

1-1-2011

Modeling User's Non-Functional Preferences for Personalized Service Ranking

Rozita Mirmotalebi
Ryerson University

Follow this and additional works at: <http://digitalcommons.ryerson.ca/dissertations>

 Part of the [Computer Sciences Commons](#)

Recommended Citation

Mirmotalebi, Rozita, "Modeling User's Non-Functional Preferences for Personalized Service Ranking" (2011). *Theses and dissertations*. Paper 1387.

This Thesis is brought to you for free and open access by Digital Commons @ Ryerson. It has been accepted for inclusion in Theses and dissertations by an authorized administrator of Digital Commons @ Ryerson. For more information, please contact bcameron@ryerson.ca.

MODELING USER'S NON-FUNCTIONAL PREFERENCES FOR PERSONALIZED SERVICE RANKING

by

Rozita Mirmotalebi

B.E. in Computer Science, Azad University, Iran, 2007

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the program of

Computer Science

Toronto, Ontario, Canada, 2011

©Rozita Mirmotalebi 2011

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research.

ROZITA MIRMOTALEBI

Signature

Date

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

ROZITA MIRMOTALEBI

Signature

Date

MODELING USER’S NON-FUNCTIONAL PREFERENCES FOR PERSONALIZED SERVICE RANKING

Rozita Mirmotalebi
Master of Science, Computer Science, 2011
Ryerson University

ABSTRACT

As the number of web services is increasing on the web, selecting the proper web service is becoming a more and more difficult task. How to make the selection results from a list of services more customized towards users’ personal preferences and help users identify the right services for their personal needs becomes especially important under this context. In this thesis, we propose a novel User Modeling approach to generate user profiles on their non-functional preferences on web services, and then apply the generated profiles to the ranking process in order to make personalized selection results. The User Modeling system is based on both implicit and explicit information from the user. Also, this is a flexible model to include different types of non-functional properties. We performed experiments using a real web service dataset with values on various non-functional properties to show the accuracy of our system.

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my supervisor, Dr. Cherie Ding, professor of Computer Science department at Ryerson University. This work would not have been possible without her support and guidance when encountering obstacles. Her valuable suggestions, patience, and encouragement during all difficulties have greatly contributed to this research.

Also, I would like to thank the faculty of the Computer Science department, Dr. Alex Ferworn, Dr. Eric Harley and Dr. Isaac Woungang who have reviewed my thesis and given me valuable comments, which enabled me to improve my thesis.

I would like to express my deepest gratitude to my husband, Ali Mirmotallebi, for his patience during these past two years.

TABLE OF CONTENTS

AUTHOR’S DECLARATION.....	ii
ABSTRACT.....	iii
ACKNOWLEDGEMENTS.....	iv
LIST OF FIGURES	vii
LIST OF TABLES.....	viii
LIST OF ACRONYMS	ix
CHAPTER 1	1
INTRODUCTION.....	1
1.1 Background of Generic User Modeling and Web Service Selection	1
1.2 Motivation and Problem Statement	3
1.3 Proposed Methodology.....	4
1.4 Objectives of the Thesis	5
1.5 Thesis Outline.....	6
CHAPTER 2	8
LITERATURE REVIEW.....	8
2.1 Introduction	8
2.2 Generic User Modeling	8
2.3 Personalization and Recommender Systems	11
2.4 Web Services and Quality of Services	15
2.5 Web Service Selection and Recommendation.....	18
2.6 Understanding WSDL in a UDDI Registry	19
2.7 Summary.....	20
CHAPTER 3	21
USER MODELING FOR WEB SERVICE SELECTION	21
3.1 Introduction	21
3.2 User Model Properties.....	21
3.3 User Model Data Collecting	25
3.3.1 Explicit User Preference Data	26
3.3.2 Implicit User Preference Data	27

3.4	Implicit User Modeling	27
3.4.1	System Architecture	27
3.4.2	Log Data Processing.....	29
3.4.2.1	Fuzzy Representation of User Preferences.....	30
3.4.2.2	Implicit Data Formulation.....	31
3.4.3	Personalized Service Ranking	42
3.4.3.1	Similarity Calculation Between User Modeling system and Services	42
3.4.3.2	Total Similarity Calculation.....	49
3.5	Summary.....	50
CHAPTER 4	51
IMPLEMENTATION AND EXPERIMENT	51
4.1	Introduction	51
4.2	Implementation.....	51
4.2.1	Programming Environment	51
4.2.2	Interface Design to Get User's Explicit Preferences	51
4.3	Experiment.....	57
4.3.1	Dataset	58
4.3.2	Sample Query and Results.....	62
4.4	Evaluation and Analysis of User Modeling System	63
4.5	Summary.....	67
CHAPTER 5	68
CONCLUSIONS	68
5.1	Conclusion	68
5.2	Main Contributions.....	69
5.3	Future Works	69
APPENDIX A – Hierarchy list of all Quality of Service (QoS)[58]	70
APPENDIX B – Experiment design for all users	71	
APPENDIX C – Results	73	
REFERENCES	75

LIST OF FIGURES

Figure 1.1- The three processes involved in user model [4].....	2
Figure 2.1- A Generic User Modeling Architecture [3]	11
Figure 2.2- The relation between SOAP, WSDL, and UDDI [52]	16
Figure 2.3- Key elements for supporting QoS in web services [44]	18
Figure 3.1- The structure of our selected non-functional properties	25
Figure 3.2- System Architecture	28
Figure 3.3- provider Location flowchart.....	45
Figure 4.1- User Modeling system interface schema (GUI schema)	52
Figure 4.2- Layer 1 of User Modeling system interface: sign-in.....	53
Figure 4.3- Layer 2 of User Modeling system interface: sign-up.....	53
Figure 4.4- Provider history scales	54
Figure 4.5- Layer 3 of User Modeling system interface: user preference (1).....	55
Figure 4.6- Layer 3 of User Modeling system interface: user preference (2).....	56
Figure 4.7- Layer 4 of User Modeling system interface: search.....	57
Figure 4.8- The selected keywords and number of services used in the experiment.....	60
Figure 4.9- Top 10 result on “finance” for user <i>test</i>	63
Figure 4.10- Baseline result for “finance”	63
Figure 4.11- MAP calculation on different keywords	66
Figure 4.12- MAP comparison on different number of preferences.....	66

LIST OF TABLES

Table 4.1- Properties and ranges.....	54
Table 4.2- Experiment design preferred	61
Table 4.3- <i>test</i> user profile	62
Table 4.4- <i>MAP</i> calculation on each keyword	65

LIST OF ACRONYMS

AH: Adaptive Hypermedia

AP: Average Precision

API: Application Programming Interface

GUI: Graphical User Interface

HCI: Human Computer Interaction

HTTP: Hypertext Transfer Protocol

MAP: Mean Average Precision

QoS: Quality of Service

SOAP: Simple Object Access Protocol

UDDI: Universal Description, Discovery and Integration

UM: User Modeling

UMS: User Modeling Server

WSDL: Web Service Description Language

XML: Extensible Markup Language

CHAPTER 1

INTRODUCTION

1.1 Background of Generic User Modeling and Web Service Selection

User Modeling (UM) is an area of research which can help make systems' behaviours more personalized by focusing on users' preferences and characteristics. UM is useful for web users in different aspects, such as, personalized search, recommender systems, educational delivery systems, assisting web browsing or purchasing activities, etc. [1].

The process of personalization for each user is based on his/her individual characteristics [2]. A personalized system attempts to collect information about each user. Then by using that information and various personalization techniques (e.g. machine learning), a user model can be built. The user model includes user profiles for many different users and is based on the type of the service. Therefore there are two main steps for personalization: user modeling (making user models) and adaption (personalization based on user models) [3]. In the user modeling step, we can consider three responsibilities: user model initialization, user model updating and system process [4]. In the initialization process, users need to initialize properties, and the user model includes all the properties, and their descriptions. After initialization, the system needs to update the user profile for the specific user. User profile could be either knowledge-based or behavior-based. Knowledge-based approach dynamically matches users to the closest model by using questioners and interviewers to obtain the user's information. Behavior-based approach uses machine-learning techniques and behavior to build the user model [5]. Finally in the system

process the user model will complete all the properties not answered by the user from user's history data. Now the user model would be ready to be used for various purposes (Figure 1.1).

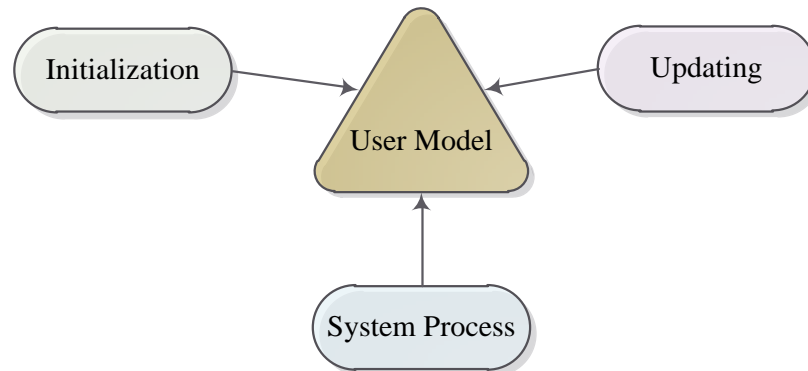


Figure1.1- The three processes involved with the user model [4]

Web services are self-contained systems that exist on the network which could be the infrastructure for other applications and software systems. Nowadays, web services are becoming more and more popular on the internet. Since the information volume is increasing rapidly, it is difficult to find and choose a specific data for the users. Also we can say if people have common interests on an item, the item chosen by one user could be recommend to another user. For instance, two people like same type of movies, then if one person likes a movie which hasn't been watched by another person yet, this movie could be recommended to the second person because he/she may like it considering he/she is sharing interest with the first person. Therefore understanding users' behaviours and characteristics while selecting and using services is important, and it could help them to find their desired services [6]. There are two general steps for web service selection: discovering and matching based on functional requirements, and then filtering and ranking based on non-functional requirements. Functional requirements specify the capability that the system must be able to support. These requirements are specified through keywords or tags e.g. hotel,

weather, economics, etc. Non-functional requirements are about the system itself and the quality of the system performance, e.g. location, language, availability, reliability, response time, etc.

As a novel idea, in this research we design a User Modeling system for web services that can capture the history of users' preferences on non-functional properties and can be used later for the personalized service ranking. Implementing User Modeling is possible in two ways: as a part of a user-adaptive application, called user shell [7], or as an independent User Modeling Server (UMS) [8]. The choice of our work is the latter: UMS: the central storage information which could be used with other applications.

1.2 Motivation and Problem Statement

The earlier research efforts on web service selection mainly used functionality based matching techniques for selecting services [6]. Because of the large number of web services, matching services with each other only on functional requirements is a huge and tedious task. Therefore, filtering and selecting services not only by functional attributes, but also by non-functional attributes emerged [9]. For service selection based on users' non-functional requirements, the biggest obstacle is that users may not define their requirements accurately and completely, or they may not want to spend time on defining these requirements. Therefore, if we could build a User Modeling system which could capture user's preferences on various non-functional properties and use this user model in the selection process, the ranking result could be more personalized and accurate. Also in the meantime, user efforts of defining their non-functional requirements each time when they have service requests could be largely reduced. Personalization based on user models have been proved to be very effective for a lot of web

related activities such as searching, browsing, product shopping or other e-Commerce activities [10,11,12].

To the best of our knowledge, there is not much work done on User Modeling for web service selection. Therefore, this motivates us to propose a User Modeling approach to get user profiles of their preferences on non-functional properties of web services, and use the user model to make service selection and ranking more personalized. We consider different aspects of user preferences and also different ways of getting this kind of information. For instance, a user may have some restrictions/preferences (e.g. location of the service provider, language of the service output, etc.), about which kind of results he/she wants to receive upon his/her request. To know the user preferences, we could either ask them to enter into the selection system and save into their personal profiles, or check their selection and invocation histories to make a reasonable guess. Then by making some experiments on user preferences in the User Modeling system and offering services based on them, we will find that the results would be more accurate and close to user's preferences and request.

