Enhancing the Results of Recommender Systems using Implicit Ontology Relations

Lamiaa Abdelazziz and Khaled Nagi
Dept. of Computer and Systems Engineering, Faculty of Engineering, Alexandria University, Alexandria, Egypt

Keywords: Recommender Systems, Ontology Mapping, Quality of Recommendation, Performance Analysis.

Abstract: Sharing unstructured knowledge between peers is a must in virtual organizations. The huge number of documents available for sharing makes modern recommender systems indispensable. Recommender systems use several information retrieval techniques to enhance the quality of their results. Unfortunately, every peer has his/her own point of view to categorize his/her own data. The problem arises when a user tries to search for some information in his/her peers' exposed data. The seeker categories must be matched with its responders categories. In this work, we propose a way to enhance the recommendation process based on using simple implicit ontology relations. This helps in recognizing better matched categories in the exposed data. We show that this approach improves the quality of the results with an acceptable increase in computation cost.

1 INTRODUCTION

A Virtual Organization is a temporary network organization, consisting of independent enterprises that come together swiftly to exploit an apparent market opportunity. The enterprises utilize their core competencies in an attempt to create a best-of-everything organization in a value-adding partnership, facilitated by information and communication technology (Fuehrer and Ashkanasy, 1998). Due to the autonomous nature of the participants of virtual organizations, knowledge sharing cannot be done in a structured and centralized way. Peers in a virtual organization have a large amount of documents and each peer (or group of peers) have their own way of classifying them. These two facts create great challenges to any document recommender system. A typical Recommender System (RS) acting in this environment should be distributed and autonomous in order to match the nature of virtual organizations. Typical RS use several Information Retrieval (IR) techniques to generate good results. Moreover, RS should also match the category structure of the seeker with that of the responder. For this to work, the search engines lying within the heart of the RS should be extended.

We base our work on KARe; which stands for Knowledgeable Agent for Recommendations (Gomez Ludermir et al., 2005), (Guizzardi-Silva Souza et al., 2007). It is a multi-agent recommender system that supports nomadic users sharing knowledge in a peer-to-peer environment with the support of a nomadic service. We extend this system in order to enhance the quality of the results coming from search component of the RS. This is done by enriching the search query and enhancing the ranking process of the result set. Our means is employing extra ontological information provided by the peers. Ontology has been used a lot in harmonizing knowledge sharing where it shows great success. In our distributed and autonomous scenario, we restrict ourselves to using simple implicit ontological relations since we do not want to burden the peer with defining their own elaborate ontology (or else they will simply not do it) or force them to use a centralized ontology (since it is not applicable in such a heterogeneous environment).

However, improving the quality of results often involves more computation. For this reason, we test our extension against the original system using the same dataset to quantify the increase in quality versus the increase in computation cost during indexing and searching. We also use a second dataset to verify the generality of our solution.

The rest of the paper is organized as follows. Section 2 provides a background on recommender systems. Our proposed system is presented in Section 3. Section 4 contains an assessment of our pro-

Abdelazziz L. and Nagi K..
Enhancing the Results of Recommender Systems using Implicit Ontology Relations.
DOI: 10.5220/0004105700050014
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2012), pages 5-14
ISBN: 978-989-8565-30-3
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

posed system and its implementation while Section 5 concludes the paper.

2 BACKGROUND

2.1 Recommender Systems

Recommender Systems (RS) are software tools and techniques providing suggestions for items to be of use to a user. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read (Ricci et al., 2011). (Burke, 2007) provides a good taxonomy for distinguishing between recommender systems. Recommender Systems can be categorized in the following classes: *content-based*, *collaborative filtering*, *demographic*, *knowledge-based*, *community-based*, and *hybrid recommender systems*.

In *content-based* RS, the system learns to recommend items that are similar to the ones that the user liked in the past. The similarity of items is calculated based on the features associated with the compared items. Content-based RS can be even found in early standard literature as in (Balabanovic and Shoham, 1997).

Collaborative filtering RS are also called "people-to-people correlation". In their simplest form, implementations of this approach recommend to the active user the items that other users with similar tastes liked in the past (Schafer et al., 2007). The similarity in taste of two users is calculated based on the similarity in the rating history of the users.

Demographic RS recommend items based on the demographic profile of the user. Many web sites dispatch their users to particular pages based on their language or country. Other criteria include age, gender, etc., if this information is collected in the user profile.

Knowledge-based RS recommend items based on specific domain knowledge about how certain item features meet users needs and preferences. Notable knowledge based recommender systems are constraint based or case-based (Bridge, 2006). In these systems, a similarity function estimates the matching degree of the recommendations to the user needs. Here the similarity score can be directly interpreted as the utility of the recommendation for the user.

Community-based RS recommend items based on the preferences of the user friends. The emergence of social networks, such as Facebook, gave rise to this type of systems. Social networks contain billions of records holding user behavioral patterns and combining them with a mapping of their social relationships.

Hybrid RS are based on the combination of the above mentioned techniques. Collaborative filtering methods suffer from new item problems, i.e., they cannot recommend items that have no ratings. This does not limit content-based approaches since the prediction for new items is based on their description (features) that are typically available. Given two (or more) basic RS techniques, several ways have been proposed for combining them to create a new hybrid system. Four different recommendation techniques and seven different hybridization strategies are compared in (Burke, 2007).

2.1.1 Complementary Role of Information Retrieval

Information Retrieval (IR) assists users in storing and searching various forms of content, such as text, images and videos (Manning, 2008). IR generally focuses on developing global retrieval techniques, often neglecting the individual needs and preferences of users.

