0.842	5.26%	0.031	0.874	0.840	4.07%	0.035

6.2.4.3 Comparison between Query-to-Query Hybrid Filtering and Collaborative Filtering

The Spearman's rank correlation-based metric was also basically used for this comparison between the query-to-query hybrid filtering and a pure collaborative filtering technique used in GroupLens [46, 60]. Table 10 shows the results of the comparisons between the query-to-query hybrid filtering and the pure collaborative filtering using the subsets of the million rating data. Unlike the previous results, the query-to-query hybrid

filtering does not outperform the pure collaborative filtering overall. Also, the pure collaborative filtering does not seem to have the new-item problem except for a test dataset selected from the million rating dataset, u1000_d10, which has items with a very small number (five) of user ratings. Such cases cannot be tested for the 100k rating dataset since there is no such test datasets having items with user ratings smaller than five. Nevertheless, it can be said that the query-to-query hybrid filtering is reasonably acceptable for the new-item situations from the results of the test datasets, u300_d10 and u1000_d10 selected from the 100k rating dataset and the million rating dataset, respectively. The results are similar or slightly better than the results of the pure collaborative filtering for such test datasets while its performance is not better overall. Furthermore, all of the Spearman's rank correlation coefficient (Spearman's rho) values of these test datasets representing the performance of the query-to-query hybrid filtering also turned out to be statistically significant at conservative significance level of 0.01 from the two-tailed paired T-test with n-2 degrees of freedom.

Table 10 Comparison between Query-to-Query Hybrid Filtering (HF) and Collaborative Filtering (CF) via the Spearman's rank correlation coefficient

	u1000_ d10	u1000_ d20	u1000_ d50	u1000_ d100	u1000_ d500	u1000_ d1000	u1000_ d1001	u1000_ overall
Spearman's Rho_HF	0.508	0.581	0.575	0.585	0.576	0.576	0.599	0.584
Spearman's Rho_CF	0.425	0.636	0.655	0.678	0.677	0.637	0.630	0.680
Difference	19.44%	-8.63%	-12.15%	-13.72%	-14.93%	-9.59%	-4.90%	-14.06%

In this experiment, the effectiveness of the query-to-query hybrid filtering method is tested by comparing it with the well-known pure collaborative filtering. As shown in the results, the pure collaborative filtering performs better than the query-to-query hybrid filtering. This result occurred because the performance was tested by comparing the predicted values to the actual user ratings. In addition, the goal of the query-to-query hybrid filtering is not to predict the user rating values for the items, but rather to refine user queries, while the pure collaborative filtering aims to predict the user rating values. Therefore, another experiment has been conducted to test the performance of the query-to-query hybrid filtering.

Initially, I found a way to use the query-to-query hybrid filtering for predicting the user rating values; that is, to complement the predictions of the pure collaborative filtering with the query-to-query hybrid filtering using Equations 14 and 15. The basic idea of this complement is to give more weights to a prediction of an active user's rating value for a data item if the data item is found to be similar to an active query refined by the query-to-query hybrid filtering as shown in Equation 14. The prediction value of a data item is also penalized if the data item is not similar to the refined query. Equation 15 denotes that the new prediction values are bounded with a scale of user rating values used in the Movielens dataset, 1 to 5 out of 5. The parameter α representing a weight of the hybrid filtering in the combination can be determined by heuristics, and 4 was found as an ideal value for α that maximize the effectiveness of the combining in predicting user rating values on data items. Table 11 shows the results of the comparisons between the complemented collaborative filtering by the query-to-query hybrid filtering and the pure

collaborative filtering using the subsets of the million rating data. The comparisons are done with Mean Absolute Error (MAE) since the prediction values produced from both methods have a same scale with the actual user rating values. As shown in the results, the complemented one is slightly better in performance (lower is better for MAE metrics).

$$p'_{u_a, d_u} = \begin{cases} p_{u_a, d_u} + \alpha(sim(q_a, d_u) - 0.5)^2 & \text{if } sim(q_a, d_u) \geq 0.5 \\ p_{u_a, d_u} - \alpha(sim(q_a, d_u) - 0.5)^2 & \text{if } sim(q_a, d_u) < 0.5 \end{cases} \quad (14)$$

$$p'_{u_a, d_u} = \begin{cases} 5.0 & \text{if } p'_{u_a, d_u} > 5.0 \\ p'_{u_a, d_u} & \text{if } 1.0 \leq p'_{u_a, d_u} \leq 5.0 \\ 1.0 & \text{if } p'_{u_a, d_u} < 1.0 \end{cases} \quad (15)$$

where p_{u_a, d_u} represents a prediction of rating value of an unseen data item, d_u for an active user, u_a calculated by the pure collaborative filtering. $sim(q_a, d_u)$ represents a normalized similarity value of the unseen data item for an active query, q_a refined by the query-to-query hybrid filtering. α is a parameter value representing a weight of the query-to-query hybrid filtering. p'_{u_a, d_u} represents a complemented prediction of an active user's rating value for an unseen data item, d_u .