1.3 Proposed Methodology

Nowadays, choosing a specific web service (e.g. flight booking, weather, etc.) from a long list of web services with the same functionality is really difficult. To solve this problem, we decided to design and implement a personalized ranking system. This system is an intermediary located between users and services. The User Modeling is the first step, and the generated user model could understand users' preferences from their past invocation histories and from characteristics of invoked services. Then the user model is applied to the service ranking process so that the personalized ranking result could be returned to users.

To get user preferences, we need to collect service descriptions in the web service definition language (WSDL) documents and quality attribute information from the service level agreement (SLA) or monitored data, and also information from registries and usage data from log files.

In this work by using User Modeling in web service selection, we could offer a new list of web services to the user which is closer to his/her requirements and preferences. From our study we see that most of the works until now deal with some specific functional and non-functional requirements; however, more general preferences of users on various non-functional properties are not taken into consideration. One of the ways for considering user general preferences is User Modeling based on invocation history. User preferences could be captured based on two different aspects: implicit feedback or explicit feedback from the user. The implicit approach comes from usage patterns, and user preferences based on implicit feedback may not be accurate because the usage data may not be complete and user identification may not be correct [13]. Explicit feedback is more accurate; however, it places a burden on users and sometimes users may not want to provide it, and also it may not be reliable (depends on users' mood and personality). Since usually on the web the implicit data is easier to get whereas users are reluctant to give explicit feedback, it is more common to use implicit feedback for User Modeling, and then combine with explicit feedback if it is available [14]. In this thesis, we consider both; however, in the experiment because of the lack of the log data, we only use explicit feedback information for our testing.

1.4 Objectives of the Thesis

The main goal of this work is to make a User Modeling system for web service selection. We propose an integrated and powerful User Modeling system for all users which later will help

them search and find the services they want based on their preferences. This User Modeling system will cover various non-functional user preferences.

The objectives of this work are given as follows: Firstly, we propose a mechanism to collect the user preference data – explicitly and implicitly, and build user models. Secondly, we define which non-functional preferences are important for users to select services. Finally, we give a sample application about how the user model can be used; in our case, we just implement it for personalized service selection; however, it could be used in the other facets of web service selection.

The major contributions of our work are as follows:

- To the best of our knowledge, it is a novel idea to propose a User Modeling system on user's non-functional preferences on web services. Those are some general preferences which are not restricted to a particular domain.
- We defined which non-functional preferences are important for the users to select services. The list of properties and the formulas of how to calculate use preferences on them from implicit user feedback was explained. The system also supports explicit preference definition from the user.
- We discussed a sample application about how a user model can be used, which is personalized service selection and ranking in our implementation. The result of the User Modeling could also be used in the other facets of web service selection.

1.5 Thesis Outline

The rest of the thesis is organized as follows:

Chapter 2 – Literature Review: this chapter reviews works related to this thesis in different areas such as: User Modeling, web service selection and personalized search and recommendation systems.

Chapter 3 – User Modeling for Web Service Selection: this chapter begins with the overview of User Modeling for web service selection and goes on to discuss all the features about the required non-functional properties. It further explains the method of collecting implicit and explicit data. It also discusses the system architecture model that we have used for our implementation. This chapter finishes with similarity and ranking calculation between users and services.

Chapter 4 – Implementation and Experiment: this chapter gives details about implementation steps and the experiment design. The chapter continues with the result analysis and comparisons between different approaches.

Chapter 5 – Conclusion: this chapter concludes and summarizes our work. Also the benefits of this work and the points of improvements are highlighted. It points out some suggestions and potential future works.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The main purpose of this chapter is to give detailed information of research works in different areas: i) User Modeling in general, ii) User Modeling for the web and personalized search or recommendation systems, iii) web services, iv) recommendation systems for web services.

2.2 Generic User Modeling

The history of User Modeling can be traced back to the works of Allen and Cohen in 1979 according to Kobsa [15, 16, 17]. User Modeling was an application system in different application environments. In the mid-eighties, developers began to separate the user model from components as an independent component [18] and User Modeling started to be a separate part of an application. In 1986, the “General User Modeling System (GUMS)” was proposed by Tim Finin [19]. GUMS can save facts and information provided by the application system and can answer queries by concerning currently saved assumptions about the user. But it never has been used by any application and just opened a new future to User Modeling systems [20]. In 1990, Kobsa made a User Modeling shell system which was the first shell system. A shell system is part of just one system [7]. There are many User Modeling shell systems after GUMS with different characteristics, such as: UMT [21], PROTUM [22], TAGUS [23], um [25]. After that User Modeling servers have become more popular. They are central server systems that can

communicate with different applications simultaneously. In contrast with shell systems, server systems are more independent [25]. There are different examples of User Modeling servers such as: BGP-MS [26, 7], UMS [27], etc. Also, there was an evolution from academic User Modeling to commercial User Modeling and Group Lens [28] is an outstanding example.

User Modeling is one of the research areas which are related to many other different areas such as human computer interaction (HCI), artificial intelligence, social psychology, etc. [4] which can help the computer systems to interact with the users easily. HCI has focused on the interaction between users and computers and is not only based on interfaces, but also is about personalization; User Modeling can provide some answers to HCI [1].

User Modeling can be combined with different areas such as: adaptive hypermedia (AH), web personalization, e-learning, etc. [29]. In adaptive hypermedia, the modeling is done automatically by the system, and the user has little knowledge about how to proceed with the work. Therefore, AH tailors all the user's goals, interests and knowledge and will offer the most appropriate ones to the user [30].

Applying the user model to personalized search, which is very similar to our goal in this research, begins with user login. This means that when we ask the user to login, all of the following steps in the system are simply based on the user's preferences. User's general preferences can help the search engines to clarify the query. Also users' personal information in different systems cannot transfer to other systems and sometimes users have to access their information in multiple systems [31], thus, by using the user model this problem could be fixed.

There are different types of personalization systems. Feng and Junghoo [32] have proposed a personalized framework user model to formalize users' interests and preferences with their click history on search results. And at the end they computed page rank on multiple topics for ranking

the results. The work in [32, 33] is concerned with user model personalization on short-term and long-term user needs documents without considering the level of document correlations and is simply derived from query matching. Because the studies show that most of the users are not willing to make explicit feedback on search results and their interests [32], search engines try to collect user preferences implicitly and automatically. In [10] Sun et al. proposed a mechanism for personalized ranking based on implicit feedback regarding user preferences from web access logs. They showed how site structure can make better results from access logs in CiteSeer academic search engine and also they showed personalized ranking is better than the other ranking features. Our work is a combination of explicit and implicit feedback and preferences from the user.

Personalized systems require knowledge about users which can be expressed as a user model which is a set of preferences from a user. Figure 2.1 is a generic (application-independent) User Modeling architecture proposed by Kobsa and Fink (2006) [3, 34]. This model contains two main functional components: user-adaptive application and User Modeling, which is our main focus in this research. User Modeling is responsible to make and maintain user models for storing different information about users such as: their goal, plan, and preferences. Also, we can make more assumptions from existing information. A user-adaptive application will deliver personalized services (e.g. personalized flight recommendation) to the users via internet, and is based on User Modeling systems.

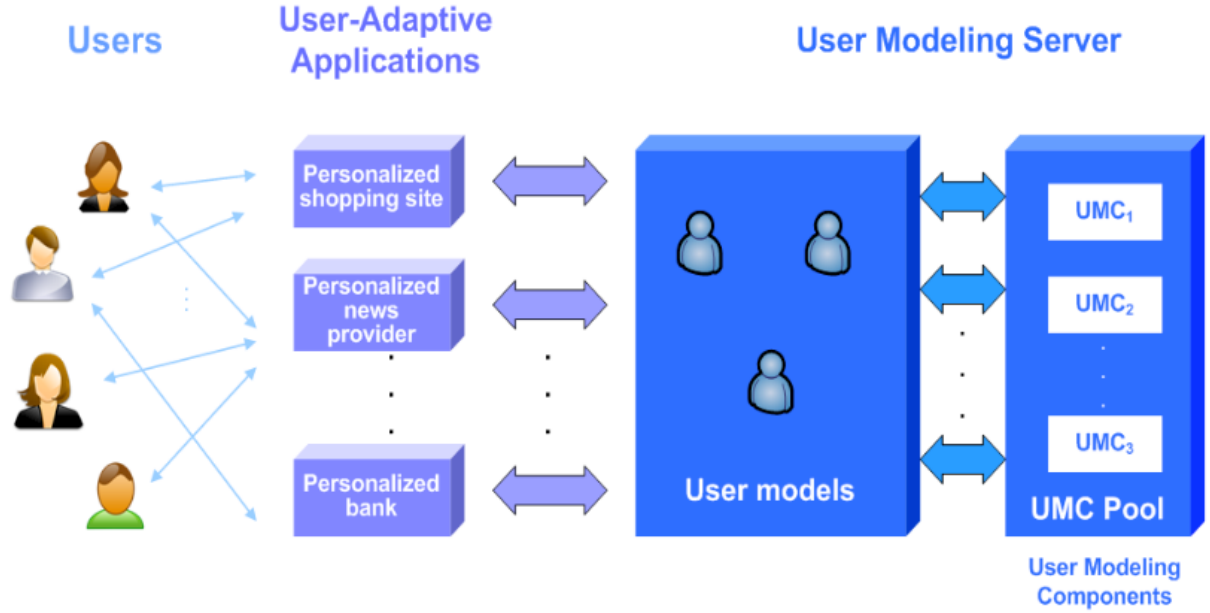


Figure 2.1- A Generic User Modeling Architecture [3]

In this thesis, we have developed a user-friendly interface for adding, updating and removing preferences with different properties in real time. In addition, our User Modeling system is utilized to support personalized ranking process to select and invoke web services.

2.3 Personalization and Recommender Systems

Personalized systems require accurate modeling of users' interests and needs. Such systems can make tailored results for the user. A typical example of a personalized system is a recommender system [35], which recommends items which users may like based on user profiles. The results will be collected by exploiting users' past history and interaction with different resources. Recommender systems have different types. One of the classifications contains three categories: content-based filtering [35, 36], collaborative filtering [35, 37,38], and hybrid recommendation. In content-based filtering, the new recommendation is based on previous ratings and items selected by the user. Content-based methods are designed mostly for

text-based items and they contain some keywords in the model. There are different techniques to implement content-based filtering: Bayesian classification [39], machine learning techniques, including clustering, decision trees and neural networks [40], data mining [41] etc. The content-based recommendation method has several limitations:

- Limited content analysis: there are some limitations for the features related to objects, so the content must be in a form that could be parsed automatically like texts, or features that could be assigned manually. For example, automatically assigning for multimedia, such as movies and images is difficult, because they are not text. Moreover, because of the limitation in resources manually assigning is not practical. Another problem with content-based approach is that, if you are assigning the same keywords or features to two different items, they are not distinguishable, so you cannot find which one is better and closely related (semantically).
- Overspecialization: it occurs when a user does not have any experience with the item. For example, when a user does not have any experience with horror movies and wants to try one for the first time, the system cannot recommend anything so it offers a random choice.
- New user: a new user has no ratings, and therefore cannot receive the correct recommendation.

The collaborative filtering method comes from the ratings of previous similar users. In this method, the systems will try to find peers of users. For instance, when two users have the same tastes in movies, they would rate the same movies similarly. Algorithms for collaborative filtering can be grouped in two categories: memory-based (or heuristic based) and model-based [42]. Memory-based algorithms use heuristics and are based on the previous ratings by the users. Model-based algorithms use the collection of ratings for learning the model. After that we can

use the algorithm for making rating prediction. For this prediction, we use probabilistic models in two ways: Cluster models and Bayesian models. In cluster model, we cluster the same users in a group and their ratings are independent. In the Bayesian model, each item in the domain is a node in a Bayesian network, and the state of each node is the rating values for each item. Collaborative filtering systems have some limitations:

- New user problem: the same problem as mentioned above in content-based systems for new users. One solution for that is using a hybrid approach, which is the combination of content-based and collaborative systems.
- New item problem: collaborative filtering is based on user preferences, so when we have a new item without any rating it would be difficult.
- Sparsity: it means that the number of ratings is less than the predictions, and the availability of users in the same place and time. One solution is to collect their profile information while collecting the similarity information.