Nevertheless, both IR and RS are faced with similar filtering and ranking problems. That's why at the heart of RS usually lies a search engine, such as the open source Lucene (Hatcher and Gospodnetic, 2004). Queries submitted to the search engine are enriched with RS-relevant attributes collected by the RS and associated to the resultset.

Nowadays, various search engines also apply some form of personalization by generating results to a user query that are not only relevant to the query terms but are also tailored to the user context (e.g., location, language), and his/her search history. Clearly, both RS and IR will eventually converge to one intelligent user assistant agent.

2.1.2 Complementary Role of Taxonomies and Ontologies

Taxonomy is a hierarchical grouping of entities. Ontologies are a machine readable set of definitions that create a taxonomy of classes and subclasses and relationships between them (Deng and Peng, 2006).

Both taxonomies and ontologies are used to enhance the quality of results suggested by RS. The RS can use the taxonomy structures and ontology to refine the filtering and adjust the ranking of the results sent by the IR internal component. Since most of our knowledge is not hierarchical, it is intuitive to assume that an ontology-based approach would lead to better results. Yet, there is an overhead in defining ontologies by the user and creating a match for the nodes of different ontologies in case of peer-based autonomous systems, such as multi-agent environments.

Clearly, a trade-off would be using *implicit* ontology relations which can easily be defined by the user and then matched by the system. The *simplicity* of the definition of the ontology is critical factor in convincing the autonomous peer to define it.

2.2 Example of Artifact Recommendation System

KARe (Knowledgeable Agent for Recommendations) is a typical example of artifact recommender systems (Gomez Ludermir et al., 2005), (Guizzardi-Silva Souza et al,. 2007). KARe is a multi-agent recommender system that supports nomadic users sharing knowledge in a peer-to-peer environment. Supporting social interaction, KARe allows users to share knowledge through questions and answers. Furthermore, it is assumed that nearby users are more suitable for answering user questions in some scenarios and it uses this information for choosing the answering partners during the recommendation request process.

The first goal of KARe is to develop a distributed system for artifact recommendation. It aims at increasing the precision of current recommendation algorithms. KARe mainly consists of the following components, illustrated in Figure 1 (Gomez Ludermir, 2005).

The *information retrieval* component is divided into two parts. The first part is the process where the user *creates an index* of the knowledge artifacts and concepts. The second is the recommendation mechanism which consists of a *searching* process for the knowledge artifacts. KARe includes the user context in the searching mechanism, providing semantics to the artifacts (i.e., relating it to the concept it is associated with). Similar documents are grouped by the user under the same concept in the context tree. Before submitting the query, the user assigns it to a specific concept. By doing this, the user gives the system extra information on the query content leading to more accurate results.

The *recommendation agent* component simulates the natural social process involved in knowledge sharing by exchanging requests (*questions*) and recommendations (*answers*). Furthermore, the agents have to control the user knowledge base, i.e., whenever a recommendation arrives, the agent stores it in his knowledge base. Social interaction involved in the recommendation process is modeled as agent interaction.

The *peer discovery* component finds potential peers based on proximity information. The system scans the neighborhood for other devices. When new bluetooth-enabled devices are found this information is forwarded to the KARe scanner to check whether the device participates in the KARe platform or not. The KARe scanner prepares a message and sends it to the *peer assistant* agent. If the peer assistant finds the agent representing the device in the KARe platform, then it sends a message back to the KARe scanner.

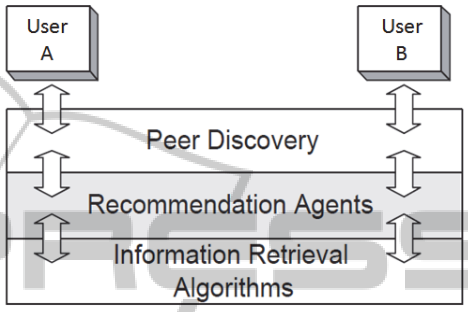


Figure 1: KARe Architecture.

We choose KARe as a base for our work due to the following reasons:

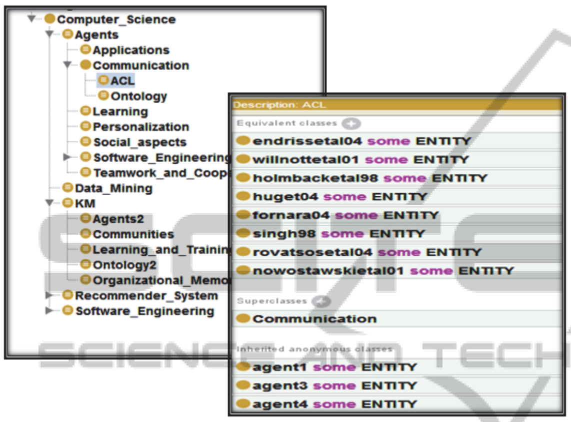
- Its multi-agent nature suits the environment of autonomous virtual organizations.
- Documents fit perfectly in the artifact concept of KARe.
- It comprises a standard search engine: Lucene (Hatcher and Gospodnetic, 2004).
- It originally uses taxonomies in structuring its recommendations.
- It is extendible due to its origins in research labs.
- It belongs to the most general class of RS; which is the *hybrid* family. It combines knowledge-based, location (similar to demographic), community-based, and content-based approaches.

In our work, we concentrate on the first (lowest component): the *information retrieval* component.

