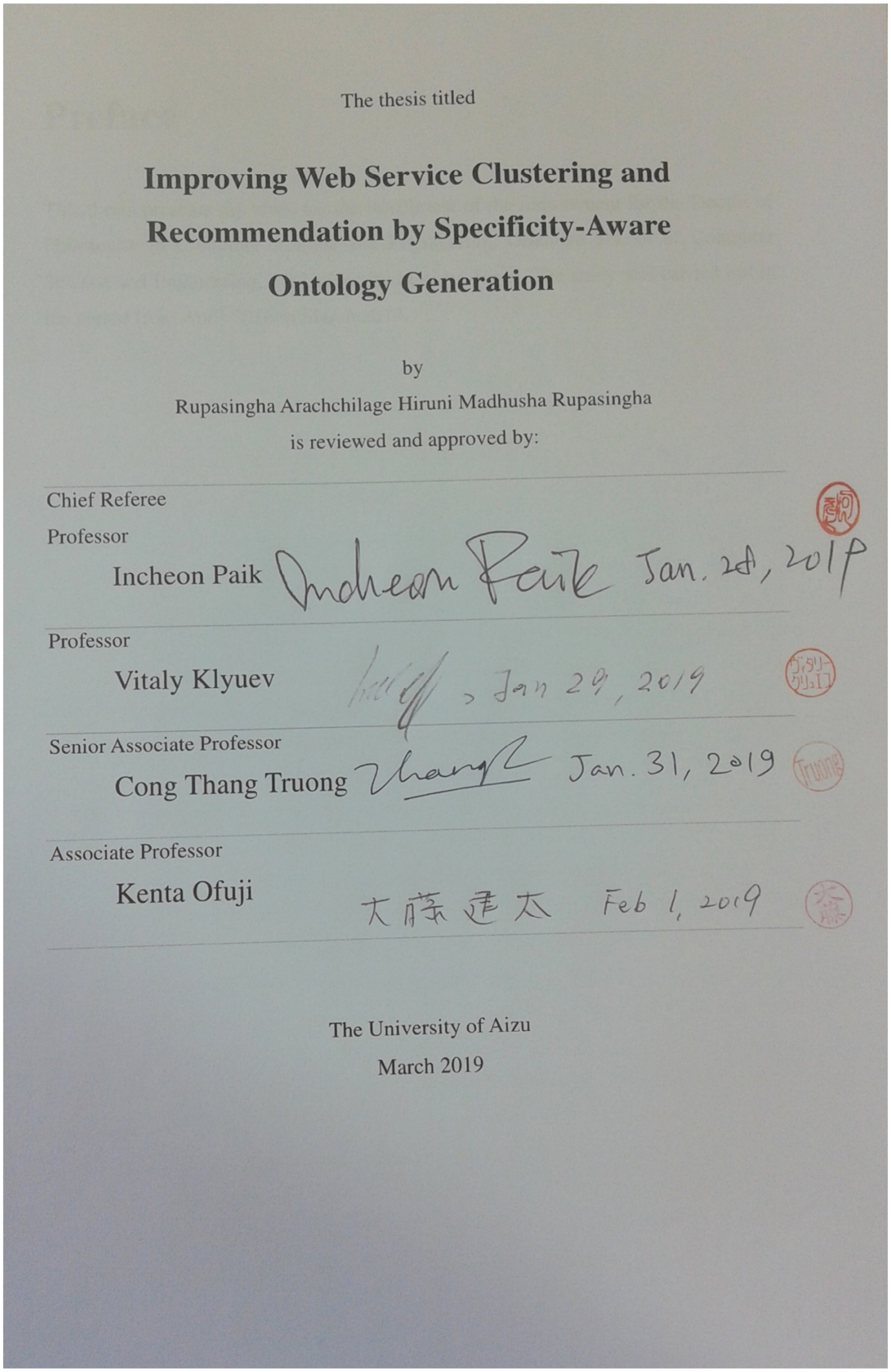
Improving Web Service Clustering and Recommendation by Specificity-Aware Ontology Generation

Rupasingha Arachchilage

Hiruni Madhusha Rupasingha

A DISSERTATION
SUBMITTTED IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE AND ENGINEERING

Graduate Department of Computer and Information Systems
The University of Aizu, Japan
March 2019



The thesis titled

Improving Web Service Clustering and Recommendation by Specificity-Aware Ontology Generation

by

Rupasingha Arachchilage Hiruni Madhusha Rupasingha

is reviewed and approved by:

Chief Referee

Professor

Incheon Paik Incheon Paik Jan. 28, 2019

Professor

Vitaly Klyuev [illegible], Jan 29, 2019 ヴィタリー クリュエフ

Senior Associate Professor

Cong Thang Truong [illegible] Jan. 31, 2019 Truong

Associate Professor

Kenta Ofuji 大藤建太 Feb 1, 2019 大藤

The University of Aizu

March 2019

Preface

This thesis presents my work for the fulfillment of the requirement for the Doctor of Philosophy in Computer Science and Engineering, Graduate School of Computer Science and Engineering, the University of Aizu, Japan. The study was carried out in the period from April 2016 to March 2019.

Acknowledgment

I would like to thank everyone, mentioned and unmentioned, who give me valuable comments, suggestions, encouragement and share inspiring ideas with me.

A special acknowledgement to my supervisor, Professor Incheon Paik, for his guidance over the past three years. He has guided me on not only just writing and research, but also on professional expectations of doctors.

I wish to express my thanks to the members at Professor Incheon Paik's Lab, both current and past for supporting me during this research, encouraging and bringing me happiness.

Abstract

The rapid development of the Internet in recent years has led to a vast increase in the numbers of Web services, which challenges the process of clustering and users' capability to find their favorite services quickly and accurately. Clustering Web services based on their functional features to different domains have started to play a major role in several service management tasks such as efficient Web service discovery and recommendations. In this thesis, we present solutions for Web services clustering and recommendations.

In this thesis, first we present a Web service clustering approach that uses novel ontology learning and a similarity calculation method based on the specificity of an ontology in a domain with respect to information theory. Instead of using traditional methods, we generate the ontology using a novel method that considers the specificity and similarity of terms. The specificity of a term describes the amount of domain-specific information contained in that term. Although general terms contain little domain-specific information, specific terms may contain much more domain-related information. The generated ontology is used in the similarity calculations. New logic-based filters are introduced for the similarity-calculation procedure. If similarity calculations using the specified filters fail, then Information-Retrieval (IR)-based methods are applied to the similarity calculations. Finally, an agglomerative clustering algorithm, based on the calculated similarity values, is used for the clustering.

As a second step we propose a recommendation approach. Among the service recommendation algorithms, Collaborative Filtering (CF) gives credence to user inputs by comparing user's correlations. Although the CF technique is one of the most successful recommendation system technologies, it suffers from data sparsity and cold-start problems, which make the incomplete and inadequate information to analyze a user predicament on Web services. This thesis proposes a CF-based recommendation approach that first alleviates the sparsity problem using a proposed ontology-based clustering. This clustering approach can easily and effectively increase the data density of the user-service dataset by assuming blank user preferences according to the history of user-favored domain(s). Then, we propose a trust-based user rating prediction by determining the trust value between users by calculating the correlation of users. Finally, recommendation was based on these predictions.

We achieved highly efficient and accurate results with this clustering approach than other existing clustering approaches. And the experimental results of recommendation approach indicate that the proposed approach can effectively alleviate the data sparsity and cold-start problems with lower prediction error with the best recommendation performance.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

IR:	Information-Retrieval
KM:	Keyword Match
OB:	Ontology-Based
CAS:	Context-Aware Similarity
SEB:	Search Engine-Based
NGD:	Normalized Google Distance
CF:	Collaborative Filtering
SVD:	Singular Value Decomposition
PLSA:	Probabilistic Latent Semantic Analysis
WSDL:	Web Services Description Language
HTS:	Hybrid Term Similarity
SVMs:	Support Vector Machines
LSI:	Latent Semantic Indexing
PCC:	Pearson's Correlation Coefficient
MAE:	Mean Absolute Error
RMSE:	Root Mean Square Error
NGD:	Normalized Google Distance
XML:	eXtensible Markup Language
BN-LFM:	Bayesian Nonparametric Latent Factor Model
QoS:	Quality of Service
3D:	three-Dimensional
2D:	two-Dimensional
QASSA:	QoS-Aware Service Selection Algorithm
SSM:	Simplified Similarity Measure
TF–IDF:	Term Frequency–Inverse Document Frequency
SoS:	Similarity of Services
SASKS:	Spherical Associated Keyword Space
GRNG:	Gaussian Random Number Generator
SBT:	Social Balance Theory

Chapter 1 Introduction

Web services are reusable software components that allow users to conduct various tasks such as it provides a method for discover, communicating and execute transactions between clients all over the world online with minimal human interaction. As an important innovation in service computing, more and more Web services are developed and published to the Internet. And also most of the business organizations interesting and adaptability is growing towards the Web services. Hence, the number of Web services published on the Web is rapidly increasing day by day.

However, as the number of services developed by different service providers grows rapidly, it is essential to have efficient discovery, selection, and recommendation for Web service complex business processes. As seeking for efficient web service discovery is the main challenge for researchers, research in cluster analysis and recommendation of web services has recently gained much attention.

In the era of service-oriented software engineering, service clustering is used to organize Web services, and it can help to enhance the efficiency and accuracy of service discovery and recommendation. Clustering will enable to group Web services by their similarity and reduces search space. Adequate methods, tools, technologies for clustering the Web services have been developed. Web services can be clustered using functional properties such as input, output, precondition, and effect [1], nonfunctional properties such as cost, reliability and response time [2] or social properties such as sociability [3]. Most of the current researches mainly focus on functionally based clustering. In this research, clustered the Web services considering functional clustering.

When we consider about the Web service clustering, Web service similarity computation is a key part of clustering for differentiating cluster groups based on similarity values. Several methods have been used to compute the Web service similarity in functionally based clustering approaches. But, there are several problems encountered in these approaches and Table 1.1 provides the summary of issues that affect the clustering approaches in existing clustering approaches.

Table 1.1 Summary of issues in current clustering approaches

Current approaches	**Problem for clustering performance**
IR-based methods (ex., Cosine similarity)	– Usually focus on plain text, whereas Web services contain much more complex structures, often with very little textual description. – Lack of up-to-date knowledge – Failed to identify synonyms or variations of terms
IR-based methods (ex., WordNet)	– Fixed, lack of up-to-date knowledge
One-to-one and Structure Matching	– Lack of up-to-date knowledge – Failed to identify synonyms or variations of terms – Consider terms only at the syntactic level
Ontology-Based (OB) method	– Shortage of high quality ontology (defining high-quality ontologies is a major challenge) – Didn't get the advantage of the domain specificity of terms – Lack of up-to-date knowledge
Search-Engine-Based (SEB) (ex., Normalized Google Distance (NGD))	– Do not encode fine grained information. Only uses page-counts of words and ignores the context in which the words appear.

These problems may lead to low recall and low precision in Web service clustering. We proposed an ontology-based clustering approach that based on the specificity and similarity of terms and it could successfully overcome from most of the existing problems.

In order to improve the Web service utilization efficiency, service recommendation techniques are proposed to assist users to find the satisfactory services. We continue our approach until the recommendation by taking the advantage of the proposed clustering approach.

Based on how recommendations are made the recommender systems are classified into the three categories, content-based, Collabarative Filtering (CF) and hybrid approaches [4]. Content-based systems [5] recommend an item to a user based on the item's descriptions and a profile of the user's interests. Here consider the user's taste and behavior and compares the items that are already positively rated by the user with the items negatively rated or didn't rate. CF [6] is based on the other users in a community that share same appreciations. It recommended items that people with similar tastes and preferences liked in the past. Hybrid approach [7] is combine content-based and collaborative methods.

In our approach we used CF methods for the recommendation. CF can be generally categorized into two classes: memory-based and model-based. Memory-based CF algorithms, specially neighborhood-based algorithms [8] compute the similarity between two users or items, and a weighted aggregate of their ratings is use to produces a prediction for the user. There are three types of neighborhood-based algorithms such as user-based, item-based and hybrid. In user-based approach ratings provided by similar users to a target user and item-based approach depends on the most similar items to target item. Hybrid approach is work with combining both user-based, item-based approaches. Model-based [9] techniques such as bayesian networks, Singular Value Decomposition (SVD), and Probabilistic Latent Semantic Analysis (PLSA) based on correlations between either users or items and it present using matrix factorization and it characterizes both items and users by vectors of factors inferred from item rating patterns. Our approach is user-based neighborhood methods CF algorithms, that ratings measured by similar users to a target user and it used to make recommendations. We calculate the similarity between users as a trust weight and assign new ratings based on the user similarities. Figure 1.1 shows the summary of the proposed clusering and recommendation both approaches.

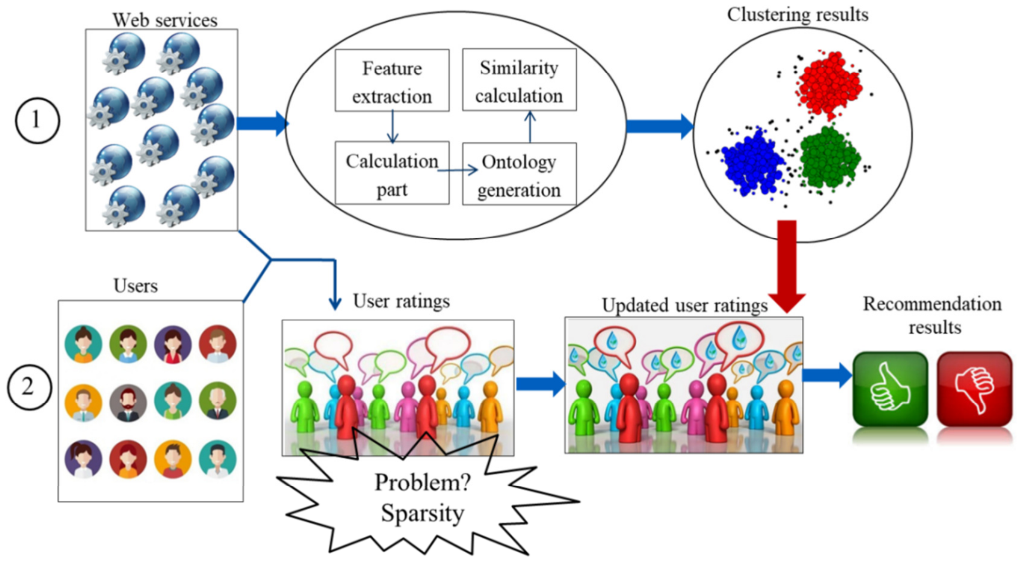


Figure 1.1 Summary of the proposed approach

1.1 Improving Web Service Clustering by Specificity-Aware Ontology Generation

Web services are self-contained, self-describing, modular applications that enable users to conduct various tasks online with minimal human interaction [10]. The rapid evolution of Web services has brought problems, with a variety of providers producing many different Web services for customers. In particular, selecting optimal Web services according to their functional or nonfunctional properties to satisfy user needs has become a challenging and time-consuming task. Therefore, effective Web-service discovery and recommendation tools are now being used to help identify groups of similar services. Clustering similar services by considering their characteristics has become a hot topic in both industry and research.

Web service clustering is used to generate service groups within a large-scale group by considering similar characteristics and functionalities [11,12]. Existing clustering approaches can be classified in terms of the properties used in the clustering process and include functionally-based [1], nonfunctionally-based [2] and social-criteria-based [3] clustering. Here, we propose a new clustering approach based on functional properties, which is the most popular research approach. Our approach is based on extracting the *service name*, *operation name*, *port name*, *input message* and *output message* features expressed in Web Services Description Language (WSDL) [13].

Existing clustering approaches use several methods to compute Web service similarity based on functional properties. These have included IR methods such as cosine similarity [14,15], SEB methods [16] and Keyword Match [11]. In addition, ontology-based methods such as the Hybrid Term Similarity (HTS) method [17,18] used an existing ontology or used their own method to generate the ontology. The CAS method [19] used Support Vector Machines (SVMs) in its similarity calculations. Existing approaches have provided encouraging results, but they still suffer from several drawbacks.

IR-based methods are inadequate for the fine-grained measuring of semantic similarity between services because of the loss of the machine-interpretable semantics. Furthermore, their calculations are best suited to plain text, with the complex structures and ambiguous words of Web services preventing fine-grained improvements. Existing approaches have also failed because of using obsolete knowledge in identifying the latest relationships and words in a corpus. The HTS method generates an ontology without considering domain-specific information, which plays a key role when classifying information than general terms. The CAS method ignores some term relationships when calculating similarities.

We propose a new clustering approach based on a novel ontology-generation method that uses text-mining techniques. It calculates the similarity between services by using newly proposed logic filters, with clustering being achieved via an agglomerative clustering algorithm, based on the cluster centroid method [17], which compares calculated similarity values. This approach helps to overcome the various problems associated with existing approaches.

Some existing clustering approaches [17, 18] use an ontology-based clustering method that generates ontology based on the characteristics of general terms. However, domain-specific information is a significant factor in ontology generation and helps to improve the performance of the clustering process. Specific terms have the ability to supply more domain-specific information than do general terms. We propose a novel ontology-generation method using two types of specific information, namely self-information and context-information. Self-information measures the specificity based on the modifiers in a compound term, which have the ability to describe the domain characteristics included in a term. Context-information can cover areas not addressed by self-information via the composition of multiword terms. It can be measured based on the entropy of the probabilistic distribution of modifiers for the term [20]. For the similarity calculations, we introduce new machine filters by comparing generated ontology relationships. If similarity calculations using these filters fail, we use IR-based methods such as thesaurus-based term similarity and SEB methods. Finally, clustering is performed using an agglomerative clustering algorithm based on the calculated similarity values.

Our new approach helps to address the issues associated with previous approaches by improving the clustering performance. Experimental results indicate that our novel ontology-generation-based clustering approach is effective.

1.2 Improving Web Service Recommendation by Specificity-Aware Ontology Generation

Web services are software modules that provide interoperable communication over the Internet [21]. With the rapidly increasing number of Web services on the Internet, target users face the growing challenge of selecting preferred Web services. There is thus an urgent need for Web service recommendations that help users discover services effectively and efficiently.

Among the three types of recommender systems, recently the CF technique (such as memory-based CF, model-based CF, and hybrid CF) [22]-[24] has become the most widely used method for service recommendation by improving the profits of service providers. The proposed approach is a memory-based CF algorithm such as the user-based neighborhood method, in which user ratings are measured by similar users to a target user and are then used to make recommendations.

However, some major problems limit the usefulness of CF; these include data sparsity and cold-start problems, system scalability, synonymy, and shilling attack, all of which need to be addressed. In this approach, we address the data sparsity [25, 26] and cold-start [27, 28] problems. These problems occur due to insufficient prior transactions and available feedback data that make it difficult to identify similar users (neighbors).

Many attempts have been made to alleviate the sparsity and cold-start problems. Transitive association-based methods [29–31], clustering-based methods [32, 33], which reduce the dimensionality using Latent Semantic Indexing (LSI) [34], binary preference-based methods [35], and correlation and cosine-based techniques [36] give good performance for recommendations while reducing sparsity. In our proposed approach, we address these issues of recommender systems by applying a clustering-based method that successfully and effectively decreases the data sparsity of the user rating dataset. It does this by assuming the nonrated data based on the previously preferred clustering groups of users and services.

In clustering-based methods, applying clustering results to Web services with high precision is affected by the performance of sparsity reduction. Although several clustering methods are available, instead of using a simple algorithm, we used a specificity-based novel ontology generation method as a clustering approach to identify the service cluster groups [37–39] and decrease the data sparsity based on clustering results.

Existing approaches used general terms for their approches. However, general terms contain little domain-related information, which is inadequate for fulfilling the domain information when generating the ontology hierarchy. In our approach, we successfully dealt with this problem and generated ontology by taking advantage of specific terms rather than using general terms. This helps to produce efficient and accurate cluster results.

The amount of domain-specific information included in the term is identified as a specificity of the term [37]. We consider two types of information: self-information and context-information when generating the ontology. Self-information describes the internal structure of terms considering a set of modifiers of the term. Context-information helps cover the issues that cannot be covered by the term modifier structure. After generating the ontology, the similarity calculation is performed based on the generated ontology hierarchy relationships and Information Retrieval (IR) methods. Finally, clustering is performed using an agglomerative clustering algorithm based on the calculated similarity values. Of note, this clustering result shows higher performance than existing approaches and it is used to identify the service domains and successfully alleviate the sparsity problem in the user-service dataset.

After applying the clustering method, a new low-sparsity matrix is used for further calculation. Pearson's Correlation Coefficient (PCC) is used to calculate the similarity between user u and user v. The Calculated similarity value is assigned as a trust weight value between the two users. Then the new rating prediction is based on the new low-sparsity matrix and calculated trust weight values. Finally, top services are recommended to the users based on the predicted ratings.

We conducted comparative experiments on the user-service dataset using well-known statistical accuracy metrics, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), on several parameters to generate ontology and other sparsity reduction methods. Significant improvement in recommendation performance is shown by giving the lowest error rate on MAE and RMSE compared with existing service recommendation mechanisms.

1.3 Summary of the Original Contributions

We proposed two reserach approaches by introducing specificity-aware ontology generation method. Main original contributions for Web service clustering and recommendation to increase the performance have been made in the work as follows.

1. Specificity-aware ontology generation method has been proposed to calculate the Web service similarity to improve the accuracy of Web service clustering.

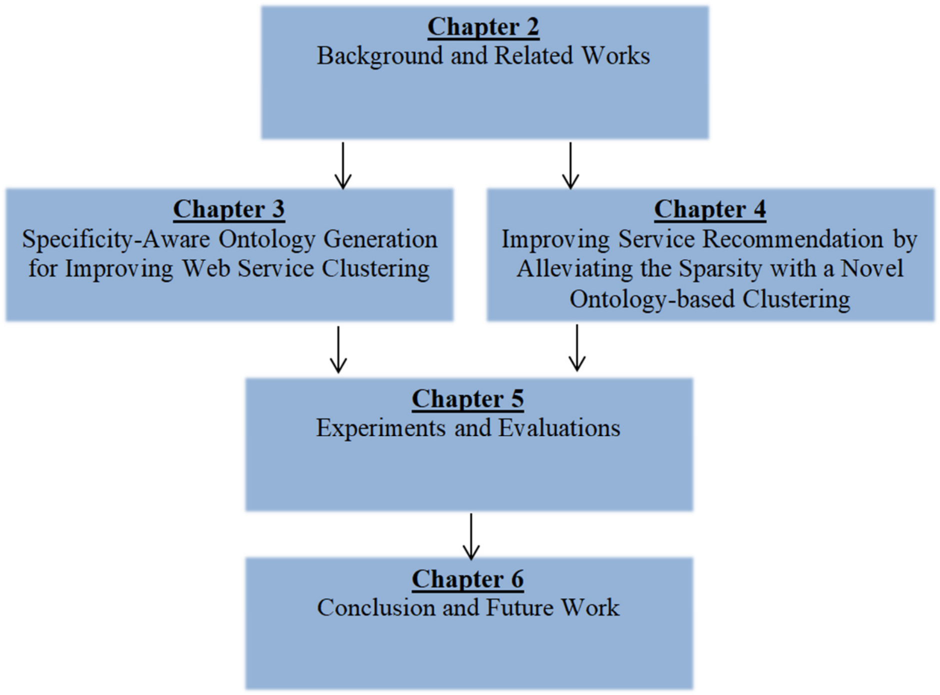
2. Proposed novel clustering apprach is used to ovecome the sparsity and cold-start problem by alleviaing the saprsity of web service-user ratings. Then recommendation method has been proposed to recommmend Web services to users based on their domain preference.

1.4 Thesis Organization

The thesis mainly consists of five parts as shown in Figure 1.2.

Section 1: Background and Related Work

In the chapter 2, the background and related work of the study will be presented. First, we give an explanation of Web service, ontology learning, similarity calculation, clustering and recommendation. Then, we discuss the work related to ontology learning, similarity calculation and Web service clustering. Next, we focus on term specificity, existing CF and challenges of user-based CF algorithms such as the sparsity problem and the cold-start problem.


Chapter 2
Background and Related Works
Chapter 3
Specificity-Aware Ontology Generation for Improving Web Service Clustering
Chapter 4
Improving Service Recommendation by Alleviating the Sparsity with a Novel Ontology-based Clustering
Chapter 5
Experiments and Evaluations
Chapter 6
Conclusion and Future Work
Figure 1.2 Thesis organization

Section 2: Specificity-Aware Ontology Generation for Improving Web Service Clustering

In chapter 3, first discuss about the of specificity-aware ontology generation. Then explain about the proposed clustering approach based on ontology generation.

Section 3: Improving Service Recommendation by Alleviating the Sparsity with a Novel Ontology-based Clustering

In chapter 4, first, discuss the sparsity alleviating using novel specificity-aware ontology generation. Then explain about the proposed recommendation approach using the updated user-service matrix.

Section 4: Experiments and Evaluations

In chapter 5, experiments and evaluations of our proposed clustering approach and recommendation approach are presented. Experimental results show that our proposed specificity-aware ontology generation approach can improve the Web service clustering and recommendation by addressing the issue of previous approaches.

Section 5: Conclusion and Future Works

In chapter 6, the thesis is concluded and the future works are presented.

Chapter 2 Background and Related Work

In this chapter, first we present overview of the Web services. Then, we describe ontology learning, similarity calculation, Web service clustering, recommendation and give the brief description of existing approaches.

2.1. Overview of the Web Services

A Web service is a set of related application functions that can be programmatically invoked over the Internet. It allows buyers and selers all over the world to discover each other, connect dynamically and execute transactions in real time with minimal human interaction.

A Web Service is can be defined by following ways:

- is a client server application or application component for communication.
- method of communication between two devices over network.
- is a software system for interoperable machine to machine communication.
- is a collection of standards or protocols for exchanging information between two devices or application.

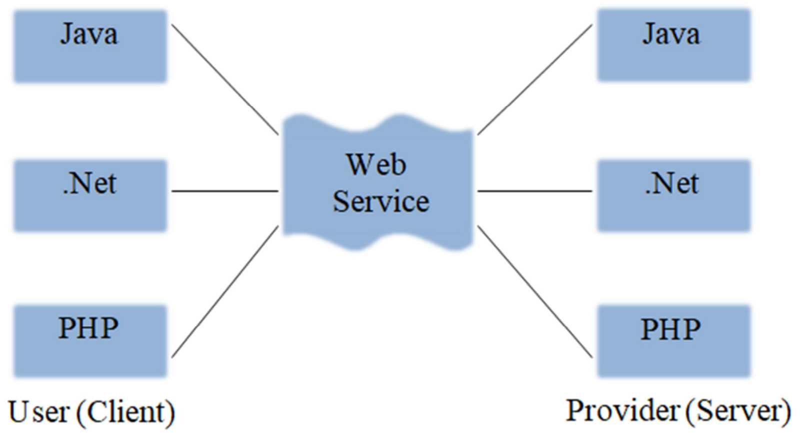


Figure 2.1 Web service

As you can see in the Figure 2.1, Java, .net or PHP applications can communicate with other applications through web service over the network. For example, Java application can interact with Java, .Net and PHP applications. So web service is a language independent way of communication.

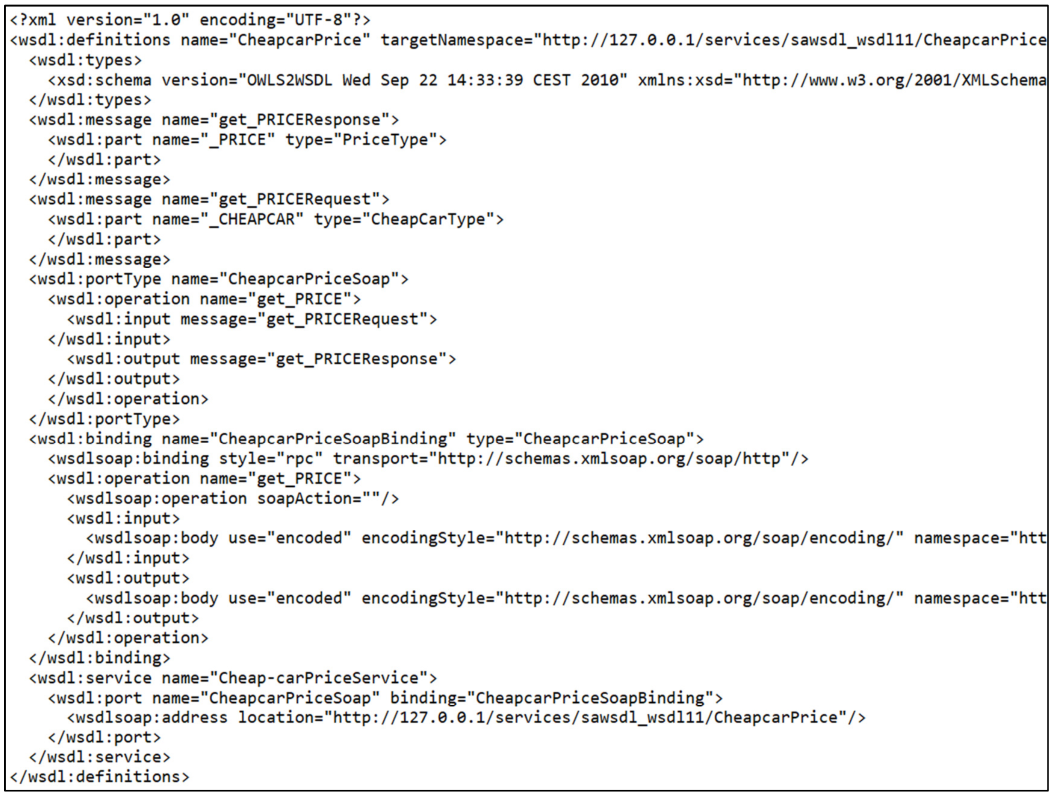
Web service describes a collection of operations that are network accessible through standardized eXtensible Markup Language (XML) messaging. XML is used to encode all communications to a Web service. For an example, a client invokes a Web service by sending an XML message, and then he waits for a corresponding XML response. Because all communication is in XML, Web services are not tied to any one operating system or programming language and hides the implementation details of the service.