Hybrid recommendation is the combination of content-based recommendation and collaborative filtering which can help to decrease the number of problems that we explained before for both content-based and collaborative methods. We can classify the combination of content-based and collaborative methods in four ways [6]:

- 1- Performing content-based and collaborative methods separately and then just combining the prediction.
- 2- Adding the content-based characteristics to collaborative systems.
- 3- Adding collaborative characteristics in content-based systems.
- 4- Merging content-based and collaborative characteristics in one model.

Recommender systems can give users high quality recommendations if they have been made based on user preferences. The task of collecting user's preference information is typically called the User Model (UM) [8], and can be done in two ways: explicit and implicit. Explicit data is collected by asking from users explicitly. Implicit data refers to applying different mechanisms to collect data by monitoring observable user's behaviours [8]. Hybrid data is the combination of those two approaches to make the UM more accurate because explicit data needs the user's effort to complete the forms. However, researches show that users do not devote much effort to complete them. Also implicit data sometimes involve incorrect translation from user's behaviours. But the combination of these two approaches can make a concrete solution to the users' preferences [40], which is the method that we use in our user model solution for web services. The characteristics of the user model can lead to better recommendations with higher quality to the users. For instance, if our UM has a lot of accurate information which is up-to-date, it will affect the quality of the top k results. The quality means offering services which are closer to the user's preferences and interests. As we mentioned before, our User Modeling system can be used in different areas such as recommender systems. And based on our former explanations on recommender systems and User Modeling, each recommender system can support a collection of user models. It means that user models are specialized for specific content or products in the recommender systems (flights, movies, etc.), and one of the specific implementation techniques for recommender systems that we explained before (collaborative filtering, content-based, etc.) [35]. This work focuses on making a User Modeling system for web service selection.

2.4 Web Services and Quality of Services

According to W3C (World Wide Web Consortium), web service “is a software system designed to support interoperable machine-machine interaction over a network” [43]. Web services are self-containing systems that exist on the internet, and they could be building blocks for other applications and software systems on the Internet [44]. They can interact with each other regardless the fact that they are based on different platforms and run by different protocols. Today, web services represent the core technology for e-businesses. They have the ability of making the whole business process a reality over the network. This brings customers, suppliers and partners together to successfully achieve the business goal.

Web services can interact with other software agents by using XML messages to exchange information. Some examples of the web services are travel and hotel reservation, auction, ticket purchase, and so on. There are three defined XML standard technologies for web services: WSDL, UDDI, and SOAP [45]. Web service Definition Language (WSDL) is a protocol or language for describing the interfaces of web services and is the format for processing them in the machine. The shortcoming of this technology is the limitation as different versions of services attempt to communicate dynamically over the network. As McIlrath and Martin (2003) demonstrated [46], to overcome this problem, we must bring semantics to web services. This can be accomplished by developing new technologies to express and fully describe contents, objects and their interrelations on the internet. Some of the languages that were created for this purpose are Resource Description Framework (RDF) [47], RDF Schema (RDFS) [48], Web Ontology Language (OWL) [49] and Web Ontology Language for Services (OWL-S) [50]. The overall benefit of using these languages is to express the semantics of the web services; for instance, OWL-S has been developed to describe web services by defining general domain classes and

properties [51]. The main goal of Semantic web service is giving a robust and meaningful description to the web services. Semantic Web extends web services' functionality by giving information well-defined meanings, and by helping both human and computing systems cooperate with each other.

SOAP (Simple Object Access Protocol) is an XML-based standard over Hypertext Transfer Protocol (HTTP) and other internet protocols to exchange web services' information between requesters and providers. UDDI (Universal Description, Discovery and Integration) is a standard for indexing business registry of web services on the internet, and provides keyword-based searching on services, which is called tModels. Figure 2.2 shows the relation between three standards of web service [52]. Commonly UDDI enables businesses to publish their service listings over the internet and also makes it possible to interact with each other. UDDI registry is a repository for web services and could be used as a database for all of their information. Providers publish WSDL for their services in the UDDI registry, and requesters can access the services by using SOAP. Furthermore UDDI can reduce the integrity and time consumption between them [53].

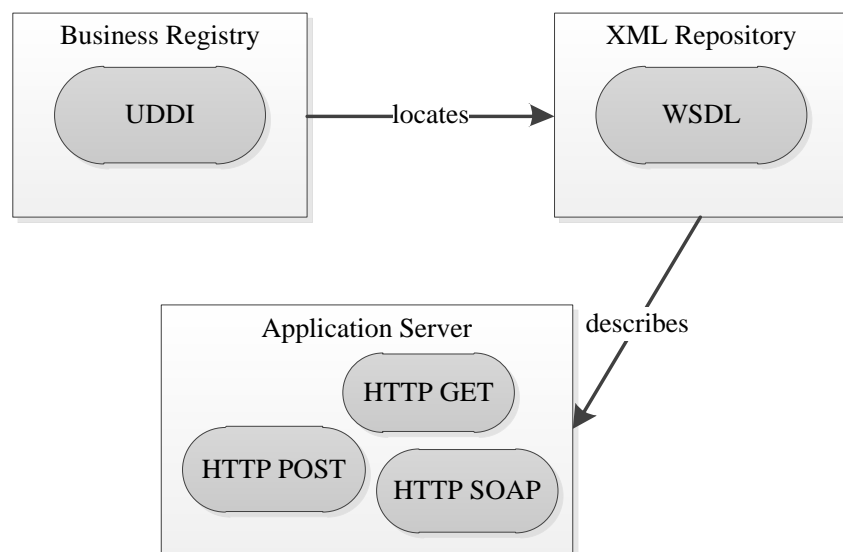


Figure 2.2- The relation between SOAP, WSDL, and UDDI [52]

Users' interaction with the web services could be stored in log files which are located in the client side or server side and contains web usage by the users. The client side log file can collect all of the usage data for the individual user on all of his/her interactions with different web services. On the other hand, the server side log file can collect all of the interactions from different users on the web services which are hosted by the server. There are different types of usage data, such as, browsing history, click-through data, and so on [54, 55, 38]. In our work, we have used the log files for collecting user's data implicitly from user's history, and also we have made our log files work with our user model. We will go through them in the next chapter.

Quality of Service (QoS) refers to the mechanism to enable web services to respond to invocations in a proper way. QoS tries to make an appropriate answer for any requests with mutual expectations from providers and users. Each service may offer a different choice of quality of services based on the technical requirements; also here QoS is part of non-functional requirements. QoS is the most important and critical part of web services because of the dynamic and unpredictable nature of web services, and all efforts on web services aim to find a way to adjust QoS based on users' tastes to make better results for requests. According to Mani [44], there are some key elements for supporting QoS in web services (Figure 2.3). Also more detail of different attributes with all of the classifications in Quality of Services can be found in Appendix A [58].

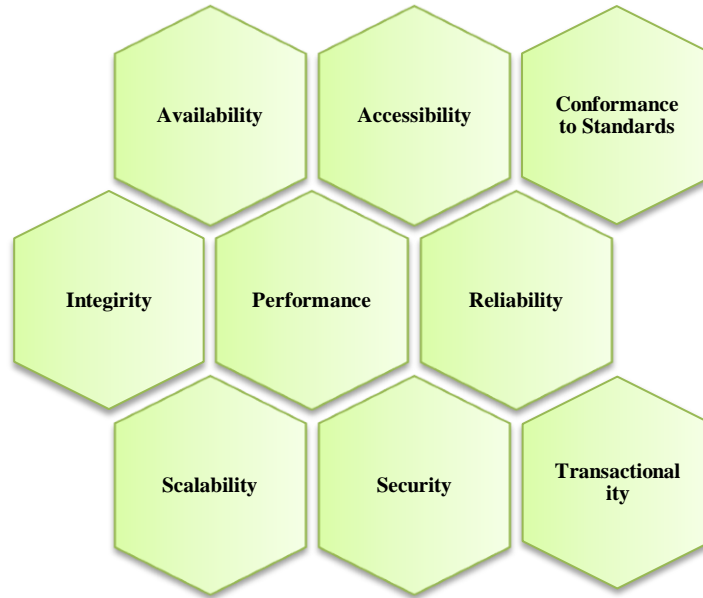


Figure 2.3- Key elements for supporting QoS in web services [44]

2.5 Web Service Selection and Recommendation

Recommendation algorithms can be used in web services in different ways: for service selection and ranking, for QoS prediction, etc. In [57], Averbakh, Krause and Skoutas calculated the scores of web services based on the feedback from similar users. In [13], Zheng et al. used collaborative filtering for the prediction purpose. They proposed WSRec, a Web service recommender system to predict the QoS value a user may experience based on other similar users' experiences. In [38], Zhang proposed a collaborative filtering based service selection algorithm. The similarity calculation is based on similar users on the past QoS queries and actual invocations. The recommendations to the users are based on their matching similarity degrees with similar users in the past. In our work we consider the previous history of a user and we recommend items by only considering his/her history.

2.6 Understanding WSDL in a UDDI Registry

To complete the required information in our user model implicitly, the knowledge about source of data is required. WSDL and UUDI are two main data sources for our work, which are related to each other. WSDL is a description of services and contains a definition of operations and messages for binding them to each other. UDDI registry provides a method to describe businesses (providers) and their services. UDDI supports different types of services and does not have direct access to WSDL; therefore mapping WSDL into a UDDI registry is needed. There are four primary data types in the UDDI registry [52]:

- BusinessEntity: includes information about business (business is a provider in our case)
- BusinessService: includes technical and business description for a web service (BusinessService is a web service in our case)
- BindingTemplate: references to one or more tModel.
- tModel: is a technical definition for a service.

WSDL documents are divided in two types:

- Service interface: a description of a service about types, import, message, portType, and binding elements.
- Service implementation: contains service elements and at least has a reference to service interface and describes an instance of a service.

Now if we want to publish a WSDL for UDDI registry, service interface in WSDL will be published as a tModel in a UDDI registry and service implementation will be published as a businessService. This is the way that WSDL document will map to a UDDI registry.

2.7 Summary

In this chapter, we explained the research works related to the User Modeling and web service personalization. To the best of our knowledge, there is no comprehensive works on making a user model for web services. In our approach as a novel idea, we will design a user model for web services, explicitly and implicitly, which can be used in different aspects of web service selection. Our user model will cover non-functional requirements on any kind of web services by emphasizing quality of services through a central mechanism.

CHAPTER 3

USER MODELING FOR WEB SERVICE SELECTION

3.1 Introduction

As we mentioned before in this research, we are going to propose a unique and comprehensive User Modeling system for web service selection. In this chapter, we will explain the process in different steps from collecting data to ranking services and presenting the results to the user. The chapter is organized as follows: (i) selecting user profile properties, (ii) collecting user data for user profiles implicitly and explicitly, (iii) calculating the similarity between user model and web services, and (iv) performing the personalized ranking process based on user's request.

3.2 User Model Properties

The User Modeling for web services should be based on user requests and needs to cover all of the user preferences for any kinds of web services. To address this issue, the generic user model for representing users' profiles is required, and a User Modeling system has a set of user profiles for different users. A user profile contains the collection of relevant characteristics related to a specific user, and a set of user profiles comprises the user model in the web environment. The User Modeling system could be used for personalization; in our work we are using it for web service selection.

As we mentioned before, web service properties are categorized in two groups: functional and non-functional properties. Functional properties include semantic information, inputs, outputs, pre-conditions and effect, etc. Non-functional properties typically focus on qualities and

characteristics of the service. Quality of Service (QoS) is the most common type of non-functional properties, which include: availability, security, reliability, performance, etc. The whole list can be found in Appendix A [58]. Non-functional requirements enforce constraints on the design time. In general, functional properties contain the description of what the system actually can provide, and non-functional properties capture how the system is supposed to do the functional part. For example, in the case of a flight booking service, booking a ticket is the functionality and might be constrained by a reliable agent (reliability as a non-functional property) and by response time of the invocation (as a non-functional property) [59]. Our User Modeling system only captures the non-functional preferences of the users.