3 PROPOSED SOLUTION

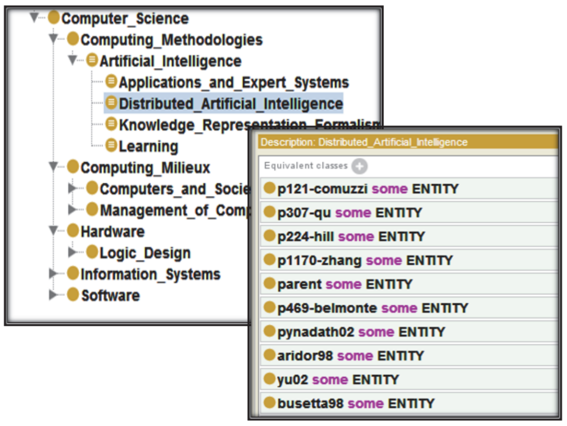
KARe inherently supports a distributed knowledge management approach. One challenge, however, is gaining user acceptance to spend more time in feeding the system with documents and classifying them. Since the way each user classifies his/her own knowledge is particular, we cannot impose a common classification for their artifacts.

Each user is allowed to define and use his/her own taxonomy represented in OWL format (OWL, 2009) to classify artifacts. Figure 2 illustrates a user-defined ontology that holds the artifacts; computer science research papers in this case. This ontology expresses the peer's point of view and does not typically match with the standard classification system of the Association of Computing Machinery (ACM, 1998) used as a reference base in KARe and illustrated in Figure 3.


Computer_Science
Agents
Applications
Communication
ACL
Ontology
Learning
Personalization
Social_aspects
Software_Engineering
Teamwork_and_Coop
Data_Mining
KM
Agents2
Communities
Learning_and_Trainin
Ontology2
Organizational_Memo
Recommender_System
Software_Engineering

Description: ACL
Equivalent classes
endrissetal04 some ENTITY
willnottetal01 some ENTITY
holmbacketal98 some ENTITY
huget04 some ENTITY
fornara04 some ENTITY
singh98 some ENTITY
rovatsosetal04 some ENTITY
nowostawskietal01 some ENTITY
Superclasses
Communication
Inherited anonymous classes
agent1 some ENTITY
agent3 some ENTITY
agent4 some ENTITY

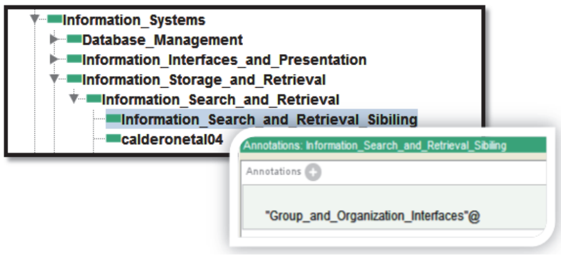
Figure 2: User-defined ontology.


Computer_Science
Computing_Methodologies
Artificial_Intelligence
Applications_and_Expert_Systems
Distributed_Artificial_Intelligence
Knowledge_Representation_Formalism
Learning
Computing_Milieux
Computers_and_Socie
Management_of_Comp
Hardware
Logic_Design
Information_Systems
Software

Description: Distributed_Artificial_Intelligence
Equivalent classes
p121-comuzzi some ENTITY
p307-qu some ENTITY
p224-hill some ENTITY
p1170-zhang some ENTITY
parent some ENTITY
p469-belmonte some ENTITY
pynadath02 some ENTITY
aridor98 some ENTITY
yu02 some ENTITY
busetta98 some ENTITY

Figure 3: ACM ontology.

We integrate the Protégé Ontology Editor (Tudorache et al., 2008) in our proposed system to enable the peer editing his/her OWL ontology file. Through the editor, the user can extend the basic taxonomy trees with other implicit relations such as *sibling*, *parent* and *related-to* between the different tree nodes. The more relations are added, the more support is given for the recommender system in detecting the best matched categories. Figure 4 illustrates a sample ontology relation added to the ACM taxonomy.


Information_Systems
Database_Management
Information_Interfaces_and_Presentation
Information_Storage_and_Retrieval
Information_Search_and_Retrieval
Information_Search_and_Retrieval_Sibiling
calderonetal04

Annotations: Information_Search_and_Retrieval_Sibiling
Annotations
"Group_and_Organization_Interfaces"@

Figure 4: Sample ontology relation added to ACM.

3.1 Component Architecture

The ontology files defined in the previous section are fed to the adapted information retrieval component. Figure 5 illustrates the integration of new components of our proposed system within the information retrieval layer.

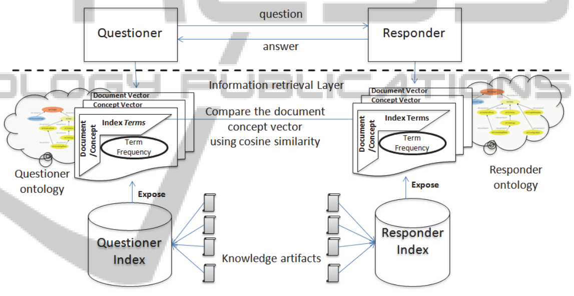


Figure 5: System components of the information retrieval layer.