A Web service is described using a standard, formal XML notion, called its service description. The WSDL is an XML-based interface definition language that is used for describing the functionality offered by a web service. WSDL document describes all the details necessary to interact with the service, including message formats, transport protocols and location. Web services fulfill a specific task or a set of tasks. Web services share business logic, data and processes through a programmatic interface, represent an important way for businesses to communicate with each other and with clients. They can be used alone or with other Web services to carry out a complex aggregation or a business transaction as we discussed in introduction part. The concept of Web services has therefore become a widely applied paradigm in research and industry.

2.1.1. WSDL structure

There are many industrial and academic standards for Web service descriptions, including WSDL [13], OWL-S [40] and Web service modeling ontology [41]. WSDL is an XML-based language used to describe the services a business offers and to provide a way for individuals and other businesses to access those services electronically. WSDL represents the most fundamental form for standard-of-service APIs.

We use the structure of WSDL to cluster services and we translate material retrieved in other formats into WSDL. Figure 2.2 represents the part of WSDL file of *cheapcar_price_service*.



```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="CheapcarPrice" targetNamespace="http://127.0.0.1/services/sawsdl_wsdl11/CheapcarPrice
  <wsdl:types>
    <xsd:schema version="OWLS2WSDL Wed Sep 22 14:33:39 CEST 2010" xmlns:xsd="http://www.w3.org/2001/XMLSchema
  </wsdl:types>
  <wsdl:message name="get_PRICEResponse">
    <wsdl:part name="_PRICE" type="PriceType">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="get_PRICERequest">
    <wsdl:part name="_CHEAPCAR" type="CheapCarType">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="CheapcarPriceSoap">
    <wsdl:operation name="get_PRICE">
      <wsdl:input message="get_PRICERequest">
    </wsdl:input>
      <wsdl:output message="get_PRICEResponse">
    </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CheapcarPriceSoapBinding" type="CheapcarPriceSoap">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="get_PRICE">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input>
        <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="htt
      </wsdl:input>
      <wsdl:output>
        <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="htt
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Cheap-carPriceService">
    <wsdl:port name="CheapcarPriceSoap" binding="CheapcarPriceSoapBinding">
      <wsdlsoap:address location="http://127.0.0.1/services/sawsdl_wsdl11/CheapcarPrice"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Figure 2.2 Structure of WSDL file

WSDL file provides definitions grouped into the following sections:

- <definitions>: The root element of a WSDL document. The attributes of this element specify the name of the WSDL document, the document's target namespace and the shorthand definitions for the namespaces referenced in the WSDL document.
- <types>: The XML Schema definitions for the data units that form the building blocks of the messages used by a service.
- <messages>: The section that contains the description of the messages exchanged during invocation of a service operation.
- <portType>: The most important WSDL element. It defines a Web service, the operations that can be performed and the messages that are involved.
- <binding>: The section that defines the message format and communication protocol details for each port. It links the port type to a transport method.
- <service>: The section that defines port elements that specify where requests should be sent.

In the past few years there are numerous researches has been done in the field of Web services. Ontology learning, similarity calculation, Web service clustering, , and recommendation are four related research topics with this new approach.

2.2. Ontology Learning

Ontology is a formal naming and definition of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain. There is, however, no general agreement on which requirements the formal representation needs to satisfy in order to be appropriately be called an ontology. Depending on the particular point of view, ontologies can be simple dictionaries, taxonomies, thesauri, or richly axiomatized top-level formalisations. Ontology learning is a subtask of information extraction. Figure 2.3 shows the ontology example.

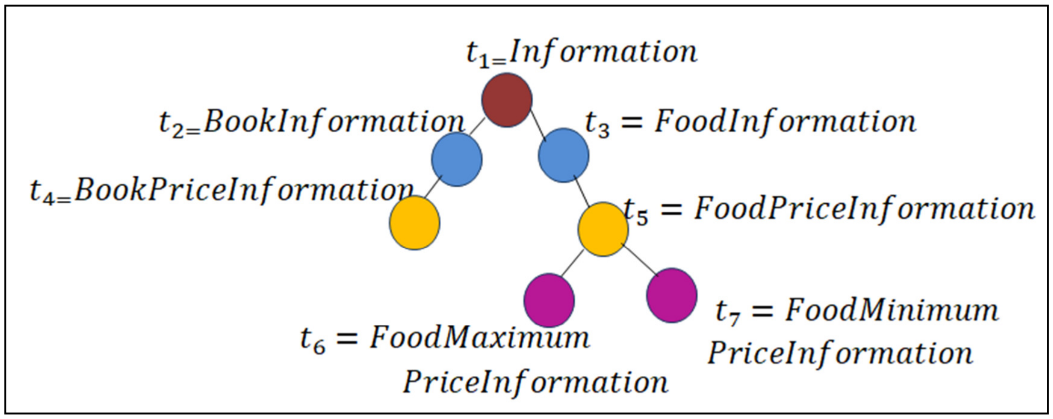

$t_1 = Information$
$t_2 = BookInformation$
$t_3 = FoodInformation$
$t_4 = BookPriceInformation$
$t_5 = FoodPriceInformation$
$t_6 = FoodMaximum PriceInformation$
$t_7 = FoodMinimum PriceInformation$

Figure 2.3 Ontology example

2.3. Calculating Web Service Similarity

Cosine similarity [42] is used to calculate the similarity of features. It measures the similarity between two sentences or documents. But, for a complex Web terms it makes difficult. To discover related Web services [16], clustered services using the four types of features based on the tree-traversing ant algorithm. They similarity computed using the NGD. However, NGD is not to consider the context of the terms occurs.

Based on the Web service semantics and clustering research work [44] presented a Web service discovery approach. They computed similarity values of Web services using WordNet and ontologies. In [43], used similarity computation methods such as Web-Jaccard, Web-Dice, Web Overlap and Web-PMI as a Web based measuring methods. In [14], researches used cosine similarity as IR based methods to similarity calculation of features. IR techniques like cosine similarity mainly focus on plaintext, for more complex terms in Web services it is very problematic.

Eg:

get_traffic_information, get_car_information, get_book_information

- get_traffic_information, get_car_information – similarity is high
- get_traffic_information, get_book_information – similarity is low

2.4. Web Service Clustering

Service clustering, which can greatly reduce the search space of service discovery, is an efficient approach to increasing the discovery performance. The idea is to organize semantically similar services into one group. As we mentioned in introduction section, service clustering can be categorized as functionally-based, non-functionally-based and social criteri-based clustering. Figure 2.4 shows the clustering example.

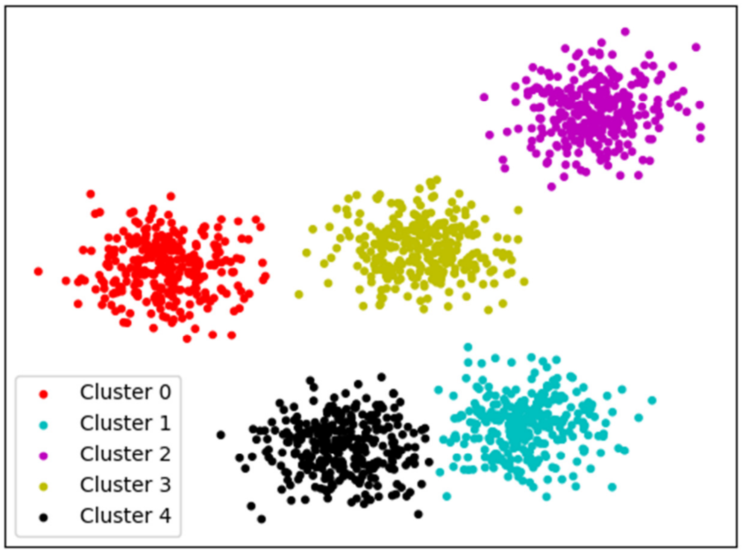


Figure 2.4 Clustering example

2.4.1. Functionally-based Web service clustering

Functional based clustering approaches use functional attributes of Web services such as *service name*, *operation name*, *port name*, *input* and *output message* in clustering process. Liu et al. [16] used tree-traversing ant algorithm to cluster the services and in similarity calculating of Web service features, they used the corpus-based method based on Normalized Google Distance (NGD). They extracted four features for their clustering approach namely host name, context, content and name from WSDL documents. Sometimes different services published by the same host. Because of this, considering context and the host name do not make better way to the clustering services. Elgazzar et al. [11] used Quality-threshold clustering algorithm for clustering and extracted features as messages, complex data types and ports with content and service names. For a similarity calculation, they used KM technique and structure matching; it helps to decide the number of similar matches between Web services. They only consider the pair wise matches and did not consider semantic patterns of complex data types.

Chen et al. [14] proposed WordNet-VSM model to calculate the feature similarity by generating vectors of service name, operation and message. They used unsupervised neural networks based on a kernel cosine-similarity measure instead of using traditional clustering algorithms.

Research work [44] presented WordNet-based similarity calculation approach to measure the similarity between service name and text description and similarity calculation method using an input and output parameters based on domain ontology. OWL-S files are used to extract the semantic information and service similarity calculates through it. Kmedoids is used as the clustering algorithm. In [45], association rule is used to identify relationships between clustering parameters using Web application description language documents. They proposed combination method to empower RESTful semantic Web services using a learning ontology and Web application description language. In [46], researches proposed post-filtering method to increase the performance of clusters. It used CAS based approach and rearranging the incorrect Web services. In [47], proposed a machine learning method to understand the service space through number of latent functional factors and Bayesian Nonparametric Latent Factor Model (BN-LFM) for clustering and service representation.

2.4.2. Non functionally-based Web service clustering

There are only limited research works related with the Non functional-based and social criteria-based clustering. In Non functional-based clustering approaches, Web services' attribute is used for clustering process namely Quality of Service (QoS). In [48], researches proposed Web service clustering based on QoS properties with genetic algorithm to improve the efficiency of service discovery. According to research work [2], the service selection algorithm also plays a significant role. They proposed a new algorithm, called QSSAC, for the solution as a service selection problem. It used for the service clustering and it can cluster a large number of Web services into an identified small groups using their different QoS properties. Algorithm is capable to covenant the near-optimal Web service selection procedure and reduce the execution time. Research work [49], QASSA used to resolves QoS-aware services using clustering techniques such as K-Means algorithm. They did service clustering in a novel way according to QOS values. Chen Wu et al. [50] proposed new credibility-aware QoS prediction approach with two-phase K-means clustering for improve the prediction accuracy through decreasing the unreliable data.

2.4.3. Social criteria-based Web service clustering

Research works [3] and [51] proposed social criteria based clustering method and service composition approach respectively. As social properties they considered sociability preference in generating the global social service network. They proposed it by connecting isolated service islands to increase the sociability on a global scale of services. It is help to improve the discovery and composition. However, in this new approach, only consider the functionally based clustering.

2.5. Web Service Recommendation

Recently, recommendation systems are attracting a lot of attention since it helps users to deal with information overloading on the Web, recommendation algorithms have been used to recommend books and CDs at Amazon.com, movies at Netflix.com and lot of other things. With the exponential growth of Internet in recent years, Web service recommendation is also become a significat task.

2.5.1. CF recommender systems

"Recommend items that similar users liked." CF recommender systems are basic forms of recommendation engines. In this type of recommendation engine, filtering items from a large set of alternatives is done collaboratively by users' preferences.

The basic assumption in a CF recommender system is that if two users shared the same interests as each other in the past, they will also have similar tastes in the future. If, for example, user *A* and user *B* have similar movie preferences, and user *A* recently watched *Titanic,* which user *B* has not yet seen, then the idea is to recommend this unseen new movie to user *B*. The movie recommendations on Netflix are one good example of this type of recommender system.

Figure 2.5 shows the collaborative Filtering types and Figure 2.6 shows the CF recommendation example.

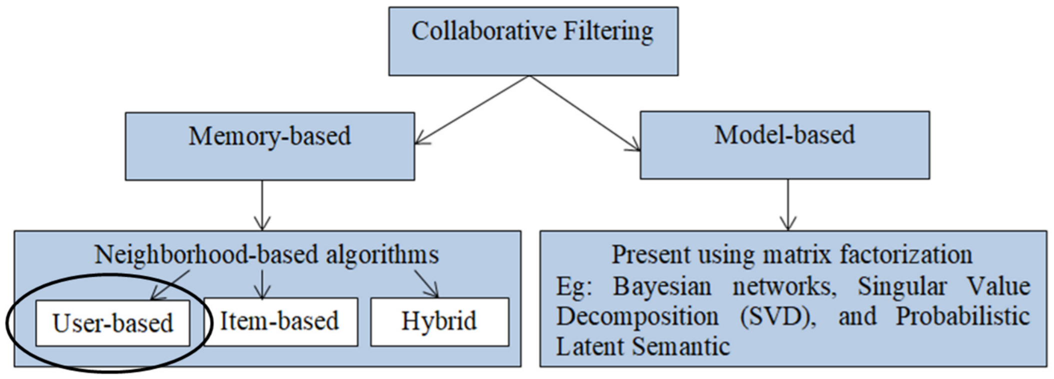

Collaborative Filtering
Memory-based
Model-based
Neighborhood-based algorithms
User-based
Item-based
Hybrid
Present using matrix factorization
Eg: Bayesian networks, Singular Value Decomposition (SVD), and Probabilistic Latent Semantic

Figure 2.5 Collaborative Filtering types

2.5.2. Content-based recommender systems

"Recommend items that are similar to those the user liked in the past." Content-based recommendation systems analyze item descriptions to identify items that are of particular interest to the user. Content-based recommendation systems may be used in a variety of domains ranging from recommending web pages, news articles, restaurants, television programs, and items for sale.

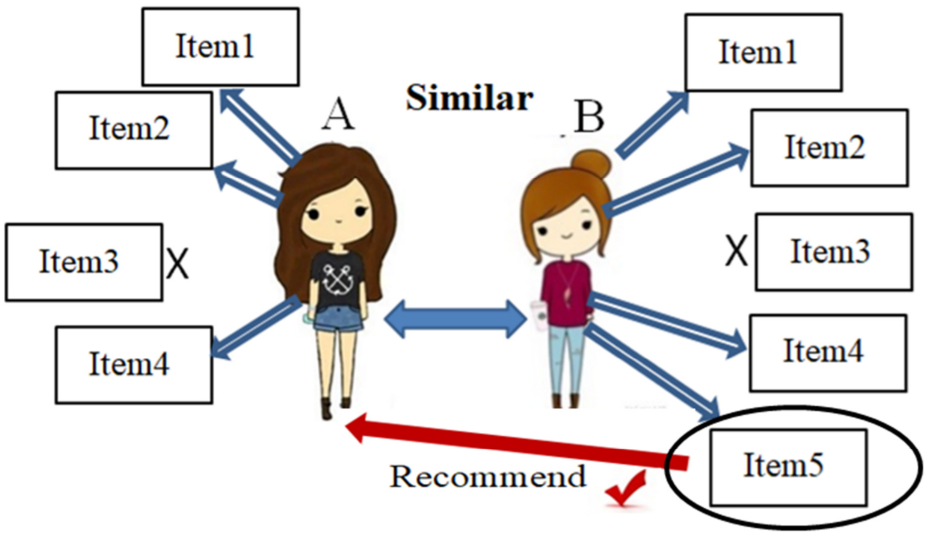


Figure 2.6 CF recommendation example

2.6. Related Work

With the exponential growth of Web applications in recent years, various approaches have been proposed for Web-service clustering, discovery and recommendation. In this section, we describe several representative works related to ontology learning, similarity calculation, Web-service clustering, term specificity, CF, and challenges of user-based CF algorithms.

2.6.1. Ontology learning

Previous ontology-based HTS clustering approaches [17,18] extract relevant features from WSDL documents, namely *service name*, *domain name*, *operation name*, *input message* and *output message*. An ontology is then generated via the hidden semantic patterns of the extracted features. This is achieved by splitting complex terms into individual words before generating the ontology. An ontology hierarchy is generated through two types of relations, namely concept hierarchy (*Subclass–Superclass*) and triples (*Subject–Predicate–Object*). Similarity calculation and clustering are achieved by comparing generated ontology relationships.

Fang et al. [52] proposed an agility-oriented and fuzziness-embedded cloud-service ontology model, which adopted agility-centric design. The model enabled comprehensive service specification by capturing cloud concept details and their interactions. Utilizing the model as a knowledge base, a service-recommendation system prototype was developed. In Xie et al. [53], a domain-ontology hierarchy was defined to describe the conceptual semantic information. They used a weighted domain-ontology method to calculate functional similarity using input and output parameters. A domain ontology was developed using semantic dictionaries and existing ontologies from the Internet. Xia and Yoshida [54] proposed Web-service recommendation via an ontology-based similarity measure. By proposing a similarity assessment model, they visualized a Web-service recommendation framework. However, this method could not capture the real semantics existing among Web services, and therefore did not address search-space issues.

2.6.2. Similarity calculation

Kumara et al. [17] used ontology-based similarity calculations that involved proposed filter values to identify the ontology relationships. By assigning different weights to the filters, they calculated similarity via edge-count-based similarity calculations. Banage et al. [19] proposed new similarity calculations using machine-learning methods such as SVMs. Their method generated feature vectors through extracted terms from Google and Wikipedia. Similarity calculations involved converting the SVM output into posterior probabilities. Our previous approach [18] proposed a new similarity calculation method that combined an ontology-based method [17] and an SVM-based method [19] that gave more-efficient results.

Shi et al. [55] acquired the semantic similarity between Web-application-program interfaces and mash-ups by proposing an enhanced cosine-similarity calculation method, where a penalty term for the dissimilarity of two vectors was introduced. Lei et al. [56] proposed both a Web-service similarity-measurement method and a recommendation method based on ontology and IR techniques. Their method calculated similarities and classified services according to their topics, functionality and semantics. Chen et al. [14] proposed an unsupervised self-organizing-map neural network algorithm, again based on a kernel cosine-similarity measure, for clustering Web services automatically. Paik et al. [43] used the Web-based measuring methods Web-PMI, Web-Dice, Web-Jaccard and Web-Overlap for similarity computation. Latent terms hidden in the Web document could be provided via a similarity calculation measure using a search engine, thereby giving more flexibility to finding similar terms to terms in a query.

2.6.3. Web service clustering

In previous studies [17,18], an agglomerative clustering algorithm has been used in the cluster-center identification approach to clustering the Web services. This is a bottom-up hierarchical clustering method and starts by assigning each service to its own cluster. The CAS method uses a spatial clustering technique called the associated keyword space, which was effective for noisy data [19]. It projected the clustering results for a three-Dimensional (3D) sphere onto a two-Dimensional (2D) spherical surface for 2D visualization.

Zhou et al. [57] used unsupervised clustering with K-means clustering on attrition rates to determine an appropriate segmentation and number of segments. The clustering method starts by choosing appropriate attributes. Final clustering results are converted into a score for a recommendation. Mabrouk et al. [49] investigated clustering techniques that used a K-means algorithm based on a QoS-Aware Service Selection Algorithm (QASSA). It grouped service candidates associated with an activity into several clusters according to their QoS values. QASSA defines service selection under global QoS requirements as a set-based bi-level optimization problem, representing a mathematical model for the problem. Liu and Wong [16] proposed an integrated feature-mining and automatic-clustering approach dedicated to Web-service clustering. A tree-traversing ant algorithm [58] was used for clustering Web services by introducing a new semantic-relatedness measure based on a combination of four types of extracted features.

2.6.4. Term specificity

Term specificity has not been discussed in recent work. Caraballo et al. [59] used a large text corpus and proposed a method for determining the relative specificity of nouns, i.e., some nouns are more specific than others. The approach calculated the specificity of general nouns using the distribution of modifiers. It was based on the assumption that general nouns are usually modified, whereas specific nouns are rarely modified. Aizawa [60] measured term specificity through information-theoretic methods based on the pairwise mutual information of terms. Ryu et al. [61] used specificity-measuring methods for terms based on information theory, such as measures that use the compositional and contextual information of terms.

2.6.5. CF

CF [29] leverages the experiences of similar users in the system to predict the target users' personalized preferences and make recommendations. It only requires past user ratings to predict the remaining unknown ratings; then, items can be recommended to users according to the predictions [62, 63]. Many CF approaches have been developed in industrial and academic fields in both memory-based and model-based categories. Memory-based methods simply recall the rating matrix and make recommendations based on the correlation between the users and items. Model-based methods build a model that can predict the user's ratings using a given rating matrix and then make recommendations based on the fitted model.

In [64], presented a mashup service recommendation approach by integrating the implicit API correlations regularization into the matrix factorization model. When determining the future invocation of APIs by a target mashup, they considered both the content features of APIs and the historical invocation relations between APIs and mashups are essential. Yu et al. [65] proposed an item-based CF method that explores the latent bundling relationships between products and integrates the semantics of bundling sets with building sequential patterns to model user-item behaviors. Adeniyi et al. [66] applied memory-based methods such as the k-nearest neighbor classification method to measure the user–item similarity through the entire user–item matrix and thereby make recommendations. Zuo et al. [9] proposed the use of a model-based method, such as a neural network-based method, to extract the in-depth features from tag space layer by layer. Based on those extracted abstract features, the user's profile was then updated and recommendations were made. Engelbert et al. [67] outlined a recommendation method based on a Bayesian model; their proposed system adapted the method for use in the application area of television and analyzed the user's behavior to present new content choices.

2.6.6. Challenges of user-based CF algorithms

Figure 2.7 explained the way of selecting a proposed clustering approach for the new recommendation process.

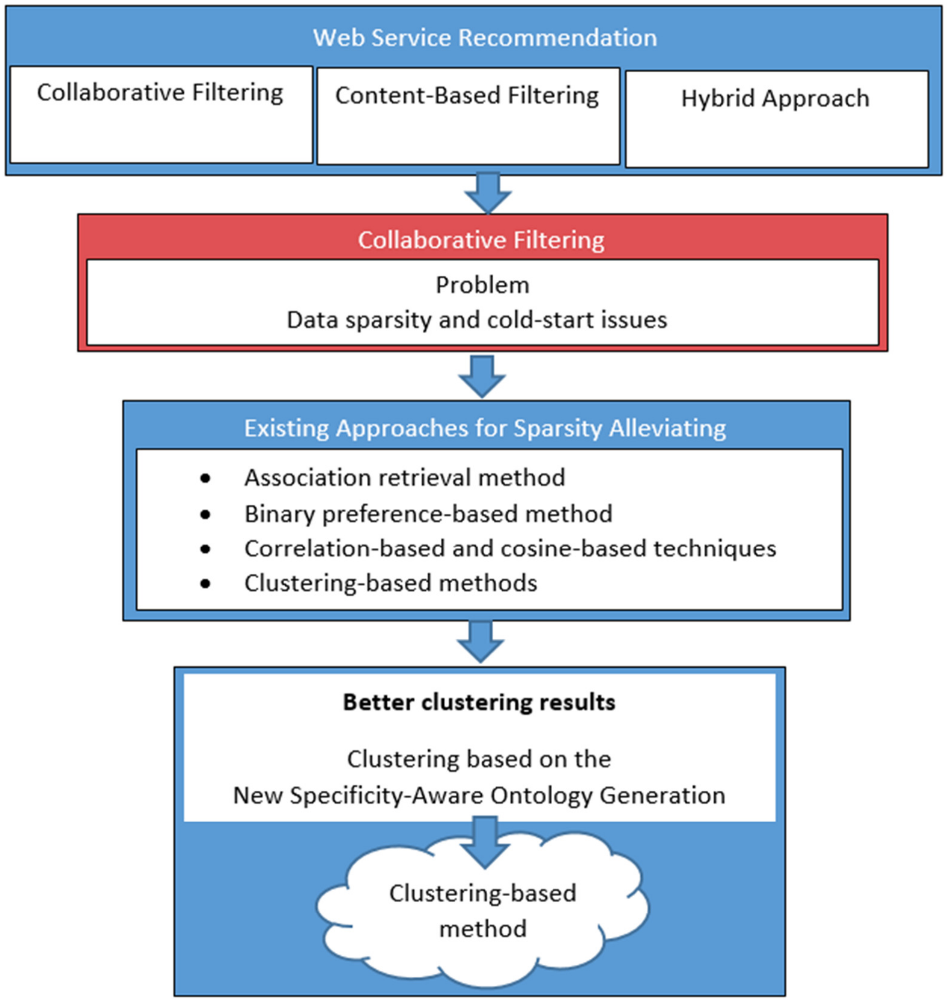

Web Service Recommendation
Collaborative Filtering
Content-Based Filtering
Hybrid Approach
Collaborative Filtering
Problem
Data sparsity and cold-start issues
Existing Approaches for Sparsity Alleviating
Association retrieval method
Binary preference-based method
Correlation-based and cosine-based techniques
Clustering-based methods
Better clustering results
Clustering based on the
New Specificity-Aware Ontology Generation
Clustering-based
method

Figure 2.7 Selecting a sparsity alleviating method

2.6.6.1. The sparsity problem

A major issue that limits the performance of CF is the sparsity problem, which occurs due to the lack of previous user-service feedback data and causes difficulties in identifying similarities between users' preferences.

Some existing approaches have been used to alleviate the sparsity problem. Chen et al. [29] managed the sparsity problem successfully by using association retrieval technology and proposed a new CF algorithm to improve recommendation performance. They examined the transitive associations based on the user's feedback data. They proposed a direct similarity and an indirect similarity between users and computed the similarity matrix through the relative distance between the users' rating. To obtain the recommendation matrix, the association retrieval approach and the direct similarity matrix were combined and the sparsity problem was thereby managed with increasing recommendation precision. Yildirim and Krishnamoorthy [31] proposed a novel item-oriented algorithm, the random walk recommender, which first infers transition probabilities between items based on their similarities, and models finite length random walks on the item space to compute predictions. Huang et al. [31] used a bipartite graph to represent the consumer-product matrix; using the graph, they proposed exploring the global graph structure to facilitate CF under sparse data.

In [68], improved the CF-based Web service recommendation approach by considering social balance theory (SBT), and put forward a novel data-sparsity tolerant recommendation approach. Instead of looking for similar friends of the target user or similar services of the target services in traditional CF-based recommendation approaches, they first looked for the "enemies" of the target user, and further determined the "possible friends" of the target user indirectly based on SBT. Shrivastava and Singh [32] used k-mean clustering, k-medoid clustering, and a combination of the harmonic mean and Euclidean distance method to solve the sparsity problem. They examined the sparsity problem in a movie recommendation system that can recommend movies to a new user as well as to others. Sarwar et al. [34] showed that by reducing the dimensionality of the product space, density can be increased and thereby more ratings can be found. Their approach successfully dealt with the sparsity problem and LSI [69] was used to reduce the dimensionality of the customer–product ratings matrix. Li et al. [35] proposed a Simplified Similarity Measure (SSM) for CF recommendation to handle the sparsity problem. By converting the value of the user-item matrix into a binary preference value (0/1), they found similar groups of users and proposed an SSM for speeding up the process for the sparsity problem. Instead of computing similarities between users, Sarwar et al. [36] proposed a method that used the same correlation-based and cosine-based techniques to compute similarities between items and results used to address both the scalability and sparsity problems. There is a hidden correlation among users and Web services and in [70] defined such hidden correlation with an asymmetric matrix. Their goal was to employ such asymmetric correlation among users and Web services to alleviate the data sparsity problem and further enhance the prediction accuracy in Web service recommendation.

Table 2.1 shows the summary of the existing sparsity alleviating methods.

Table 2.1 Summary of the existing sparsity alleviating methods

Current approaches	**Description**
Association retrieval method	Explored the transitive associations based on the user's feedback data using association retrieval technology.
Binary preference-based method	By converting the value of user-item matrix into binary preference value found the similar group of users and propose a method for sparsity problem.
Correlation-based and cosine-based techniques	Instead of computing similarities between users, the method proposed using the same correlation-based and cosine-based techniques to compute similarities between items and result used to addressed sparsity problem.
Clustering-based methods	Reduce the sparsity by applying the clustering result.

2.6.6.2. The cold-start problem

It is very difficult to make a recommendation to a new user or new item if the profile is empty and there are no available ratings. This is the so-called cold-start problem, which decreases the effectiveness of a recommendation. Wei et al. [26] proposed a hybrid recommendation model to address the cold-start problem. Their model explores the item content features learned from a deep learning neural network and applies them to the time SVD++ CF model. In [71], a novel inverse CF approach is introduced to help alleviate the cold-start problem in Web service recommendation. They first looked for the target user's enemy, and then determine the target user's "possible friends" based on SBT (e.g., "enemy's enemy is a friend" rule). Afterwards, “the Web services preferred by “possible friends” of target user” or “the Web services disliked by enemies of target user” are recommended to the target user, so as to alleviate the cold-start problem.

Ye and Wang [27] proposed a new method to solve the cold-start problem by using worker performance in different types of human intelligence tasks published by different requesters. They also proposed a method to differentiate homogeneous workers, a new similarity to improve the accuracy of predictions, and a novel trust subnetwork extraction approach to tackle the data sparsity and cold-start problems. Barjasteh et al. [72] proposed an algorithmic framework based on matrix factorization that exploits similarity information about users and items to manage the cold-start problem. The proposed method successfully decoupled the completion of unobserved ratings and transduction of knowledge and used that result for the computation.

Chapter 3 Specificity-Aware Ontology Generation for Improving Web Service Clustering

3.1. Motivation for New Clustering Approach

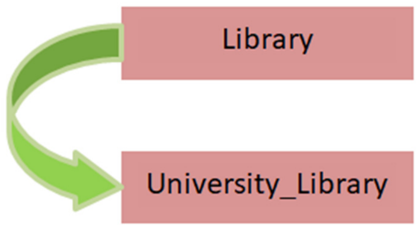
3.1.1. Motivation for Selecting a Ontology-based Clustering Approach

Accurate similarity calculation is the main point for achieving better clustering results. There are some existing approaches to similarity calculation. But, as described in Table 1.1 existing clustering approaches still suffer from several drawbacks.

Figure 3.1 explained the motivation to a new approach with the existing approaches problems. Here HTS [17] shows better results than the existing methods, but they also didn't consider the domain specificity of information when generating the ontology.

Ontology generation is showing better results than other similarity calculation methods. As shown in the following example ontology-based methods can identify the similarity between terms more accurately.

Eg:



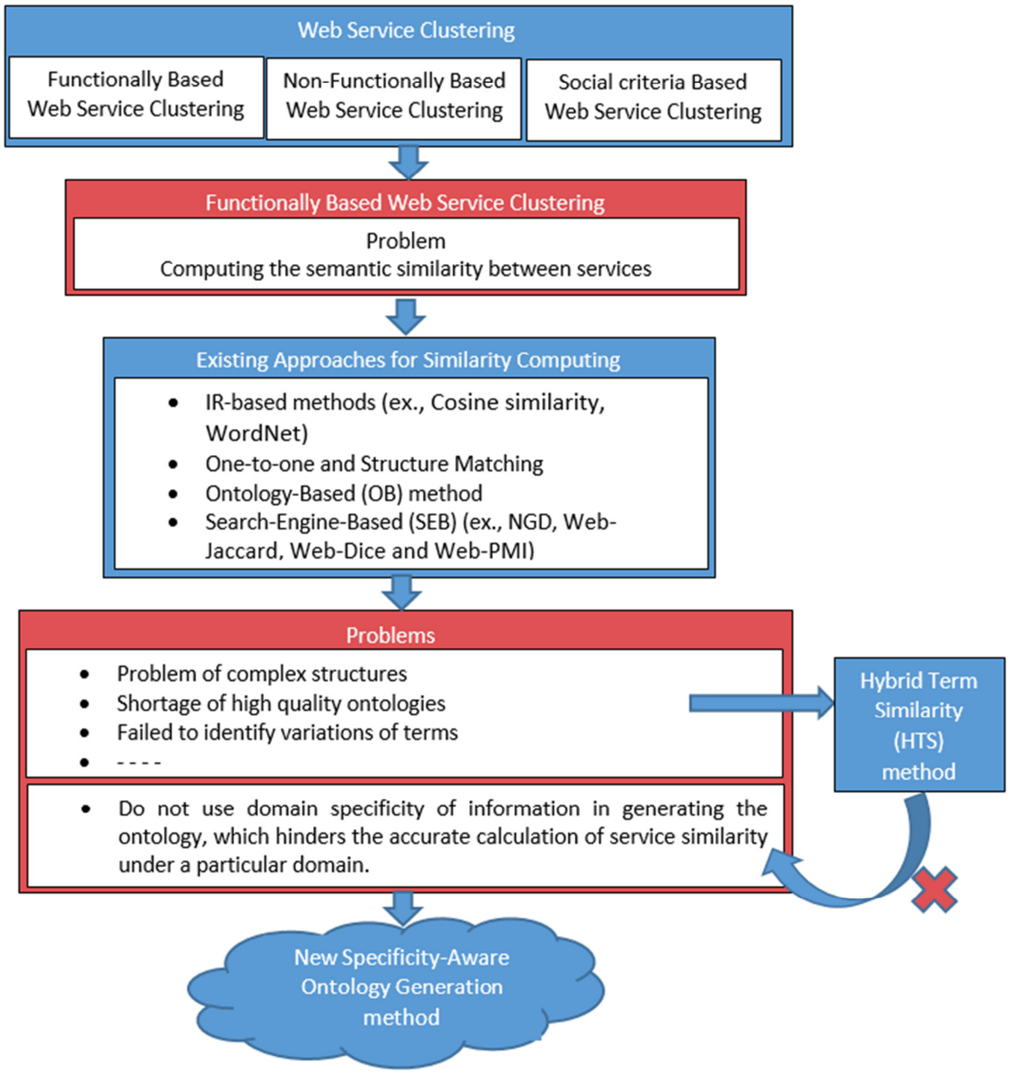
Using other similarity calculation methods,

Library – University_Library → 0.5/0.6

Using ontology generation-based methods,

Library – University_Library → 0.8/0.9

In this example, *Library* and *University_Library* should have a high similarity value like 0.8/0.9 based on the super-sub like term relationship. It proves that ontology-based similarity calculation shows the more accurate results.



Web Service Clustering
Functionally Based Web Service Clustering
Non-Functionally Based Web Service Clustering
Social criteria Based Web Service Clustering
Functionally Based Web Service Clustering
Problem
Computing the semantic similarity between services
Existing Approaches for Similarity Computing
IR-based methods (ex., Cosine similarity, WordNet)
One-to-one and Structure Matching
Ontology-Based (OB) method
Search-Engine-Based (SEB) (ex., NGD, Web-Jaccard, Web-Dice and Web-PMI)
Problems
Problem of complex structures
Shortage of high quality ontologies
Failed to identify variations of terms
- - - -
Do not use domain specificity of information in generating the ontology, which hinders the accurate calculation of service similarity under a particular domain.
Hybrid Term Similarity (HTS) method
New Specificity-Aware Ontology Generation method

Figure 3.1 Motivation to new approach

There have been several approaches for ontology learning/generation such as Formal Concept Analysis method, Lexical Syntactic method, Specificity and Similarity-based method, and Terms Clustering/Dividing based method. In our approach, we have used a combination of lexical-syntactic method, and specificity and similarity-based method. Because they are suitable for applying to service data.

3.1.2. Motivation from Comparing New Apporach with Previous HTS Apporach

Our proposed new approach can give better clustering results than the previous HTS method. Figure 3.2 shows the incorrectly clustered services in the HTS method. But the new approach successfully identified those services. The new approach is considered the specificity of terms and it compares each term using specificity calculations. Because of that, it can identify key terms of every term successfully and improve the clustering performance.

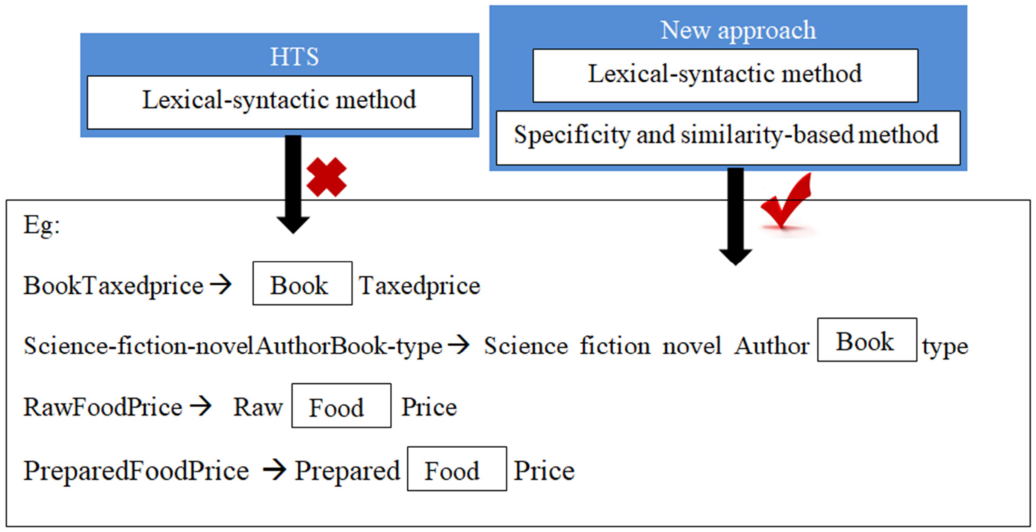
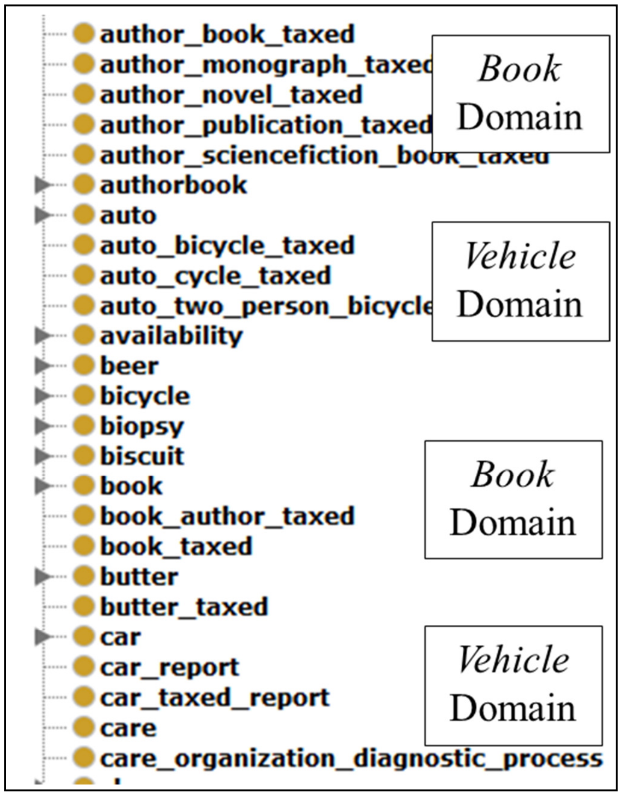

HTS
Lexical-syntactic method
New approach
Lexical-syntactic method
Specificity and similarity-based method
Eg:
BookTaxedprice → Book Taxedprice
Science-fiction-novelAuthorBook-type → Science fiction novel Author Book type
RawFoodPrice → Raw Food Price
PreparedFoodPrice → Prepared Food Price

Figure 3.2 Examples of incorrectly placed cluster groups in the HTS method

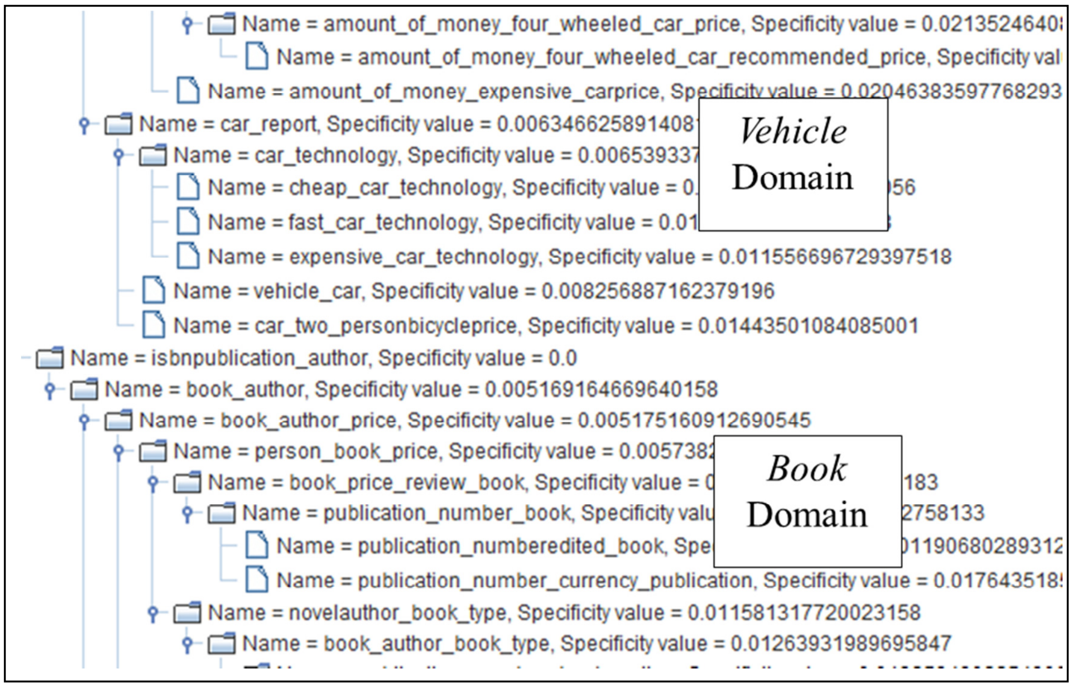
And also new approach is introduced new machine filters (Eg: Near-Descendants, Shared-Ancestor, Far-Descendants) for the similarity calculation. It also helps to identify the correct relationships between terms and improve the clustering performance.

When checking the generated ontology hierarchy as shown in Figure 3.3 and Figure 3.4, in the HTS method ontology nodes spread diffusely over the ontology hierarchy in many places. But in the new method, they have clearly separated automatically into hierarchies according to the domains. This hierarchy overview also helps to improve the clustering performance in the new method.