Table 11 Comparison between Query-to-Query Hybrid Filtering (HF) and Collaborative Filtering (CF) via MAE

	u1000_ d10	u1000_ d20	u1000_ d50	u1000_ d100	u1000_ d500	u1000_ d1000	u1000_ d1001	u1000_ overall
HF	0.748	0.664	0.685	0.690	0.698	0.689	0.690	0.682
CF	0.832	0.672	0.677	0.696	0.693	0.702	0.718	0.699
Difference	-10.14%	-1.25%	1.20%	-0.83%	0.69%	-1.79%	-3.94%	-2.43%
One-tailed T-test	< 0.01	0.35	0.56	0.38	0.52	0.29	0.041	0.18

6.2.4.4 The Effectiveness of Collaborative Weight Determination of Query-to-Query Hybrid Filtering

Table 12 shows the result of an experiment testing the validity and effectiveness of the query-to-query hybrid filtering in terms of the weight determination for a multi-dimensional query. The weight distribution is a key problem of information retrieval and Web search area as such systems suffer from the ordering of the immense amount of result data found by a query. Also, the weight distribution is a key issue of the data mining and machine learning area since the weights of dimensions actually represent which dimensions of a data item impacted, and by how much, on the users' selectiveness of the data item.

As shown in Table 12, the country and genre features have quite low correlation coefficient values which show that the features would not have any influence on determining users' preference for movie data items. The director and actor features have relatively higher correlation values with the users' actual ratings. This would mean that user queries may be answered more meaningfully, if they select movies based on their favorite directors or actors, rather than selecting movies based on genres or countries.

However, it would be inaccurate for the country feature because the Movielens dataset has very skewed data in terms of the country feature (e.g., about 65 percent of movies in the Movielens datasets are from only one specific country, USA).

Even the country and genre features have such low correlation coefficient values, the similarity values obtained by using all features, e.g., `sim_all` have similar or higher correlation coefficient values compared to the coefficient values based on any of the features. Table 12 also shows the precision and recall values for this comparison. The results are quite similar to the results of the comparison using the correlation coefficient measure; the query-to-query hybrid filtering slightly degrades precision when it uses all the features instead of using only the director feature or the actor feature, but recall is noticeably improved when it uses all the features instead of using any of only one feature. Based on the results, it can be concluded that the weight distribution of the query-to-query hybrid filtering works quite well. If a dimension reduction algorithm is used, it would have better performance. However, the dimension reduction algorithm only works for an information filtering system or an information retrieval system oriented toward only one domain of data and using a fixed feature list. Therefore, the weight distribution mechanism surely fits better in a dynamic environment in which an information retrieval system like Knowledge Sifter would deal in a dynamic way with different domains and different sets of information content features.

Table 12 Performance of Query-to-Query Hybrid Filtering Using Multi-Features (sim_all) or Using Only One Feature

	sim_all	sim_genre	sim_keyword	sim_country	sim_director	sim_actor
Spearman's Rho	0.576	0.02	0.421	0.114	0.557	0.532
Precision	0.849	0.727	0.817	0.749	0.881	0.872
Recall	0.848	0.526	0.779	0.116	0.667	0.76

6.2.4.5 The Effectiveness of Using Ontology-based Concept Generalization/Specialization in Query-to-Query Hybrid Filtering

This experiment validates the effectiveness of using semantics in the query-to-query hybrid filtering. That is, to use ontologies for calculating similarities of data items to user profile queries. The linguistic ontology, WordNet, is used to extend the concepts in the user profile queries for the keyword feature, which has values which having general concepts. A cluster-based simple ontology for country feature was also created. The clusters of countries are created based on social and cultural similarities [27] as shown in Table 13.

Table 13 Clusters of Countries based on Social/Cultural Similarities

Clusters	Countries
Anglo Cultures	"Australia," "Canada," "Ireland," "New Zealand," "South Africa," "UK," "USA."
Confucian Asia	"China," "Hong Kong," "Japan," "South Korea," "Taiwan."
Eastern Europe	"Algeria," "Georgia," "Greece," "Hungary," "Kazakhstan," "Poland," "Russia," "Slovenia."
Germanic Europe	"Austria," "Germany," "Netherlands," "West Germany," "Switzerland."
Latin America	"Argentina," "Brazil," "Mexico."
Latin Europe	"France," "Israel," "Italy," "Portugal," "Spain."
Nordic Europe	"Denmark," "Finland," "Sweden."
Southern Asia	"India," "Iran," "Philippines."

Table 14 shows the precision and recall values of the query-to-query hybrid filtering both with and without using semantics by employing the MLTest_1000 dataset. As shown in the features, the precision values are a bit improved when the ontologies for keyword and country features are used, and the recall values are noticeably improved especially for the country feature. This would be because the social/cultural similarity-based country clusters somewhat diminish the problem of the skewness of the test dataset for the country feature, which is caused by the USA dominance of producing the movies. A simple word matching employing synonyms and hypernyms found from WordNet was used for the keyword ontology. The improvement might be higher if a more robust domain ontology is used for the IMDB keyword tags.