The *question* contains the name of the document; the *questioner* is searching for and expecting recommendations around it. It is important to note that the answer will contain this document- if found - as a special case, since the *answer* of the *responder* involves recommendations related to this document and not only the document in question. The *knowledge artifacts* represent the document libraries exposed by each peer and constitute the search pool for the recommendation seekers. The *index terms* are the words contained in each document after performing text preprocessing steps. The *term frequency* vector defines the frequency of occurrence of each specific term in a document. The *document* vector represents the terms in the documents together with their frequencies. It is used to formulate the *document matrix* where the documents are represented as rows and the terms as columns and the term frequencies as cell values. The *concept vectors* represent concepts or nodes in the ontology. The dimension size is that of the vocabulary (i.e., number of indexed terms). In order to create the vectors it is necessary to read all indexed terms and count their occurrences in the documents during the indexing process. The *questioner* and *responder indices* represent the store for the concept and document vectors representing the indexed terms and term frequencies.

3.2 The Indexing Process

Figure 6 shows the indexing sequence diagram in UML notation. The indexing process is triggered by the user usually after adding documents to the knowledge artifact store. The *Indexer* is the class that receives the method call `createIndex` from the user and is responsible for handling the process. The Indexer receives two parameters: *a list of documents to be indexed* and *the ontology that classifies them*. The first step towards the creation of the index is to parse each concept of the ontology and the related relations associated with each concept. During the concept parsing, the indexer parses each knowledge artifact to create the vocabulary and index terms. Once indexing is completed, the Indexer creates the vectors for the concepts and knowledge artifacts. During this step, the weight for each term in each document and concept is calculated and stored within the index.

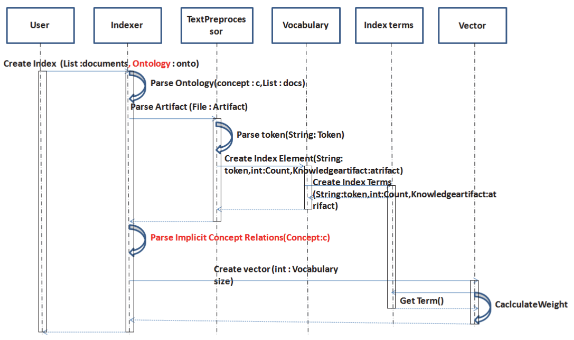

User
Indexer
TextPreprocessor
Vocabulary
Index terms
Vector
Create Index (List :documents, Ontology : onto)
Parse Ontology(concept : c,List : docs)
Parse Artifact (File : Artifact)
Parse token(String: Token)
Create Index Element(String: token,int:Count,Knowledgeartifact:atrifact)
Create Index Terms (String:token,int:Count,Knowledgeartifact:atrifact)
Parse Implicit Concept Relations(Concept:c)
Create vector (int : Vocabulary size)
Get Term()
CaclculateWeight

Figure 6: Indexing sequence diagram.

3.3 The Searching Process

The searching sequence diagram is shown in Figure 7. The major contribution is shown in the last part of the sequence diagram after returning the best matching concepts. Here, we use the implicit ontology relations saved during the indexing process and retrieve the corresponding concept vectors and their artifacts to include them in the calculation process. The similarity is calculated and the query vector and the list of the retrieved documents are ranked according to the similarity. The following is a description of the methods in Figure 7.

- `Query`: is an indication from the user that a question is posted. The parameters are the question itself, the vocabulary spoken by the questioning peer and the concept vector associated with the question.
- `ParseQuery`: is a method for pre-processing the question. It performs stemming and removes the stop words from the query.
- `CreateQueryVector`: compares the question with the knowledge artifacts. For this method to work, we must create a vector representation of the question itself.
- `GetVector`, `StoreSimilarity` and `GetBestConcepts`: the questioning concept vector is compared to each concept vector on the destination taxonomy. For that, we retrieve each vector and check its similarity with the questioning concept vector. At the end of the process, we are able to retrieve the best matching concepts with the questioning concept.
- `NormalizeConceptVector`: aligns the questioning concept vector with the targeted vectors and vocabulary.
- `GetListOfRelatedConcepts`: for each of the best matched concepts, we retrieve the related concepts to include in the next comparison.
- `RecalculateSimilarity`: in this step, we check the similarity between the newly related concepts and the best matched concepts.
- `ReplaceBestMatchedconcepts`: if the similarity calculation shows better concepts we reorder and replace the selected *three* concepts with better ones. The number three is arbitrary chosen and can be changed in the configuration files.
- `GetConcept` and `GetAtrifact`: once we have a good concept, we retrieve its artifacts.
- `StoreArtifactSimilarity`: calculates the similarity among the documents from the related concepts and the question vector. The method returns the resulting documents with associated similarities.

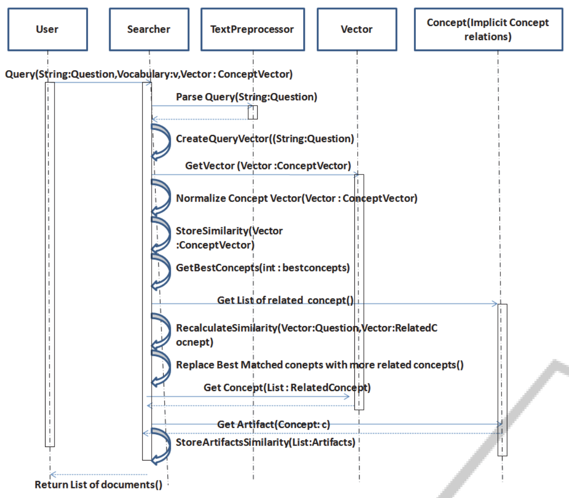

User
Searcher
TextPreprocessor
Vector
Concept(Implicit Concept relations)
Query(String:Question,Vocabulary:v,Vector : ConceptVector)
Parse Query(String:Question)
CreateQueryVector((String:Question)
GetVector (Vector :ConceptVector)
Normalize Concept Vector(Vector : ConceptVector)
StoreSimilarity(Vector :ConceptVector)
GetBestConcepts(int : bestconcepts)
Get List of related concept()
RecalculateSimilarity(Vector:Question,Vector:RelatedConcept)
Replace Best Matched concepts with more related concepts()
Get Concept(List : RelatedConcept)
Get Artifact(Concept: c)
StoreArtifactsSimilarity(List:Artifacts)
Return List of documents()

Figure 7: Searching sequence diagram.