The first step to make a generic User Modeling system for web services is collecting a set of non-functional properties, which are crucial for decision making to express user requirements when selecting web services. The most important source for collecting non-functional service properties is from the service provider or a third party (third party such as: network monitoring agency, a registry, a certification authority, etc.). After our study on different sources for providers and third parties, we made a list of mandatory non-functional properties, which cover preferences on providers and preferences on services. We save their values from different services in our provider information repository and service information repository.

Provider-related properties can be represented as a 4-tuple component:

Provider Properties: $\langle pName, pLocation, pHistory, pPopularity \rangle$

- **pName:** identifies the names of the provider's preferred by the user, and is based on the effective top level domain of the server, and they are available in the UDDI registry or a third party. The type of data for pName is a set of nominal data.

- *pLocation*: determines the location of each provider based on the physical location of the provider's server. The information can be found in the UDDI registry or the third party. The provider location itself includes two elements as defined here:

Provider Location: $\langle pContinent, pCountry \rangle$

pContinent and *pCountry* show the continent and the country of the provider respectively and each provider can have one location at a time. The type of the stored data is set of nominal data.

- *pHistory*: represents the provider history which is the life time for each provider. This information could be found under each provider's description in the UDDI registry or a third party.
- *pPopularity*: shows the provider popularity among various requestors. Popularity could be measured by how frequent services from this provider are invoked or the number of service requestors by looking at the log file or provider invocation history [4].

Service related properties can be represented as an 8-tuple component:

Service Properties: $\langle sLanguage, sLiftTime, sFreshness, sRating, sPopularity, sAvailability, sResponseTime, sDocumentation \rangle$

- *sLanguage*: states the language(s) of the output from a service. This information can be found in the WSDL repository on the server side.
- *sHistory*: indicates the service history, which means the length of the time that the service is existing and can be found in the UDDI registry.
- *sFreshness*: expresses the service freshness, which is the last time that the service has been updated.

- sRating: denotes the service rating, which is the average rating for a service by all users in its life time. A larger number means a higher rating.
- sPopularity: shows the service popularity. The calculation is the same as provider popularity. A larger number shows a higher popularity.

As we stated before, non-functional properties of services include various QoS attributes. In our work because of the dataset that we used for the experiment (Seekda) and also the available QoS values from in this dataset, we have selected three QoS parameters for service properties:

- sAvailability: represents the service availability, which measures the degree of service accessibility and functionality when the user requests it. A larger number is better.
- sResponseTime: is the elapsed time between the end of a request to a service and the beginning of the service's response time. In other words, it is the total waiting time before the service begins to respond to the user. A smaller response time is better because it means the waiting time for the user is less.
- sDocumentation: shows whether the service is well documented in WSDL or not (such as: API explanation). A well-documented service has a more complete functional description than services without or with poor documentation, so that the functional matching for these kinds of services is more accurate and the services have higher chances to be retrieved by requestors. The type of the data is a set of nominal data.

Figure 3.1, shows the hierarchical structure of the fundamental non-functional properties that we have chosen for our User Modeling system in this research:

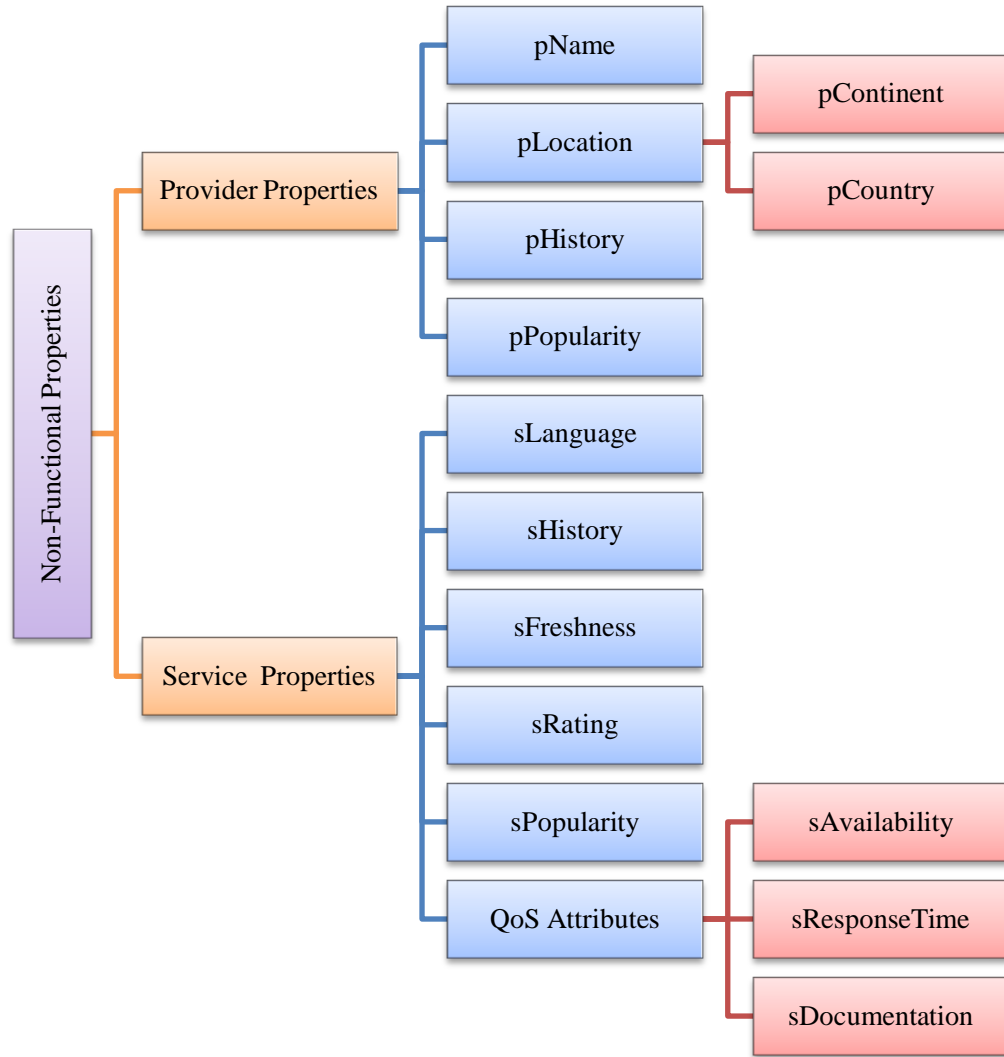


Figure 3.1- The structure of our selected non-functional properties

3.3 User Model Data Collecting

In the previous section, we have reviewed all of the required properties for user preferences in our User Modeling system with their data types. Now we need to collect the source data for all of the properties.

As we explained before, explicit data collection has some problems such as: some users are not willing to answer all the questions in a form, they don't have any idea or background about

the question, and sometimes their answers do not reflect their interests. Thus we need to propose an implicit user data collection in the user model. Here both explicit and implicit data collecting are explained.

3.3.1 Explicit User Preference Data

In the personalization mechanism explicit preference declarations would be possible by asking the user (by indicating the consent) to complete a series of forms and questions. Then we can sort and save all of them in the user profile. In other words, the explicit user model stores the information elicited from the user's favourites [60].

In a user-centric service selection system designing a good user interface is very important. The reason is that, on one hand, it should be compatible with all of data sources, and on the other hand, it should not put too much burden on users to complete complex forms. Therefore, in the user model design time, we have considered these points and have added some guided process for the user [61].

Upon navigating our profile input forms, a user can explicitly specify his/her preferences in a personalized system for a set of pre-defined questions with different options. Or if the user leaves them unspecified, in which case, it is the system's responsibility to complete them implicitly by considering the user's previous history information. We have divided our forms into two parts: part one is the information related to the provider and part two is the information about the service. The system will save users' preferences under their names in the database, and the users can change them later on. The details of the form design will be discussed in Chapter 4.

3.3.2 Implicit User Preference Data

For implicit data collection the same preference information as in implicit can be extracted or calculated from the usage logs.

As we explained in Chapter 2, there are different types of usage data that can be used in User Modeling systems. In search engine systems different types of data collection such as the click-through data and searching history data have been used to increase the accuracy in the search results [55, 38]. Similarly in our work we need to analyze service invocation logs to discover user's interests and preferences and extract them [38].

3.4 Implicit User Modeling

In this section we will go through the implicit User Modeling process in detail, which is one of the most important parts of the work. We will first present our system architecture design, then describe the User Modeling process, and finally discuss how we can use the generated user model to implement a personalized service ranking system.

3.4.1 System Architecture

The main components of the system reside on the server side, which in our case is an extended UDDI registry, and the user interface components are located on the user side. This architecture has extended from the Zhang's work [38] based on our assumptions and definitions. The architecture model of our system is shown in Figure 3.2.

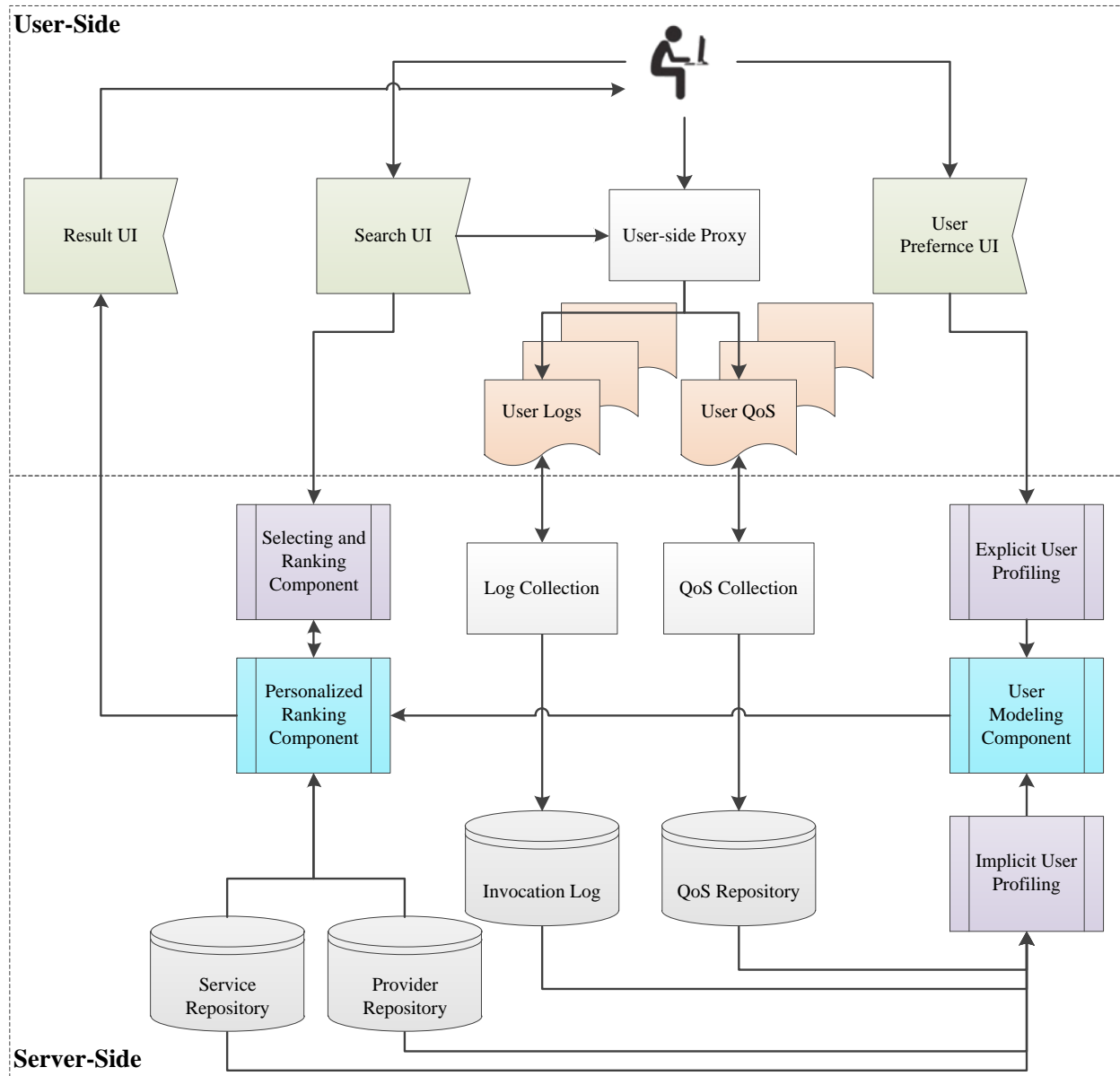


Figure 3.2- System Architecture

On the user side, we have a user-side proxy to relay all the user requests to the server. It can collect the usage data in the log files. Also the proxy collects the actual QoS data by monitoring invocations and that information will be saved in the QoS repository. The proxy component can be installed on the client machine.