Table 14 Performance of Query-to-Query Hybrid Filtering (HF) With/Without Semantics

	Precision			Recall		
	Using only Keywords	Using only Country	Using all features	Using only Keywords	Using only Country	Using all features
HF w/ Semantics	0.864	0.805	0.866	0.868	0.780	0.898
HF w/o Semantics	0.817	0.749	0.849	0.779	0.116	0.848
Difference	5.71%	7.48%	1.90%	11.43%	572%	5.81%
One-tailed T-Test	0.029	0.021	0.13	< 0.01	< 0.005	0.028

6.3 Conclusion of Validation

As the goal of the query-to-query hybrid filtering is to improve the effectiveness of information search via collaborative query refinement, the comparison between the query-to-query hybrid filtering and the pure content-based filtering using the nearest neighbor algorithm is the primary experiment for the validation. As shown in the experimental results, the query-to-query hybrid filtering method performed better than a pure content-based filtering in terms of accuracy, both overall and for the new-user problem. In detail, the query-to-query hybrid filtering improved precision and recall by 5.3% and 4.1% overall, respectively. For the new-user situation, the query-to-query hybrid filtering improved precision and recall by 10.6% for both.

On the other hand, the query-to-query hybrid filtering did not perform better than the GroupLens collaborative filtering overall, but the query-to-query hybrid filtering performed better in the new-item situation. It would be because the other characteristics of movies such as quality and scenario perfectness would be more important factors for deciding the users' likeness rather than the general movie content values such as genre and actor. For example, a user may like a particular drama movie, but does not necessarily like all drama movies. In addition, a result is presented, which shows the GroupLens collaborative filtering, when complemented by the query-to-query hybrid filtering improves predictability of user ratings overall. In this case, the complemented approach performed slightly better than the GroupLens collaborative filtering overall and improved the performance by 8 to 11% for the new item problem.

Lastly, the experiments on the Movielens datasets has shown the validity and effectiveness of the query-to-query hybrid filtering algorithm for the weight determination of a multi-dimensional query in terms of information search accuracy. The result of an experiment for evaluating the effect of using an ontology for the query-to-query hybrid filtering is also presented. The result suggests that using the semantic technologies for hybrid filtering improves the accuracy of refining user queries in representing user information needs, as the accuracy of the movie retrieval increased by using the refined queries.

7 Conclusion

This research is highly interdisciplinary since building an intelligent and personalizable information search system over heterogeneous data sources requires knowledge of many areas such as information retrieval, information systems, data mining, software engineering, artificial intelligence, statistics, etc. This research is also highly practical in designing and developing both a system and a methodology for searching information over heterogeneous sources while considering user preference and semantics of user queries. Most importantly, a new hybrid filtering method, which we call “*query-to-query hybrid filtering*,” is introduced for automatic refinement of user queries based on opinions of a community of users who have similar preferences.

7.1 Contributions

First, I have developed a new hybrid filtering method which combines content-based filtering and collaborative filtering to take advantage of the entire user search histories including user feedback for improving the effectiveness of information search in a collaborative manner. The hybrid filtering is called “*query-to-query hybrid filtering*,” and it refines a user query based on 1) emergent semantics via use cases 2) mined preferences based not only on an active user’s search history, but also on neighbor users’ search history who have similar preferences. Query-to-query hybrid filtering is developed

by taking into account the ternary relationships among users, their queries, and data items; other filtering systems use only the binary relationship between users and data items. An algorithm and methodology for query-to-query hybrid filtering are presented, together with the mathematical equations to determine the weights and statistics. The validity and effectiveness of query-to-query hybrid filtering, in providing more relevant and preferable information to users, have been validated with numerous experiments and comparisons.

A case-based reasoning framework has been proposed for capturing, maintaining, and reusing the artifacts produced during the users' entire session. An XML-based meta-model is used to store all the artifacts produced during the search process including user query and its refined queries, user feedback, related ontology, data sources, etc. In the case-based reasoning framework, user query cases are created according to an XML-schema which is used to represent user queries with the semantics of user concepts together with their related explicit ontology concepts. An ontology-based indexing scheme is also introduced to effectively and efficiently retrieve other user query cases related to concepts and topics of an active user query case.

Semantic Web standards and technologies such as XML, RDF, OWL, and Semantic Web Services are used to implement semantic search. A prototype application, Knowledge Sifter, has been developed based on these open-standards and a service-oriented architecture. The agent architecture implemented by using Semantic Web Services and the automatic refinement of user queries via the query-to-query hybrid filtering enable a system to be dynamically configured based on user preferences.