The pseudo-code of the searching steps is shown in Figure 8.

```
Process createAnswer (Vector: conceptVector, String: question)   {
// step 1:  Create the query vector
 Get Question Terms
   Calculate Terms frequencies
   Set Terms frequencies in the query vector
  // step 2:  normalize the concept vector
 Get Intersection between questioner and responder vectors
   Prepare projected responder document weight matrix
  // step 3:  select best matching concept
 Loop on all responder concepts
  Compute similarity between responder vector and normalized questioner
  vector
  Sort the concepts list by similarity
  Return the best matched three concepts
// step 4:  Use indexed implicit ontology relations
 Get the implicit indexed relations for the retrieved concepts
   Load the related concepts
   Compute similarity between responder vector and normalized related
         concept vector
   Add the concepts to the concepts list
   Resort the new list
   Return the three best matched  concepts
// step 5:  Load the artifacts and send results to user
   Load all the artifacts under the three best matched concepts
   Compute similarity between query vector and best matched concepts
         vectors
   Return the list to the user.
}
```

Figure 8: Recommendation algorithm.

3.4 Example

The following is a simplified example of the recommendation process performed by our system. Figure 9 shows the questioner categorization of its library while Figure 10 illustrates that of the responder after adding few implicit ontological relationships.

During the indexing process, concepts and document are parsed to build the document matrix. Assuming the following six books: “Design Patterns Java Workbook”, “Effective Java Programming Language Guide”, “Micro JAVA Game Development”, “Java Collections”, “Client-Side Java Script Reference”, and “Java 2 Network Security” under the concept “Java”, the resulting document matrix is illustrated in Table 1 and the concept vector for “Java” is illustrated in Table 2.

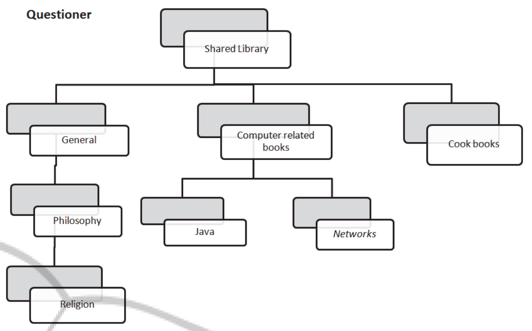


Figure 9: Questioner library categorization.

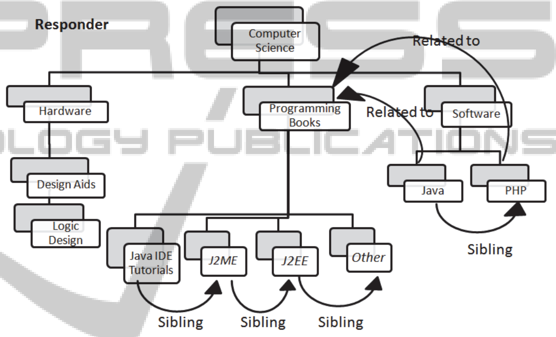


Figure 10: Responder-specific implicit ontology relations.

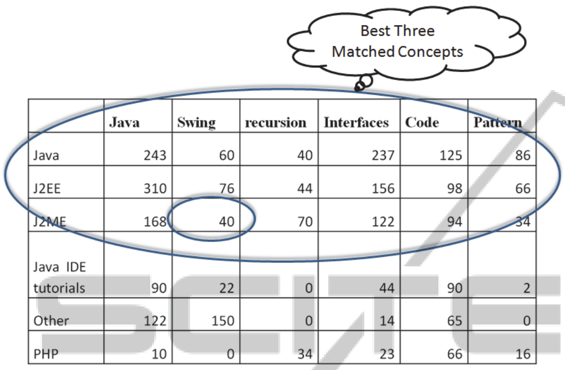
Table 1: Sample document matrix.

	Java	Swing	recursion	Interfaces	Code	Pattern
Design Patterns Java Workbook	103	0	0	35	67	156
Effective Java Programming Language Guide	234	57	12	211	167	20
Micro JAVA Game Development	176	36	87	150	55	55
Java Collections	223	145	123	456	87	98
Client-Side Java Script Reference	311	34	67	52	83	45
Java 2 Network Security	180	0	0	26	33	22

Table 2: Sample concept vector for term “Java”.

	Java	Swing	Recursion	Interfaces	Code	Pattern
JAVA	204.5	45.3	48.1	155	82	66

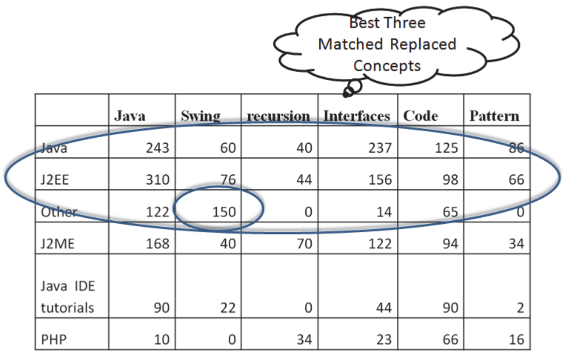
Searching for the document “Swing Basic Components” under the concept “java”, the system tokenizes the question into terms, i.e., “Swing”, “Basic”, and “Components” and assigns a frequency for those terms (in this case all have the frequency 1 due to the absence of duplication). The system compares the associated concept vector with each concept vector of the responder. This process is done with an intersection between the questioner vector and the responder vector. The questioner selected concept vector (calculated above as the average of the document vectors belonging to that concept) is shown in Table 2. Using the cosine similarity measure, the projected vector is compared to each of the responder concept vectors. The best three matched concept vectors are shown in Figure 11.