```
author_book_taxed
author_monograph_taxe
author_novel_taxed
author_publication_taxed
author_sciencefiction_book_taxed
authorbook
auto
auto_bicycle_taxed
auto_cycle_taxed
auto_two_person_bicycle
availability
beer
bicycle
biopsy
biscuit
book
book_author_taxed
book_taxed
butter
butter_taxed
car
car_report
car_taxed_report
care
care_organization_diagnostic_process
```

Figure 3.3 Part of a generated ontology – HTS



```
            Name = amount_of_money_four_wheeled_car_price, Specificity value = 0.0213524640
                Name = amount_of_money_four_wheeled_car_recommended_price, Specificity val
          Name = amount_of_money_expensive_carprice, Specificity value = 0.02046383597768293
      Name = car_report, Specificity value = 0.006346625891408
        Name = car_technology, Specificity value = 0.006539337
          Name = cheap_car_technology, Specificity value = 0.                56
          Name = fast_car_technology, Specificity value = 0.01
          Name = expensive_car_technology, Specificity value = 0.011556696729397518
        Name = vehicle_car, Specificity value = 0.008256887162379196
        Name = car_two_personbicycleprice, Specificity value = 0.01443501084085001
  Name = isbnpublication_author, Specificity value = 0.0
    Name = book_author, Specificity value = 0.005169164669640158
      Name = book_author_price, Specificity value = 0.005175160912690545
        Name = person_book_price, Specificity value = 0.0057382
          Name = book_price_review_book, Specificity value = 0              183
            Name = publication_number_book, Specificity valu              2758133
              Name = publication_numberedited_book, Spe               01190680289312
              Name = publication_number_currency_publication, Specificity value = 0.01764351
          Name = novelauthor_book_type, Specificity value = 0.011581317720023158
            Name = book_author_book_type, Specificity value = 0.01263931989695847
```

Figure 3.4 Part of a generated ontology – New

3.1.3. Motivating Example Involving Domain Specificity

3.1.3.1. Term specificity and term similarity

In this paper, we propose a novel ontology-generation method using hybrid information produced by combining self-information and context-information. In addition, a simple similarity calculation is used for the ontology generation to increase the accuracy of the ontology structure. Figure 3.5 shows two terms, t_1 and t_2, in specificity and similarity relationships.

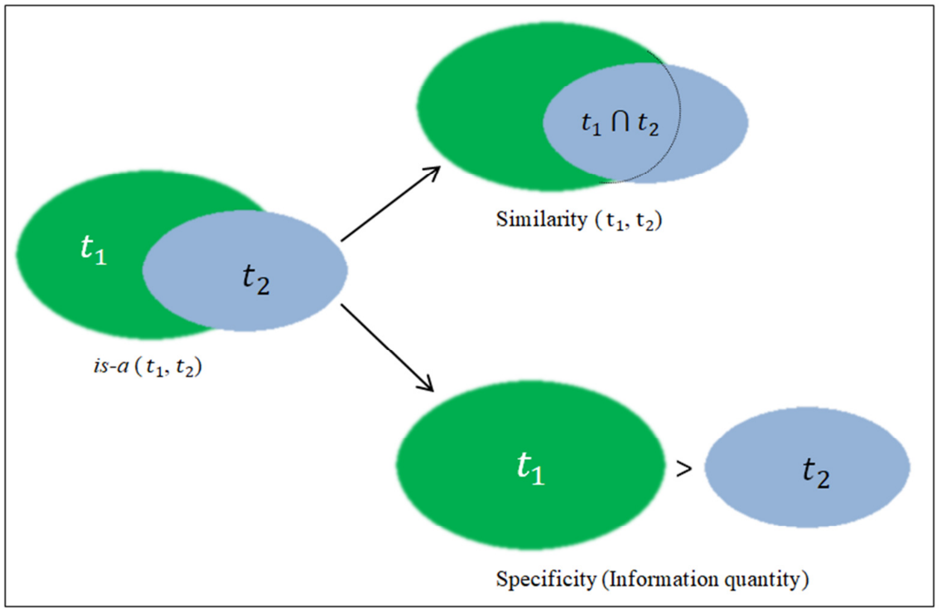


Figure 3.5 Specificity and similarity relationship for two terms t_1 and t_2

Research approaches in the last decade have focused on general terms in generating an ontology and did not take advantage of the domain-specific information contained in the terms. However, the specificity of a term explains the quantity of domain-specific information contained in the term. Specific terms contain a larger quantity of domain-specific information than general terms. The performance of similarity calculations and service clustering can be improved by focusing on the ontology-generating procedure and considering domain-specific information.

Specificity can be measured for each term as its included information quantity. Specificity plays a significant role in generating hierarchical relationships between terms [20], and highly specific terms tend to be located at deeper levels of the hierarchy. That is, if t_1, t_2, t_3, t_4 and t_5 are five terms included in a hierarchy, as shown in Figure 3.6, the terms' specificity can be estimated according to their level in the hierarchy:

$$Spec(t_1) < Spec(t_2) < Spec(t_3) \approx Spec(t_4) \approx Spec(t_5) \tag{3.1}$$

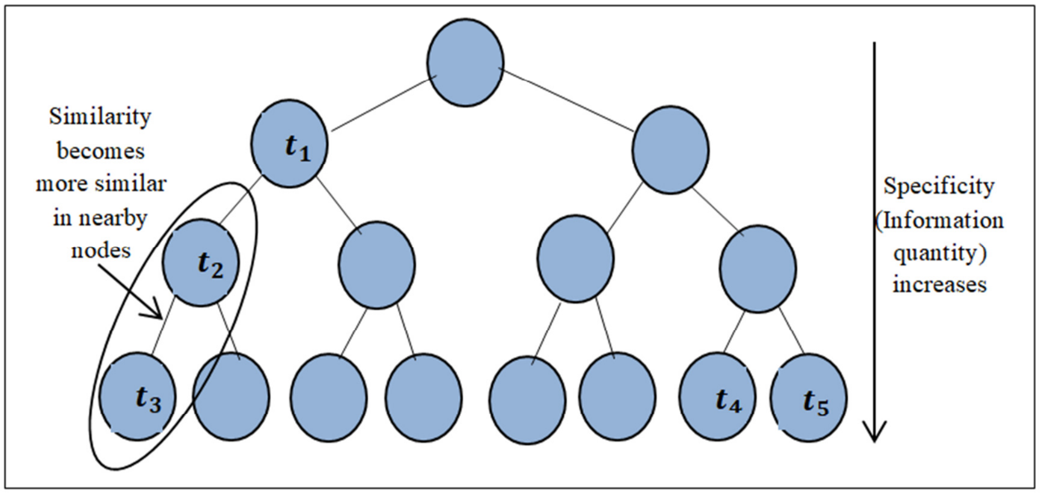

Similarity becomes more similar in nearby nodes
t_1
t_2
t_3
t_4
t_5
Specificity (Information quantity) increases

Figure 3.6 Term specificity and term similarity differences in a hierarchy

As shown in Figure 3.6, if term t_2 is a subclass of term t_1 and term t_3 a subclass of term t_2 in a hierarchy, then the specificity of t_3 is greater than that of t_2 and the specificity of t_2 is greater than that of $t_{1.}$ Based on this, there is a high probability that t_1, t_2 and t_3 have an ancestor relationship, and therefore that t_1, t_2 and t_3 have a semantically similar relationship. However, t_4 and t_5 are not semantically similar to t_1 and t_2 and do not have an ancestor relationship, but, based on the hierarchy levels, the specificity of t_4 and t_5 will be greater than that of t_1 and t_2.

In this paper, we introduce a new term-specificity measuring method by classifying information into two categories, namely self-information and context-information, based on the composition of component words. The final specificity value is measured as a combination of self-information and context-information. Most terms are compound terms with a set of modifiers, enabling self-information to be significant for representing a set of domain characteristics. Context-information helps to cover any shortage of self-information.

3.1.3.2. Self-information

Self-information includes information in compositional words and the internal anatomy of terms. Most domain-specific terms are compound terms, and that helps represent the meaning of terms. The characteristics of each component word and/or the internal anatomy of terms are both useful for measuring term specificity. When measuring self-information, modifiers play a major role in creating new terms by adding modifiers to existing terms.

Using the modifier-head structure, the specificity of the term can be calculated incrementally, starting from the head word. If the term contains more than one word, the specificity of the term is always larger than that of the head term. With this condition, we can assume that a more specific term has higher specificity:

t_1= *NovelAuthor*
t_2= *FictionNovelAuthor*
t_3= *ScienceFictionNovelAuthor*

Consider these three terms t_1, t_2 and t_3 as modifier-head structures. In each term, *Author* represents the head and *Novel*, *Fiction,* and *Science* represent modifiers. t_1 has one modifier, t_2 has two modifiers and t_3 has three modifiers. The compound term t_1 is created by adding the modifier *Novel* to the existing word *Author*. Here, *Author* is considered an ancestor of *Novel.* The meaning of the compound term t_1 can be predicted by using the two compounding words *Novel* and *Author* that describe their unique characteristics. t_2 is created by adding a new modifier *Fiction* to the term t_1 and t_3 is created by adding a new modifier *Science* to the term t_2. In this manner, as the number of modifiers is increased, the compound term can achieve a more specific meaning. Therefore, multiword terms have a higher specificity value than single-word terms. The specificity of the terms t_1, t_2 and t_3 are ordered as follows:

$$Spec(t_1) < Spec(t_2) < Spec(t_3) \quad (3.2)$$

3.1.3.3. Context-information

Some information cannot be accessed by self-information derived from the composition of multiword terms. Some terms have the ability to describe their own characteristics independently without sharing common words, such as:

t_1= *FilmInformation*

t_2= *MovieDetails*

Consider these two terms t_1 and t_2, which do not share any common words. These new terms would have been created independently of existing terms. In this word format, it is ineffective to measure the specificity values using self-information because the compounding words of t_1 and t_2 are completely different from each other. In such cases, assessing compound words independently cannot give the correct information for the compound term.

These limitations can be overcome using context-information that represents the characteristics of the terms indirectly. General terms are usually modified by other words, but domain-specific terms are rarely modified by other words because they already contain sufficient information [20,59]. Using this idea, we can use the probabilistic distribution of modifiers as context-information to measure the specificity of terms [20].

Based on this theory of information and basic similarity calculation, we propose a new method to improve the accuracy of the ontology hierarchy, Web-service similarity calculations and the clustering procedure.

3.2. Proposed New Approach

Figure 3.7 shows clustering example using five domains. Figure 3.8 shows five phases of our new approach and the architecture of the proposed approach is shown Figure 3.9.

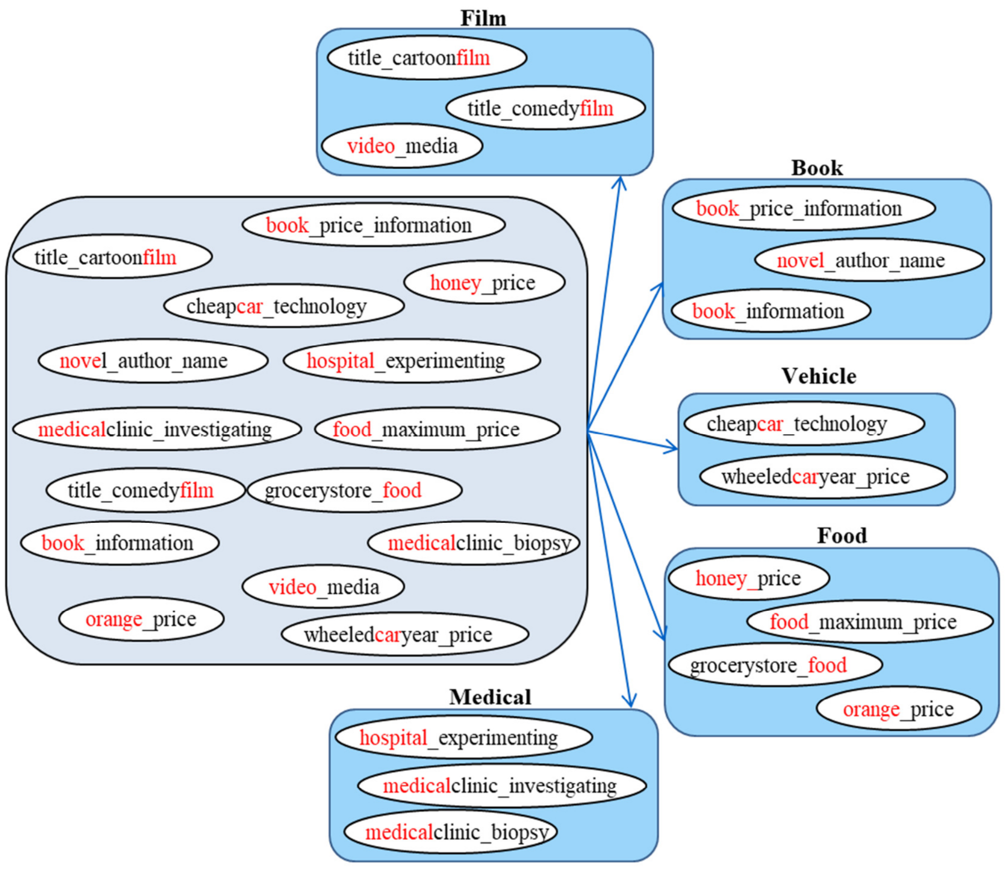


Figure 3.7 Clustering example using five domains

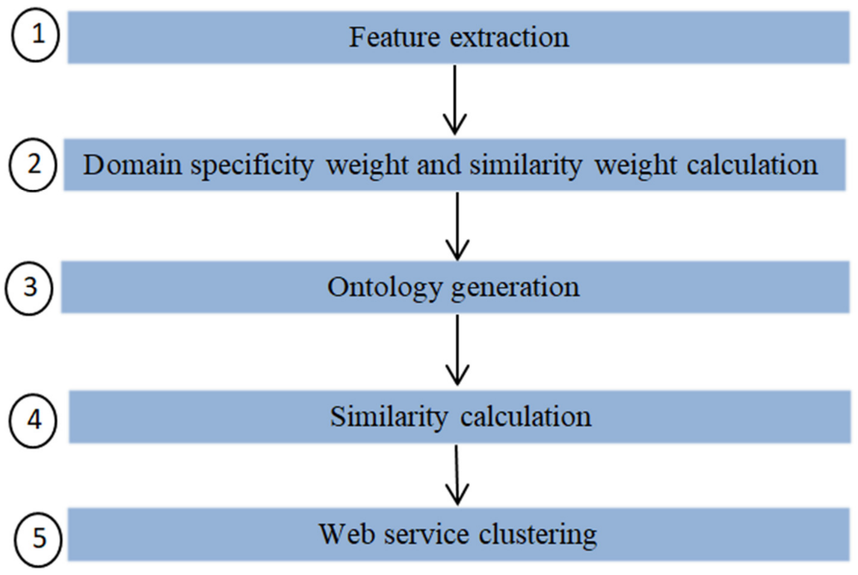


Figure 3.8 The five phases of the new approach

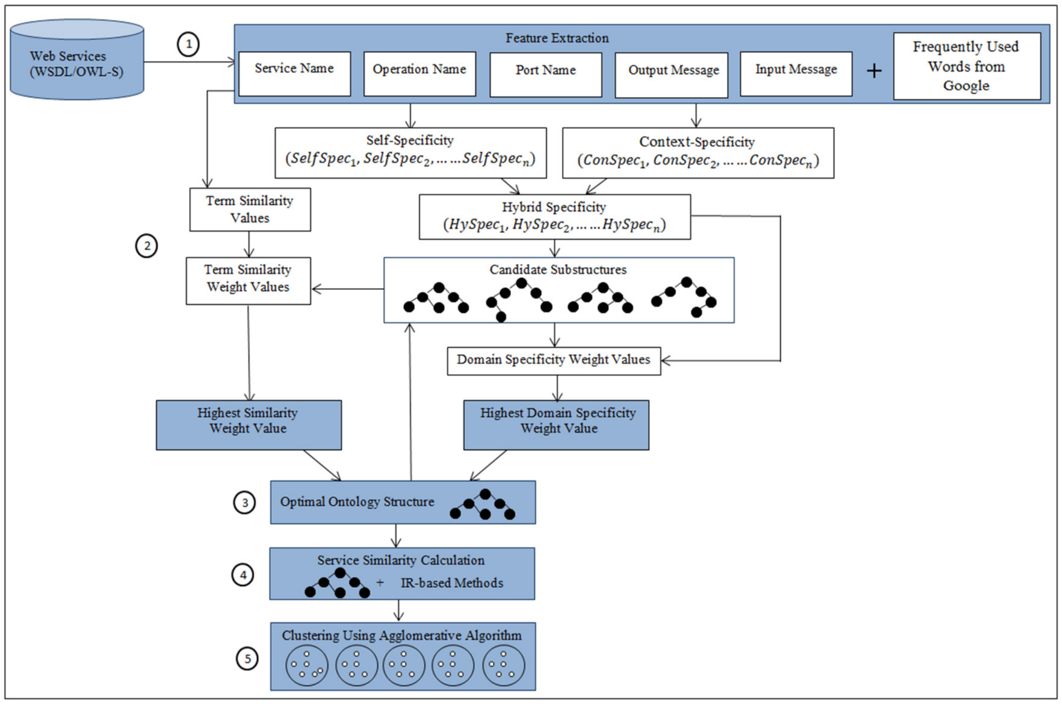

Web Services (WSDL/OWL-S)
1
Feature Extraction
Service Name
Operation Name
Port Name
Output Message
Input Message
+
Frequently Used Words from Google
Self-Specificity ($SelfSpec_1, SelfSpec_2, \ldots SelfSpec_n$)
Context-Specificity ($ConSpec_1, ConSpec_2, \ldots ConSpec_n$)
Term Similarity Values
Hybrid Specificity ($HySpec_1, HySpec_2, \ldots HySpec_n$)
2
Term Similarity Weight Values
Candidate Substructures
Domain Specificity Weight Values
Highest Similarity Weight Value
Highest Domain Specificity Weight Value
3
Optimal Ontology Structure
4
Service Similarity Calculation
+ IR-based Methods
5
Clustering Using Agglomerative Algorithm

Figure 3.9 Architecture of the new approach

3.2.1. Feature extraction

There are many standards for Web service descriptions, including WSDL [13], OWL-S [40], semantic annotations for WSDL [73], XML schema [74] and Web service modeling ontology [41]. We used WSDL documents in extracting features that describe the characteristics included in the Web services. Real world Web service repositories and the OWL-S (http://projects.semwebcentral.org/projects/owls-tc/) test collection were used as the services dataset for the WSDL documents related to five domains, namely *Vehicle*, *Medical*, *Film*, *Food* and *Book*. We extracted five features from WSDL documents, namely *service name* from the WSDL documents' main element <*service*>, *operation name* from <*port type*>, *port name* from <*Port*>, and *input message* and *output message* from <*message*>. These features help to differentiate the characteristics and functionalities of each Web service:

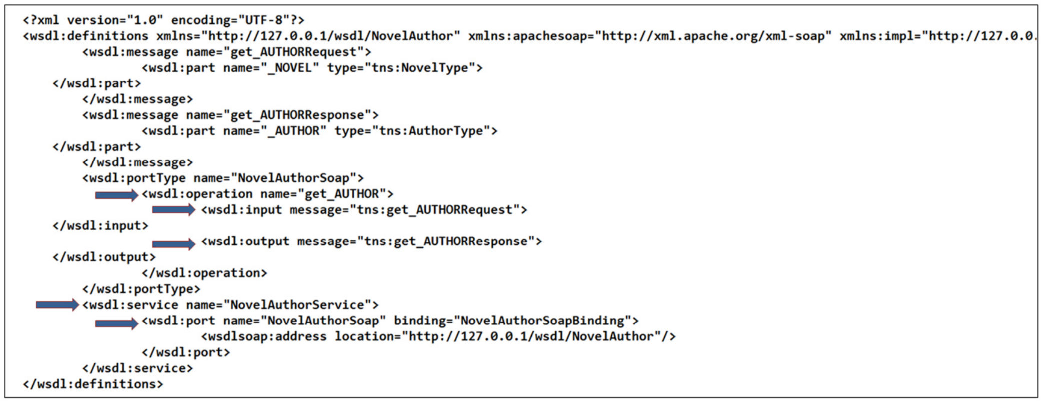
Feature 1: *Service name*. The WSDL file contains a Web-service name extracted as a feature. It provides a unique name among all services using composite names.

Feature 2: *Operation name*. An operation is an abstract description of an action supported by the service and is not required to be unique.

Feature 3: *Port name*. A port defines an individual endpoint by specifying a single address for a binding.

Feature 4: *Input message*. This element in an WSDL document describes the names and format of the messages that are a parameter of the Web services sent to the Web service provider from consumers.

Feature 5: *Output message*. This is similar to an *input message*, but sent from the Web service provider to consumers.