Finally, most of the information filtering and information retrieval systems having the ability to mine user preferences or profiles use a *static structure* (e.g., a pre-specified and fixed list of criteria/features) for representing content patterns of data items found preferable by a user. This is due to the difficulty of dealing with changes in such structure itself for the mining. The framework and methodologies presented in this research are developed to refine user queries in terms of modifying not only the feature values, but also the features of the user-preferable content patterns with emergent semantics found from query-to-query hybrid filtering. The following are the key aspects of the framework and methods that enable a system to handle the changes also on the query structure during the refinement:

- *The XML-based user-query case representation with a flexible query structure enables a system to refine user queries with the structure changes;*
- *Emergent semantics via using query-to-query hybrid filtering can be formed not only with emergent feature (metadata) values, but also with emergent features; and*
- *The dynamic feature-weight distribution, developed as a part of query-to-query hybrid filtering, works well with the flexible query structure.*

7.2 Further Research

In this research, a domain-specific query structure is introduced in order to represent the general user concepts specified in the conceptual query in terms of domain-specific concepts. The domain-specific queries are semi-automatically formulated and refined based on simple ontology-concept linkages found from the neighbor user query

cases. However, the ontology-concept linkages are created to represent only synonym and hypernym relationships for finding domain-specific concepts related to the general user concepts or vice versa. Therefore, if the case-based reasoning framework and the query-to-query hybrid filtering method were to incorporate more complex reasoning techniques, such as the rule-based inference providing by Jena [16] and Pellet [64], and take advantage of other ontological relationships among the domain concepts defined in domain ontologies were used in the query refinement process, than more accurate and serendipitous emergent semantics could be found. As a result, the effectiveness of information search in a system would be improved.

Appendix A: XML Schemas for Data Specification

A.1 XML Schema of User Query: Managed by Query Formulation Agent

```
<xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="userTerm">
    <xs:sequence>
      <xs:attribute name="domain" type="xs:integer" use="required"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="userQuerySpec">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SO" type="userTerm" minOccurs="1" maxOccurs="1"/>
        <xs:element name="AoI" type="userTerm" minOccurs="0" maxOccurs="1"/>
        <xs:element name="PoI" type="userTerm" minOccurs="0" maxOccurs="1"/>
        <xs:element name="LoI" type="userTerm" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A.2 XML Schema of WordNet Concept: Managed by Ontology Agent

```
<xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="sense">
    <xs:sequence>
      <xs:element name="senseIndex" type="xs:integer" maxOccurs="1"/>
      <xs:element name="synonyms" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="dataPrefSpec">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SOSyn" type="sense" minOccurs="1" maxOccurs="1"/>
        <xs:element name="AoISyn" type="sense" minOccurs="0" maxOccurs="1"/>
        <xs:element name="PoISyn" type="sense" minOccurs="0" maxOccurs="1"/>
        <xs:element name="LoISyn" type="sense" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A.3 XML Schema of Source-Specific Query for GNIS: Managed by Ontology Agent

```
<xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="GNISQuerySpec">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="featureName" type="xs:string" minOccurs="1" maxOccurs="1"/>
        <xs:element name="state" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="variant" type="xs:boolean" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A.4 XML Schema of Data Preference: Managed by Preference Agent

```
<xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name='weightedElement'>
    <xs:sequence>
      <xs:attribute name='weight' type='xs:double' use='required'/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name='dataPrefSpec'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='term' type='weightedElement' maxOccurs='1'/>
        <xs:element name='location' type='weightedElement' maxOccurs='1'/>
        <xs:element name='date' type='weightedElement' maxOccurs='1'>
          <xs:complexType>
            <xs:sequence>
              <xs:element name='dateFrom' type='xs:date' maxOccurs='1'/>
              <xs:element name='dateTo' type='xs:date' maxOccurs='1'/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name='size' type='weightedElement' maxOccurs='1'>
          <xs:complexType>
            <xs:sequence>
              <xs:element name='width' type='xs:integer' maxOccurs='1'/>
              <xs:element name='height' type='xs:integer' maxOccurs='1'/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name='theme' type='weightedElement' maxOccurs='1'>
          <xs:complexType>
            <xs:sequence>
              <xs:element name='photoW' type='xs:double' maxOccurs='1'/>
              <xs:element name='topoW' type='xs:double' maxOccurs='1'/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



```

        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name='source' type='weightedElement' maxOccurs='1'>
      <xs:complexType>
        <xs:sequence>
          <xs:element name='terraW' type='xs:double' maxOccurs='1'>
            <xs:element name='yahooW' type='xs:double' maxOccurs='1'>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

A.5 XML Schema of Search Result Retrieved from Various Sources: Managed by Web Services Agent

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="specType">
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:element name="Search">
    <xs:complexType>
      <xs:attribute name="source" type="xs:string" use="required"/>
      <xs:element name="Request" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="status" type="xs:string" use="required"/>
          <xs:attribute name="uri" type="xs:anyUri" use="required"/>
          <xs:attribute name="posedTime" type="xs:dateTime" use="required"/>
          <xs:attribute name="endedTime" type="xs:dateTime" use="required"/>
          <xs:sequence>
            <xs:element name="Spec" type="specType" maxOccurs="unbounded"/>
            <xs:element name="Result" maxOccurs="unbounded">
              <xs:complexType>
                <xs:attribute name="uri" type="xs:anyUri" use="required"/>
                <xs:element name="Spec" type="specType" maxOccurs="unbounded"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

A.6 XML Schema of Source-Specific Query for Yahoo Image Search Engine: Managed by Web Services Agent

```
<xs:element name="Spec" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="query" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="format" type="xs:string" maxOccurs="1"/>
      <xs:element name="coloration" type="xs:string" maxOccurs="1"/>
      <xs:element name="site" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