Best Three Matched Concepts

	Java	Swing	recursion	Interfaces	Code	Pattern
Java	243	60	40	237	125	86
J2EE	310	76	44	156	98	66
J2ME	168	40	70	122	94	34
Java IDE tutorials	90	22	0	44	90	2
Other	122	150	0	14	65	0
PHP	10	0	34	23	66	16

Figure 11: Best three matched concepts.

"Java", "J2EE", and "J2ME" are the three most related concepts considering the cosine similarity of the concept vectors. This list is updated taking into consideration the *related concept list* from the implicit ontological relations generated during the indexing phase. Assuming that "Programming books" is related to "Java", "other" to "J2EE", and "J2EE" to "J2ME", the concept called "other" represents one of the user classifications and indicates that matching concepts does not necessarily depend on the name of the classification and that all of them are included in the cosine similarity calculation. In this particular example, searching for a book about Swing, the system searches in all siblings of java, J2EE and J2ME concepts and replace related concepts. According to the cosine similarly, the system replaces "J2ME" with the concept "other" since it has a relative term frequency of 150 for the term


Best Three Matched Replaced Concepts

	Java	Swing	recursion	Interfaces	Code	Pattern
Java	243	60	40	237	125	86
J2EE	310	76	44	156	98	66
Other	122	150	0	14	65	0
J2ME	168	40	70	122	94	34
Java IDE tutorials	90	22	0	44	90	2
PHP	10	0	34	23	66	16

Figure 12: Best three matched concepts after replacement.

"swing" as compared to 40 for the same term in the concept "J2ME" as illustrated in Figure 12.

4 ASSESSMENT

4.1 Datasets

We use *two* different datasets. The first dataset is the same one used in evaluating the original KARe system. We insist on using the same dataset to quantify the increase in quality versus the increase in computation cost during indexing and searching. A summary of the dataset is found in Table 3 (Dataset I). Taxonomy *A* collects papers and classifies them according to user specific point of view. Taxonomy *B* is taken from the ACM Classification System. In our assessment, we simulate the questions and answers using the title and the body of the scientific papers. We test whether the algorithm is able to retrieve a paper giving its title or keywords from its abstract.

We also use a second dataset to ensure the generality of our solution. The second dataset is shown in Table 3 (Dataset II). The second dataset is a real library of programming books found at a medium sized software company. It is classified from two points of views and is used as questioner and responder exposed libraries. This dataset consists of 125 programming books as a questioner source and 206 as a responder target; thus slightly smaller than dataset I but has the advantage of having much higher terms frequencies, since every book contains hundreds of pages unlike the papers in dataset I with maximum of 20 pages per paper.

Table 3: Summary of datasets.

Dataset	I		II	
Taxonomy	*A*	*B*	*A*	*B*
Number of documents	250	315	125	206
Number of concepts	28	15	32	24
Average doc/concept	9	21	4	9

4.2 Input and Output Settings

To start the experiment execution, we give the system the following as input:

- the title of a document, and
- the concept associated with the paper.

The outputs are:

- the list of matched concepts,
- the list of documents classified under the resulting concepts, and

- the cosine similarity measure attached with each concept.

To distinguish between the false/true positive/negative alarms, we search for a specific paper or book where we previously know that it already exists in the target peer shared data pool.

4.3 Performance Measures

In our assessment, we use the following standard performance indices:

- Number of document *hits*: the average number of hits returned by the responder.
- *Recall*: is the fraction of the documents that are relevant to the query that are successfully retrieved.
- *Precision*: the fraction of retrieved documents relevant to the search.
- *F1-Measure*: is a measure of test accuracy. It considers both the precision and the recall. The F1-Measure can be interpreted as a weighted average of the precision and recall, where an F1-Measure reaches its best value at 1 and worst score at 0.

In addition to scalability measures such as:

- *Indexing time vs. the number of documents*, and
- *Searching time vs. the number of documents*.

4.4 Results of Dataset I

In this set of experiments, we perform 75 queries. A summary of the results is shown in Table 4. Under the original implementation, the number of documents found is 42. Using our ontology-based solution, the number increases to 59. The average recall is also increased from 0.573 to 0.786. Figure 13 shows the detailed plotting of the recall versus the number of queries. The precision is also enhanced from 0.153 to 0.238. Figure 14 shows the detailed plotting of the concept precision versus the number of queries. Consequently, the derived F1-Measure is enhanced from 0.24 to 0.365.

Table 4: Summary of the result of dataset I.

	3 concepts (taxonomy)	3 concepts (ontology)
Document recall	0.573	0.786
Concept precision	0.153	0.238
F1-Measure	0.24	0.365

Considering the scalability measures, our proposed solution incurs a higher cost of computation - as expected - due to the increase in the result quality.