```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns="http://127.0.0.1/wsdl/NovelAuthor" xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://127.0.0.
        <wsdl:message name="get_AUTHORRequest">
                <wsdl:part name="_NOVEL" type="tns:NovelType">
    </wsdl:part>
        </wsdl:message>
        <wsdl:message name="get_AUTHORResponse">
                <wsdl:part name="_AUTHOR" type="tns:AuthorType">
    </wsdl:part>
        </wsdl:message>
        <wsdl:portType name="NovelAuthorSoap">
                <wsdl:operation name="get_AUTHOR">
                        <wsdl:input message="tns:get_AUTHORRequest">
    </wsdl:input>
                        <wsdl:output message="tns:get_AUTHORResponse">
    </wsdl:output>
                </wsdl:operation>
        </wsdl:portType>
        <wsdl:service name="NovelAuthorService">
                <wsdl:port name="NovelAuthorSoap" binding="NovelAuthorSoapBinding">
                        <wsdlsoap:address location="http://127.0.0.1/wsdl/NovelAuthor"/>
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

Figure 3.10 Part of a WSDL file that shows the structure of *NovelAuthorService*

Figure 3.10 shows part of the WSDL file structure for *NovelAuthorService*. Each extracted term was used as an ontology node without splitting into words. For the specificity and similarity calculations, we split each complex term into individual words based on several assumptions. For example, the *NovelAuthor, Novel-Author* and *Novel_Author* terms would be divided into two parts, namely *Novel* and *Author*, based on the assumption that capitalized characters indicate the start of a new word, that a hyphen (-) is used to join two words and that an underscore (_) is also used to join two words. The *Author-of-Novel* term would be divided into three parts, namely *Author, of,* and *Novel*, then stop-word filtering would be performed to remove any stop words such as *of*. This set of words then forms the corpus to be used for the specificity and similarity weight calculations.

The specificity and similarity weights depend on the set of all domain-related words contained in the corpus. The accuracy of the ontology generation can be improved by adding more domain-related terms to the corpus. We added more domain-specific terms by extracting frequently used terms in the particular domains related to *Food, Vehicle, Medical, Book* and *Film* domains, which are the domains we selected for our WSDL documents. We used the domain name as the search query and used Google as the search engine. We obtained 100 snippets from the search engine. The final word set was compiled by computing the Term Frequency-Inverse Document Frequency (TF-IDF) values for all terms in each domain, as expressed in (3.3), and selecting the set of terms with the highest TF–IDF values.

$$TFIDF_{i,s} = tf_{i,s} \text{ X } log\ (\frac{n}{df_{i,s}}) \tag{3.3}$$

Here, $TFIDF_{i,s}$ is the TF-IDF value for term i in snippet s, $tf_{i,s}$ is the term frequency for term i in snippet s, $df_{i,s}$ is the number of snippets that contain term i and n is the total number of snippets. Figure 3.11 and Algorithm 3.1 show the process of extracting frequently used terms.

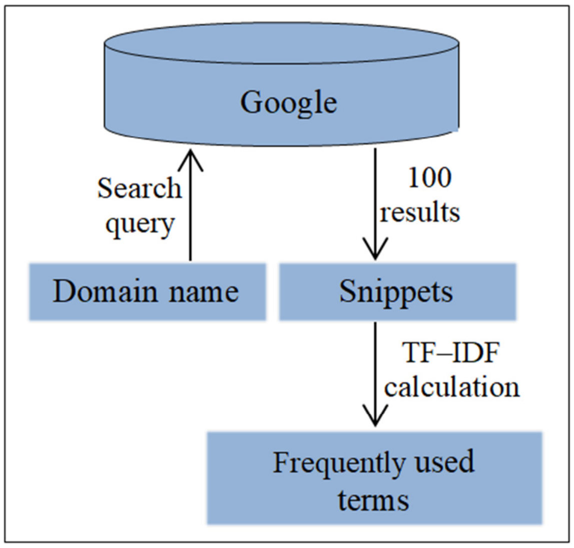


Figure 3.11 Process of extracting frequently used terms

Only service-specific terms relevant to the service domain and the more meaningful terms would be identified using this TF-IDF calculation procedure.

Each term extracted from the WSDL documents and the Google search engine is used in the hybrid specificity calculation and as an ontology node. Five separate ontologies, one for each of the five feature types, were generated by considering domain specificity and similarity weights.

Algorithm 3.1: Extracting Frequently Used Terms

```
Input I : Array of domain names
Output O : Frequently used terms
 1:  For each domain $d_i$ in I do
 2:    S = getSnippets ($d_i$); //get total 100 snippets from Google
       search engine
 3:  end-for
 4:  For each domain $d_i$ in I do
 5:    For each snippets $S_j$ in S do
 6:      StopWord_filtering(S);
 7:    end-for
 8:   $Td_i$ = Calculate_term_frequency();
 9:  end-for
10:  For each domain $d_i$ in I do
11:     For each term $t_x$ in $Td_i$ do
12:       $T_{ix}$= calculate_TF-IDF($t_x$);
13:     end-for
14:  $H_{TI}$ = FrequentlyUsedTerms (); //select terms with highest
     TF-IDF value
15:  end-for
```

3.2.2. Domain-specificity weight and similarity weight calculations for ontology generation

According to the hybrid specificity and the similarity of terms, we calculated the domain-specificity weight and similarity weight, respectively. Figure 3.12 shows the steps of calculating domain specificity weight and similarity weight.

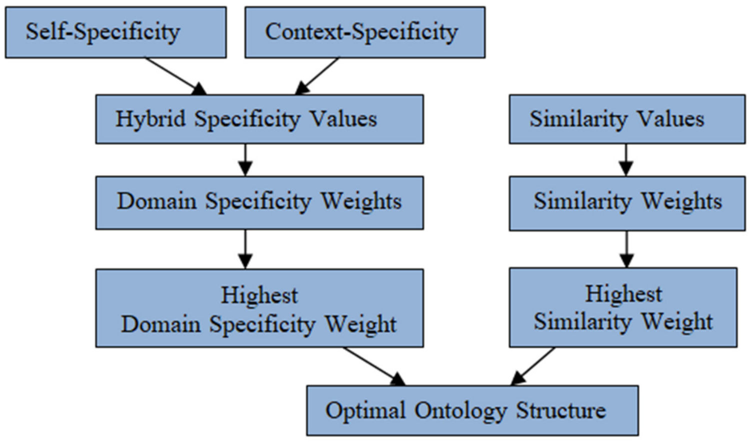

Self-Specificity
Context-Specificity
Hybrid Specificity Values
Similarity Values
Domain Specificity Weights
Similarity Weights
Highest Domain Specificity Weight
Highest Similarity Weight
Optimal Ontology Structure

Figure 3.12 Steps of calculating domain specificity weight and similarity weight

3.2.2.1. Domain-specificity weight for ontology generation

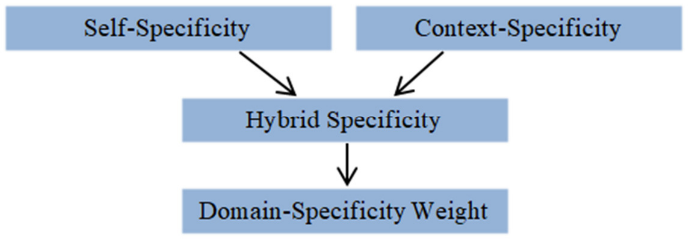


Figure 3.13 Phases of the domain-specificity weight calculation

As shown in Figure 3.13, we first calculate the self-specificity and context-specificity using extracted features of terms from WSDL and extracted terms from the Google search engine. Those values are then combined to evaluate the hybrid specificity, based on information theory from theoretical specificity-measuring methods [20]. Finally, we calculate the domain-specificity weight values from the hybrid specificity values.

Self-specificity value

As described in Section 3.1.3.2, self-specificity is based on a set of compound terms. Figure 3.14 shows an example corpus with frequencies of compound terms and their component words found in the corpus. Each t_i describes a term and a_j describes the individual words contained in each term.

$n_{t_1} = 1$

$n_{t_2} = 2$

$n_{t_3} = 3$

$n_{t_4} = 1$

$n_{t_5} = 1$

$T = \{t_1, t_2, t_3, t_4, t_5\}$

$N_T \rightarrow$ Total number of terms = 8

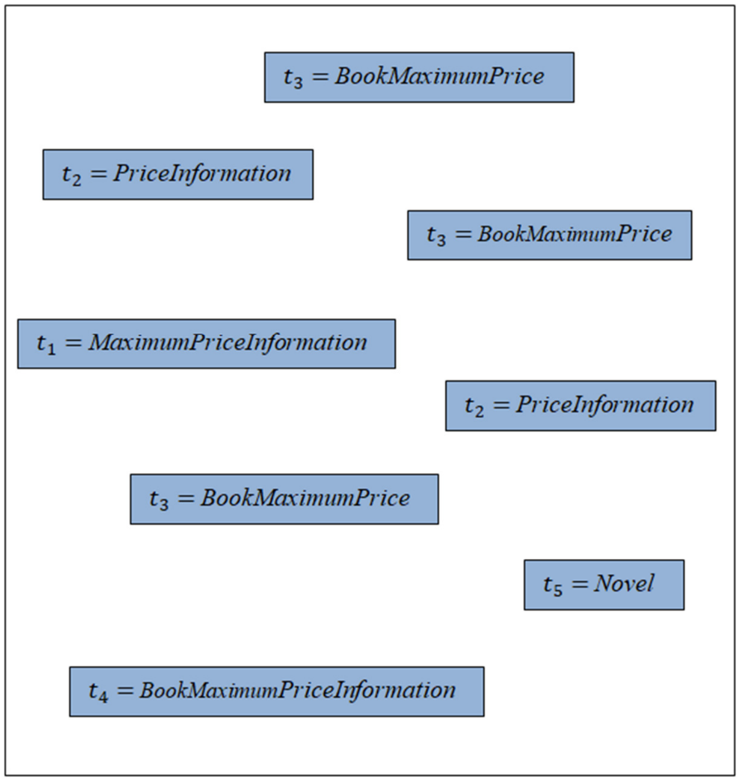


Figure 3.14 Example of a service corpus

$n_{Information} = 4$

$n_{Price} = 7$

$n_{Maximum} = 5$

$n_{Book} = 4$

$n_{Novel} = 1$

$A = \{a_1, a_2, \dots a_h\}$

$\quad = \{Information, Price, Maximum, Book, Novel\}$

N_A → Total number of words = 21

Here, N_T is used to describe the total number of terms in a corpus. $t_i = \{t_1, t_2, \dots\ t_g\}$ is the set of terms found in the corpus and n_{t_i} is used to count each term separately. Terms consist of one or more words. The count of all words in the corpus is given by N_A. Here, $a_j = \{a_1, a_2, \dots a_h\}$, and n_{a_j} is used to count each word separately.

If a term t_i is found in the corpus, the information quantity of the event of t_i is observed by $I(m_i)$ and can be measured via information-theoretic methods [20]. Based on this, the specificity of t_i is assigned as the following (3.4).

$$Spec(t_i) = I(m_i) \tag{3.4}$$

The joint probability distribution $P\{c_i, d_j\}$ is given by $c_i \in C$ and $d_j \in D$. We can assume c_i for selecting a term from T and d_j for selecting a word from A. Events $\{c_1, c_2, \ldots c_g\}$ and $\{d_1, d_2, \ldots d_h\}$ are defined by the random variables C and D. D_{t_i} is a set of d_j that are associated with the words A_{t_i}. The mutual information between c_i and d_j compares the probability of observing c_i and d_j together and independently, as given by (3.5).

$$I(c_i, d_j) = log \frac{P(c_i, d_j)}{P(c_i)P(d_j)} \tag{3.5}$$

The specificity of term t_i is represented by the value $I(c_i, D)$, which indicates the mutual information between c_i and D. $I(c_i, D)$ is estimated using the frequency of terms and words in a corpus according to the following equation (3.6).

$$Spec\ (t_i) \approx I(c_i, D)$$

$$= \sum_{d_j \in D_{t_i}} P(c_i, d_j)\ log \frac{P(c_i, d_j)}{P(c_i)P(d_j)}$$

$$= \sum_{d_j \in D_{t_i}} P(c_i|d_j)P(d_j)\ log \frac{P(c_i|d_j)}{P(c_i)}$$

$$\approx \sum_{a_j \in A_{t_i}} \frac{(n_{a_j}.t_i)}{n_{t_i}} \frac{n_{t_i}}{N_T}\ log \frac{(n_{a_j}.t_i)}{n_{t_i}} \frac{N_A}{n_{a_j}}$$

$$SelfSpec(t_i) \approx \frac{1}{N_T} \sum_{a_j \in A_{t_i}} (\alpha\, .\, log \frac{N_A}{n_{t_i}.n_{a_j}}) \tag{3.6}$$

Extracted terms from WSDL documents contain one or two words, according to the normal format of WSDL. Because of this consideration, $(n_{a_j}.t_i)$, the number of the words in t_i, is assumed to be 1. The weighting scheme for the specificity of the modifier represented by α is based on linguistic knowledge [20]. Adding two or three words of head-modifiers enables the final specificity to be high. Because of this consideration and from experimental results, α=1 is selected from the range 0≤ α≤1. According to the final result (3.6), n_{t_i} and n_{a_j} alone contribute to the self-specificity value because N_T, N_A and α become fixed values. Therefore, the self-specificity depends on the number of compound words that contain the term, the frequency of the term and the frequency of each term containing its words.

Context-specificity value

Context-specificity is based on the entropy of the probabilistic distribution of modifiers for a term [20].

$$E_{mod(t_i)} = -\sum_{1 \leq m \leq F} P(mod_m, t_i)\ log\, P(mod_m, t_i) \tag{3.7}$$

Here, F is the set of modifiers of t_i. The probability that mod_m modifies t_i is given by (mod_m, t_i). The relative frequency of (mod_m, t_i) in all $\left(mod_f, t_i\right)$ pairs in a corpus is estimated for $1 \leq f \leq F$. The entropy value is given by the average information in all (mod_m, t_i) pairs. Because domain-specific terms have simple modifier distributions, specific terms have low entropy. Therefore, the result of (3.7) is converted as an inverse entropy and assigned to $I(m_i)$, as given by (3.8), giving a large quantity of information [20].

$$ConSpec(t_i) \approx I(m_i) \approx \max_{1 \leq j \leq K} E_{mod(t_j)} - E_{mod(t_i)} \tag{3.8}$$

Here, K includes all modifiers with the same head and $E_{mod(t_j)}$ is used for each modifier.

Hybrid-specificity value

The two methods just described are powerful tools for calculating specificity values. Self-specificity helps to cover the component words' characteristics, and context-information addresses areas that cannot be handled by self-information.

We therefore combine the results from (3.6) and (3.8) to form a hybrid specificity as given by (3.9) that takes advantage of both methods:

$$HySpec(t_i) \approx I(m_i) = \frac{1}{\beta\left(\frac{1}{SelfSpec(t_i)}\right) + (1-\beta)\frac{1}{ConSpec(t_i)}} \tag{3.9}$$

Here, β is in the range $0 \leq \beta \leq 1$ and was given the value 0.7 by experimentation. That is, self-specificity makes the major contribution to the final hybrid specificity value. This hybrid method is applied whenever both methods are applicable. Because of the normalization, all results for $SelfSpec(t_i)$, $ConSpec(t_i)$ and $HySpec(t_i)$ were between 0 and 1.

Domain-specificity weight value for ontology generation

The optimal ontology structure is based on the domain-specificity weight, which is calculated using sibling terms. Here we used the theoretical substructure, with sibling terms of similar specificity being assigned a higher score.

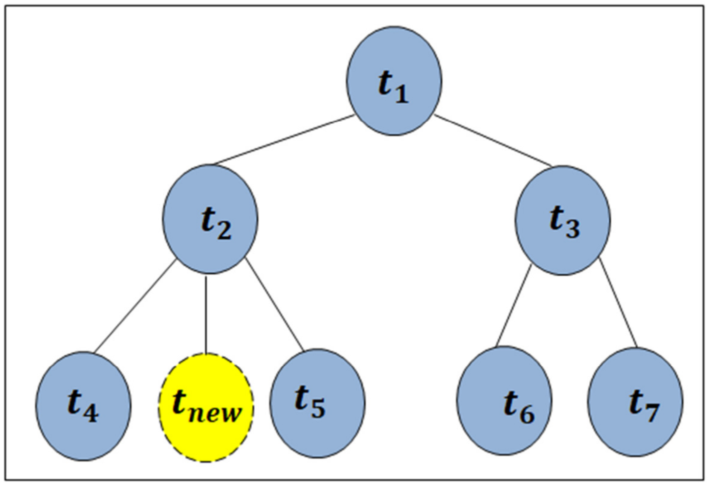


Figure 3.15 Ontology example for sibling terms

In the example of Figure 3.15,
$Spec(t_{new}) \approx Spec(t_4)$ and $Spec(t_{new}) \approx Spec(t_s)$

The domain-specificity weight of a substructure is calculated using (3.10), as follows:

$$W_{Spec}(H_i) = \begin{cases} 1 - \frac{\sum_{t_s \in F_{t_s}} |Spec(t_s) - Spec(t_{new})|}{|F_{t_s}|} \\ 0.5 \end{cases} \tag{3.10}$$

Here, F_{t_s} is the number of sibling terms of the new term t_{new} and t_s is the set of siblings in H_i. $Spec(t_s)$ describes the specificity of each sibling term and $Spec(t_{new})$ describes the specificity of new term. If F_{t_s} is empty, $W_{Spec}(H_i)$ is directly assigned as 0.5 from experimentation.

3.2.2.2. Term similarity weight for ontology generation

When generating the ontology, finding an optimal structure is also based on the similarity weight. As shown in Figure 3.16, we first calculate the similarity value using the simple similarity calculation method and its result is then used for the similarity weight calculation.

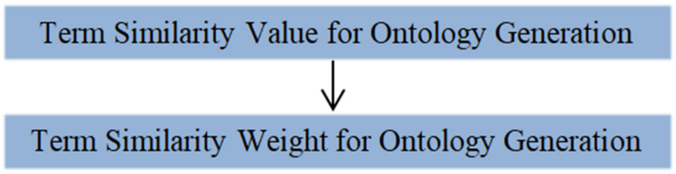

Term Similarity Value for Ontology Generation
Term Similarity Weight for Ontology Generation

Figure 3.16 Phases of similarity weight calculation

Term similarity value for ontology generation

Here, we use the basic similarity calculation procedure by comparing the words common to two terms.

$$Sim(t_x, t_{new}) = \frac{2\ F_{(t_x,t_{new})}}{|F_{t_x}| + |F_{t_{new}}|} \qquad (3.11)$$

For example,

Sim(*Fiction Novel Author*, *Novel Author*) = 0.8

Sim(*Fiction Novel Author*, *Author*) = 0.5

Here, $F_{(t_x,t_{new})}$ is the number of common words of t_x and t_{new}. F_{t_x} and $F_{t_{new}}$ describe the number of compositional words in each term. This result is used to calculate the similarity weight values, as given by (3.12) below.

<u>Term similarity weight value for ontology generation</u>

The optimal ontology structure is selected through similarity weight, which is based on the parent, child and sibling terms. Here, we assume that a substructure with more similar terms clustered around the new term will have a higher score. We use all the terms connected to the new term, which are parent, child and sibling terms, as more-similar terms.

Figure 3.17 shows two substructures H_1 and H_2 with an added t_{new}. Here, $H_1(Spec_{t_{new}}) > H_2(Spec_{t_{new}})$ because $H_1(t_{new})$ contains parent, sibling and child nodes t_2, t_4 and t_5, respectively, whereas $H_2(t_{new})$ contains t_3 and t_6 (parent and sibling) only. Therefore, there are more similar terms clustered around $H_1(t_{new})$ and $H_1(Spec_{t_{new}})$ is therefore higher than $H_2(Spec_{t_{new}})$.

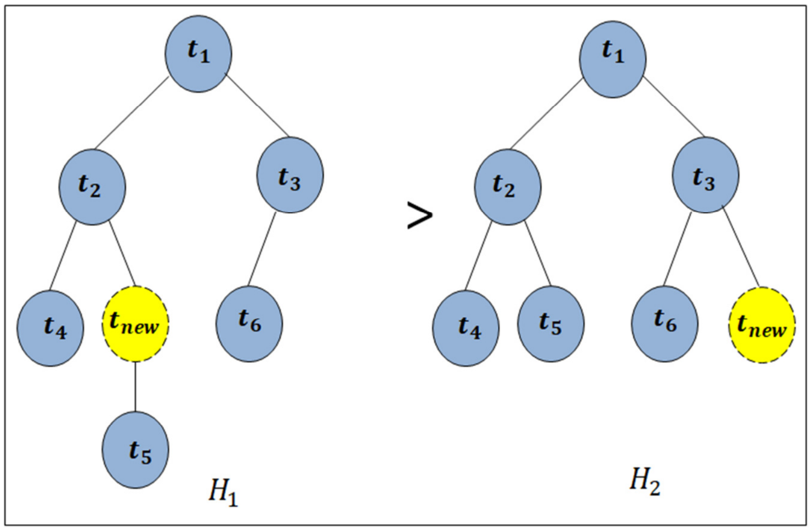


Figure 3.17 Ontology example for similar terms clustered around a new term

The similarity weight value is calculated by using the results from (3.11) to produce (3.12):

$$W_{Sim}(H_i) = \frac{\sum_{t_x \in K} |Sim(t_x, t_{new})|}{|K| - 1} \tag{3.12}$$

Here, K is the total number of parent, sibling and child terms. t_x is the set of all terms, with $t_x \in K$.

3.2.3. Ontology generation

An ontology is a formal specification of a shared conceptualization of a domain of interest. Ontology learning is a subtask of information extraction.

Here, each extracted term from the WSDL documents becomes a node in the ontology hierarchy. The ontology-generating procedure considers each term's relations in the ontology and depends on the calculated specificity weight (3.10) and similarity weight (3.12). This is a top-down approach that builds the hierarchy by starting from the top root node and adding other nodes one by one to the current hierarchy, as shown in the example of Figure 3.18. We cannot guarantee the exact order for incorporating new nodes. In principle, it can be any node in the current ontology structure.

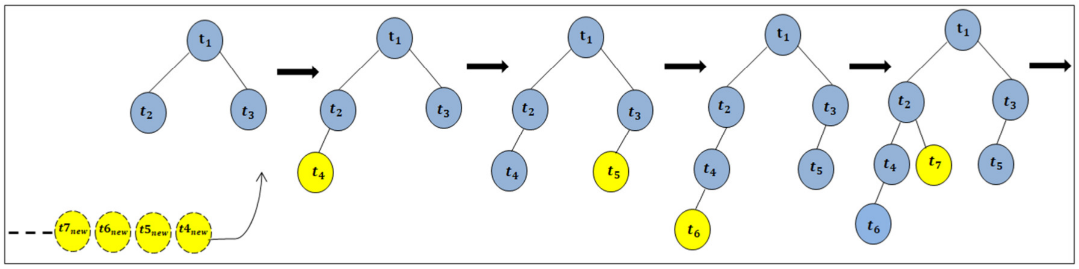


Figure 3.18 Example of the steps in ontology generation

Figure 3.19 outlines the flow of incremental taxonomy construction, which involves three elements, namely subsumption information, contextual information and optimal structure selection:

- Subsumption information (see Figure 3.5)
  - Similarity: conceptual overlapping
  - Specificity: domain-specific information in terms
- Contextual information
  - Similarity weight: semantically similar terms close together in the taxonomy
  - Specificity weight: sibling terms have similar specificity levels

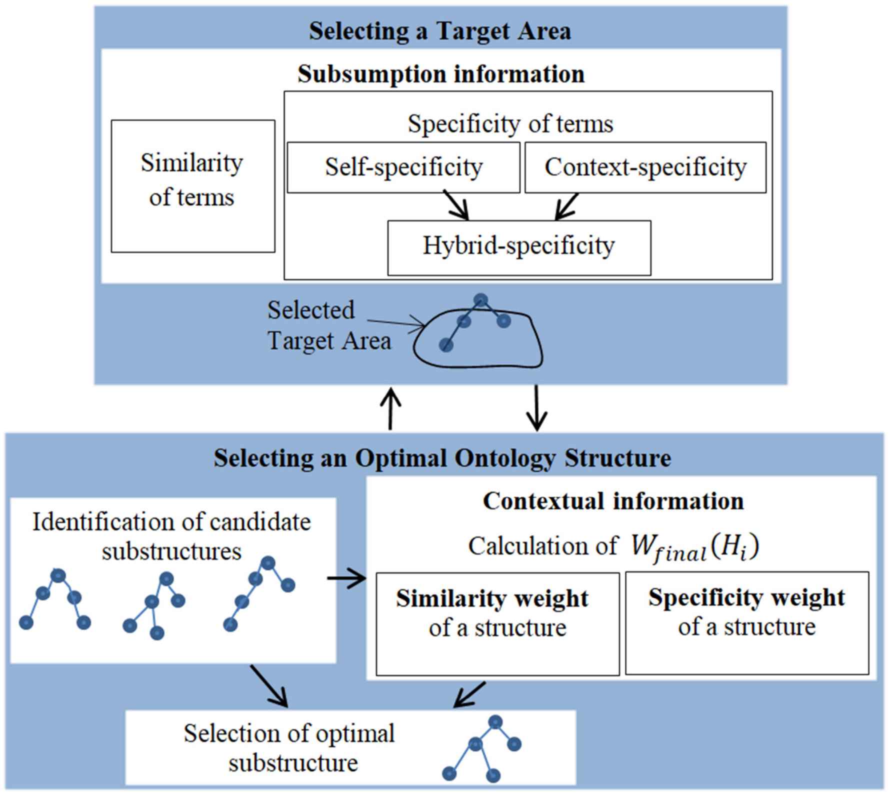

Selecting a Target Area
Subsumption information
Similarity of terms
Specificity of terms
Self-specificity
Context-specificity
Hybrid-specificity
Selected Target Area
Selecting an Optimal Ontology Structure
Identification of candidate substructures
Contextual information
Calculation of $W_{final}(H_i)$
Similarity weight of a structure
Specificity weight of a structure
Selection of optimal substructure

Figure 3.19 Flow of incremental taxonomy construction

Incremental insertion of a new term t_{new} into the taxonomy is achieved by the following steps:

Step 1. The calculated hybrid specificity values are arranged in ascending order and the first three values are selected as the first three nodes for starting the ontology generation procedure.