A.7 XML Schema of Result Data from Yahoo Image Search Engine: Managed by Web Services Agent

```
<xs:element name="Spec" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="desc" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="thumbnailUri" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="width" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

A.8 XML Schema of Source-Specific Query for TerraServer: Managed by Web Services Agent

```
<xs:element name="Spec" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="theme" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="centerLong" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="centerLat" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="scale" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="projection" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="width" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="height" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

A.9 XML Schema of Result Data from TerraServer: Managed by Web Services Agent

```
<xs:element name="Spec" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="thumbnailUri" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="centerLong" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="centerLat" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="eastLong" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="westLong" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="northLat" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="southLat" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="creationTime" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="scale" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="projection" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="width" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="height" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Appendix B: MySql Script File for KS Meta Schema

```
CREATE TABLE KSUser (  
    userID VARCHAR(50),  
    password CHAR(40) NOT NULL,  
    firstName VARCHAR(50),  
    lastName VARCHAR(50),  
    industry VARCHAR(99),  
    secQ VARCHAR(99),  
    secA VARCHAR(99),  
    currentDPID INT REFERENCES DataPreference(dataPrefID),  
    currentSPID INT REFERENCES SourcePreference(sourcePrefID),  
    PRIMARY KEY (userID)  
);  
  
CREATE TABLE Source (  
    sourceID VARCHAR(50),  
    provenance VARCHAR(100),  
    PRIMARY KEY (sourceID)  
);  
  
CREATE TABLE SourcePreference (  
    sourcePreferenceID INT NOT NULL AUTO_INCREMENT,  
    isUserSpecified BOOLEAN NOT NULL,  
    specXML VARCHAR(9999),  
    userID VARCHAR(50),  
    sourceID VARCHAR(50),  
    PRIMARY KEY (sourcePreferenceID),  
    FOREIGN KEY (userID) REFERENCES KSUser(userID),  
    FOREIGN KEY (sourceID) REFERENCES Source(sourceID),  
    UNIQUE (userID, sourceID, isUserSpecified)  
);  
  
CREATE TABLE DataPreference (  
    dataPreferenceID INT NOT NULL AUTO_INCREMENT,  
    isUserSpecified BOOLEAN NOT NULL,  
    specXML VARCHAR(9999),  
    userID VARCHAR(50),  
    PRIMARY KEY (dataPreferenceID),  
    FOREIGN KEY (userID) REFERENCES KSUser(userID),  
    UNIQUE (userID, isUserSpecified)  
);  
  
CREATE TABLE UserQuery (  
    userQueryID INT NOT NULL AUTO_INCREMENT,
```

```

        queryText VARCHAR(100),
        specXML VARCHAR(9999),
        posedTime TIMESTAMP NOT NULL,
        numOfResults INT UNSIGNED,
        userComments VARCHAR(300),
        userID VARCHAR(50),
        PRIMARY KEY (userQueryID),
        FOREIGN KEY (userID) REFERENCES KSUser(userID),
        UNIQUE (queryText, posedTime, userID)
    );

CREATE TABLE UserQueryConcept (
    userQueryConceptID INT NOT NULL AUTO_INCREMENT,
    specXML VARCHAR(9999),
    userQueryID INT,
    sourceID VARCHAR(50),
    PRIMARY KEY (userQueryConceptID),
    FOREIGN KEY (userQueryID) REFERENCE UserQuery(userQueryID),
    FOREIGN KEY (sourceID) REFERENCES Source(sourceID),
    UNIQUE (userQueryID, sourceID)
);

CREATE TABLE RefinedQuery (
    refinedQueryID INT NOT NULL AUTO_INCREMENT,
    specXML VARCHAR(9999),
    posedTime TIMESTAMP NOT NULL,
    endedTime TIMESTAMP NULL DEFAULT NULL,
    numOfResults INT UNSIGNED,
    userQueryID INT,
    sourceID VARCHAR(50),
    PRIMARY KEY (refinedQueryID),
    FOREIGN KEY (userQueryID) REFERENCES UserQuery(userQueryID),
    FOREIGN KEY (sourceID) REFERENCES Source(sourceID)
);

CREATE TABLE DataItem (
    dataItemID INT NOT NULL AUTO_INCREMENT,
    uri VARCHAR(999),
    type VARCHAR(50),
    specXML VARCHAR(9999),
    creationTime TIMESTAMP NULL DEFAULT NULL,
    lastUpdatedTime TIMESTAMP NULL DEFAULT NULL,
    sourceID VARCHAR(50),
    PRIMARY KEY (dataItemID),
    FOREIGN KEY (sourceID) REFERENCES Source(sourceID)
);

CREATE TABLE UserQueryResult (

```

```

        userQueryResultID INT NOT NULL AUTO_INCREMENT,
        userSimilarity DOUBLE,
        systemSimilarity DOUBLE,
        rank INT UNSIGNED,
        userComments VARCHAR(300),
        userQueryID INT,
        dataItemID INT,
        dataPrefID INT,
        sourcePrefID INT,
        PRIMARY KEY (userQueryResultID),
        FOREIGN KEY (userQueryID) REFERENCES UserQuery(userQueryID),
        FOREIGN KEY (dataItemID) REFERENCES DataItem(dataItemID),
        FOREIGN KEY (dataPrefID) REFERENCES DataPreference(dataPrefID),
        FOREIGN KEY (sourcePrefID) REFERENCES SourcePreference(sourcePrefID),
        UNIQUE (userQueryID, dataItemID)
    );