There are two main processes performed on the server side: User Modeling process, and personalized ranking process. The User Modeling process includes three components: explicit profiling, implicit profiling, and user modeling component. In fact, implicit and explicit components are two sub-components of user modeling component. A user can have access to the user preference interface to complete his/her preferences. All of the preferences will go to the explicit profiling component for processing. On the other hand, implicit user profiling component will complete all of the user's preferences by using logs and repositories. Then the user modeling component will combine all of the collected preferences from both explicit and implicit user profiles, and will send them to personalized ranking component.

The personalized ranking process includes two components: service selecting and ranking component and personalized ranking component. When the user searches for a service through the search UI, the request will be sent to proxy to save them in the log and in the meantime, the request goes to the selecting and ranking component. At first all the available services just based on the user's request will be selected and ranked in this component and then those services will be sent to the personalized ranking component. The personalized ranking component will re-rank those services based on the generated user preferences from user modeling component and information from service repository and provider repository. The top k ($k=10$) personalized results will be sent to the result interface for the user's knowledge.

3.4.2 Log Data Processing

Now we have designed the system architecture and identified all of the relationships between different components. If a user enters all the necessary information for generating the user profile, the entered information will be saved as the user profile. It is a simple process, and

it will be explained in more detail later when we discuss our system implementation and the interface design. If a user does not enter any information or only enters partial information, the implicit profiling component will be called to generate the implicit user profile. In this section, we will explain how we process the log data.

3.4.2.1 Fuzzy Representation of User Preferences

Before we move on to discuss the implicit User Modeling process, we need to explain the data types for representing the user preferences. For instance, we want to know the user preference on the provider life time. Suppose the unit for life time is year. We can ask this question in different ways. We could ask user to enter the exact number of years (e.g. 3 years), or a range of the years (e.g. > 3 years and < 10 years), or a fuzzy value which represents a range of years in a relative way (e.g. long). To define an exact number, users have to have some knowledge about the data distribution, which is usually difficult for the users. Therefore a more reasonable option and an easier way for users would be to use fuzzy values. In the user interface we demonstrated user preferences by an n-point Likert scale for required attributes, which includes *no preference* as well [62]. For this particular property (i.e. provider life time), we have selected a 4-point Likert scale (*no preference, short, medium, long*). The reason is that by analyzing the year of establishment for all of the providers from our testing dataset, we found that four is a good choice. We will explain our chosen fuzzy Likert scales for all other properties in Chapter 4.

3.4.2.2 Implicit Data Formulation

As we previously mentioned, our user model includes a list of properties which are common for all requestors. Here is the list of provider related and service related properties:

- **Provider-related:**
 - o Provider Name
 - o Provider History
 - o Provider Location
 - o Provider Popularity
- **Service-related:**
 - o Service History
 - o Service Language
 - o Service Rating
 - o Service Popularity
 - o Service Freshness
 - o Service QoS attributes (QoS value)

Suppose there are M users registered in our system, N providers in the provider directory and Q services in the service directory. Also we should remember each service is associated with one provider. The set of users could be represented as $U = \{u_1, u_2, u_3 \dots u_M\}$ where u_i is the i -th user in the set. The set of providers could be represented as $P = \{p_1, p_2, p_3 \dots p_N\}$ where p_j is the j -th provider. In addition, the set of services could be represented as $S = \{s_1, s_2, s_3 \dots s_Q\}$ where s_k is the k -th service in the set. Because of the relation between

providers and services we can show them as a correlation: for each $p_j \in P$ there is a list of services $\{s_{j1}, s_{j2}, s_{j3} \dots s_{jh}\} \subset S$, where $1 \leq jh \leq Q$.

In our implicit User Modeling process, for each property, we define how we can calculate the user preference based on the past invocation history. The following describes the calculation methods for all the non-functional properties we include in this work.

- **Provider**

- o **Provider Name**

To calculate user u_i 's preference on the provider name, we need to track user's history. The first step is going through the invocation log to count all the services invoked by this user in the past. Then we count all of the providers associated with those services from the provider directory. By calculating the invocation frequency on provider p_j by user u_i and comparing this with a threshold (T_{PNF}) value we can know whether this user has preference on this particular provider. We go through the list of all providers to check their invocation frequencies to get the set of providers the user preferred (PN_{u_i}). The provider name frequency is calculated as follows:

$$PNF_{p_j} = \frac{NIv_{p_j}}{NIv}, \quad (3.1)$$

where PNF_{p_j} represents the invocation frequency on provider p_j , NIv_{p_j} represents the number of invoked services from p_j by u_i , and NIv in the denominator represents the number of all invoked services by u_i from all providers. If $PNF_{p_j} \geq T_{PNF}$, then this provider would be in the list of provider name preferences for u_i , but if none of the providers has met the threshold, then the provider name preference property for u_i would be *no preference*. T_{PNF} is a pre-defined

number, which shows the lower boundary in the conditions to check the frequency. The value could be assigned in the implementing time based on the number of services and providers in the repositories. All the possible results on provider name on user u_i (PN_{u_i}) are as follows:

output(PN_{u_i}): A nominal set $\{p_1, p_2, \dots, p_n\} \subset P$, "no preference"

o **Provider History**

The steps for calculating the user preference on the provider history are similar as those for the provider name, but in the provider directory we should find the established date for each provider instead of the provider name. Before going through the actual calculation, we need to explain the method of calculating provider's life time. The calculation formula is as follows:

$$LT_{p_j} = \text{Current Date} - \text{Date Established Time}_{p_j} \quad (3.2)$$

Where LT_{p_j} represents the life time of provider p_j . Based on the values of LT_{p_j} , we can categorize the providers in different nominal sets: providers with *short* lifetime, *medium* lifetime and *long* lifetime. The boundary range for each nominal set would be defined in the user model implementation time by considering all of the providers' life time. We can now compute the provider history frequency by using this formula:

$$PHF_{(short/medium/long)} = \frac{NIv_{(short/medium/long)}}{NIv}, \quad (3.3)$$

where PHF represents the history frequency for each category (*short*, *medium*, *long*), how many services have been invoked for each category. NIv represents the number of invoked services by u_i from each category: *short*, *medium* or *long* life time, and NIv in the denominator

represents the total number of all invoked services by user u_i . After calculating PHF for each lifetime category, the result needs to be compared with a threshold (T_{PHF}) value. For instance, if the value of PHF_{short} is bigger than the threshold value, then the user preference on provider history would be *short*. If PHF values for two categories are greater than the threshold, the user preference on provider history would be both of them, for all the rest of conditions user preference would be *no preference*. Below shows the possible results for provider history and PH_{u_i} represents provider history on the user u_i .

Output(PH_{u_i}): "short", "medium", "long", "no preference"

o **Provider Location**

Since in our work we don't need all of the details on the address, we have decided to divide it into two parts: the continent and country for each provider. If the user has not entered any preferences for the provider location, we need to calculate it based on his/her history. After discovering all of the services which have been invoked by this user in the past from his/her invocation log, the providers' location could be found in the provider information directory. The formulas for provider location frequency on continent x ($PLF_{continent_x}$), and provider location frequency on country y ($PLF_{country_y}$) for u_i is as follows:

$$PLF_{continent_x} = \frac{Nlv_{continent_x}}{Nlv} \quad (3.4) \quad , \quad PLF_{country_y} = \frac{Nlv_{country_y}}{Nlv}, \quad (3.5)$$

where $Nlv_{continent_x}$ represents the number of invoked services from $continent_x$ by user u_i , and $Nlv_{country_y}$ represents the number of invoked services from $country_y$ by user u_i . Also,

Nlv in the denominator represents the number of all invoked services by user u_i . A provider can have only one location. Then, by looking at those providers who have met the threshold (T_{PLF}), $PLF_{continent_x} \geq T_{PLF}$ or $PLF_{country_y} \geq T_{PLF}$, we can make two lists of the provider locations for user u_i . However, if none of the providers have PLF values greater than or equal to the threshold T_{PLF} , then the provider location preference for user u_i would be *no preference*. All the possible results on provider location on user u_i (PL_{u_i}) are as follows:

Output(PL_{u_i}): Two nominal sets $\{p_1, p_2, \dots, p_n\} \subset P$, "no preference"

o **Provider Popularity**

As we explained earlier, provider popularity is the level of popularity on each provider by all users. We need to find invoked services by u_i from the invocation log and find their related providers from the service directory. Afterwards, from the provider directory, we need to count all of the providers for this user. From the invocation log we should count all of the service invocations on provider p_j . The formula for provider popularity calculation is as follows:

$$PP_{p_j} = \frac{Nlv_{p_j}}{Nlv}, \quad (3.6)$$

where PP_{p_j} represents the provider popularity on provider p_j , and Nlv_{p_j} represents the number of all service invocations on provider p_j by all users, also Nlv in the denominator represents the number of all service invocations on all providers by all users. Similar to the provider history, we need to categorize the provider popularity in three ranges: *low*, *medium* and *high*. Then we should calculate provider popularity frequency with this formula:

$$PPF_{low/medium/high} = \frac{NIv_{(low/medium/high)}}{NIv}, \quad (3.7)$$

where PPF represents the popularity frequency for each category (*low/medium/high*), NIv represents the number of invoked services by user u_i for services with *low*, *medium* or *high* popularity, and NIv in the denominator represents the number of all invoked services by user u_i . After that, based on the results for *low*, *medium* or *high*, if one of them has passed the threshold (T_{PPF}): $PPF \geq T_{PPF}$, it defines the user preference on the provider popularity. If the combination of two of them have met the threshold, both of them are considered as user preference, for the rest of conditions the user preference would be *no preference*.