Step 2. The terms are then added to the ontology hierarchy in ascending order. Target substructures for a t_{new} to be combined as a new term, as shown in Figure 3.20, are selected based on the hybrid specificity values of the new node and the existing nodes of the ontology. We select target nodes that satisfy $Spec(t_{new}) - 0.3 < Spec(t_x) < Spec(t_{new}) + 0.3$, where t_x is an existing node of the ontology.

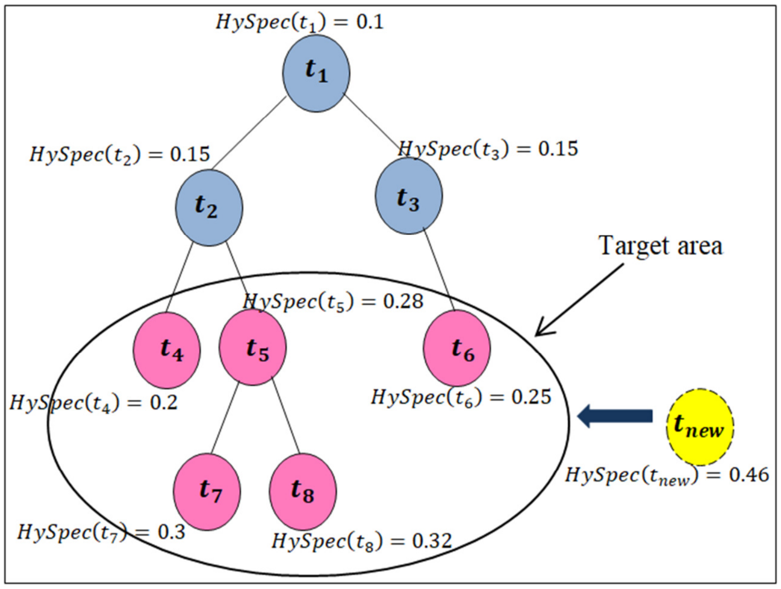


Figure 3.20 Target area of the existing ontology

Step 3. With this set of target nodes, we can identify a set of candidate substructures, as shown in Figure 3.21.

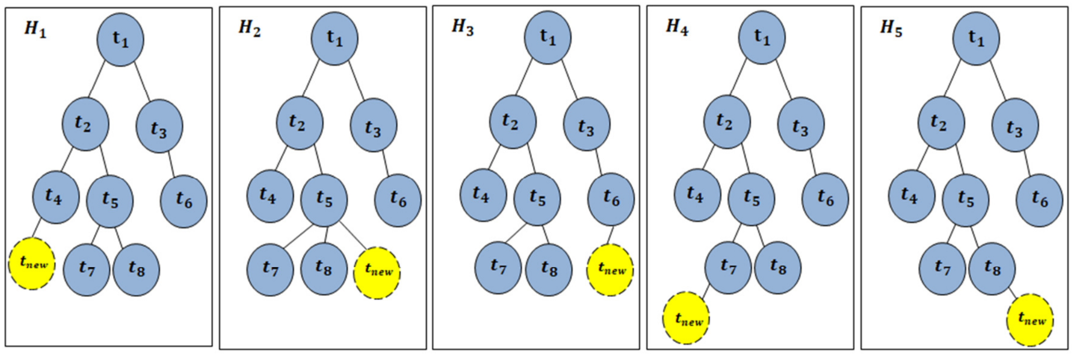


Figure 3.21 Candidate target-area substructures for the example of Figure 3.20

Step 4. The optimal substructure is found by calculating the specificity weight (3.10) and similarity weight (3.12) for each candidate substructure associated with the t_{new} and finding the maximum $W_{final}(H_i)$ by combining them:

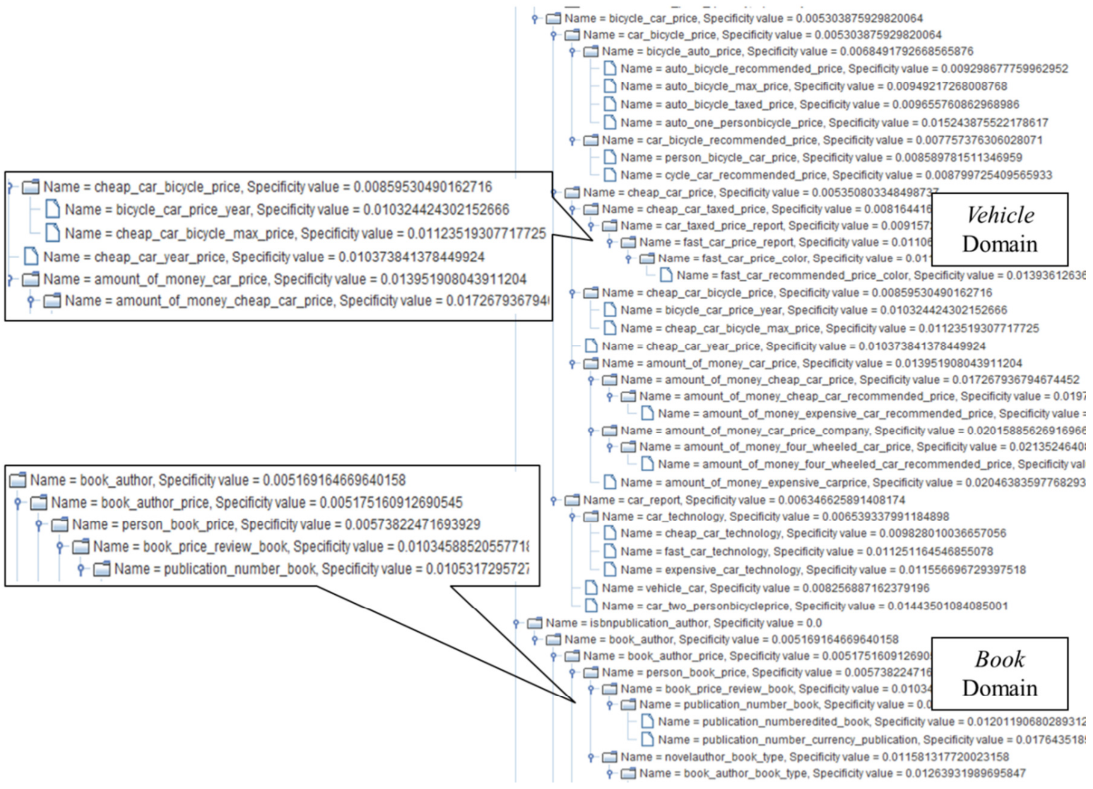
$$H_O = \arg \max_{H_i \in H_A} W(H_i | H_T, t_{new}) \tag{3.13}$$

The optimal structure H_O is selected by comparing all H_A candidate substructures and finding the maximum weight value. Equation (3.14) gives the final calculation by which we select the optimal ontology structure that has the highest $W_{final}(H_i)$ value.

$$W_{final}(H_i) = \Upsilon . W_{Spec}(H_i) + \ (1 - \Upsilon) . W_{Sim}(H_i) \tag{3.14}$$

Here, Υ was assigned as 0.4 through experimentation. This procedure iterates until the ontology hierarchy is complete, with all terms added in the prepared order.

Figure 3.22 shows a screenshot of part of a generated ontology that contains terms from the *vehicle* and *book* domains. They are separated automatically into hierarchies according to the domains. Algorithm 3.2 describes the ontology-generation procedure.



```
Name = bicycle_car_price, Specificity value = 0.005303875929820064
  Name = car_bicycle_price, Specificity value = 0.005303875929820064
    Name = bicycle_auto_price, Specificity value = 0.0068491792668565876
      Name = auto_bicycle_recommended_price, Specificity value = 0.009298677759962952
      Name = auto_bicycle_max_price, Specificity value = 0.00949217268008768
      Name = auto_bicycle_taxed_price, Specificity value = 0.009655760862968986
      Name = auto_one_personbicycle_price, Specificity value = 0.015243875522178617
    Name = car_bicycle_recommended_price, Specificity value = 0.007757376306028071
      Name = person_bicycle_car_price, Specificity value = 0.008589781511346959
      Name = cycle_car_recommended_price, Specificity value = 0.008799725409565933
  Name = cheap_car_price, Specificity value = 0.005350803348498737
    Name = cheap_car_taxed_price, Specificity value = 0.008164416
      Name = car_taxed_price_report, Specificity value = 0.009157
        Name = fast_car_price_report, Specificity value = 0.01106
          Name = fast_car_price_color, Specificity value = 0.01
            Name = fast_car_recommended_price_color, Specificity value = 0.01393612636
    Name = cheap_car_bicycle_price, Specificity value = 0.00859530490162716
      Name = bicycle_car_price_year, Specificity value = 0.010324424302152666
      Name = cheap_car_bicycle_max_price, Specificity value = 0.01123519307717725
    Name = cheap_car_year_price, Specificity value = 0.010373841378449924
    Name = amount_of_money_car_price, Specificity value = 0.013951908043911204
      Name = amount_of_money_cheap_car_price, Specificity value = 0.017267936794674452
        Name = amount_of_money_cheap_car_recommended_price, Specificity value = 0.0197
          Name = amount_of_money_expensive_car_recommended_price, Specificity value =
      Name = amount_of_money_car_price_company, Specificity value = 0.02015885626916966
        Name = amount_of_money_four_wheeled_car_price, Specificity value = 0.02135246408
          Name = amount_of_money_four_wheeled_car_recommended_price, Specificity val
      Name = amount_of_money_expensive_carprice, Specificity value = 0.02046383597768293
  Name = car_report, Specificity value = 0.006346625891408174
    Name = car_technology, Specificity value = 0.006539337991184898
      Name = cheap_car_technology, Specificity value = 0.009828010036657056
      Name = fast_car_technology, Specificity value = 0.011251164546855078
      Name = expensive_car_technology, Specificity value = 0.011556696729397518
    Name = vehicle_car, Specificity value = 0.008256887162379196
    Name = car_two_personbicycleprice, Specificity value = 0.01443501084085001
Name = isbnpublication_author, Specificity value = 0.0
  Name = book_author, Specificity value = 0.005169164669640158
    Name = book_author_price, Specificity value = 0.005175160912690
      Name = person_book_price, Specificity value = 0.005738224716
        Name = book_price_review_book, Specificity value = 0.01034
          Name = publication_number_book, Specificity value = 0.0
            Name = publication_numberedited_book, Specificity value = 0.01201190680289312
            Name = publication_number_currency_publication, Specificity value = 0.017643518
        Name = novelauthor_book_type, Specificity value = 0.011581317720023158
          Name = book_author_book_type, Specificity value = 0.01263931989695847
```

```
Name = cheap_car_bicycle_price, Specificity value = 0.00859530490162716
  Name = bicycle_car_price_year, Specificity value = 0.010324424302152666
  Name = cheap_car_bicycle_max_price, Specificity value = 0.01123519307717725
Name = cheap_car_year_price, Specificity value = 0.010373841378449924
Name = amount_of_money_car_price, Specificity value = 0.013951908043911204
  Name = amount_of_money_cheap_car_price, Specificity value = 0.017267936794
```

```
Name = book_author, Specificity value = 0.005169164669640158
  Name = book_author_price, Specificity value = 0.005175160912690545
    Name = person_book_price, Specificity value = 0.00573822471693929
      Name = book_price_review_book, Specificity value = 0.01034588520557718
        Name = publication_number_book, Specificity value = 0.0105317295727
```

Figure 3.22 Screenshot of part of a generated ontology

Algorithm 3.2: Ontology Generation

```
Input $I_1$ =: Array of WSDL extracted terms
Input $I_2$ =: Array of Google search engine extracted terms
Output $O_s$ : Ontology
 1:  For $I_1$ and $I_2$ do
 2:      I = Array of ($I_1$+ $I_2$) ascending order; //create an array using all terms
     according to the hybrid specificity
 3:  end-for
 4:  $I_s$ = Select 1st three nodes from I;
 5:  Start ontology $O_s$ with $I_s$;
 6:  For ontology $O_s$ do
```

```
7:  For each remaining term $t_{new}$ in I do
8:      if (Spec($t_{new}$)–0.3 < Spec(existing nodes of ontology ($T_n$)) < Spec($t_{new}$)+0.3)
9:          select target nodes as $T_n$; //target area nodes will be selected
10:     end
11:     For each $T_n$ in $O_s$ do
12:         add $t_{new} \longrightarrow H_i$ = candidate substructures; // set of candidate substructures will
    be generated
13:         $W_{Spec}(H_i)$ = Calculate domain-specificity weight;
14:         $W_{Sim}(H_i)$ = Calculate similarity weight;
15:         $W_{final}(H_i)$= Find final maximum weight;
16:         $O_s$ = Substructure of maximum weight (from $W_{final}(H_i)$); //select optimal
    substructure
17:     end-for
18: end-for
19: end-for
```

3.2.4. Service-similarity calculation in an ontology

Similarity calculation is achieved by comparing generated ontology term relationships by calculation.

Service similarity calculation steps:

Step 1. To measure the similarity of two Web services, we first take the relevant extracted features from the two WSDL files.

Step 2. We then match these two terms via the generated ontology without splitting the words.

Step 3. There are two possibilities: they can be identical or they can differ. If two terms match exactly, we assign the term to the *extract* filter (see below) and give it the highest similarity value, i.e., 1. If there is no exact match in the ontology, we define a procedure for similarity calculation involving six different filters, weighted toward different relationships. If we find that the two extracted terms in our generated ontology satisfy one of our defined filters, then a similarity calculation is performed using the appropriate equation.

Step 4. If the two extracted terms do not satisfy any of the defined filters, then we use an IR-based method to calculate the similarity.

3.2.4.1. Ontology-based similarity calculation

We now describe the machine filters we used in the similarity calculations. Three of the filters were used in previous research work [17,18] and there are four new filters in our proposed approach. The seven machine filters are called *extract*, *siblings, parent–child, near-descendants, shared-ancestor, far-descendants* and *fail*. They are used to compute the degree of semantic similarity for a pair of services.

Figure 3.23 shows an example ontology that contains nine terms describing nine WSDL files for one feature among five features such as *service name*, *operation name,* etc. The functions of the seven filters, together with examples from Figure 3.23. are:

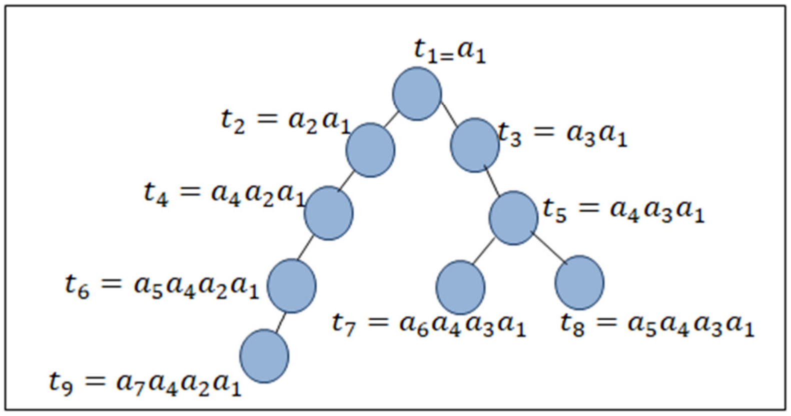


Figure 3.23 Example ontology hierarchy

Filter 1: *Exact.* If term t_i and term t_j are the same and represent the same feature, then the services exactly match.

Example: $t_2 = a_2a_1$ and $t_2 = a_2a_1$

Filter 2: *Siblings.* Term t_i and term t_j plug into term t_p with $t_i \in$ *DirectChildren*(t_p) and $t_j \in$ *DirectChildren*(t_p).

Example: $t_2 = a_2a_1$ and $t_3 = a_3a_1$

Filter 3: *Parent–Child.* Term t_i plugs into term t_j with $t_i \in$ *DirectChildren*($\mathrm{t_j}$).

Example: $t_2 = a_2a_1$ and $t_1 = a_1$

Filter 4: *Near-Descendants.* Term t_c plugs into term t_i and term t_j plugs into term t_c with t_c ∊ *DirectChildren*($\mathrm{t_i}$) and t_j ∊ *DirectChildren*($\mathrm{t_c}$).

Example: $t_2 = a_2 a_1$ and $t_6 = a_5 a_4 a_2 a_1$

Filter 5: *Shared-Ancestor.* Term t_i and term t_j plug into a child term of term t_q with t_q ∊ *Ancestor*(t_i) and t_q ∊ *Ancestor*(t_j).

Example: $t_4 = a_4 a_2 a_1$ and $t_5 = a_4 a_3 a_1$

Filter 6: *Far-Descendants.* Term t_i and term t_j have a far-descendants relationship with t_j ∊ *FarAncestor*(t_i).

Example: $t_1 = a_1$ and $t_8 = a_5 a_4 a_3 a_1$

Filter 7: *Fail.* If none of the other filters generates a match, then there is a fail.

To investigate the strength of the defined filters, we conducted experiments and assigned weight values for each filter in the following order, based on the strength in logic-based matching: *Exact* > *Siblings* > *Parent–Child* > *Near-Descendants* > *Shared-Ancestor* > *Far-Descendants* > *Fail*.

If the filter is an exact match, then the similarity is assigned the highest value of 1. The remaining filters and their weight assignments are listed in Table 3.1.

Table 3.1 Assigned matching filters and weights

Matching filter	**Weight**	
	$\boldsymbol{W_1}$	$\boldsymbol{W_2}$
Extract	Similarity = 1	
Siblings	0.9	0.1
Parent–Child	0.8	0.2
Near-Descendants	0.78	0.22
Shared-Ancestor	0.65	0.35
Far-Descendants	0.62	0.38
Fail	Used IR-based methods	

If the matching filter is a one of *Siblings, Parent–Child, Near-Descendants, Shared-Ancestor* or *Far-Descendants*, then we use equations (3.15) and (3.16) below with the relevant weight values to calculate the similarity:

$$S_{Onto}(t_1, t_2) = -log\frac{d(t_1, t_2)}{2D} \quad (3.15)$$

This calculation is an edge-count-based method. Here, $d(t_1, t_2)$ describes the shortest distance between the two terms t_1 and t_2, and D describes the maximum depth of the generated ontology.

$$Sim_F(t_1, t_2) = W_1 + W_2\, S_{Onto}(t_1, t_2) \quad (3.16)$$

Here, $S_{Onto}(t_1, t_2)$ is assigned using the results from (3.15) and W_1 and W_2 are assigned according to the relevant filter values in Table 3.1. For normalization purposes in the similarity values, we select values that satisfy $W_1 + W_2$=1. Further, the final similarity value is between 0 and 1.

3.2.4.2. IR-based similarity calculation

If the two selected terms do not match any of the above five relationships or exhibit exact matching, then we calculate similarity using IR-based methods such as thesaurus-based term similarity or SEB term similarity, as described below.

Thesaurus-based term similarity

Our thesaurus-based term similarity calculation uses WordNet as the knowledge base. It helps to cover any failed ontology relationships by providing a large lexical database for expressing distinct concepts and synonym rings that are interlinked by means of conceptual-semantic and lexical relations [75]. This method can be considered as a knowledge-rich similarity-measuring technique, which requires a semantic network or a semantically tagged corpus.

SEB term similarity

Some terms used in Web services may not be included in a thesaurus. For example, “*IphoneInformation*” and “*SamsungInformation*” are absent from WordNet. Furthermore, some latent semantics of terms fail to be identified by WordNet, such as “*Apple*” and “*Computer.*” SEB term similarity is used to cover new technological and Internet-related data that are omitted from WordNet. We use three algorithms, namely Web-Jaccard (3.17), Web-Dice (3.18) and Web-PMI (3.19) for the calculations [76].

$$Web-Jaccard(t_1,t_2) = \begin{cases} 0, & \text{if } (H(t_1 \cap t_2)) \leq c \\ \frac{H(t_1 \cap t_2)}{H(t_1)+H(t_2)-H(t_1 \cap t_2)}, & otherwise \end{cases} \quad (3.17)$$

$$Web-Dice(t_1,t_2) = \begin{cases} 0, & \text{if } (H(t_1 \cap t_2)) \leq c \\ \frac{2H(t_1 \cap t_2)}{H(t_1)+H(t_2)}, & otherwise \end{cases} \quad (3.18)$$

$$Web-PMI(t_1,t_2) = \begin{cases} 0, & \text{if } (H(t_1 \cap t_2)) \leq c \\ log_2\left[\frac{\frac{H(t_1 \cap t_2)}{N}}{\frac{H(t_1)}{N}+\frac{H(t_2)}{N}}\right], & otherwise \end{cases} \quad (3.19)$$

Here, $H(t_1)$ and $H(t_2)$ are page counts for the queries t_1 and t_2, respectively. $H(t_1 \cap t_2)$ is the conjunction query t_1 and t_2. All the coefficients are set to zero if $H(t_1 \cap t_2)$ is less than a threshold, *c*, because two terms may appear by accident on the same page. *N* is the number of documents indexed by the search engine.

Algorithm 3.3 shows the similarity calculation procedure using both ontology-based and IR-based methods.

Algorithm 3.3: Similarity Calculation

```
Input I: Array of ontology-contained terms
Output O: Array of similarity values
 1:  For each term in I do
 2:     Take term $t_1$;
 3:     For each term in I do
 4:       Take term $t_2$;
 5:       Compare with ontology defined filters;
 6:       If (filter is available)
 7:         Calculate by equations;
 8:       End
 9:         Calculate by IR-based methods;
10:     end-for
11:  end-for
```

3.2.5. Web-service clustering

3.2.5.1. Feature integration

Generating the ontology and calculating the similarities is performed separately for each extracted feature such as *service name*, *operation name, port name, input message* and *output message*. We then integrate the five similarity values for the five different features, with clustering being achieved according to the integrated similarity values. The similarity values were integrated as follows (3.20):

$$\begin{aligned}&Similarity_{Final}(t_1, t_2) = \\ &W_a\, S_F\left(Sname_{t_1}, Sname_{t_2}\right) + W_b\, S_F\left(Oname_{t_1}, Oname_{t_2}\right) + \\ &W_c\, S_F\left(Pname_{t_1}, Pname_{t_2}\right) + W_d\, S_F\left(Inmsg_{t_1}, Inmsg_{t_2}\right) + \\ &W_e\, S_F\left(Outmsg_{t_1}, Outmsg_{t_2}\right) \qquad (3.20)\end{aligned}$$

The final similarity value $Similarity_{Final}(t_1, t_2)$ integrates the individual similarity values in terms of weights W_a, W_b, W_c, W_d and W_e (each in the range 0–1) [20].

3.2.5.2. Clustering

We used an agglomerative clustering algorithm based on the cluster-center method using TF-IDF values for Web-service clustering [17]. This is a bottom-up hierarchical clustering method. It starts by assigning every Web service to its own cluster and continues, using the TF–IDF values of the *service name*s, until the number of clusters reduces to 5. To merge clusters, a cluster-center identification approach is used [17, 77]. Finally, the services are grouped into five different clusters, namely *Food, Book, Medical, Film* and *Vehicle.* Algorithm 3.4 describes the method used in our clustering approach.

Algorithm 3.4: Clustering Algorithm

Input I_1: Array of calculated Web-service similarity values

Input I_2: Number of required clusters

Output O: Five domain clusters

```
 1:  Let each service be a cluster;
 2:  ComputeProximityMatrix(C);
 3:  N=Number of services;
 4:  while N != $I_2$ do
 5:        Merge two closest clusters;
 6:        N=getNumberOfCurrentClusters();
 7:        Calculate center value of all services in all clusters;
 8:        Select service with highest value of each cluster as
           cluster centers;
 9:        UpdateProximityMatrix();
10:  end-while
```

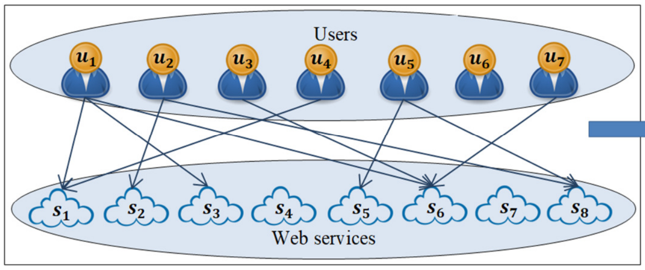
Chapter 4 Improving Web Service Recommendation by Alleviating the Sparsity with a Novel Ontology-based Clustering

4.1. Motivation for Sparsity Reduction

4.1.1. Problem identification

In CF systems, users and services are typically represented by user ratings or purchase history. When the numbers of services increase, the number of selection possibilities increases. For example, if users have rated only a few services among the total number of available services in a dataset or if services have been rated by only a few of the total available users in the dataset, then the sparsity problem occurs. The cold-start problem occurs with new users or new items that enter into the system. If there is a new user without a rating on any items, or a new item with no rating from any users, then there is no information for the prediction. CF requires a large number of available ratings; thus, with a sparse rating matrix, it is challenging to identify the relationship between users and services and thereby make an effective recommendation.

Figure 4.1 shows a simple Web service-user rating graph of a social network consisting of seven Web service users $u = \{u_1, u_2, \ldots. u_7\}$ and eight Web services $s = \{s_1, s_2, \ldots. s_8\}$. In this example, u_1 invokes only three Web services, u_2 and u_5 invoke only two Web services, u_3, u_4, and u_7 invoke just one service among eight Web services, and u_6 invokes no Web service.

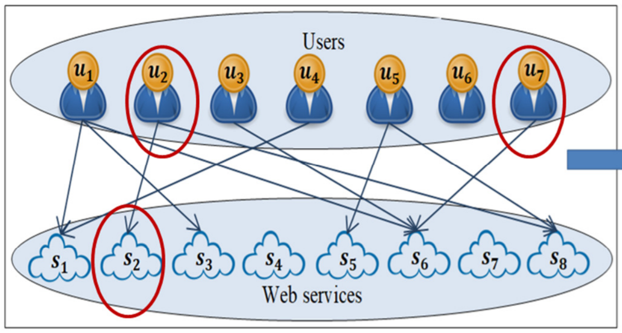


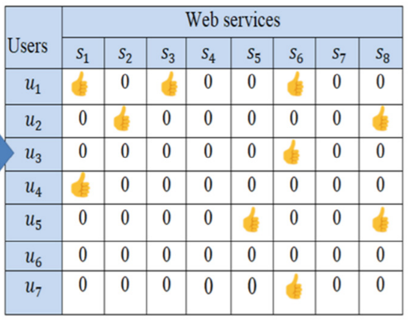
Users	Web services							
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
u_1	👍	0	👍	0	0	👍	0	0
u_2	0	👍	0	0	0	0	0	👍
u_3	0	0	0	0	0	👍	0	0
u_4	👍	0	0	0	0	0	0	0
u_5	0	0	0	0	👍	0	0	👍
u_6	0	0	0	0	0	0	0	0
u_7	0	0	0	0	0	👍	0	0

Figure 4.1 Example of a Web service-user rating graph

As shown in Figure 4.1, even if users are very active, they may invoke only a few Web services among the available web services. In practice, when the numbers of users and Web services become large, the number of invoking ratings becomes limited. In addition, as the number of users or services keeps growing, the data matrix becomes sparse and does not have enough ratings to make accurate and reliable predictions in the recommendation algorithm.

Example of the sparsity problem:



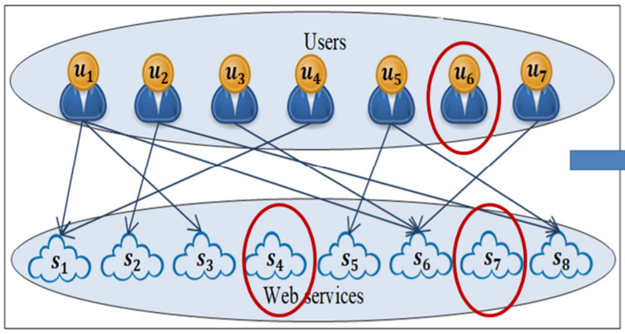


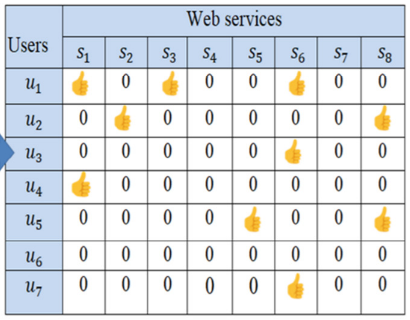
Users	Web services							
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
u_1	👍	0	👍	0	0	👍	0	0
u_2	0	👍	0	0	0	0	0	👍
u_3	0	0	0	0	0	👍	0	0
u_4	👍	0	0	0	0	0	0	0
u_5	0	0	0	0	👍	0	0	👍
u_6	0	0	0	0	0	0	0	0
u_7	0	0	0	0	0	👍	0	0

Figure 4.2 Example of the sparsity problem

Let u_7 be the active user, and we need to determine whether s_2 should be recommended to u_7. To do this, the neighborhood-based (trust-based) CF algorithm first finds neighbor(s) of u_7 by calculating a similarity (trust weight) value. In this example, similar neighbors will be user(s) who invoked s_2 previously. However, in Figure 4.2, s_2 is only previously invoked by u_2, and there is no common previously invoked experience between u_7 and u_2. This situation occurs because of sparsity limitations.