CREATE TABLE RefinedQueryResult (
    refinedQueryResultID INT NOT NULL AUTO_INCREMENT,
    systemSimilarity DOUBLE,
    rank INT UNSIGNED,
    refinedQueryID INT,
    dataItemID INT,
    PRIMARY KEY (refinedQueryResultID),
    FOREIGN KEY (refinedQueryID) REFERENCES RefinedQuery(refinedQueryID),
    FOREIGN KEY (dataItemID) REFERENCES DataItem(dataItemID),
    UNIQUE (refinedQueryID, dataItemID)
);

CREATE TABLE AccessProtocol (
    accessProtocolID VARCHAR(50),
    type VARCHAR(50),
    spec VARCHAR(1000),
    creationTime TIMESTAMP NOT NULL,
    lastUpdatedTime TIMESTAMP NULL DEFAULT NULL,
    sourceID VARCHAR(50),
    PRIMARY KEY (accessProtocolID),
    FOREIGN KEY (sourceID) REFERENCES Source(sourceID)
);

CREATE TABLE QoSSLAs (
    qosSlaID VARCHAR(50),
    availability DOUBLE,
    minThroughput DOUBLE,
    responseTime DOUBLE,
    authority VARCHAR(50),
    sourceID VARCHAR(50),
    PRIMARY KEY (qosSlaID),
    FOREIGN KEY (sourceID) REFERENCES Source(sourceID)
);

```

);

```
CREATE TABLE Coverage (  
    coverageID VARCHAR(50),  
    domain VARCHAR(300),  
    class VARCHAR(300),  
    refSource VARCHAR(50),  
    sourceID VARCHAR(50),  
    PRIMARY KEY (coverageID),  
    FOREIGN KEY (sourceID) REFERENCES Source(sourceID),  
    FOREIGN KEY (refSource) REFERENCES Source(sourceID)  
);
```

```
CREATE TABLE Event (  
    eventID VARCHAR(50),  
    type VARCHAR(50),  
    spec VARCHAR(300),  
    occurredTime TIMESTAMP NOT NULL,  
    sourceID VARCHAR(50),  
    PRIMARY KEY (eventID),  
    FOREIGN KEY (sourceID) REFERENCES Source(sourceID)  
);
```

```
CREATE TABLE Certificate (  
    certificateID VARCHAR(50),  
    type VARCHAR(50),  
    authority VARCHAR(1000),  
    expirationDate DATE,  
    rating VARCHAR(50),  
    sourceID VARCHAR(50),  
    PRIMARY KEY (certificateID),  
    FOREIGN KEY (sourceID) REFERENCES Source(sourceID)  
);
```

```
INSERT INTO Source (sourceID) VALUES ('WORDNET2.1');  
INSERT INTO Source (sourceID) VALUES ('GNIS');  
INSERT INTO Source (sourceID) VALUES ('YAHOO');  
INSERT INTO Source (sourceID) VALUES ('TERRA');
```

Appendix C: XML Schemas for Case-Based KS Framework

C.1 XML Schema of User Query Case

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Case">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CaseID" type="xs:string"/>
        <xs:element name="UserName" type="xs:string"/>
        <xs:element name="ConceptualQuery">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Concept" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="UserTerm" type="xs:string"/>
                    <xs:element name="WordNetSenseID" type="xs:string" minOccurs="0"
                      maxOccurs="unbounded"/>
                    <xs:element name="ConceptWeight" type="xs:double"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="DomainSpecificQuery">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Info_Domain" type="xs:string"/>
              <xs:element name="Feature" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="FeatureName" type="xs:string"/>
                    <xs:element name="FeatureValue" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="value" type="xs:string"/>
                          <xs:element name="valueWeight" type="xs:double"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="featureWeight" type="xs:double"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



```

        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

C.2 XML Schema of Ontology Index

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="OntologyIndices">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DomainConceptID" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="WordNetSenseID" type="xs:string" minOccurs="0"
                maxOccurs="unbounded"/>
              <xs:element name="CaseID" type="xs:double" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

REFERENCES

REFERENCES

- [1] “Extensible Markup Language (XML),” <http://www.w3.org/XML/>.
- [2] “USGS Geographic Names and Information System (GNIS),” <http://geonames.usgs.gov/>.
- [3] “NGA, The GEOnet Names Server (GNS),” <http://earth-info.nga.mil/gns/html/index.html>.
- [4] “KML - Google Code,” <http://code.google.com/apis/kml/>.
- [5] “The Internet Movie Database (IMDb),” <http://www.imdb.com/>.
- [6] “DCMI Metadata Terms,” 2008, <http://dublincore.org/documents/dcmi-terms/>.
- [7] A. Aamodt and E. Plaza, “Case-based reasoning: foundational issues, methodological variations, and system approaches,” *AI Commun.*, vol. 7, no. 1, pp. 39-59, 1994.
- [8] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 6, pp. 734-749, 2005.
- [9] C. Amaral, D. Laurent, A. Martins, A. Mendes, C. Pinto, and P. Informática, “Design and Implementation of a Semantic Search Engine for Portuguese,” in *In Proceedings of 4th International Conference on Language Resources and Evaluation (LREC 2004*, pp. 26–28, 2004.
- [10] G. Amato and U. Straccia, “User Profile Modeling and Applications to Digital Libraries,” in *Research and Advanced Technology for Digital Libraries*, vol. 1696, S. Abiteboul and A. Vercoustre, Eds. Springer Berlin / Heidelberg, 1999, pp. 670-670.
- [11] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press, New York, 1999.