There could be multiple preferences if more are bigger than threshold. All the possible results on provider popularity on user u_i (PP_{u_i}) are as follows:

Output(PP_{u_i}): " *low* ", " *medium* ", " *high* ", " *no preference* "

- **Service:**

- **Service History**

Service History is about the life time of the service, therefore we need to go through the invocation log and find all the invoked services by u_i . We will then be granted access to the detailed information about each service in the service directory. After finding all the invoked services by user u_i the services need to be categorized. As we described in the provider history, the life time for each service could be in a different scope: *short*, *medium* or *long*. The formula for calculating the history frequency on u_i for s_k is as follows:

$$SHF_{short/medium/long} = \frac{Nlv_{(short/medium/long)}}{Nlv}, \quad (3.8)$$

where SHF represents the history frequency in each category (*short*, *medium*, *long*), Nlv represents the number of invoked services by user u_i for services with *short*, *medium* or *long* life time, and Nlv in the denominator represents the number of all invoked services by u_i . The difference between SHF and PHF is that in SHF the provider for each service is not important. After calculating the history frequency, we need to compare them with the threshold (T_{SHF}) to check whether it satisfies the condition $SHF \geq T_{SHF}$. If two of them have met the threshold the user preferences would be both of them are considered as user preferences, in the rest cases user preference would be *no preference*. The summary of all possible results could be shown as:

Output(SH_{u_i}): "*short*", "*medium*", "*long*", "*no preference*"

o Service Freshness

Similar to the service history, we can find the service's last updated time from the service directory. To calculate service freshness, evaluating the service's last updated time is needed, where LU_{s_k} represents the last time that this service has been updated:

$$LU_{s_k} = Current\ Date - Lat\ Updated\ Time_{s_k} \quad (3.9)$$

After calculating LU_{s_k} , we can categorize services in different sets: *low*, *medium* and *high*. The boundary could be assigned in the implementation time based on all of the services. The

unit could be day, month or year. In this step, calculating service freshness frequency is needed: (*low*, *medium*, *high*)

$$SFF_{low/medium/high} = \frac{NIv_{(low/medium/high)}}{NIv} \quad (3.10)$$

In this formula, SFF shows service freshness frequency for each category (*low/medium/high*), NIv represents the number of invoked services by user u_i for services with *low*, *medium* or *high* last updated time, and NIv in the denominator represents the number of all invoked services by user u_i on all providers. Then, by comparing all of the numbers by threshold (T_{SFF}), $SFF \geq T_{SFF}$; if it is greater than the threshold, it indicates the user preference on service freshness for user u_i . If two of them have met the threshold those two would be user preferences, for the rest of conditions user preference would be *no preference*. All the possible results on service freshness on user u_i (SF_{u_i}) are as follows:

Output(SF_{u_i}): "*low*", "*medium*", "*high*", "*no preference*"

o **Service Language**

We have a list of languages in our database which is shown as: $\{l_1, l_2, l_3 \dots l_F\} = L$. Tracking the language(s) of each service would be the same as service history. The difference is that in the service directory we are looking for service languages instead of the service established time. The next step is calculating language frequency on invoked services for each language by user u_i :

$$SLF_z = \frac{Nlv_{(language_z)}}{Nlv} \quad (3.11)$$

In this formula, SLF_z shows the service language frequency on language z , Nlv represents the number of services with $language_z$, and Nlv in the denominator represents the number of all invoked services by user u_i . The next step is comparing all the results with threshold (T_{SLF}). If $SLF_z \geq T_{SLF}$, then the language used in calculation is the preferred language(s) by the user u_i . However, if the result was empty, the preference on the service language for user u_i would be *no preference*. All the possible results on service language on user u_i (SL_{u_i}) are as follows:

Output(SL_{u_i}): A nominal set $\{l_1, l_2, \dots, l_n\} \subset L$, "no preference"

o Service Rating

As we explained before overall service rating comes from rating or voting from all the users who have invoked this service. Access to user rating would be possible from the service repository. At first, we have to go through the log file to find those services that have been invoked by user u_i by looking at the ratings through the service directory. For different services and considering some boundaries, we should divide them into different scopes of *low*, *medium* and *high*. It is possible in the implementation time, depending on the volume of ratings, to decide on the boundaries. The formula to calculate user rating frequency is as follows: (*low*, *medium*, *high*)

$$SRF_{low/medium/high} = \frac{Nlv_{(low/medium/high)}}{Nlv}, \quad (3.12)$$

where SRF represents user rating frequency for each category (*low/medium/high*) for user u_i , Nlv represents the number of services invoked by u_i with *low*, *medium* or *high* rating, and Nlv in the denominator represents the number of all invoked services by u_i . The next step is comparing all the results with threshold (T_{SRF}). If just one or two categories have met the threshold, $SRF \geq T_{SRF}$, then results show the rating, but for the rest of conditions the results would be *no preference*. All the possible results on service rating on user u_i (SR_{u_i}) are as follows:

$$\mathbf{Output}(SR_{u_i}): "low", "medium", " high", "no preference"$$

o Service Popularity

Service popularity is similar to provider popularity. The formula for service popularity calculation is as follows:

$$SP_{s_k} = \frac{Nlv_{s_k}}{Nlv}, \quad (3.13)$$

where SP_{s_k} represents the service popularity on service s_k , and Nlv_{s_k} represents the number of all invocations on all services by all users. Also Nlv in the denominator represents the number of all service invocations on all providers by all users. Similar to provider history, we need to categorize the provider popularity in three ranges: *low*, *medium* and *high* the only difference is instead of calculating all invocations on providers we should compute it for services in the service directory. The frequency formula is represented as:

$$SPF_{low/medium/high} = \frac{NIv_{(low/medium/high)}}{NIv}, \quad (3.14)$$

where SPF_{u_i} represents service user rating frequency for user u_i , NIv represents the number of invoked services by u_i for services with *low*, *medium* or *high* popularity, and NIv represents the number of all invoked services by user u_i . After $SPF \geq T_{SPF}$, if one or two of them have met the threshold (T_{SPF}), this would be the user preference. For the reset of possible conditions the user preference would be *no preference*. All the possible results on service popularity on user u_i (SP_{u_i}) are as follows:

$$\mathbf{Output}(SP_{u_i}): "low", "medium", "high", "no preference"$$

o QoS attributes

In Chapter 2, we have listed all of the quality of service attributes for web services. At first, we should find out a set of attributes that the user has a requirement on by using this formula:

$$att_h = \frac{NQ_{att_h}}{NQ} \quad (3.15)$$

This formula will return the preferred attribute att_h that the user u_i has preference on. NQ_{att_h} represents the number of queries having a request on att_h by user u_i , and NQ in the denominator represents the number of all submitted queries by u_i . For each acquired attribute from 3.15, there is a definition on value fuzzy terms, for example *low*, *medium* or *high*. After setting up the boundaries for all attributes' values we should find the user's preference on

att_h by using formula 3.16, where $QoSF_{att_h}$ shows the selected att_h attribute frequency in the User Modeling system for user u_i for service in category (*low/medium/high*):

$$QoSF_{att_h} = \frac{Nlv_{att_h(low/medium/high)}}{Nlv}, \quad (3.16)$$

where Nlv_{att_h} represents the number of invoked services with *low*, *medium* or *high* values on att_h by user u_i , and Nlv in the denominator represents the number of all services associated with a query from u_i . If one or two of the fuzzy values for $QoSF_{att_h}$ have passed the threshold (T_{att_h}) in this formula, $QoSF_{att_h} \geq T_{att_h}$, then the value for that specific QoS will be defined. For the rest of conditions would be *no preference*. We should note that the range and the number of the fuzzy values in different attributes are different. All the possible results on QoS for specific attribute on user u_i (QoS_{u_i}) are as follows:

Output(QoS_{u_i}): "low", "medium", "high", "no preference"

3.4.3 Personalized Service Ranking

In the earlier sections, we have designed our User Modeling system with user preferences on a set of non-functional properties. The user model could be generated explicitly by asking users to complete the user preference forms or implicitly by calculating from past invocation history data. Now, we need to calculate the similarity between all user preferences in the User Modeling system and services.

3.4.3.1 Similarity Calculation Between User Modeling system and Services

Whenever a user submits a service request to the system, after the functional matching is done based on user's functional requirements, our personalized ranking component will do the similarity calculation. The similarity calculation is between user's non-functional preferences saved in the User Modeling system and the actual non-functional property values of the functionally matching services. These services will be ranked on their similarity scores and the services with high matching degrees with the user profile will be ranked higher. In this section we will explain this similarity calculation process.

Suppose the user profile of $u_i \in U$ contains 10 non-functional properties, which could be represented as:

$$u_i = \langle PN_i, PH_i, PL_i, PP_i, SH_i, SF_i, SL_i, SR_i, SP_i, QoS_{att_i} \rangle$$

We explained before the first four properties represent: the user preferred provider name, provider history, provider location and provider popularity respectively. Also, PN_i and PL_i can have more than one value and could a set $PN_i = \{pn_1, pn_2, pn_3 \dots pn_n\}$, $n \geq 1, n \leq N$; and $PL_i = \{pl_1, pl_2, pl_3 \dots pl_f\}$, $f \geq 1, f \leq F$; for the provider name and the provider location, respectively.

The next five components are mainly used to show service properties including its history, freshness, language, rating and popularity. Also SL_i can have more than one value and could appear as a set $SL_i = \{sl_1, sl_2, sl_3 \dots sl_n\}$, $n \geq 1$.

The last property QoS_{att_i} includes all of the QoS attributes preferred by user u_i . In this research, because of the data source that we are using, three attributes have been defined: availability, documentation and response time. On the other hand, each service $s_k \in S$ also can be represented as a 10-tuple with the same 10 properties, which is shown as below.

$$s_k = \langle PN_k, PH_k, PL_k, PP_k, SH_k, SF_k, SL_k, SR_k, SP_k, QoS_{att_k} \rangle$$

In order to compare the similarity between the service and the user profile, we need to compare their similarities on each property separately first, and then get the overall similarity value. Next we will explain how we calculate the similarity for each property.

- **Provider:**

- o **Provider Name (PN)**

The value for the provider's name in the user model for user u_i could be more than one and we should match them with services' provider names one by one. Consider PN_i as a set of provider names: $PN_i = \{p_1, p_2, p_3 \dots p_n\}$ where $p_n \in P$ and $i \geq 0, n \leq N$; since P is the active domain of providers, the cardinality of the set is finite. Based on user preferences (both explicit and implicit), different scenarios may happen.

Firstly, if the cardinality of the set PN_i was zero, which means the preferred name on the PN_i for u_i in the User modeling system is *no preference*. Secondly, if at least one of the providers in the user preferences from the set PN_i was the same as the provider's name of the service, the degree of similarity would be one. Finally, if none of the providers' names in user preferences in UM was the same as the provider of the service, the degree of similarity for PN_i would be zero. The following similarity equation shows various possible conditions and degrees of similarity results between user u_i in the user model and one of the services on the provider name PN_i , where $Sim_{PN}(u_i, s_k)$ represents the similarity degree on the provider name between the user (u_i) preference and the service.

$$Sim_{PN}(u_i, s_k) = \begin{cases} 1 & \text{One or more similar services on } PN_i \text{ Or "no Preference"} \\ 0 & \text{No Similar Service on } PN_i \end{cases}$$

o **Provider Location (PL)**

Provider location includes a 2-tuple: $\langle \text{Continent}, \text{Country} \rangle$. In order to define the similarity degree between user u_i and s_k on provider location, different scenarios may occur. The user can select more than one provider location and for each selection different conditions can happen: 1- continent has been specified, but country has not been specified, 2- both continent and country are specified, 3- none of them have been specified by the user which means *no preference*. Based on these conditions and the provider directory, there will be different similarity results. Figure 3.3 shows the flowchart of them:

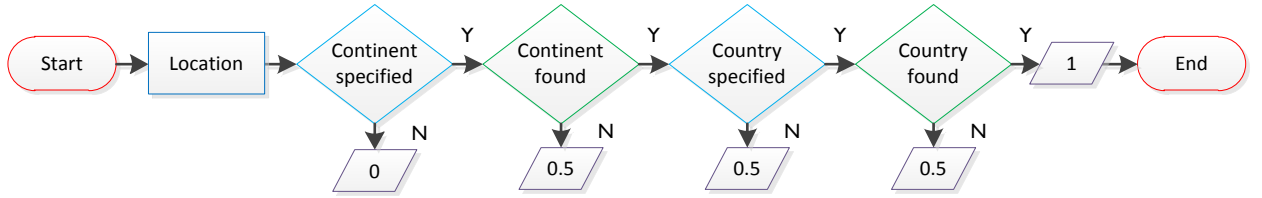


Figure 3.3- Provider Location flowchart

As a result of the above algorithm, the values for the similarity degree on the provider location $Sim_{PL}(u_i, s_k)$ would be:

$$Sim_{PL}(u_i, s_k) = \begin{cases} 1 \\ 0.5 \\ 0 \end{cases} , no\ preference = 1$$

o **Provider History (PH)**

To calculate the similarity degree on the provider history and some of the other properties, we need to assign fuzzy values to our Likert scale [62] variables. *Provider history* is in a 4-point Likert scale with different domains of $\langle no preference, short, medium, long \rangle$. To calculate similarity degree we don't need to consider *no preference*, and the fuzzy values for *short*, *medium*, and *long* are 0, 1, and 2, respectively. To calculate the similarity degree between fuzzy values, we adopt the array of semantic similarity proposed by Chen and Singh [63, 64] as shown in formula 3.15. There are some modifications have been made for this research.

$$Sim_{PH}(u_i, s_k): \begin{matrix} & & & s_k \\ & & 2 & 1 & 0 \\ u_i & 2 & \begin{bmatrix} 1 & 0.7 & 0 \\ 0.7 & 1 & 0.7 \\ 0 & 0 & 0.7 & 1 \end{bmatrix} \end{matrix} \quad (3.17)$$

Where Sim_{PH} shows the similarity degree between the user (u_i) preference and a service (s_k) on provider history. We have changed the size and values of the array to better fit our work. If the user preference has not been given by the user, this means *no preference* and the default similarity value will be set as 1. The values in the above matrix represent their similarity in evaluating the service's provider history from the user preference with a web service. For instance, in the first row and second column the value of the array is 0.7, which means the semantic similarity for user u_i and service s_k on provider history will be 0.7. In other words, when the user preference on provider history is long and the actual provider history of the service is medium, their similarity is 0.7. Also, if the similarity has not been found, the value would be zero.

o **Provider Popularity (PP)**

Provider popularity can be considered as a 4-tuple Likert scale $\langle no\ preference, low, medium, high \rangle$. The calculation is similar to provider history and we consider *low*, *medium*, *high* with 0,1,2, respectively. The rest of the process is the same as the provider history.