Example of the cold-start problem:





Users	Web services							
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
u_1	👍	0	👍	0	0	👍	0	0
u_2	0	👍	0	0	0	0	0	👍
u_3	0	0	0	0	0	👍	0	0
u_4	👍	0	0	0	0	0	0	0
u_5	0	0	0	0	👍	0	0	👍
u_6	0	0	0	0	0	0	0	0
u_7	0	0	0	0	0	👍	0	0

Figure 4.3 Example of the cold-start problem

In Figure 4.3, Web services s_4 and s_7 are not invoked by any users and u_6 does not invoke any Web service. As there is no information available about these items, if we attempt to make recommendations related to s_4, s_7, and u_6, then it becomes difficult without an existing history between users and Web services. This situation occurs because of cold-start limitations.

In our approach, the main task is to add missing rating values in the user-service matrix using a new clustering approach and reduced sparsity. By increasing the user interaction between service items, we can compare more invocations between users and improve the recommendation performance.

4.1.2. Choosing a better clustering method

Web service clustering is used to create single databases within a large-scale database based on their characteristics, aggregating them according to their similarities. To alleviate sparsity, choosing a better clustering method is the main means of gaining improved performance in the recommendation result. A review of several existing Web service clustering approaches indicates that they use different methods for similarity calculation and clustering. Based on clustering performance, we mainly consider three approaches.

(i) The HTS method [17] uses an ontology learning method for the clustering. The HTS method identifies hidden semantic patterns such as subclass–superclass, data property, and object property relationships from complex terms in Web services description language (WSDL) documents to generate the ontology. After generating the ontology, similarity is calculated using the generated ontology relationships. If this fails, IR-based methods are used for the similarity calculation. Finally, the agglomerative clustering algorithm is used for the clustering.

(ii) The CAS method [19] uses a machine learning-based method for the clustering. It uses a support vector machine (SVM) in its similarity calculation. SVMs are trained to produce a model for computing the similarity of services (SoS) for different domains. A spherical associated keyword space (SASKS) algorithm [78] is applied to visualize the service clusters. It projects the clustering results for a three-dimensional (3D) sphere onto a two-dimensional (2D) spherical surface for 2D visualization.

(iii) In addition, our previously proposed clustering approach [79] uses the novel specificity-aware ontology generating method for the similarity calculation and the agglomerative clustering algorithm for the clustering.

We chose these approaches for the comparison because they use interesting clustering methods rather than a simple clustering method; moreover, they have shown better clustering performance, and also use the same Web service dataset that we used.

These methods [17], [19] are based on the formal concepts and super-subrelations of general terms and do not consider the specificity of the terms. However, the specificity of a term explains the quantity of domain-specific information contained in the term, which is essential in generating a more efficient detailed hierarchy. Our proposed clustering approach [79] takes advantage of this domain specificity and we selected the method to alleviate sparsity by evaluation with the high performance of the clustering results as well as the recommendation.

As an example of the better clustering performance of our approach, Figure 4.4 shows the clustering results with a comparison of the HTS and CAS methods. Each method uses five clustering groups, *Vehicle*, *Medical*, *Film*, *Food*, and *Book* for clustering. Figure 4.4 shows cluster groups with some Web service clustering examples with incorrectly placed clusters. The novel specificity-based clustering method was able to identify the real ontological concept very well by differentiating *Book*, *Food*, *Vehicle*, *Car*, *Hospital*, and *Medical* as key terms that failed to be identified in the previous methods and thus showed better clustering performance.

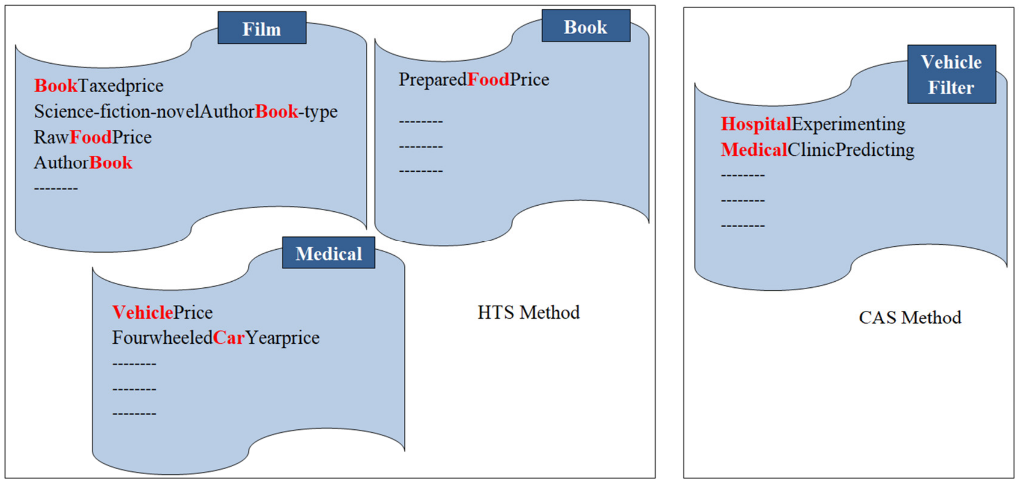


Figure 4.4 Examples of incorrectly placed cluster groups in the HTS and CAS methods

4.2. Overview of the Proposed Recommendation Approach

The architecture of the proposed approach is shown in Figure 4.5. It contains five steps, namely collect user-service rating data, alleviate sparsity by the ontology-based clustering approach, calculate the trust weight between users, user's rating prediction, and service recommendation.

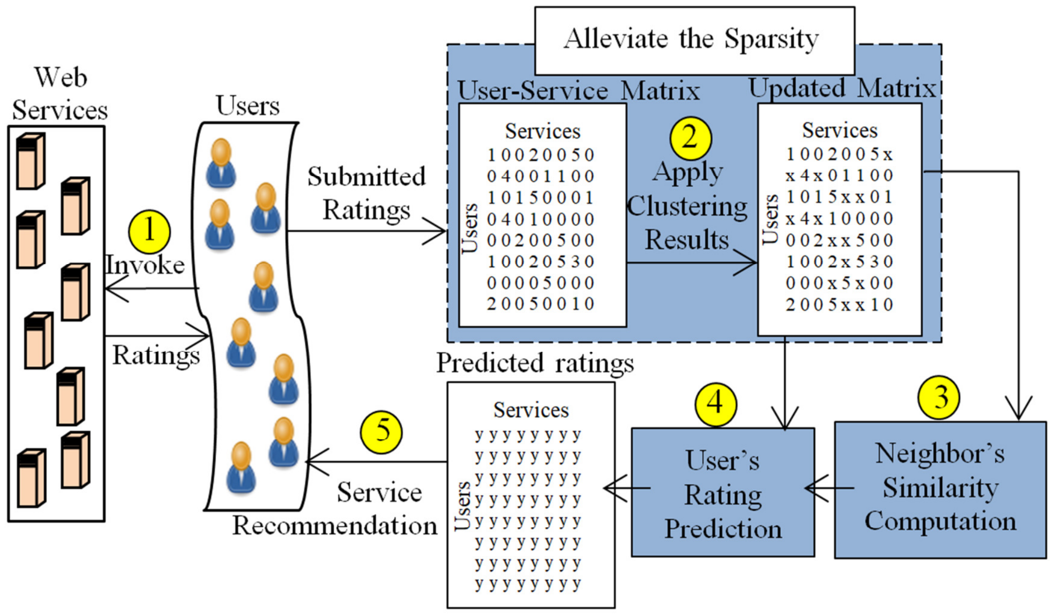


Figure 4.5 Architecture of the proposed method for recommending web services

As shown in Figure 4.5, first we collect the user-service rating data after the user invokes available Web services. Following the next steps shown in the Figure 4.5, we fill the nonrated values using the clustering results and continue until the recommendation.

4.2.1. Collect user-service rating data

The user-service rating graph ($u\ X\ s$) shows each user's (u) ratings (r_{us}) for each Web service (s). As shown in (4.1), the ratings indicate whether user u had invoked service s in the past or not and their level of preference, which ranges from 1 to 5 where 5 is the highest possible rating. If user u did not previously invoke the Web service s, then $r_{us} = 0$.

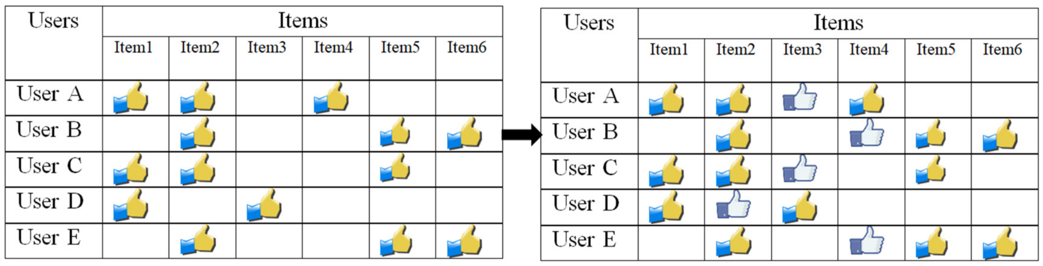
$$r_{us} \begin{cases} r,\ r = 1, 2, 3, 4, 5 \text{ if user } u \text{ rated service } s, \\ 0, \text{ otherwise} \end{cases} \tag{4.1}$$

4.2.2. Alleviate sparsity by the ontology-based clustering approach

4.2.2.1. Proposed ontology-based clustering approach

Here, we apply our proposed ontology-based clustering approach. With comparing the previous clustering approaches we selected this for a sparsity alleviating with the best performance.

4.2.2.2. Sparsity alleviation



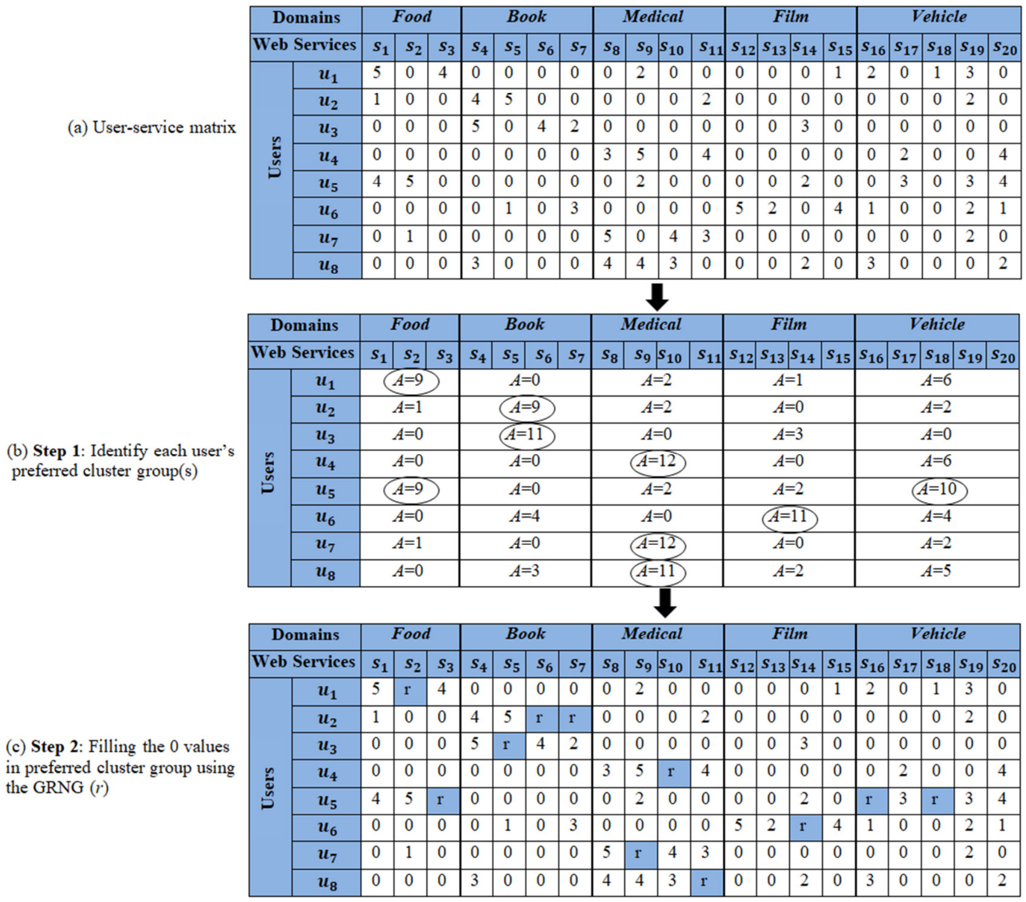
Users	Items					
	Item1	Item2	Item3	Item4	Item5	Item6
User A	👍	👍		👍		
User B		👍			👍	👍
User C	👍	👍			👍	
User D	👍		👍			
User E		👍			👍	👍

Users	Items					
	Item1	Item2	Item3	Item4	Item5	Item6
User A	👍	👍	🖒	👍		
User B		👍		🖒	👍	👍
User C	👍	👍	🖒		👍	
User D	👍	🖒	👍			
User E		👍		🖒	👍	👍

Figure 4.6 Sparsity alleviation example

Figure 4.6. shows sparsity alleviation example using clustering results. The user-service matrix explains each user's invoking history on Web services with their preference in the range 1 to 5. When we consider a larger user-service matrix with a high number of users and services, it contains more 0 values with a high sparsity level. Figure 4.7 shows the user-service matrix for eight users and twenty Web services. The steps for alleviating the sparsity are as follows.

Step 1: We identify each user's preferred cluster group(s) according to the history of each user's ratings. The new ontology-based clustering approach outputs the five clustering groups (*Food*, *Book*, *Medical*, *Film*, and *Vehicle*) related to different domains of Web services. First, we obtain the addition (*A*) of ratings for each of the five separate identified service cluster groups and using a threshold value we decide on the highest *A* group(s) as the user's preferred cluster group(s). We continue this process for every user. The one limitation of existing approaches is that they do not consider the situation when a user prefers more than one cluster domain and assumes that each user belongs to a single cluster. However, we deal with this situation by giving a threshold value to consider the rating information.



(a) User-service matrix

Domains		Food			Book				Medical				Film				Vehicle				
Web Services		s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}
Users	u_1	5	0	4	0	0	0	0	0	2	0	0	0	0	0	1	2	0	1	3	0
	u_2	1	0	0	4	5	0	0	0	0	0	2	0	0	0	0	0	0	0	2	0
	u_3	0	0	0	5	0	4	2	0	0	0	0	0	0	3	0	0	0	0	0	0
	u_4	0	0	0	0	0	0	0	3	5	0	4	0	0	0	0	0	2	0	0	4
	u_5	4	5	0	0	0	0	0	0	2	0	0	0	0	2	0	0	3	0	3	4
	u_6	0	0	0	0	1	0	3	0	0	0	0	5	2	0	4	1	0	0	2	1
	u_7	0	1	0	0	0	0	0	5	0	4	3	0	0	0	0	0	0	0	2	0
	u_8	0	0	0	3	0	0	0	4	4	3	0	0	0	2	0	3	0	0	0	2

(b) **Step 1**: Identify each user's preferred cluster group(s)

Domains		Food	Book	Medical	Film	Vehicle
Web Services		s_1 s_2 s_3	s_4 s_5 s_6 s_7	s_8 s_9 s_{10} s_{11}	s_{12} s_{13} s_{14} s_{15}	s_{16} s_{17} s_{18} s_{19} s_{20}
Users	u_1	$\boxed{A=9}$	$A=0$	$A=2$	$A=1$	$A=6$
	u_2	$A=1$	$\boxed{A=9}$	$A=2$	$A=0$	$A=2$
	u_3	$A=0$	$\boxed{A=11}$	$A=0$	$A=3$	$A=0$
	u_4	$A=0$	$A=0$	$\boxed{A=12}$	$A=0$	$A=6$
	u_5	$\boxed{A=9}$	$A=0$	$A=2$	$A=2$	$\boxed{A=10}$
	u_6	$A=0$	$A=4$	$A=0$	$\boxed{A=11}$	$A=4$
	u_7	$A=1$	$A=0$	$\boxed{A=12}$	$A=0$	$A=2$
	u_8	$A=0$	$A=3$	$\boxed{A=11}$	$A=2$	$A=5$

(c) **Step 2**: Filling the 0 values in preferred cluster group using the GRNG (*r*)

Domains		Food			Book				Medical				Film				Vehicle				
Web Services		s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	s_{20}
Users	u_1	5	r	4	0	0	0	0	0	2	0	0	0	0	0	1	2	0	1	3	0
	u_2	1	0	0	4	5	r	r	0	0	0	2	0	0	0	0	0	0	0	2	0
	u_3	0	0	0	5	r	4	2	0	0	0	0	0	0	3	0	0	0	0	0	0
	u_4	0	0	0	0	0	0	0	3	5	r	4	0	0	0	0	0	2	0	0	4
	u_5	4	5	r	0	0	0	0	0	2	0	0	0	0	2	0	r	3	r	3	4
	u_6	0	0	0	0	1	0	3	0	0	0	0	5	2	r	4	1	0	0	2	1
	u_7	0	1	0	0	0	0	0	5	r	4	3	0	0	0	0	0	0	0	2	0
	u_8	0	0	0	3	0	0	0	4	4	3	r	0	0	2	0	3	0	0	0	2

Figure 4.7 Alleviating the sparsity of the user-service rating matrix using a clustering process

Eg:

Figure 4.7 (b) shows the addition (A) of ratings for each of the five separate service cluster groups for each user. Here, A=8 is used as a threshold value, and as shown in Figure 4.7 (b), the highest A group(s) are selected as the user's preferred cluster group(s). According to the addition values A, u_5 preferred two service cluster groups (*Food* and *Vehicle*) and users u_1 (*Food*), u_2(*Book*), u_3(*Book*), u_4(*Medical*), u_6(*Film*), u_7(*Medical*), and u_8(*Medical*) preferred only one cluster group.

Step 2: If user u contains nonrated data (set as 0) inside their most preferred cluster(s), we fill it using a Gaussian distribution random number generator. This process continues for all users by filling the 0 values and the updated user-service matrix is used for further calculations.

Eg:

As shown in Figure 4.7 (b), u_1's preferred cluster group is *Food.* Then, 0 values of u_1 under the *Food* cluster group are filled using the GRNG (*r*) as shown in Figure 4.7 (c). This process continues for all users.

It shows how the sparsity is successfully alleviated by random number *r* after applying this clustering process.

4.2.3. Calculation of the similarity between users

Similarity calculation is based on the history of ratings of users who have similar preferences for Web services. Existing recommendation approaches use different ways to calculate the similarity, in particular PCC and cosine-based methods, which depend on the user ratings of items that both users have rated. We assign the similarity as a trust weight and use PCC for the calculation since it can be easily implemented and can achieve high accuracy. A trust-based system analyzes not only user group similarity but also the social relationships between users.

$$Sim(u_i, u_j) = \frac{\sum_{s \in S}\left(r_{u_i,s} - \overline{r_{u_I}}\right)\left(r_{u_j,s} - \overline{r_{u_J}}\right)}{\sqrt{\sum_{s \in S}\left(r_{u_i,s} - \overline{r_{u_I}}\right)^2}\sqrt{\sum_{s \in S}\left(r_{u_j,s} - \overline{r_{u_J}}\right)^2}} \tag{4.2}$$

Here, $S = S_{u_i} \cap S_{u_j}$ is the subset of Web service items that both users u_i and u_j have invoked previously. In addition, $r_{u_i,s}$ and $r_{u_j,s}$ denote the rating values of Web service *s* invoked by service user u_i, and Web service *s* invoked by service user u_j, respectively. $\overline{r_{u_I}}$ and $\overline{r_{u_J}}$ represent the average rating values of different Web services observed by service users u_i and u_j, respectively. According to this measurement, the final trust weight value of two service users is in the range [–1, 1], where a larger trust weight value indicates that service users *i* and *j* are more similar. In addition, if S = *null*, this means that we cannot measure the similarity between two users due to the lack of history of information between them. In this situation, we assumed that they don't have common invocations means their trust value becomes low and $Sim(u_i, u_j) = 0$.

We improve the calculated similarity value by applying the significance weighting calculation [80], which helps to overestimate the similarities of service users who are actually not similar but happen to have similar rating experience of Web services. The calculation helps to reduce the number of similar users who do not otherwise share much similarity.

$$Sim(u_i, u_j)' = \frac{2 \times |W_{u_i} \cap W_{u_j}|}{|W_{u_i}| + |W_{u_j}|} Sim(u_i, u_j) \tag{4.3}$$

$|W_{u_i} \cap W_{u_j}|$ indicates the number of Web services that are invoked by both users u_i and u_j. $|W_{u_i}|$ and $|W_{u_j}|$ are the number of Web services invoked by users u_i and u_j, respectively. The new similarity value $Sim(u_i, u_j)'$ is also in the range [–1, 1]. As the number of common invoked services $|W_{u_i} \cap W_{u_j}|$ increases, the final similarity also becomes high. Similar users for each user are identified and the final similarity $Sim(u_i, u_j)'$ is assigned as a trust value between those users. This weight should be related to the trust from user u_i toward user u_j.

4.2.4. User's rating prediction

Then, each user's rating is predicted based on the updated user-service rating matrix and the calculated trust weight value from the previous step. This approach creates the trust network between users and makes predictions based on the user ratings that are directly or indirectly trusted by the user seeking a recommendation [81].

To recommend service s to user u, the first task is to find similar users who rated the same service s. If we want to compute a prediction $(P_{u_i,s})$ for user u_i on target service s and if we discover that user u_j rated the same service s and users u_i and u_j trusted each other, then we can aggregate their ratings for the calculation with confidence. Equation (4.4) is used for the prediction calculation:

$$(P_{u_i,s}) = \overline{r_{u_I}} + \frac{\sum_{u_j \in U} W_{u_i,u_j} \left(r_{u_j,s} - \overline{r_{u_J}}\right)}{\sum_{u_j \in U} W_{u_i,u_j}} \tag{4.4}$$

where W_{u_i,u_j} is the trust value *($Sim(u_i, u_j)'$)* between users u_i and u_j calculated by equation (4.3), which describes the effect of user u_j on user u_i. $\overline{r_{u_\imath}}$ and $\overline{r_{u_J}}$ are the average ratings of users u_i and u_j, respectively.

4.2.5. Service recommendation

All the rating values of the user-service matrix are predicted using equation (4.4); finally, the top *s* services are recommended to the users based on the predicted ratings. The overall process of service recommendation is described in algorithm 4.1.

Algorithm 4.1: Web Service Recommendation

```
Input  S: Web Service dataset
       U: User's dataset
       R: User-service invokes data
       C: Web service clustering results
Output O: Recommendation results
1:    For each user U do
2:        C_i=Calculate summation of ratings data (r_us) in R separately
          for each
3:        C cluster group; //for each user separately for five domain
4:        clusters
5:        M=Maximum cluster(s) C_i;
6:          For M do
              R'=Update nonrated services in M using Gaussian
            distribution;
          end-for
7:    end-for
8:    For each user U do
9:        T=calculate trust rate between each user;
10:   end-for
11:   For each user, invoke data (r_us) in R' do
12:       P=New predicted ratings using T and updated R';
13:   end-for
14:   O=Do recommendations using P; //select top P
```

Chapter 5 Experiments and Evaluations

5.1 Experiments and Evaluations on Web Service Clustering Process

5.1.1. Comparison of clustering approach

Table 5.1 presents the comparison of clustering approaches. Here, we consider the term similarity calculation approach that use in current clustering approaches and proposed clustering approach.

Table 5.1 Comparison of clustering approaches

Feature						**Proposed Approach**
	String-based (KM, Cosine similarity, etc.)	**Knowledge-based (Ontology, WordNet)**	**Corpus-based (Web-PMI, NGD, etc.)**	**HTS**	**CAS**	**Based on specificity-aware ontology**
Domain specific context is used	No	No	No	No	Yes	Yes
Use up-to-date knowledge from the Web data	No	No	Yes	No	Yes	Yes
Consideration for different type of relationship among Web services	No	No	No	Yes	No	Yes
Need assistance	No	Yes	No	No	No	No

from domain expertise						
Use domain knowledge	No	Yes	Yes	Yes	Yes	Yes
Identification of real semantic exist between services under particular domain (Multi-domain nature)	Nil	Low	Medium	Medium	High	Medium
Encode fined grained information	No	No	No	Yes	Yes	Yes
Use unnecessary information	No	No	Yes	No	No	No

According to the experiment results, we can show that our new approach addressed the issues of current clustering approaches. We summarized those issues in Table 1.1.

5.1.2 Experiments and evaluations

The experimental platform used Microsoft Windows 10 on a PC with an Intel Core i7-6500 at 2.59 GHz and 8.00 GB of RAM. Java was used for programming the ontology generation and the service-clustering procedure. The generated ontology was displayed visually by using JTree. In each experiment, we used a set of extracted features from 400 WSDL files.

Performance evaluation of the clustering results involved precision, recall, F-measure, purity and entropy in a comparison to previous approaches. (Note that the "entropy" measure here is totally different from that defined in equation (3.7).)

Precision is used to measure result relevancy and recall is used to quantify the numbers of properly related results being returned. F-measure is a combination of precision and recall. These three criteria can be expressed as equations (5.1), (5.2) and (5.3):

$$Precision(x,y) = \frac{N_{xy}}{N_y} \tag{5.1}$$

$$Recall(x,y) = \frac{N_{xy}}{N_x} \tag{5.2}$$

$$F(x,y) = \frac{2 \times Precision(x,y) \times Recall(x,y)}{Precision(x,y) + Recall(x,y)} \tag{5.3}$$

Here, N_{xy} is the number of members of class *x* in cluster *y*, N_y is the number of members in cluster *y* and N_x is the number of members in class *x*. The F-measure of cluster *x* with respect to class *y* is defined in (5.3).

The proportion of properly classified classes per cluster is measured by purity:

$$Purity = \frac{1}{N} \sum_{i=1}^{k} max\{N_i^j\} \tag{5.4}$$

Here, *N* is the total number of services, *k* is the number of clusters and N_i^j is the numbers of services in cluster *i* belonging to domain class *j*.

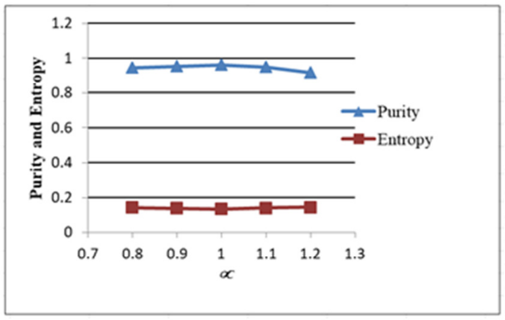
The distribution of semantic classes within each cluster is measured by entropy. A lower entropy means better clustering. The entropy of a cluster is defined as:

$$E(C) = -\frac{1}{log\ L} \sum_{q=1}^{L} \frac{n_p^q}{n_p}\ log\ \frac{n_p^q}{n_p} \tag{5.5}$$

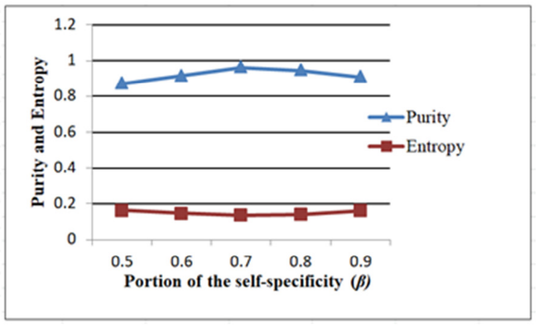
Here, L is the number of domain classes in the data set, n_p^q is the number of services of the q^{th} domain class that was assigned to the p^{th} cluster and n_p is the number of services in cluster *p*. The entropy of the entire cluster is defined as:

$$Entropy = \sum_{p=1}^{L} \frac{n_p}{N} E(C) \tag{5.6}$$

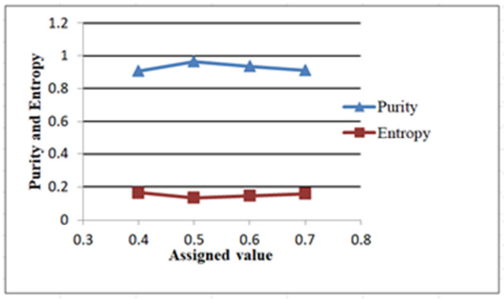
5.1.2.1. Evaluation of domain-specificity weight and similarity weight calculations



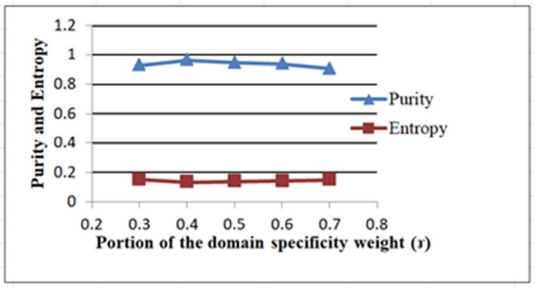
(a): Effect of the parameter ∝



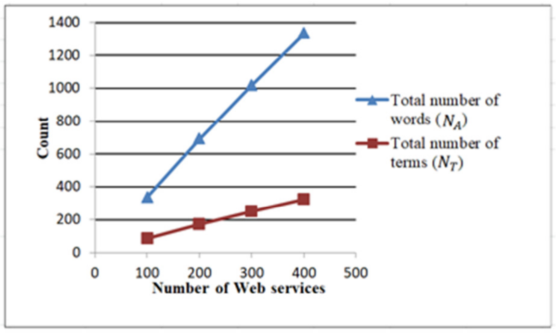
(b): Effect of the parameter β



(c): Effect of the assigned value for the $W_{Spec}(G_i)$



(d): Effect of the parameter ɤ



(e): Count of the (N_A) and (N_T)

Figure 5.1 Evaluation of domain-specificity weight and similarity weight calculations

First, to show the impact of purity and entropy for the self-specificity calculation in equation (3.6), we varied the value of ∝ from 0.8 to 1.2. Figure 5.1(a) shows that optimal purity and entropy values occur when ∝=1.