- [12] M. Balabanović and Y. Shoham, "Fab: content-based, collaborative recommendation," *Commun. ACM*, vol. 40, no. 3, pp. 66-72, 1997.
- [13] S. Berkovsky, T. Kuflik, and F. Ricci, "Mediation of user models for enhanced personalization in recommender systems," *User Modeling and User-Adapted Interaction*, vol. 18, pp. 245-286, 2008.
- [14] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities," *Scientific American*, vol. 284, no. 5, pp. 34-43, 2001.
- [15] A. Burton-Jones, V. C. Storey, V. Sugumaran, and S. Purao, "A Heuristic-Based Methodology for Semantic Augmentation of User Queries on the Web," in *Lecture Notes in Computer Science*, vol. 2813, Springer Berlin / Heidelberg, 2003, pp. 476-489.
- [16] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pp. 74-83, 2004.
- [17] R. Chinnici, "Web services description language (WSDL) version 2.0 part 2: Adjuncts," *W3C Candidate Recommendation*, <http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060327>, 2006.
- [18] S. Deerwester, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391-407, 1990.
- [19] L. Ding et al., "Swoogle: a search and metadata engine for the semantic web," in *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pp. 652-659, 2004.
- [20] Farrell, Joel and Lausen, Holger, "Semantic Annotations for WSDL and XML Schema," 2007, <http://www.w3.org/TR/sawSDL/>.
- [21] D. Fensel, "Ontology-based knowledge management," *Computer*, vol. 35, no. 11, pp. 56-59, 2002.
- [22] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," UNIVERSITY OF CALIFORNIA, 2000.

- [23] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an agent communication language," in *Proceedings of the third international conference on Information and knowledge management*, pp. 456-463, 1994.
- [24] N. Good et al., "Combining Collaborative Filtering with Personal Agents for Better Recommendations," in *PROCEEDINGS OF THE SIXTEENTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pp. 439--446, 1999.
- [25] T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," in *FORMAL ONTOLOGY IN CONCEPTUAL ANALYSIS AND KNOWLEDGE REPRESENTATION*, Deventer, The Netherlands: Kluwer Academic Publishers, 1993.
- [26] R. Guha, R. McCool, and E. Miller, "Semantic search," in *Proceedings of the 12th international conference on World Wide Web*, pp. 700-709, 2003.
- [27] V. Gupta, P. J. Hanges, and P. Dorfman, "Cultural clusters: methodology and findings," *Journal of World Business*, vol. 37, no. 1, pp. 11-15, 2002.
- [28] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.
- [29] J. Heflin and J. Hendler, "Searching the Web with SHOE," presented at the In AAAI-2000 Workshop on Artificial Intelligence for Web Search, Menlo Park, CA, pp. 35-40, 2000.
- [30] M. Hepp, "Semantic Web and semantic Web services: father and son or indivisible twins?," *Internet Computing, IEEE*, vol. 10, no. 2, pp. 85-88, 2006.
- [31] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 230-237, 1999.
- [32] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5-53, 2004.
- [33] T. Hofmann, "Probabilistic latent semantic indexing," in *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 50--57, 1999.
- [34] M. Huhns, "Agents as Web services," *Internet Computing, IEEE*, vol. 6, no. 4, pp. 93-95, 2002.