- **Service:**

- o **Service History (SH)**

The similarity calculation for service history is similar to the provider history.

- o **Service Freshness (SF)**

The similarity on service freshness is the same as the service history.

- o **Service Language (SL)**

The user can have preference on more than one service languages and the set of service languages could be shown as $SL = \{l_1, l_2, l_3 \dots l_f\}, f \leq F$. When the user has preference on some specific languages, the similarity degree for those services containing user's preferred languages are one and the rest would be zero. If the user has *no preference* the similarity degree for all services would be 1, and if there was not any similar services the result is zero. All the possible similarity degrees could be shown as:

$$Sim_{SL}(u_i, s_k) = \begin{cases} \mathbf{1} & \text{One or more similar services on } SL_i \text{ Or "no Preference"} \\ \mathbf{0} & \text{No Similar Service on } SL_i \end{cases}$$

- o **Service Rating (SR)**

Service rating could be considered as a 6-tuple Likert scale: $\langle no\ preference, very\ low, low, medium, high, very\ high \rangle$. Without considering *no preference*, by converting them to the proportional values in the fuzzy scale the values would be 0,1,2,3,4 for *very low*, *low*, *medium*, *high* and *very high*, respectively. To find similar services by considering the user's preferred rating, we have adopted the matrix by Chen and pal Singh [63]. We then modified the cardinality and values in the matrix to fit our problem. The semantic similarity matrix for service rating is as follows:

$$Sim_{SR}(u_i, s_k): \mathbf{u_i} \begin{matrix} & \begin{matrix} 4 & 3 & 2 & 1 & 0 \end{matrix} \\ \begin{matrix} 4 \\ 3 \\ 2 \\ 1 \\ 0 \end{matrix} & \begin{bmatrix} 1 & 0.7 & 0.2 & 0 & 0 \\ 0.7 & 1 & 0.3 & 0 & 0 \\ 0.2 & 0.3 & 1 & 0.3 & 0.2 \\ 0 & 0 & 0.3 & 1 & 0.7 \\ 0 & 0 & 0.2 & 0.7 & 1 \end{bmatrix} \end{matrix} \quad (3.18)$$

When the service rating from user model is "*no preference*", the similarity is 1.

- o **Service Popularity (SP)**

The similarity degree on service popularity is the same as provider popularity and all the steps are the same and on the services.

- o **Quality of Services (QoS)**

We have listed all of the quality of services in Appendix A, but only here we will go through the ones that we have access to their data for related web services.

- o **Service Availability (SA)**

To calculate the similarity degree of services' availability, we have a 6-tuple Likert scale: $\langle no\ preference, very\ low, low, medium, high, very\ high \rangle$. Without considering *no preference* the converted fuzzy values are: 0, 1, 2, 3 and 4 for *very low*, *low*, *medium*, *high* and *very high*, respectively. To calculate service availability similarity, the matrix is the same as service rating.

- o **Service Documentation (SD)**

Similarity degree on service documentation is the same as the provider history.

- o **Service Response Time (SR)**

Similarity on service response time is the same as service rating.

3.4.3.2 Total Similarity Calculation

Early parts explained different calculations to find similar services matching with the user profile on each user model properties. Relevance computations returned different lists of services based on the similarity degree between user preferences and the actual values of services. In order to return a single ranked list of services to the user, we need to combine the similarity degree on each property linearly to get a composite similarity score for the final ranking. The combination formula is shown below:

$$\mathbf{Sim}(u_i, s_k) = \sum_{i=1}^n \alpha_n * Sim_{a_n}(u_i, s_k); (\alpha_1 + \alpha_2 + \dots + \alpha_n = 1) \quad (3.19)$$

In this formula, $Sim(u_i, s_k)$ represents the total similarity degree between user u_i and a web service (s_k), based on all of the properties $Sim_{a_n}(u_i, s_k)$ represents the similarity degree on each

property for user u_i in the User Modeling system and service s_k and a_n shows each of the properties. α_n is coefficient which determines the weight for each property and it depends on the significance we can decide on them in the design time. Also, n is the number of properties that has been considered.

Based on the above formula, we can calculate the overall similarity score between the user profile and the service. We can rank all the services based on their matching degrees with the user profile. In this way, personalized ranking can be achieved. The top k results are most likely complying with user's interest and preference, which makes the selection result more accurate and takes user less time to locate the desired services.

3.5 Summary

In this chapter, we proposed the architecture of our User Modeling approach with all of the required properties in terms of different variables and their operators. Moreover, in this chapter we explained all of the non-functional properties which are crucial for decision making on service selection. The way of collecting data explicitly and implicitly for every single property had been considered. Finally, the method of making a personalized ranking list of top k services according to the user preferences was presented. In the next chapter, the implementation and experiment part will be explained.

CHAPTER 4

IMPLEMENTATION AND EXPERIMENT

4.1 Introduction

As we discussed before the main contribution of this thesis is to make a user model for personalized web service ranking. In this chapter, we demonstrate the implementation on the explicit User Modeling and the personalized ranking system. We also conducted some experiments to measure the performance of the personalized ranking algorithm. This chapter includes two parts, in the first part we will go through various steps of the implementation, and in the second part we will explain the experimental design and different parameter settings and the final results.

4.2 Implementation

4.2.1 Programming Environment

The framework has been developed as a windows-based application, using the Java language, in the Eclipse environment. In the database part, we used MySQL workbench 5.2 to make directories as a database and also using it to help users interact with the user model and input their preferences on various non-functional properties.

4.2.2 Interface Design to Get User's Explicit Preferences

User Interface (*UI*) design in User Modeling system is very important, because this is the way that the user can interact with the system. In this work we design a simple User Profile

solicitation interface which is understandable for users at any level. Our user model interface has a multi-layer and modular architecture to increase its customization level. Figure 4.1 shows the preliminary schema of the interface and also the relation between different pages; we will go through them in details in the next section.

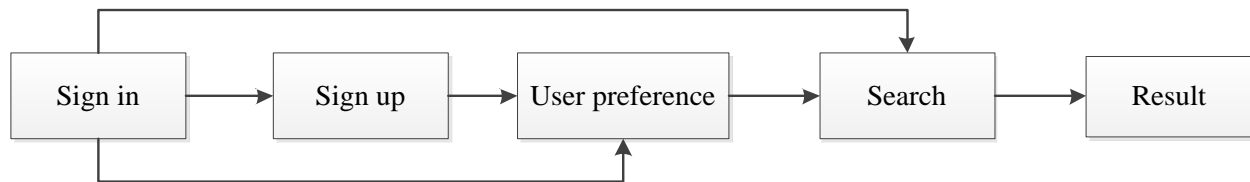


Figure 4.1- User Modeling system interface schema (GUI schema)

Our user model interface consists of five layers: sign in, sign up, user preference specification, search, and result presentation. The first layer defines the sign-in page. If the user is new to the system, he/she should go to the second layer which is the sign-up page. In the sign-in page the current user should enter the user name and password to login in. The user can go either to the search page or to his/preferences page from this page. The first time when a user signs up to the system, he/she should specify the user preferences, and later user preferences could be updated when necessary. We explain different layers by making a sample user (user *test*). Figure 4.2 presents the sign-in page for this sample user.

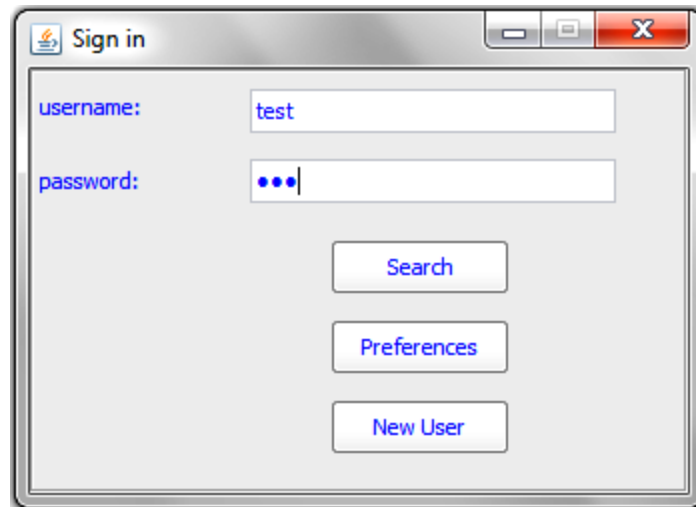

 A screenshot of a 'Sign in' window. It has a title bar with a minimize button, a maximize button, and a close button. The window contains two text input fields: 'username:' with the value 'test' and 'password:' with three dots. Below the fields are three buttons: 'Search', 'Preferences', and 'New User'.

Figure 4.2- Layer 1 of User Modeling system interface: sign-in

Then the second layer, as shown in Figure 4.3, illustrates the sign-up form. The sign-up layer is just for new users and it is mandatory to answer starred (*) questions. From this page they go to the preference page, also it is possible to reset their information in the page by clicking the reset button.

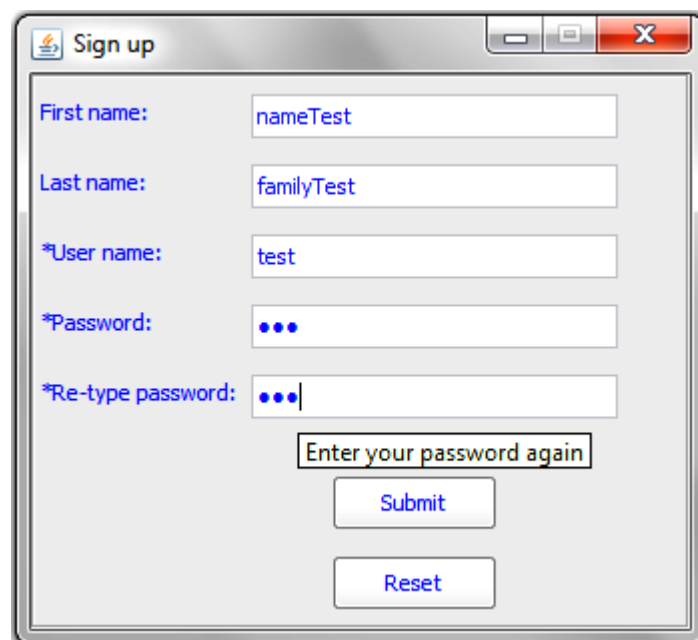

 A screenshot of a 'Sign up' window. It has a title bar with a minimize button, a maximize button, and a close button. The window contains five text input fields: 'First name:' with the value 'nameTest', 'Last name:' with the value 'familyTest', '*User name:' with the value 'test', '*Password:' with three dots, and '*Re-type password:' with three dots. Below the fields is a text label 'Enter your password again' and two buttons: 'Submit' and 'Reset'.

Figure 4.3- Layer 2 of User Modeling system interface: sign-up

The third layer, which is the most important layer in the User Modeling system interface, is user preference specification. As we explained before about user preferences, this layer contains two parts, provider preferences and service preferences. Provider preferences part includes all the properties related to the provider. The service preferences part includes two sections: general properties and QoS properties. For those properties which work with Likert Scale, we added an information button to explain the value range for each scale. Figure 4.4 shows an example which is for provider history.