Next, we measured the purity and entropy values while changing the β parameter in the hybrid specificity calculation (see equation (3.9)). It gives optimal results for purity and entropy when β=0.7 within the range 0.5 to 0.9, as shown in Figure 5.1(b). We can note that self-specificity makes a higher contribution to the final hybrid specificity value than does context-specificity.

The domain-specificity weight of a substructure, as calculated in equation (3.10), is dependent on the number of sibling terms of the new term t_{new}. If t_{new} has no sibling terms, then we assign a fixed value of 0.5 for the domain-specificity weight. It was found by experimentation with values from 0.4 to 0.7 that 0.5 gives optimal results for purity and entropy. Figure 5.1(c) shows the results for purity and entropy as the assigned value varies.

Figure 5.1(d) shows variations in the parameter γ for the final calculation (see equation (3.14)) of the domain-specificity weight and similarity weight combination that we used to select the optimal ontology structure with the highest weight value. We assigned γ =0.4 from the range 0.3 to 0.7, given the high contribution from similarity weight.

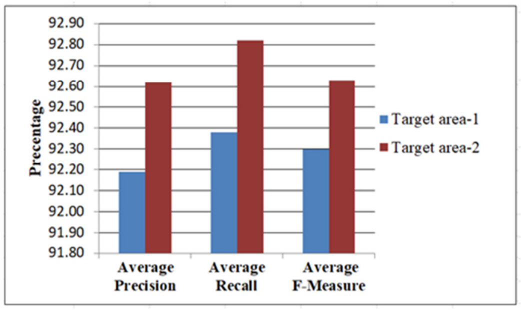
Figure 5.1(e) shows the variation in the total number of words (N_A) and the total number of terms (N_T) as the number of services varies from 100 to 400. Equation (3.6) uses these values in the calculation of self-specificity. The total numbers of words and terms both increased as the number of services increased. Note that the rate of increase for words is greater than the rate of increase for terms as the number of Web services increases.

5.1.2.2. Evaluation of ontology generation

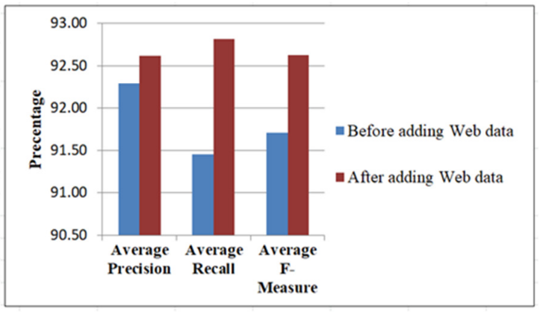
Here, we measure the extent to which ontology learning helps to improve similarity calculations and clustering by generating the ontology in a more readable manner. We experimented with several factors as we sought improvements in ontology generation.

When adding a new term to a partially generated ontology, we first need to select a suitable target area of nodes. Expanding the target area will provide opportunities for more nodes to be tested and will indicate the most suitable nodes for adding the new term. Figure 5.2(a) shows the different results for precision, recall and F-measure for two different target areas in an existing ontology. Target area-1 ranged from $Spec(t_{new}) - 0.1$ to $Spec(t_{new}) + 0.1$ and Target area-2 ranged from $Spec(t_{new}) - 0.3$ to $Spec(t_{new}) + 0.3$. Expanding the target area will also increase the computation time for the program. We chose to use a target area in the existing ontology that ranged from $Spec(t_{new}) - 0.3$ to $Spec(t_{new}) + 0.3$ because this would help to improve the performance with respect to clustering results.

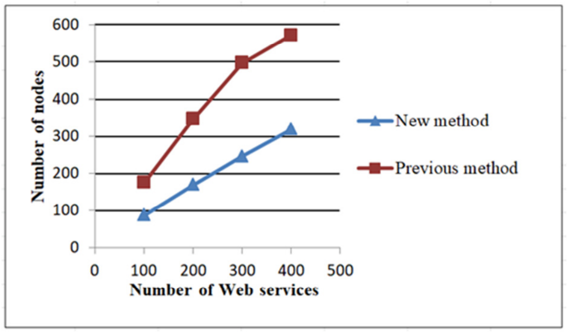
In addition, we found that we could improve the ontology performance by adding more domain-specific terms to it. Adding frequently used terms extracted from the Google search engine resulted in better precision, recall and F-measure values. Figure 5.2(b) shows the change in average precision, recall and F-measure values before and after adding Web-based data.



(a): Evaluation with different target areas



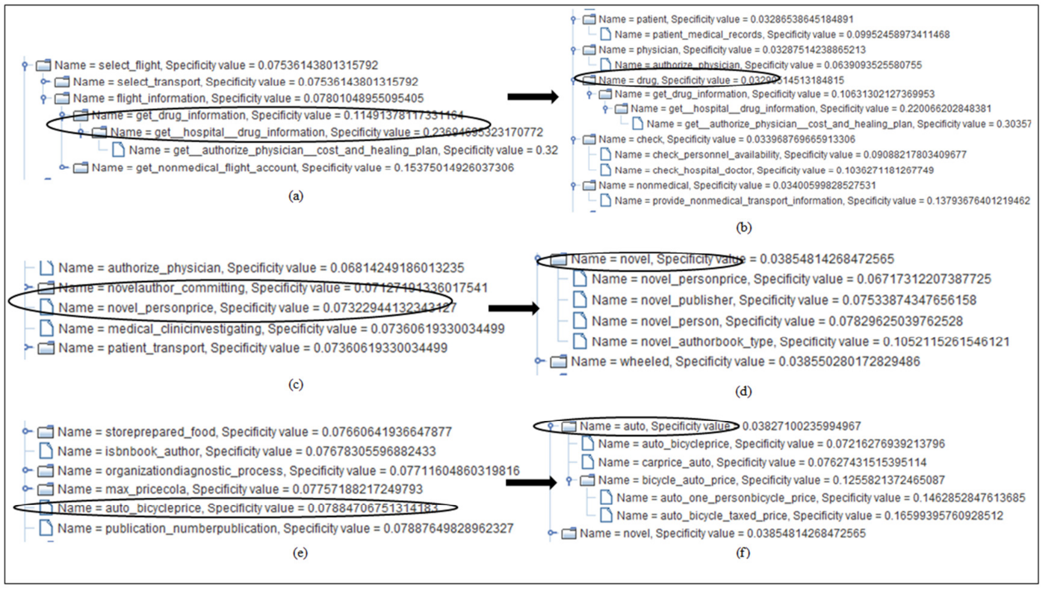
(b): Evaluation with additional input data



(c): Num.of nodes vs. num.of Web services

Figure 5.2 Evaluation of ontology generation

Figure 5.3 shows the differences in the ontology hierarchy before and after adding the data from the Google search engine. In this figure, (a), (c) and (e) show the status before adding the Web-based data, with (b), (d) and (f) showing the status afterwards. The second version has an improved arrangement of domain-specific data because of



```
Name = select_flight, Specificity value = 0.07536143801315792
    Name = select_transport, Specificity value = 0.07536143801315792
    Name = flight_information, Specificity value = 0.07801048955095405
        Name = get_drug_information, Specificity value = 0.11491378117331164
            Name = get__hospital__drug_information, Specificity value = 0.23694695323170772
                Name = get__authorize_physician__cost_and_healing_plan, Specificity value = 0.32
        Name = get_nonmedical_flight_account, Specificity value = 0.15375014926037306
```

(a)

```
Name = patient, Specificity value = 0.03286538645184891
    Name = patient_medical_records, Specificity value = 0.09952458973411468
Name = physician, Specificity value = 0.03287514238865213
    Name = authorize_physician, Specificity value = 0.0639093525580755
Name = drug, Specificity value = 0.03290514513184815
    Name = get_drug_information, Specificity value = 0.10631302127369953
        Name = get__hospital__drug_information, Specificity value = 0.220066202848381
            Name = get__authorize_physician__cost_and_healing_plan, Specificity value = 0.30357
Name = check, Specificity value = 0.033968769665913306
    Name = check_personnel_availability, Specificity value = 0.09088217803409677
    Name = check_hospital_doctor, Specificity value = 0.1036271181267749
Name = nonmedical, Specificity value = 0.03400599828527531
    Name = provide_nonmedical_transport_information, Specificity value = 0.13793676401219462
```

(b)

```
Name = authorize_physician, Specificity value = 0.06814249186013235
Name = novelauthor_committing, Specificity value = 0.07127191336017541
Name = novel_personprice, Specificity value = 0.07322944132343127
Name = medical_clinicinvestigating, Specificity value = 0.07360619330034499
Name = patient_transport, Specificity value = 0.07360619330034499
```

(c)

```
Name = novel, Specificity value = 0.03854814268472565
    Name = novel_personprice, Specificity value = 0.06717312207387725
    Name = novel_publisher, Specificity value = 0.07533874347656158
    Name = novel_person, Specificity value = 0.07829625039762528
    Name = novel_authorbook_type, Specificity value = 0.1052115261546121
Name = wheeled, Specificity value = 0.038550280172829486
```

(d)

```
Name = storeprepared_food, Specificity value = 0.07660641936647877
Name = isbnbook_author, Specificity value = 0.07678305596882433
Name = organizationdiagnostic_process, Specificity value = 0.07711604860319816
Name = max_pricecola, Specificity value = 0.07757188217249793
Name = auto_bicycleprice, Specificity value = 0.07884706751314183
Name = publication_numberpublication, Specificity value = 0.07887649828962327
```

(e)

```
Name = auto, Specificity value = 0.03827100235994967
    Name = auto_bicycleprice, Specificity value = 0.07216276939213796
    Name = carprice_auto, Specificity value = 0.07627431515395114
    Name = bicycle_auto_price, Specificity value = 0.1255821372465087
        Name = auto_one_personbicycle_price, Specificity value = 0.1462852847613685
        Name = auto_bicycle_taxed_price, Specificity value = 0.16599395760928512
Name = novel, Specificity value = 0.03854814268472565
```

(f)

Figure 5.3 Differences in the ontology hierarchy before and after adding Web-based data

the new Web-based words "*drug,*" "*novel*" and "*auto*" in these three areas, respectively.

Next, we compared the number of nodes in our new ontology with a previous HTS-based ontology [17]. (We checked the ontologies without adding the Web-based data in our method.) As shown in Figure 5.2(c), the previous method contains more ontology nodes, which means that complex terms were more likely to be divided into individual terms in generating their ontology. Because we generate our ontology directly using the original terms, the new method will contain fewer nodes.

A comparison of the ontologies generated by the previous HTS approach and by the new approach is given in Table 5.2, for various ontology criteria, characteristics and parameters.

5.1.2.3. Evaluation of similarity calculations

We evaluated the assignment of different weight values for W_1 and W_2 in the similarity-calculation equation (3.16), as shown in Table 5.3.

We chose the Weight 2 option from this table for W_1 and W_2 because the similarity-calculation results for the seven matching filters were better than for other values.

Table 5.2 Comparison of ontologies for the previous HTS approach and for the new approach

Criteria	Evaluation	
	Previous HTS approach	**New approach**
Correctness of class hierarchy	- Same domain classes are attached to the ontology in groups. However, they are spread diffusely over the ontology hierarchy in many places, with each of the many domain groups containing few nodes.	- Each level of ontology classes contains more classes of similar specificity values. Traversing the ontology hierarchy from top to bottom, the specificity increases level by level. - Same domain classes are attached to the ontology in groups. Most of the nodes for a domain are attached together in one group.
Graph shape	Ontology graph is balanced with a number of ontology levels and a number of child classes.	Ontology graph is balanced with a number of ontology levels and a number of child classes.
Node characteristics	More non-child classes at the top.	Child classes are diffused equally over the ontology.
Graph characteristics		
Number of classes	Many classes, because component words also become nodes in the ontology.	Number of nodes is the same as for the original nodes. Therefore, the graph is uncluttered, with few nodes.
Number of edges	Edges are spread diffusely across the classes.	Edges are spread diffusely across the classes.
Total weight of specificity		More terms of similar specificity exist at the same level of the ontology. Total weight of specificity in the ontology is balanced across sibling classes and ontology levels.
Total weight of similarity		Total weight of similarity in the ontology is equally balanced across parent, child and sibling classes.
Parameter		
Total number of input terms (Options were 100, 200, 300 and 400)	- Final clustering performance and the correctness of ontology structure decrease when the numbers of inputs increase.	- The correctness of ontology structure improves as the number of inputs increases. - Final clustering result deteriorates as the numbers of inputs are increased, but the rate of deterioration is low in comparison with that of the previous approach.

Table 5.3 Experimental values for W_1 and W_2

Machine filter	Weight 1		Weight 2		Weight 3		Weight 4	
	W_1	W_2	W_1	W_2	W_1	W_2	W_1	W_2
Siblings	0.95	0.05	0.9	0.1	0.87	0.13	0.82	0.18
Parent–Child	0.85	0.15	0.8	0.2	0.78	0.22	0.75	0.25
Near-Descendants	0.8	0.2	0.78	0.22	0.75	0.25	0.73	0.27
Shared-Ancestor	0.7	0.3	0.65	0.35	0.62	0.38	0.6	0.4
Far-Descendants	0.65	0.35	0.62	0.38	0.6	0.4	0.57	0.43

5.1.2.4. Evaluation of Web-service clustering

Five different domains were considered, namely *Vehicle, Medical, Film, Food* and *Book*. Performance evaluation of clustering results involved purity, entropy, precision, recall and F-measure in a comparison with previous approaches. We compared an edge-count-based method, an HTS approach that used ontology learning [17], a CAS approach that used machine learning [19] and our new approach, which uses specificity-based ontology learning.

Table 5.4 gives the experimental results, comparing precision, recall and F-measure values. From these clustering results, the best cluster performance was achieved by our new approach, which placed services correctly for more of the clusters than did the other methods. Our new approach offered improvements in the average precision values of 21.46%, 1.56% and 6.03%, the average recall values of 28.38%, 1.38% and 0.94 %, and the average F-measure values of 26.45%, 1.73% and 3.57%, over those for the edge-count-based, HTS and CAS methods, respectively. In fact, all results for the new approach exceed 84%.

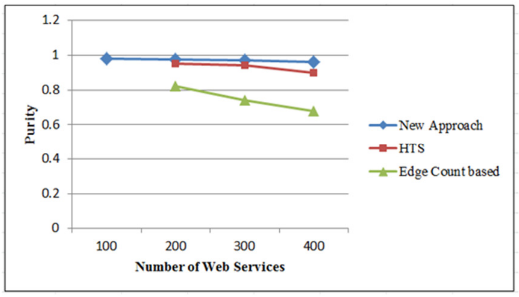
Based on these clustering results, we can note that extracting features from WSDL documents alone is insufficient to identify the correct cluster for some terms.

Figure 5.4(a) and Figure 5.4(b) show the variation in purity and entropy values, respectively, as the number of Web services increases from 100 to 400. As the number of Web services increases, our approach gives increasingly better results than previous approaches in terms of purity decrease and entropy increase. From these results, the new approach gives better accuracy for a high number of inputs. In addition, our new approach gave lower entropy and higher purity values throughout, with the rate of purity-value decrease and the rate of entropy-value increase both being smaller. This comparative study of alternative approaches supports the validity of the proposed approach.

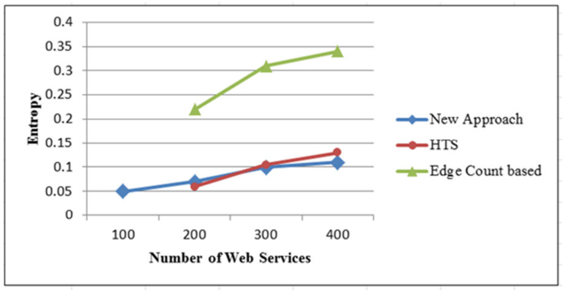
Table 5.4 Performance measures for clusters using precision, recall and F-measure

Cluster	Edge-Count-Based (Using WordNet) (%)			HTS Approach (%)		
	Precision	*Recall*	*F-Measure*	*Precision*	*Recall*	*F-Measure*
Vehicle	56.00	80.90	66.20	81.60	94.80	87.70
Medical	100.00	70.00	82.40	100.00	83.10	90.80
Food	55.00	60.00	57.40	96.00	91.30	93.60
Book	67.10	61.30	64.10	86.70	100.00	92.90
Film	77.70	50.00	60.80	91.00	88.00	89.50
Average	**71.16**	**64.44**	**66.18**	**91.06**	**91.44**	**90.90**

Cluster	CAS Approach (%)			New Approach (%)		
	Precision	*Recall*	*F-Measure*	*Recall*	*Precision*	*F-Measure*
Vehicle	89.47	89.47	89.47	90.80	91.86	91.33
Medical	88.10	100.00	93.67	94.64	84.13	89.08
Food	85.71	93.10	89.26	92.86	96.30	94.55
Book	82.14	92.00	86.79	90.59	96.25	93.33
Film	87.50	84.85	86.15	94.20	95.59	94.89
Average	**86.59**	**91.89**	**89.07**	**92.62**	**92.82**	**92.63**



(a): Cluster performance of new and existing approaches (Purity)



(b): Cluster performance of new and existing approaches (Entropy)

Figure 5.4 Cluster performance of new and existing approaches

5.2 Experiments and Evaluations on Web Service Recommendation Process

In the recommendation process, we first collect input data as the user-service rating preference, which includes each user's invoking history to available Web services. As a user-service dataset, we simulated 200 users' ratings using 400 real Web services. We extracted five Web service features from the services dataset, including real-world Web service repositories and the OWL-S (http://projects.semwebcentral.org/projects/owls-tc/) test collection dataset for the WSDL documents related to the five domains.

Although we used real Web services data, it was difficult to obtain real recommendation data for those Web services. We therefore had to use a simulated dataset by generating data using some method. There are some existing approaches that are used to simulate data [81]–[83] and there are some existing methods for generating data such as Gaussian random number generators (GRNG) [84], [85] and scale-free network theory [86], which propose a method to generate a large dataset using neighborhood data nodes. They generate a large dataset using a small dataset based on each service node relationship. However, service recommendation is not the area of scale-free network theory and, usually, recommendation and sparsity follow the GRNG. So, we chose a GRNG and then confirmed its correctness through evaluation by taking the same result compared with the simple manually generated dataset. When generating the numbers, we used a GRNG for two situations.

(i) To select rated and nonrated services, random numbers are generated as a binary option. It helps to create a space matrix.

(ii) Then, random numbers are generated to assign ratings for the above-selected rated services in the range of 1 to 5.

The GRNG is described in equation (5.7). Here μ is the mean, σ is the standard deviation, and σ^2 is the variance.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right] \tag{5.7}$$

Evaluation was carried out based on the ontology-based clustering and on the sparsity-alleviating methods using MAE and RMSE. We also used different sparsity levels, such as 85%, 70%, and 55% by varying the data density from 15% to 45%.

$$Sparsity\ level = \frac{Number\ of\ Nonspecified\ ratings}{Number\ of\ all\ possible\ ratings} \tag{5.8}$$

MAE measures the deviation of predictions generated by the recommender system from the existing rating values invoked by the user. RMSE is the square root of the average of squared differences between prediction and actual observation. Smaller values of MAE and RMSE indicate a better prediction result. These two criteria can be expressed as equations (5.9) and (5.10):

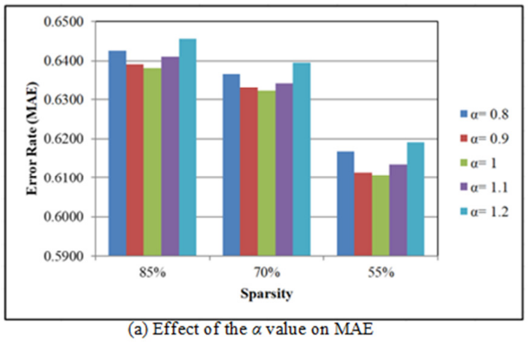
$$MAE = \frac{1}{N}\sum_{s=1}^{N}|r_{u,s} - p_{u,s}| \tag{5.9}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{s=1}^{N}\left(r_{u,s} - p_{u,s}\right)^2} \tag{5.10}$$

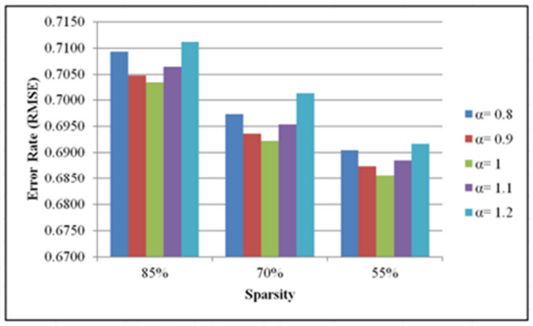
where $r_{u,s}$ and $p_{u,s}$ denote the existing rating value invoked by user *u* on Web service *s* and the predicted rating value of user *u* on Web service *s*, respectively. *N* is the number of predicted values.

5.2.1. Evaluation based on specificity-aware ontology-based clustering

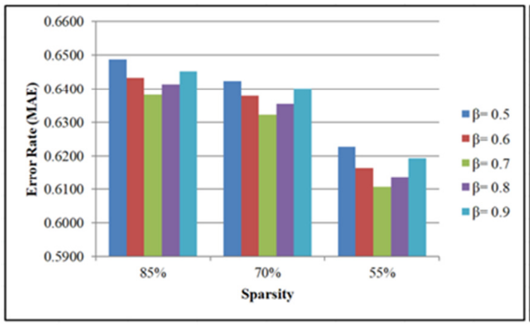
We measured the recommendation performance by changing different parameters of the ontology generation procedure. This showed that recommendation performance has a positive correlation with the clustering results.



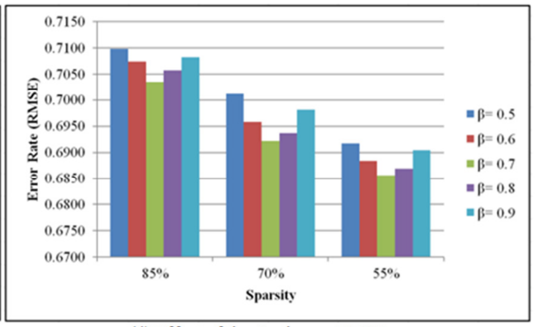
(a) Effect of the *α* value on MAE



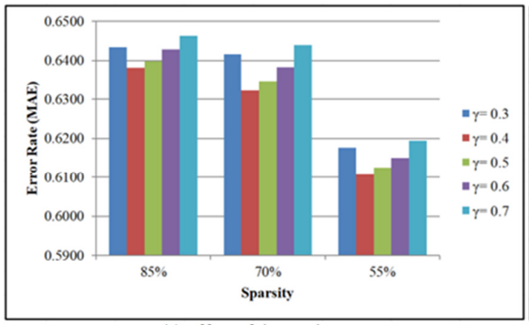
(b) Effect of the *α* value on RMSE



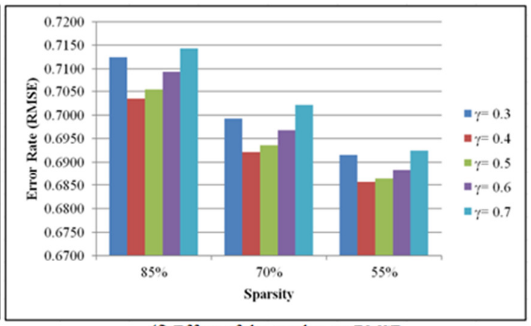
(c) Effect of the *β* value on MAE



(d) Effect of the *β* value on RMSE



(e) Effect of the *γ* value on MAE



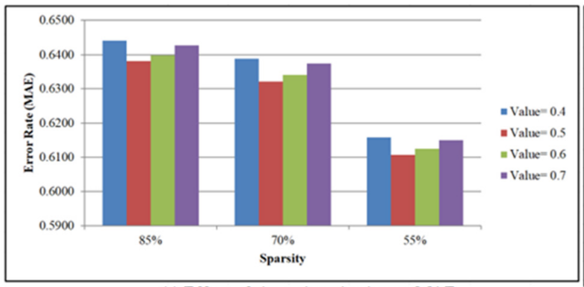
(f) Effect of the *γ* value on RMSE

Figure 5.5 Evaluation based on different ontology generation parameters

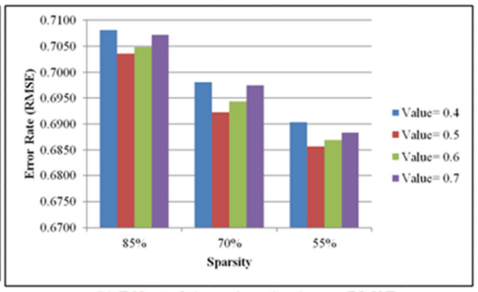
The self-specificity calculation depends on the α value (see equation (3.6)) and we varied α from 0.8 to 1.2. Figure 5.5(a) and Figure 5.5(b) show that the lowest MAE and RMSE values occurred when $\alpha = 1$.

Then, we changed the β value in the hybrid specificity calculation (see equation (3.9)) from 0.5 to 0.9. Figure 5.5(c) and Figure 5.5(d) show that the lowest MAE and RMSE values occurred when $\beta = 0.7$. This demonstrates that self-specificity provides the main contribution to hybrid specificity compared with context specificity.

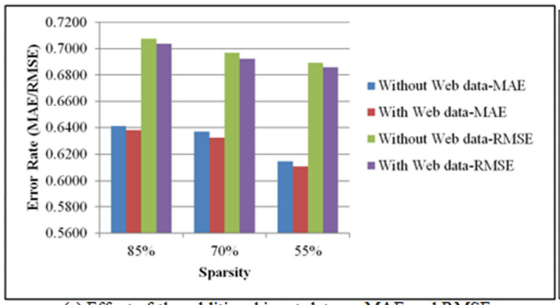
Then, we checked the effect of the γ value (see equation (3.14)) when combining the domain-specificity weight and the similarity weight to ascertain the optimal ontology hierarchy. The best performance was obtained when $\gamma = 0.4$ within the range 0.3 to 0.7 (Figure 5.5(e) and Figure 5.5(f)).



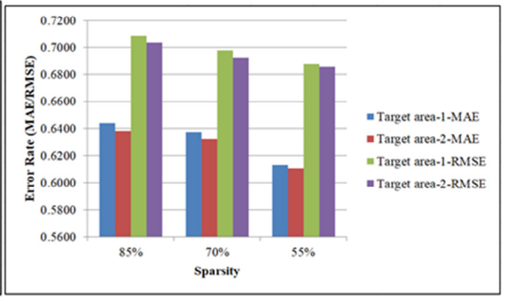
(a) Effect of the assigned value on MAE



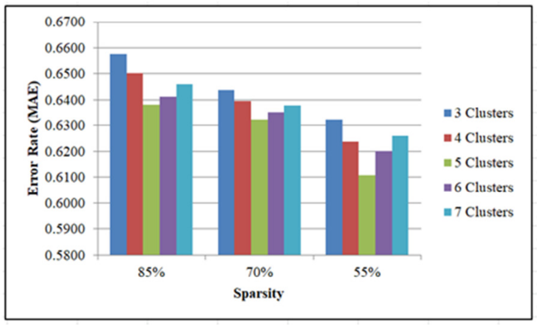
(b) Effect of the assigned value on RMSE



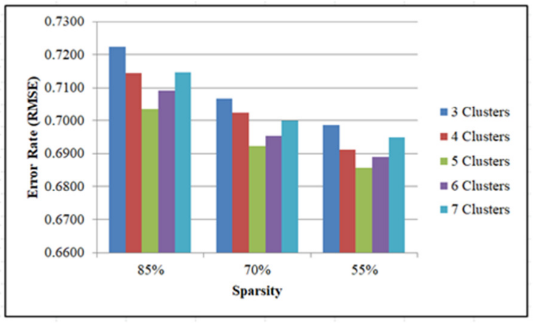
(c) Effect of the additional input data on MAE and RMSE



(d) Effect of the different target areas on MAE and RMSE



(e) Effect of the number of clusters on MAE



(f) Effect of the number of clusters RMSE

Figure 5.6 Evaluation based on different ontology generation parameters and ontology generation steps

When calculating the domain-specificity weight (see equation (3.10)), we should assign a direct value if no sibling terms are contained in the new term. That value was selected as 0.5 by experimentation in the range from 0.4 to 0.7 (Figure 5.6(a) and Figure 5.6(b)).

Then, to check how the Google extracted terms affected the recommendation performance, we performed the experiments shown in Figure 5.6(c). We established that adding more domain-related terms helped to improve the performance.

Then, we checked the performance by changing the target area by adding a new term to the existing ontology. We used Target area-1, which ranged from $Spec(t_{new})$ – 0.1 to $Spec(t_{new})$ + 0.1 and Target area-2, which ranged from $Spec(t_{new})$ – 0.3 to $Spec(t_{new})$ + 0.3. Target area-2 showed the lowest error rate as shown in Figure

5.6(d). It can be seen that expanding the target area provided more opportunities to select the best node but it reason for the computation time for the program.

Finally, evaluation was conducted by changing the number of clusters in the agglomerative clustering algorithm from three to seven. The lowest error rate occurred with the number of clusters set at five (Figure 5.6(e) and Figure 5.6(f)).