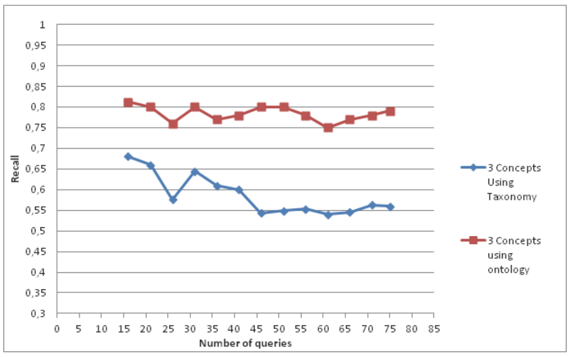


Figure 13: Recall vs. number of queries for dataset I.

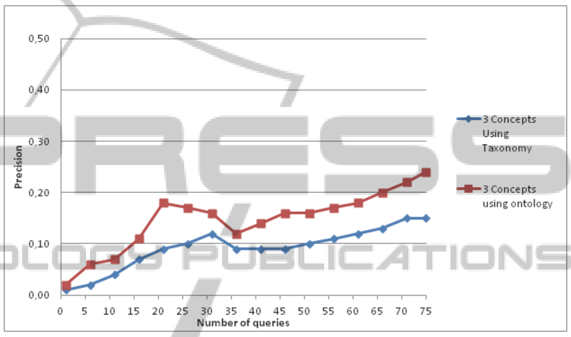


Figure 14: Precision vs. number of queries for dataset I.

The good news is that both indexing time, illustrated in Figure 15, and searching time, illustrated in Figure 16, increase with the same rate as the original KARe implementation. The relative increase in processing compared to KARe does not increase above 10% which is a fair price to pay for the improvement in quality especially that the absolute values for both indexing and searching times are very acceptable.

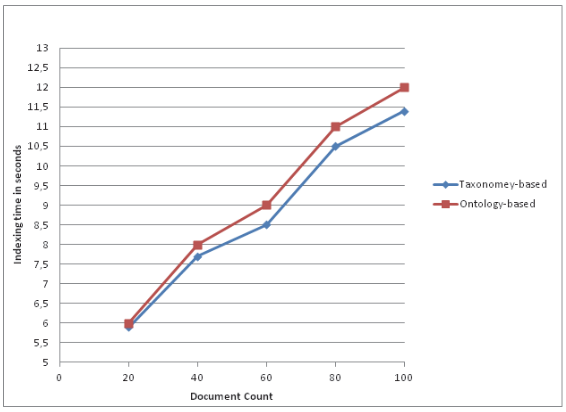


Figure 15: Indexing time for dataset I.

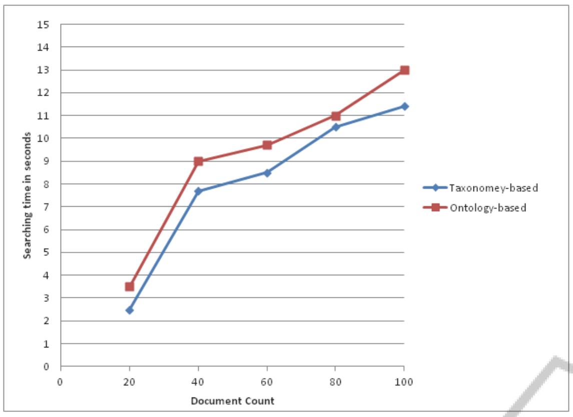


Figure 16: Searching time for dataset I.

4.5 Results of Dataset II

In this second set of experiments, we perform 20 queries. The summary of the results is shown in Table 5. The number of documents found using our proposed solution doubles from 6 to 12 when compared to the KARe implementation. The recall measure also doubles from 0.3 to 0.6. The precision increases from 0.12 to 0.2; and boosting the F1-Measure from 0.17 to 0.3. It is worth mentioned that the relative improvement for this dataset is very similar to the results of the first set of experiments.

Table 5: Summary of the result of dataset II.

	3 concepts (taxonomy)	3 concepts (ontology)
Document recall	0.3	0. 6
Concept precision	0.12	0.2
F1-Measure	0.17	0.3

In Figure 17 and Figure 18, the recall and precision are respectively plotted versus the number of queries.

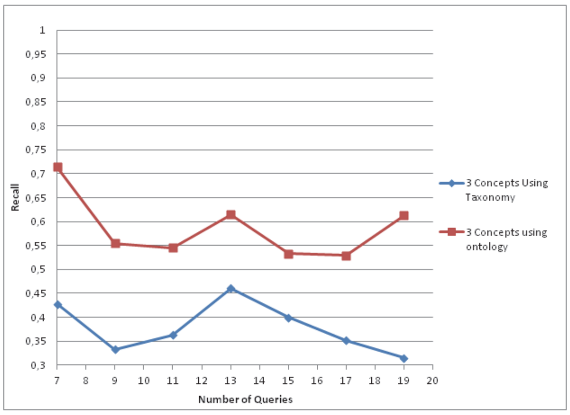


Figure 17: Recall vs. number of queries for dataset II.

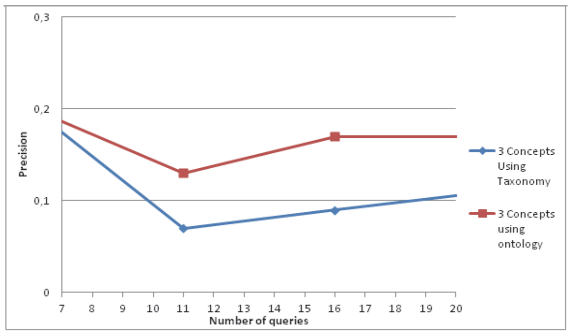


Figure 18: Precision vs. number of queries for dataset II.