- [35] L. Kerschberg, "The role of intelligent software agents in advanced information systems," in *Advances in Databases*, 1997, pp. 1-22.
- [36] L. Kerschberg, "Knowledge Management in Heterogeneous Data Warehouse Environments," in *Data Warehousing and Knowledge Discovery*, 2001, pp. 1-10.
- [37] L. Kerschberg et al., "Knowledge Sifter: Agent-Based Ontology-Driven Search over Heterogeneous Databases Using Semantic Web Services," in *Semantics of a Networked World*, 2004, pp. 278-295.
- [38] L. Kerschberg et al., "Knowledge Sifter: ontology-driven search over heterogeneous databases," in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pp. 431-432, 2004.
- [39] L. Kerschberg, H. Jeong, and W. Kim, "Emergent Semantics in Knowledge Sifter: An Evolutionary Search Agent Based on Semantic Web Services," in *Journal on Data Semantics VI*, 2006, pp. 187-209.
- [40] L. Kerschberg, H. Jeong, Y. Song, and W. Kim, "A Case-Based Framework for Collaborative Semantic Search in Knowledge Sifter," in *Case-Based Reasoning Research and Development*, 2007, pp. 16-30.
- [41] L. Kerschberg, W. Kim, and A. Scime, "A semantic taxonomy-based personalizable meta-search agent," in *Web Information Systems Engineering, 2001. Proceedings of the Second International Conference on*, vol. 1, pp. 41-50 vol.1, 2001.
- [42] L. Kerschberg, W. Kim, and A. Scime, "Intelligent Web Search via Personalizable Meta-search Agents," in *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, 2008, pp. 1345-1358.
- [43] L. Kerschberg, W. Kim, and A. Scime, "Personalizable semantic taxonomy-based search agent," U.S. Patent 711720703-Oct-2006.
- [44] W. Kim, L. Kerschberg, and A. Scime, "Learning for automatic personalization in a semantic taxonomy-based meta-search agent," *Electronic Commerce Research and Applications*, vol. 1, no. 2, pp. 150-173, 2002.
- [45] J. L. Kolodner, "An introduction to case-based reasoning," *Artificial Intelligence Review*, vol. 6, no. 1, pp. 3-34, Mar. 1992.
- [46] J. A. Konstan et al., "GroupLens: Applying collaborative filtering to usenet news," 1997, <http://eprints.kfupm.edu.sa/43400/>.

- [47] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell, "SAWSDL: Semantic Annotations for WSDL and XML Schema," *Internet Computing, IEEE*, vol. 11, no. 6, pp. 60-67, 2007.
- [48] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76-80, 2003.
- [49] C. Manning, *Foundations of statistical natural language processing*. Cambridge Mass.: MIT Press, 1999.
- [50] F. Manola and E. Miller, "RDF Primer," 2004, <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.
- [51] D. Martin et al., "Bringing Semantics to Web Services with OWL-S," *World Wide Web*, vol. 10, no. 3, pp. 243-277, 2007.
- [52] D. L. McGuinness and F. Van Harmelen, "Owl web ontology language overview," *W3C recommendation*, vol. 10, pp. 2004-03, 2004.
- [53] D. Menasce, "QoS issues in Web services," *Internet Computing, IEEE*, vol. 6, no. 6, pp. 72-75, 2002.
- [54] G. A. Miller, "WordNet: a lexical database for English," *Commun. ACM*, vol. 38, no. 11, pp. 39-41, 1995.
- [55] A. Morikawa and L. Kerschberg, "MAKO: Multi-Ontology Analytical Knowledge Organization based on topic maps," in *Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on*, pp. 459-463, 2004.
- [56] G. Paliouras, C. Papatheodorou, V. Karkaletsis, C. Spyropoulos, and V. Malaveta, "Learning User Communities for Improving the Services of Information Providers," in *Research and Advanced Technology for Digital Libraries*, vol. 1513, Springer Berlin / Heidelberg, 2009, pp. 508-508.
- [57] D. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles, "Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-Based Approach," IN *PROCEEDINGS OF THE SIXTEENTH CONFERENCE ON UNCERTAINTY IN ARTIFICIAL INTELLIGENCE*, pp. 473--480, 2000.
- [58] K. Porkaew and K. Chakrabarti, "Query refinement for multimedia similarity retrieval in MARS," in *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pp. 235-238, 1999.

- [59] A. M. Rashid et al., "Getting to Know You: Learning New User Preferences in Recommender Systems," 2002, <http://eprints.kfupm.edu.sa/42983/>.
- [60] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175-186, 1994.
- [61] P. Resnick and H. R. Varian, "Recommender systems," *Commun. ACM*, vol. 40, no. 3, pp. 56-58, 1997.
- [62] B. Ricardo and R. Berthier, *Modern information retrieval*. Addison-Wesley, 1999.
- [63] Y. Rui, T. Huang, M. Ortega, and S. Mehrotra, "Relevance feedback: a power tool for interactive content-based image retrieval," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 8, no. 5, pp. 644-655, 1998.
- [64] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51-53, Jun. 2007.
- [65] N. Stojanovic, "On Analysing Query Ambiguity for Query Refinement: The Librarian Agent Approach," in *Lecture Notes in Computer Science*, vol. 2813, Springer Berlin / Heidelberg, 2003, pp. 490-505.
- [66] W3C OWL Working Group, "OWL 2 Web Ontology Language Document Overview," <http://www.w3.org/TR/owl2-overview/>.
- [67] I. H. Witten and E. Frank, *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

CURRICULUM VITAE

Hanjo Jeong received his Bachelor of Science in Industrial Engineering from Chonbuk National University, South Korea in 2000. He received his Master of Science in Information Systems from George Mason University in 2003.

He worked as a Research Assistant for the Knowledge Sifter project at E-Center for E-Business (ECEB) at George Mason University, which was sponsored by the National Geospatial-Intelligence Agency (NGA), for his first four years of PhD study. He also served as a Graduate Teaching Assistant for the courses: Introduction to E-Commerce, Software Engineering for World Wide Web, Information Retrieval, Advanced Database Management, and Computer Networks.

His research interests are in Semantic Web, intelligent search systems, recommender systems, and Web Services-based Software Architectures such as Service-oriented Architecture and Cloud Computing.