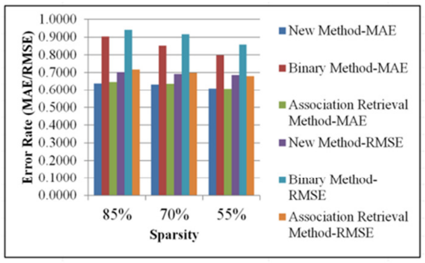
5.2.2. Evaluation based on sparsity-alleviating methods

Rather than applying clustering methods, some existing approaches for sparsity alleviation can be applied, for example, the association retrieval method [29] and the binary method (through assigning 0/1) [35].

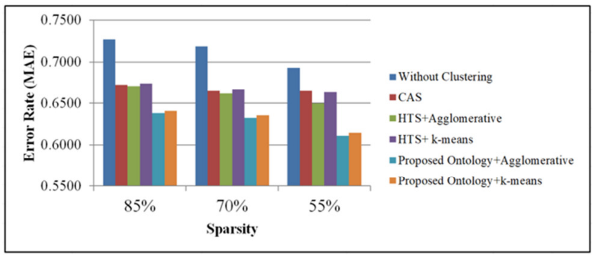
(i) In the association retrieval method, the sparsity problem was managed successfully and a new CF algorithm was proposed to improve recommendation performance. The transitive associations based on the user's feedback data were examined. A direct similarity and an indirect similarity between users were proposed and the similarity matrix was calculated through the relative distance between the users' ratings. To obtain the recommendation matrix, the association retrieval approach and the direct similarity matrix were combined and the sparsity problem was thereby managed with increasing recommendation precision.

(ii) The binary method proposed a simplified similarity measure (SSM) for CF recommendation to handle the sparsity problem. By converting the value of the user-item matrix into a binary preference value, similar groups of users were found and an SSM was proposed for speeding up the process for the sparsity problem. The so-called binary preference value means the feedback rating is greater than the average feedback level.

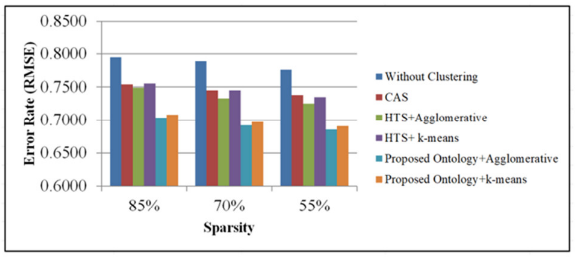
We compared those methods as shown in Figure 5.7(a). According to the results, our clustering approach showed better performance with the lowest MAE and RMSE values. It seems that the clustering approach can successfully identify the user preference than the above existing approaches. Clustering result can identify the domain of each service successfully than identifying an association between users. And also clustering result give higher accuracy than considering the binary preference.



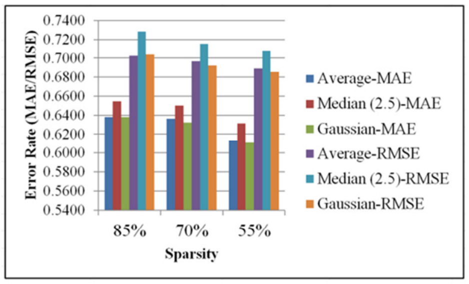
(a) Comparison with existing methods



(b) Evaluation using different clustering methods(MAE)



(c) Evaluation using different clustering methods(RMSE)



(d) Using different value assigning methods

Figure 5.7 Evaluation based on sparsity-alleviating methods

We then checked which clustering approach was best for sparsity alleviation. As shown in Figure 5.7(b) and Figure 5.7(c), we verified this without using clustering, using the CAS clustering method, using the HTS clustering method, and with our proposed clustering method, while changing the agglomerative clustering and k-means clustering. According to the results, our proposed approach with agglomerative clustering showed the lowest error rate. Finally, we conducted evaluation to ascertain the suitable value for assigning the 0 nonrated values in the matrix to increase the data matrix density. We evaluated this by assigning an average value of the user ratings in the specific cluster that the user had previously invoked, that is, the median value of the ratings (2.5) from the ratings range of 1–5, and by assigning a random value using a Gaussian distribution random number generator. As shown in Figure 5.7(d), the lowest error rate was given for Gaussian distribution.

According to the experimental results, the proposed approach shows better performance in not only the sparsity problem but also in the cold-start problem. When we are adding new Web services to the system, we don't have any user rating information related to them. But by applying the clustering process, we identified those services related domain cluster and alleviate the user ratings based on its domain and continue the recommendation process. So, we can successfully overcome the item-based cold-start problem through this.

Here, if contain only very few numbers (less than 5) of ratings from a user, we consider it as user-based cold start problem. By considering the available ratings' related domains we alleviate the sparsity and continue the recommendation. Using this process we can successfully overcome the user-based cold-start problem.

Chapter 6 Conclusion and Future work

We have first proposed a new domain-specificity-based ontology-generation method, with Web-service clustering being achieved via similarity calculations based on the generated ontology relationships. New machine filters are proposed for the similarity calculations that compare ontology relationships. This new approach is expected to help improve the clustering performance of Web services.

The new approach takes advantage of the information in specific terms instead of relying on more-general terms. Specific terms are more significant than general terms when classifying domain-related information. Previous approaches have focused on general terms and have not taken advantage of specific terms. Our measurements of specificity values showed that we could achieve high accuracy. Furthermore, according to our experimental results, our new information-theory-based approach gave improved validity and accuracy when compared with previous methods such as the edge-count-based, HTS and CAS approaches. We achieved superior clustering results in terms of precision, recall, F-measure, entropy and purity.

We then proposed a Web service recommendation approach using our novel clustering approach. The performance of existing Web service recommendation approaches is lacking due to the data sparsity and cold-start limitations. Our main objective here was to improve recommendation quality even if we lacked information about user-service ratings. We proposed to deal with these problems by applying a novel clustering approach based on domain specificity and it was used to improve the density of the user-service matrix. This clustering approach could successfully alleviate the sparsity problem and then the service user similarity was measured using PCC. Finally, new rating values were predicted using the updated user-service matrix and calculated PCC values, with the recommendation based on the predicted rating values. Our experimental results verify that our approach successfully eliminated the data sparsity and cold-start problems and significantly improved the prediction accuracy with the best recommendation performance.

In our future research, We hope to be able to investigate other significant aspects of service discovery and recommendation. We aim to apply deep learning-based methods to Web service recommendation and consider other CF problems, such as scalability, synonymy, and shilling attack.

References

[1] S. Dasgupta, S. Bhat, & Y. Lee, Taxonomic clustering and query matching for efficient service discovery, *In Proceeding of the 9th IEEE International Conference on Web Services*, Washington,DC, 2011, pp. 363– 370, doi:10.1109/ICWS.2011.112.

[2] Y. Xia, P. Chen, L. Bao, M. Wang , & J. Yang, A QoS-aware web service selection algorithm based on clustering, *In Proceeding of the 9th IEEE International Conference on Web Services*, Washington, DC, 2011, pp. 428–435, doi:10.1109/ICWS.2011.36.

[3] W. Chen, I. Paik and P.C.K Hung, Constructing a Global Social Service Network for Better Quality of Web Service Discovery, *IEEE Transactions on Services Computing*, 2013.

[4] G. Adomavicius and A. Tuzhilin, Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 2005, 17(6), pp.734-749.

[5] T. Peng, W. Wang, X. Gong, Y. Tian, X. Yang and J. Ma, A Graph Indexing Approach for Content-Based Recommendation System, In *Multimedia and Information Technology (MMIT)*, *Second International Conference* on, Kaifeng, 2010, pp. 93-97, doi: 10.1109/MMIT.2010.84.

[6] J. Li, J. Wang, Q. Sun, A. Zhou, emporal Influences-Aware Collaborative Filtering for QoS-Based Service Recommendation, *IEEE International Conference on Web Services* (ICWS), June, 2017, pp. 471-474.

[7] L. Yao, Q.Z. Sheng, , A.H. Ngu, , J. Yu and A. Segev, Unified collaborative and content-based web service recommendation. *IEEE Transactions on Services Computing*, , 2015, pp.453-466.

[8] Z. Zheng, H. Ma, M. R. Lyu and I. King, Collaborative Web Service QoS Prediction via Neighborhood Integrated Matrix Factorization, in *IEEE Transactions on Services Computing*, July-Sept. 2013, vol. 6, no. 3, pp. 289-299.

[9] Y. Zuo, J. Zeng, M. Gong et al., Tag-aware recommender systems based on deep neural networks[J]. *Neurocomputing*, 204, 2016, pp.51-60.

[10] D. Lau and J. Mylopoulos, Designing Web services with Tropos, *in: Proceedings of the 2004 IEEE International Conference on Web Services (ICWS 2004)*, 2004, pp. 306–313, doi:10.1109/ICWS.2004.1314752

[11] K. Elgazzar, A. E. Hassan, and P. Martin. Clustering WSDL documents to bootstrap the discovery of Web services, *in: IEEE International Conference on Web Services (ICWS 2010)*, Miami, Florida, USA, July 5–10, 2010, pp. 147–154, 2010, doi:10.1109/ICWS.2010.31

[12] L. Chen, L. Hu, Z. Zheng, J. Wu, J. Yin, Y. Li, and S. Deng. WTCluster: Utilizing

tags for Web services clustering, *in Service-Oriented Computing, Springer*, 2011, pp. 204–218. doi: 10.1007/978-3-642-25535-9_14

[13] Web Services Description Language (WSDL) 1.1, https://www.w3.org/TR/wsdl (accessed 03.03.2017).

[14] L. Chen, G. Yang, Y. Zhang and Z. Chen, Web services clustering using SOM based on kernel cosine similarity measure, *in: Proceedings of the 2nd International Conference on Information Science and Engineering*, Hangzhou, China, December 4–6, 2010, pp. 846–850. do i:10.1109/ICISE.2010.5689254

[15] J. Ma, Y. Zhang and J. He, Efficiently finding Web services using a clustering semantic approach, *in: Proceedings of the International Workshop on Context-enabled Source and Service Selection, Integration and Adaptation, Organized with the 17th International World Wide Web Conference, Beijing, China*, April 22, 2008, pp. 1–8. doi:10.1145/1361482.1361487

[16] W. Liu and W. Wong, Web service clustering using text mining techniques, *International Journal of Agent-Oriented Software Engineering*, 3(1), 2009, pp. 6–26. doi:10.1504/IJAOSE.2009.022944

[17] B.T.G.S. Kumara et al., Web service clustering using a hybrid term-similarity measure with ontology learning, *in: International Journal of Web Services Research (IJWSR)*, vol. 11, no. 2, 2014, pp. 24–45, doi: 10.4018/ijwsr.2014040102

[18] R.A.H.M. Rupasingha, I. Paik, and B.T.G.S. Kumara, Calculating Web service similarity using ontology learning with machine learning, *in: 2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, IEEE, 2015, pp. 1-8, doi: 10.1109/ICCIC.2015.7435686

[19] B.T.G.S. Kumara, I. Paik, H. Ohashi, Y. Yaguchi and W. Chen, Context-Aware Web Service Clustering and Visualization, *in: International Journal of Web Services Research*, 2014

[20] P. Buitelaar, An information-theoretic approach to taxonomy extraction for ontology learning, Ontology Learning from Text: Methods, Evaluation and Applications, *Frontiers in Artificial Intelligence and Applications*, IOS Press, Amsterdam, vol. 123, July, 2005, p. 15.

[21] A. Alnahdi, and S. H. Liu, Identifying characteristic attributes for estimating cost of service in service oriented architecture. *IEEE International Conference on Services Computing*, 2017, pp. 467–470.

[22] D. Kluver, M. D. Ekstrand, and J. A. Konstan, Rating-based collaborative filtering: algorithms and evaluation. *Social Information Access*. In: Brusilovsky P., He D. (eds) Lecture Notes in Computer Science, vol. 10100, Springer, Cham, 2018, pp. 344–390.

[23] L. Ren, W. Wang, An SVM-based collaborative filtering approach for Top-N web services recommendation. *Future Generation Computer Systems*, 2018, 78(Part

2), pp. 531–543.

[24] S. Meng, W. Dou, X. Zhang, and J. Chen, KASR: a keyword-aware service recommendation method on mapreduce for big data applications. *IEEE Transactions on Parallel and Distributed Systems*, 2014, 25(12), pp.3221-3231.

[25] S. Kant, T. Mahara, Merging user and item based collaborative filtering to alleviate data sparsity. *International Journal of System Assurance Engineering and Management*, 2018, 9(1), pp. 173–179.

[26] M. Polato, F. Aiolli, Exploiting sparsity to build efficient kernel based collaborative filtering for top-N item recommendation. *Neuro computing*, 2017, 268, pp. 17-26.

[27] J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang, Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, 2017, 69: 29-39.

[28] B. Ye, Y. Wang, Crowdrec: Trust-aware worker recommendation in crowdsourcing environments. *IEEE International Conference on Web Services (ICWS)*, 2016, pp.1-8.

[29] Y. Chen, C. Wu, M. Xie, and X. Guo, Solving the sparsity problem in recommender systems using association retrieval. *Journal of computers*, 2011, 6(9), pp. 1896-1902.

[30] H. Yildirim, M. S. Krishnamoorthy, A random walk method for alleviating the sparsity problem in collaborative filtering. *In Proceedings of the 2008 ACM conference on Recommender systems*, 2008, pp. 131-138.

[31] Z. Huang, D. Zeng, and H. Chen, A link analysis approach to recommendation under sparse data. *AMCIS 2004 Proceedings*, 2004, p. 239.

[32] R. Shrivastava, H. Singh, K-Means Clustering based solution of sparsity problem in rating based movie recommendation system. *International Journal of Engineering and Management Research (IJEMR) 2017*, 7(2), pp. 309-314.

[33] M. Ahmed, M. T. Imtiaz, and R. Khan, Movie recommendation system using clustering and pattern recognition network. *In: Computing and Communication Workshop and Conference (CCWC), IEEE*, 2018, pp. 143-147.

[34] B. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl, Application of dimensionality reduction in recommender systems: No. TR-00-043. *Minnesota Univ Minneapolis Dept of Computer Science*, 2000.

[35] L. H. Li, F. M. Lee, B. L. Chen, and S. F. Chen, A simplified method for improving the performance of product recommendation with sparse data. *IEEE 8th International Conference on Awareness Science and Technology*, 2017, pp. 318-323

[36] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, Item-based collaborative filtering recommendation algorithms. *In Proceedings of the 10th International*

World Wide Web Conference. ACM 2001, pp. 285–295.

[37] R. A. H. M. Rupasingha, I. Paik, B. T. G. S. Kumara, and T. H. A. S. Siriweera. Domain-aware web service clustering based on ontology generation by text mining. In *Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual*, IEEE, 2016, pp. 1-7.

[38] R. A. H. M. Rupasingha, I. Paik, and B. T. G. S. Kumara. Improving Web service clustering through a novel ontology generation method by domain specificity. In *Web Services (ICWS), 2017 IEEE International Conference on*, IEEE, 2017, pp. 744-751.

[39] R. A. H. M. Rupasingha, I. Paik, Domain information measure with novel ontology generation for Web service clustering. *IEICE Service Computing Branch Meeting Technical Report, 2017,* Vol. 117.

[40] M. Burstein, J. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, SNarayanan, M. Paolucci, B. Parsia, T. payne, E. Sirin, N. Srinvasan, K. Sycara, D. Martin(ed.), *OWL-S:* Semantic Markup for Web Services, *W3C Member Submission*, 2004.

[41] H. Lausen, A. Plleres, Web Service Modeling Ontology (WSMO), *W3C Member Submission*, 2005.

[42] C. Platzer, F. Rosenberg and S. Dustdar, Web service clustering using multidimensional angles as proximity measures, *ACM Transactions on Internet Technology TOIT* , vol. 9, no. 3, pp. 1–26, July 2009.

[43] I. Paik, E. Fujikawa, Web Service Matchmaking Using Web Search Engine and Machine Learning, *International Journal of Web Engineering*, SAP, 1(1), pp. 1-5, 2012.

[44] T. Wen, G. Sheng, Y. Li, & Q. Guo, Research on web service discovery with semantics and clustering. *In Proceeding of the 6th IEEE Joint International Information Technology and Artificial Intelligence Conference*, Chongqing, China 2011, pp. 62–67, doi:10.1109/ITAIC.2011.6030151

[45] Y. Lee, & C. Kim, A learning ontology method for RESTful semantic web services, *In Proceeding of the 9th IEEE International Conference on Web Services*, Washington, DC, 2011, (pp. 251–258). doi:10.1109/ICWS.2011.59

[46] B. T. G. S. Kumara, I. Paik, K. R. C. Koswatte, W. Chen, Improving Web service clustering through post filtering to bootstrap the service discovery, *International Journal of Services Computing* (ISSN 2330-4472) Vol. 2, No. 3, July – Sept. 2014. pp. 1-13.

[47] Q. Yu, H. Wang, L. Chen, Learning Sparse Functional Factors for Large-scale Service Clustering, *Proceedings of International Conference on Web Service (ICWS 2015)*, New York, June 27 2015-July 2 2015.

[48] J. Zhou and S. Li, Semantic Web service discovery approach using service clustering, *in Proc. International Conference on Information Engineering and Computer Science*, pp. 1-5, 2009.

[49] N. B. Mabrouk, N. Georgantas, and V. Issarny, Setbased Bi-level Optimisation for QoS-aware Service Composition in Ubiquitous Environments, *Proceedings of International Conference on Web Service (ICWS 2015)*, New York, June 27 2015-July 2 2015.

[50] C. Wu, W. Qiu, Z. Zheng, X. Wang, X. Yang, QoS Prediction of Web Services based on Two-Phase K-Means Clustering, *Proceedings of International Conference on Web Service (ICWS 2015)*, New York, June 27 2015-July 2 2015.

[51]C. Wuhui, and I. Paik. Toward Better Quality of Service Composition Based on a Global Social Service Network. *Parallel and Distributed Systems*, IEEE Transactions on 26.5 (2015): 1466-1476.

[52] D. Fang et al, An agility-oriented and fuzziness-embedded semantic model for collaborative cloud service search, retrieval and recommendation, *in:Future Generation Comp. Syst.*, vol. 56, 2016, pp. 11–26, doi: 10.1016/j.future.2015.09.025

[53] L.-L. Xie, F.-Z. Chen and J.-S. Kou, Ontology-based semantic Web services clustering, *in:Proceedings of the 18th IEEE International Conference on Industrial Engineering and Engineering Management*, 2011, pp. 2075–2079, doi: 10.1109/ICIEEM.2011.6035578

[54] H. Xia and T. Yoshida, Web Service recommendation with ontology-based similarity measure, *in: 2nd International Conference on Innovative Computing*, Information and Control, Kumamoto, 2007, pp. 412–412, doi:10.1109/ICICIC.2007.620

[55] M. Shi, J. Liu, D. Zhou, M. Tang, F. Xie and T. Zhang, A Probabilistic Topic Model for Mashup Tag Recommendation, *in: 2016 IEEE International Conference on Web Services (ICWS)*, IEEE, June, 2016, pp. 444–451, doi: 10.1109/ICWS.2016.64

[56] Y. Lei, Z. Jiantao, Z. Junxing, W. Fengqi and W. Juan, Time-aware semantic Web service recommendation, *in: IEEE International Conference on Services Computing*, New York, 2015, pp. 664–671, doi: 10.1109/SCC.2015.95

[57] N. Zhou, W.M. Gifford, J. Yan and H. Li, End-to-End Solution with Clustering Method for Attrition Analysis, *in: 2016 IEEE International Conference on Services Computing (SCC)*, 2016, pp. 363–370. doi:10.1109/SCC.2016.54

[58] W. Wong, W. Liu and M. Bennamoun, M, Tree-traversing ant algorithm for term clustering based on featureless similarities, *in: Data Mining and Knowledge Discovery*, vol. 15, no. 3, 2007, pp. 349–381, doi:10.1007/s10618-007-0073-y

[59] S.A. Caraballo and E. Charniak, Determining the Specificity of Nouns from Text, *in: Proceedings of the Joint SIGDAT Conference on EMNLP and Very Large Corpora*, 1999, pp.63-70.

[60] A. Aizawa, An information-theoretic perspective of tf-idf measures, *in:Journal of Information Processing and Management* 39(1), pp.45-65, (2003), doi:10.1016/S0306-4573(02)00021-3

[61] P.M. Ryu and K.S Choi, Determining the Specificity of Terms based on Information Theoretic Measures, insulin, 18(452.297), p.267., 2004.

[62] S. Zhang, W. Wang, and J. Ford, Learning from incomplete ratings using non-negative matrix factorization. *SIAM*, 2006, pp. 549–553.

[63] Y. Koren, Collaborative filtering with temporal dynamics. Communications of the ACM, 2010, 53(4), pp. 89-97.

[64] L. Yao, X. Wang, Q.Z. Sheng, B. Benatallah, and C. Huang, Mashup Recommendation by Regularizing Matrix Factorization with API Co-Invocations, *IEEE Transactions on Services Computing*, 2018.

[65] W. Yu, L. Li, X. Xu, D. Wang, J. Wang, and S. Chen, ProductRec: Product Bundle Recommendation Based on User's Sequential Patterns in Social Networking Service Environment. *IEEE International Conference on Web Services (ICWS)*, 2017, pp. 301-308.

[66] D. A. Adeniyi, Z. Wei, and Y. Yongquan. Automated web usage data mining and recommendation system using K-Nearest Neighbor (KNN) classification method. *Applied Computing & Informatics*, 2016, 12(1), pp. 90-108.

[67] B. Engelbert, M.B. Blanken, R. Kruthoff-Brüwer, and K. Morisse, A user supporting personal video recorder by implementing a generic Bayesian classifier based recommendation system. *IEEE International Conference on Pervasive Computing and Communications Workshops*, 2011, pp. 567-571.

[68] L. Qi, Z. Zhou, J. Yu and Q. Liu, Data-sparsity tolerant web service recommendation approach based on improved collaborative filtering. *IEICE Transactions on Information and Systems*, 2017, 100(9), pp.2092-2099.

[69] N. E. Evangelopoulos, Latent semantic analysis. Wiley Interdisciplinary Reviews: Cognitive Science, 4(6), 2013, pp. 683-692.

[70] Q. Xie, S. Zhao, Z. Zheng, J. Zhu, and M.R. Lyu, Asymmetric correlation regularized matrix factorization for web service recommendation. *IEEE International Conference on Web Services (ICWS)*, 2016, pp. 204-211.

[71] L. Qi, W. Dou, and X. Zhang, An inverse collaborative filtering approach for cold-start problem in web service recommendation. *In Proceedings of the Australasian Computer Science Week Multi conference, ACM* , 2017, p. 46.

[72] I. Barjasteh, R. Forsati, F. Masrour, A. H. Esfahanian, and H. Radha, Cold-start item and user recommendation with decoupled completion and transduction. *9th ACM Conference on Recommender Systems (RecSys '15).* ACM, 2015, 91-98.

[73] Semantic Annotations for WSDL, https://www.w3.org/TR/2006/WD-sawsdl-20060928/ (accessed 03.03.2017).

[74] XML Schema, https://www.w3.org/standards/xml/schema (accessed 03.03.2017).

[75] WordNet-A lexical database for English, https://wordnet.princeton.edu/ (accessed 06.02.17).

[76] D. Bollegala, Y. Matsuo and M. Ishizuka, Measuring Semantic Similarity between Words using Web Search Engines, *in: Proceedings of the 16th International World Wide Web Conference (WWW2007)*, May, 2007, pp.757-766, doi: 10.1145/1075389.1075392

[77] B.T.G.S. Kumara, I. Paik and W. Chen, Web-service clustering with a hybrid of ontology learning and information-retrieval-based term similarity, *in:Proceedings of the IEEE International Conference on Web Services, USA*, 2013, pp. 340–347, doi: 10.1109/ICWS.2013.53

[78] Y. Yaguchi and R. Oka, Spherical visualization of image data with clustering. *In Awareness Science and Technology (iCAST)*, 2012 4th International Conference on (pp. 200-206). IEEE.

[79] R. A. H. M. Rupasingha, I. Paik, and B. T. G. S. Kumara, Specificity-aware ontology generation for improving Web service clustering, *IEICE Transactions on Information and Systems*, 2018 Aug 1, 101(8), pp.2035-2043.

[80] Z. Zheng, H. Ma, M. R. Lyu, and I. King, Qos-aware web service recommendation by collaborative filtering. *IEEE Transactions on services computing*, 2011, 4(2), pp. 140-152.

[81] Moghaddam, S., Jamali, M., Ester, M. and Habibi, J., 2009, October. FeedbackTrust: using feedback effects in trust-based recommendation systems. *In Proceedings of the third ACM conference on Recommender systems* (pp. 269-272). ACM.

[82] A.M. Rashid, G. Karypis, and J. Riedl, Learning preferences of new users in recommender systems: an information theoretic approach. *Acm Sigkdd Explorations Newsletter*, 2008, 10(2), pp.90-100.

[83] K. Reschke, A. Vogel, and D. Jurafsky, Generating recommendation dialogs by extracting information from user reviews. *In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013, Vol. 2, pp. 499-504.

[84] D.B.Thomas, W. Luk, P.H. Leong and J. D. Villasenor, Gaussian random number generators, *ACM Computing Surveys (CSUR)*, 2007, 39(4), p.11.

[85] T. Symul, S. M. Assad, and P. K. Lam, Real time demonstration of high bitrate quantum random number generation with coherent laser light. *Applied Physics Letters*, 2011, 98(23), p.231103.

[86] S.C. Oh, D.Lee, and S.R Kumara, Effective web service composition in diverse and large-scale service networks. *IEEE Transactions on Services Computing*, 2008, 1(1), pp.15-32.

Publications

[Academic Journals (Refereed)]

[1] R. A. H. M. Rupasingha, I. Paik, and B. T. G. S. Kumara, Specificity-Aware Ontology Generation for Improving Web Service Clustering, *IEICE Transactions on Information and Systems*, 2018 Aug 1, 101(8), pp.2035-2043.

[2] R. A. H. M. Rupasingha, I. Paik, Alleviating Sparsity by Specificity-Aware Ontology-Based Clustering for Improving Web Service Recommendation, *IEEJ Transactions on Electrical and Electronic Engineering*, (In review).

[Proceedings at International Conferences (Refereed)]

[3] R. A. H. M. Rupasingha, I. Paik, Evaluation of Web Service Recommendation Performance via Sparsity Alleviating by Specificity-Aware Ontology-Based Clustering, *In 2018 9th International Conference on Awareness Science and Technology (iCAST),* IEEE, Fukuoka, Japan, 19-21 Sep. 2018, DOI: 10.1109/ICAwST.2018.8517251, pp. 279-284.

[4] T. Miyagi , R. A. H. M. Rupasingha, I. Paik, Analysis of Web Service Using Word Embedding by Deep Learning, *In 2018 9th International Conference on Awareness Science and Technology (iCAST),* IEEE, Fukuoka, Japan, 19-21 Sep. 2018, DOI: 10.1109/ICAwST.2018.8517167, pp. 336-341.

[5] R. A. H. M. Rupasingha, I. Paik, Improving Service Recommendation by Alleviating the Sparsity with a Novel Ontology-Based Clustering, *In 2018* 25th *IEEE International Conference on Web Services (ICWS),* San Francisco, CA, USA, 2-7 July 2018, DOI: 10.1109/ICWS.2018.00059, pp. 351-354.

[6] R. A. H. M. Rupasingha, I. Paik, and B. T. G. S. Kumara, Improving Web Service Clustering through a Novel Ontology Generation Method by Domain Specificity, *In Web Services (ICWS), 2017 24th IEEE International Conference on, IEEE*, Honolulu, HI, USA , 25-30 June 2017, DOI: 10.1109/ICWS.2017.134, pp. 744-751.

[7] R. A. H. M. Rupasingha, I. Paik, B. T. G. S. Kumara, and T. H. A. S. Siriweera, Domain-aware web service clustering based on ontology generation by text mining. *In Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual,* Vancouver, BC, Canada, 13-15 Oct. 2016, DOI: 10.1109/IEMCON.2016.7746301, pp. 1-7.

[Proceedings at Technical Committee (Non-Refereed)]

[8] R. A. H. M. Rupasingha, I. Paik, Evaluation of the Effectiveness of Recommendation while Managing the Data Density of the Web Service-User Preference, 09-10 Nov 2018, *IEICE technical report*, Japan, In press.

[9] R. A. H. M. Rupasingha, T. Yui, I. Paik, Readability Categorization of Japan EIKEN Document using Machine Learning with TF-IDF, 01 June 2018, *IEICE technical report*, Japan, In press.

[10] R. A. H. M. Rupasingha, I. Paik, Domain Information Measure with Novel Ontology Generation for Web Service Clustering, 02 June 2017, *IEICE technical report*, Japan, Vol. 117, Issue 75, pp 27-32.

[11] R. A. H. M. Rupasingha, I. Paik, and B. T. G. S. Kumara, Web Service Clustering through Calculating Semantic Similarity of Web Services using Novel Ontology Learning Method, 04 Nov. 2016, *IEICE technical report*, Japan, Vol. 116, Issue 287, pp 13-18.

[12] H. Sakai, R. A. H. M. Rupasingha, I. Paik, Performance Evaluation of Taxonomy Classification Using Machine Learning, 03 June 2016, *Poster presented at: IEICE technical report*, Japan.

[13] R. A. H. M. Rupasingha, I. Paik, and B. T. G. S. Kumara, Improving Web Service Clustering using Ontology Learning with Machine Learning, 03 June 2016, *IEICE technical report*, Japan, Vol. 116, Issue 76, pp. 23-28.