The scalability measures reveal a slight increase in the computation time here too. Again, the relative increase in both the indexing time, illustrated in Figure 19, and the searching time, illustrated in Figure 20, are around 10% for all values of the document counts. An interesting observation, however, is made when comparing the absolute indexing times of experiment I, illustrated in Figure 15, with that of experiment II, illustrated in Figure 17. The large increase in indexing time is attributed to the large document size of dataset II (books) as compared to the document size of dataset I (papers). This difference is not present in the searching time due to the scalable nature of the B+-Trees of Lucene regarding retrieval.

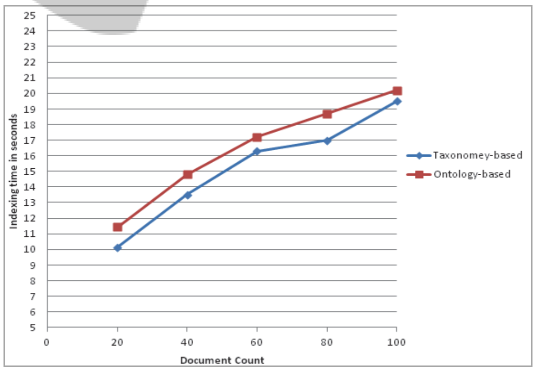


Figure 19: Indexing time for dataset II.

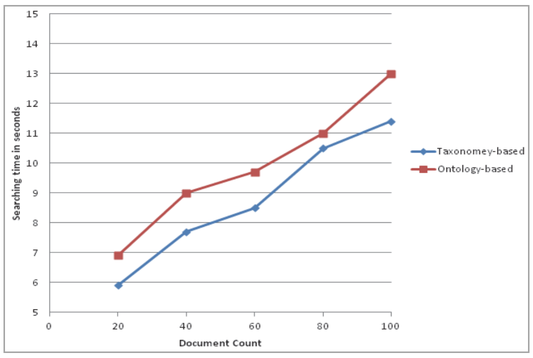


Figure 20: Searching time for dataset II.

5 CONCLUSIONS

Our main contribution in this work is integrating ontological concepts into the recommendation process. We extend the information retrieval part of multi-agent recommender system KARe by allowing the definition of simple ontological relations.
The simple and implicit ontological relations, such as sibling, parent/child and related-to relations,are presented as data properties in the OWL file. Saving those concepts during the indexing process and using them in the searching process gives additional information to support the search and retrieval of better concepts. Instead of increasing the results with more concepts, we focus on keeping the same number of concepts constant while improving their relevance which prevents the precision value from decreasing.

We assess the performance of our proposed system on two datasets to measure the recall, precision and F1-Measure. The results show good improvement in recall and precision. We also measure the indexing and searching time to see the effect of adding related concepts. The results show that adding ontology relations have a slight increase of 10% indexing and searching times.

REFERENCES

ACM, 1998. *The ACM computing classification system*, http://www.acm.org/about/class/ccs98-html.

Balabanovic, M., and Shoham, Y., 1997. Content-based, collaborative recommendation. In *Communication of ACM* 40(3), 66–72.

Bridge, D., Göker, M., McGinty, L., Smyth, B., 2006. Case-based recommender systems. In *The Knowledge Engineering review*, 20(3), 315–320.

Burke, R., 2007. Hybrid web recommender systems. In *The Adaptive Web*, pp. 377–408. Springer, Berlin/Heidelberg.

Deng, S. and Peng, H., 2006. Document Classification Based on Support Vector Machine Using A Concept Vector Model. In *the IEEE/WIC/ACM International Conference on Web Intelligence*.

Fuehrer, E. C., Ashkanasy, N. M., 1998. The Virtual organization: defining a Weberian ideal type from the inter-organizational perspective. *Paper presented at the Annual Meeting of the Academy of Management*, SanDiego, USA.

Gomez Ludermir, P., Guizzardi-Silva Souza, R., and Sona, D., 2005. Finding the right answer: an information retrieval approach supporting knowledge sharing. In *Proceedings of AAMAS 2005 Workshop. Agent Mediated Knowledge Management*, The Netherlands.

Gomez Ludermir, P., 2005. *Supporting Knowledge Management using a Nomadic Service for Artifact Recommendation*. Thesis for a Master of Science degree in Telematics, from the University of Twente Enschede, The Netherlands.

Guizzardi-Silva Souza, R., Gomes Ludermir, P., and Sona, D., 2007. A Recommender Agent to Support Knowledge Sharing in Virtual Enterprises. In *Protogeros, N. (Ed.). Agent and Web Service Technologies in Virtual Enterprises*, Idea Group Publishing.

Hatcher, E., Gospodnetic, O. 2004. *Lucene in Action*. Manning Publications.

Manning, C., 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge.

OWL, 2009. OWL2 Web Ontology Language. Document Overview. In *W3C Recommendation, http://www.w3.org/TR/owl2-overview/*.

Ricci, F., Rokach, L., and Shapira, B., 2011. *Recommender Systems Handbook*, Springer Science+Business Media.

Schafer, J. B., Frankowski, D., Herlocker, J., Sen, S., 2007. Collaborative filtering recommender systems. In *The Adaptive Web*, pp. 291–324. Springer, Berlin / Heidelberg.

Tudorache, T., Noy, N. F., Tu, S. W., Musen, M.A., 2008. Supporting collaborative ontology development in Protégé. In *Seventh International Semantic Web Conference*, Karlsruhe, Germany, Springer.