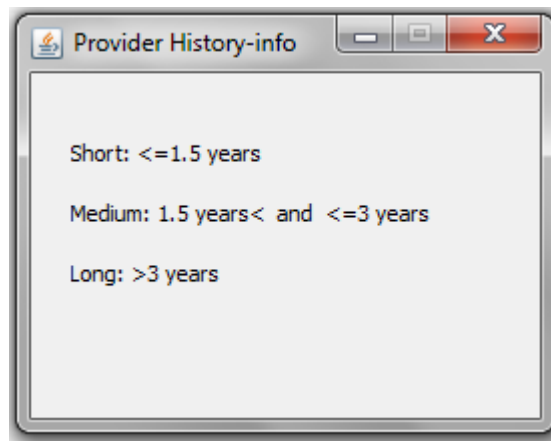


Figure 4.4- Provider history scales

The complete list of all possible selections for all of the properties, based on the source of the dataset that we are using for the experiment are shown in Table 4.1.

Table 4.1- properties and ranges

Property	Value ranges of its scales
Provider History (year)	short: ≤ 1.5 , medium: $1.5 < \text{ and } \leq 3$, long: > 3
Service History (year)	short: ≤ 1.5 , medium: $1.5 < \text{ and } \leq 3$, long: > 3
Service Freshness (month)	low: ≤ 6 , medium: $> 6 \text{ and } \leq 12$, high: > 12
Service Rating (range)	very bad: ≥ 0 , bad: > 1 , medium: > 2.5 , good: ≥ 3.5 , excellent: ≥ 4.5
Service Availability (percent)	very bad: ≥ 0 , bad: > 50 , medium: > 70 , good: ≥ 80 , very good: ≥ 95
Service Response time (ms)	very low: ≥ 0 , low: ≥ 700 , medium: > 750 , high: ≥ 770 , very high: ≥ 790

As we explained before, there are different types of data in provider and service preferences such as: ordinal, nominal, etc. To represent them in the interface of the User Modeling system we have converted them in two schemas: multiple-choice items and drop-down list. In multiple-choice items, the user can select one item at a time. In the drop-down list, the first item is *no preference*, and when it has been selected adding the other items is impossible. The user can add any number of items that he/she is willing into the right-hand list. Figure 4.5 shows provider preference and general service preference.

User Preferences-Step1

Provider Preference

Provider Name: perceval.net Add Remove seek.com.au perceval.net

Provider Location: Continent: north america Add Remove europe,austria north america,All

Country: All

Provider History: ☒ No Preference ☐ Short ☐ Medium ☐ Long

Service Preference

Service Language: french Add Remove french

Service History: ☐ No Preference ☐ Short ☒ Medium ☐ Long

Service Freshness: ☐ No Preference ☐ Low ☐ Medium ☒ High

Service Rating: ☐ No Preference ☐ Very bad ☐ Bad ☒ Medium ☐ Good ☐ Excellent

Reset Next

Figure 4.5- Layer 3 of User Modeling system interface: user preference (1)

Since there are many QoS attributes and we can add them to the user model, and also QoS is the most important part of web services, we made a new page for them. In the second page of the third layer, there are all of the supported QoS attributes. For instance, in our work, because of the source of dataset that we used for our experiment, we chose three QoS attributes. We converted them to multiple-choice items to show them in the user preferences. Also we are trying to make it clear for the user; therefore, we added an explanation to the relevant QoS attributes on this page to facilitate it. Figure 4.6 represents the second step of user preferences layer.

The screenshot shows a window titled "User Preferences-Step1" with a standard Windows-style title bar. The window is divided into three main sections. The top section, titled "QoS Preference", contains two lists: "Attributes:" on the left and "Selected attributes:" on the right. The "Attributes:" list is currently empty, while the "Selected attributes:" list contains "Documentation", "Availability", and "Response time". Between these lists are "Add" and "Remove" buttons. The middle section, titled "QoS' Definitions", provides explanations for the selected attributes: "Documentation" shows the technology used; "Availability" is the degree of service accessibility and functionality, where a larger number is better; and "Response time" is the elapsed time between the end of a request and the beginning of the response time by service, where a smaller number is better. The bottom section, titled "Selected QoS", contains three groups of radio button options for each attribute. For "Documentation", the options are "No Preference", "Bad", "Partially" (which is selected), and "Good". For "Availability", the options are "No Preference", "Very bad", "Bad", "Medium", "Good" (which is selected), and "Very good". For "Response time", the options are "No Preference", "Very low", "Low", "Medium", "High", and "Very high" (which is selected). At the bottom of the window are three buttons: "Reset", "Back", and "Submit".

Figure 4.6- Layer 3 of User Modeling system interface: user preference (2)

The forth layer defines the search page. The user would be forwarded to this page by clicking on “submit” button in the second page of the user preferences or from the sign-in page by clicking on the “search” button. The search page includes keyword selection which is an item to lead the system to produce results compatible with user’s functional requirement, e.g. finance, economics, travel, etc. Search page, ideally should be text box where the users can type in any words, but here for the experiment purpose, we have dropped-down menu selection. The user will be guided to this page from sign in page, or from user preference page for the new users. Search page is shown in Figure 4.7. When the “search” button is clicked, the functional matching services will be ranked based on the explicit user profile if the user has specified his/her non-functional preferences through these interfaces, or the implicit user profile generated from the past invocation histories if the user has chosen to skip the user preference specification step.

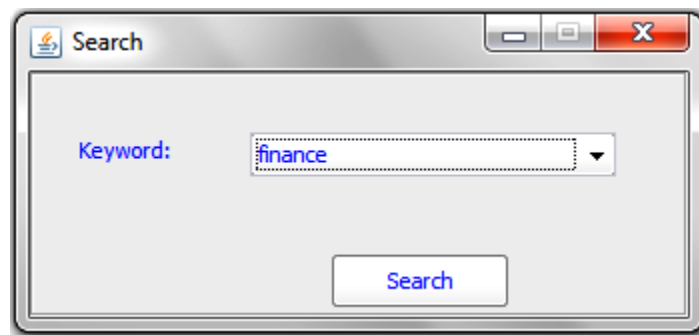


Figure 4.7- Layer 4 of User Modeling system interface: search

The fifth layer is the result layer, which shows the ranked result list. The sample of the result page with an example will be shown in the sample query in section 4.3.2.

4.3 Experiment

The main purpose of the experiments in this work is evaluating the accuracy of the personalized service ranking algorithm based on our generated user model. In other words, we

want to show that by using our system, users can find more relevant results matching their preferences. To implement all of the proposed algorithms in chapter 3, we need to have access to the invocation and query logs collected from real users. However, it is hard to find such a dataset. Due to the time constraints, we also didn't have time to run the simulation to generate the log data. Therefore in the experiment, we only test the explicit User Modeling system and run the personalized ranking algorithm on the explicit user profile. Also our User Modeling system, can find similar services for each property, by using the formulas in personalized ranking section in chapter 3 (3.4.3). Then for total similarity of for each service we used formula 3.19 and we considered all the coefficients (α_i) equal to 0.1. Because we have 10 properties in our User Modeling system and we assumed equal weight for all of them.

4.3.1 Dataset

Currently there is no public UDDI registry that we can query. Based on our research, Seekda [65] is the most comprehensive global search engine for public web services. Seekda is an Austrian search engine for web API (web services at the moment) and web services. The services are collected by the focused crawling process and some of their QoS values are daily monitored. Seekda includes 7739 providers and 28606 services over more than 28000 descriptions to date (Aug.02.2011). The outstanding point about Seekda is that the information about a service is either taken from its WSDL file or from its provider's website (e.g. by monitoring on its availability, its wiki description, etc.) [65]. For each provider it contains the real data on its country, the number of services it published, the actual list of services, its wiki history, etc. The wiki history is about the different versions of providers. Also for each service there is information about its provider, the hosting server, its WSDL file, the monitoring time, its rating,

etc. In addition, for QoS attributes Seekda covers documentation, availability, and response time. Monitoring time on web services in Seekda started in 2006. Hence, the oldest web services in our dataset were published in 2006. All of the services can be tested at Seekda with the Seekda online tester tool directly. To summarize the benefits of Seekda in comparison with the other available web service directory resources such as XMethods [66] in the absence of a UDDI registry, we have listed the following points:

- Not just focusing around the programmatic access to the registry: they make it simple, even for the first time learner about web service technologies, to explore services and their specifications.

- Updated availability: there is a mechanism to check the availability of web services to find whether they are working at a particular time.

- Community feedback: there is a support for community feedback features.

From all the above descriptions of Seekda, we found it the best source for data collection to test our algorithm. We can search Seekda for web services with their tags (keywords) in different ways, for instance, based on the provider country, service tag cloud, recent services, and most used services. We have used all of them in our data collecting.

We have collected our dataset by crawling and monitoring online web services in Seekda during a six-month period (December, 2010 to May, 2011). Our dataset includes altogether 537 providers and 1208 services; each provider contains at least one service. We used them as our provider directory and service directory respectively. There are 287 service keywords in our dataset collection of which we chose 30. We have categorized our web services in three groups based on the number of services for each group. Group one: 10 keywords with equal to or less than 50 services per keyword; Group two: 10 keywords with equal to or less than 100 services

per keyword; and Group three: 10 keywords with greater than 100 services per keyword. Then we chose 3 most popular keywords per group as the functional keyword queries, such as "traffic", "university" and "travel", to test our algorithm. The selected keywords and the number of services are shown in Figure 4.8.

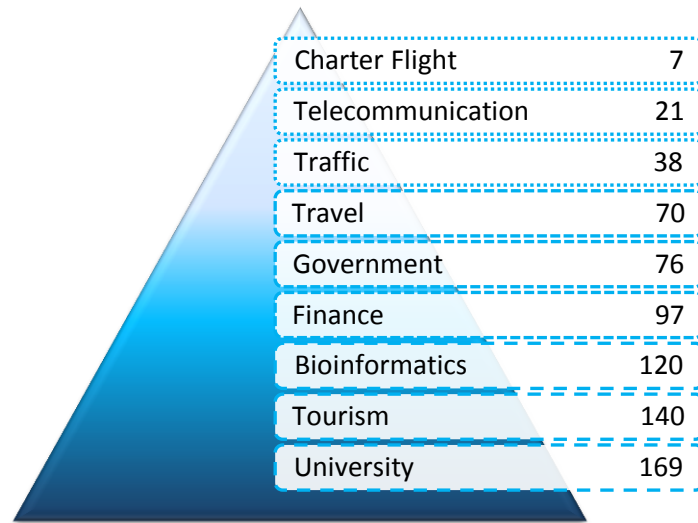


Figure 4.8- The selected keywords and number of services used in the experiment

As we explained before, we are currently just testing the personalized ranking part of our model with explicitly collected user profiles. The reason is that, in the first part, we need real data from log files, which are currently not possible for us to collect. In addition, we collected the required dataset from Seekda. The reason is that it is the only public web service search engine with a good number of services in its directory, larger than any other datasets we know (e.g. XMethods, QWS)[66,67], and it has the web service data on a few non-functional properties.

Our simulator program, which is the combination of three simulators, was implemented in Java using Eclipse under Windows XP platform, to generate random users with random user profiles and random functional queries for each user. Based upon our experiment design, we

have generated 60 random users with different random profiles. Randomly generated users have been clustered in 6 groups, and for each group there is a fixed number of properties and the rest of them are defined as *no preferences*. For instance, 10 users have preferences on all non-functional properties, or 10 users have preferences on two different non-functional properties and have no preferences on the rest of the properties. The user profile with all non-functional properties defined as *no preference* would be the baseline to compare with the other user profiles. Inside the simulator we used a configuration file to adjust the number of preferred non-functional properties and related users.

Each query record includes a functional and a set of non-functional requirements. In the current experiment setting, the functional query makes use of 9 keywords listed in Figure 4.8. QoS attributes are availability, response time and documentation, which are supported by Seekda. User's query is in accordance with user's functional requirement on the one hand as well as the non-functional requirements on the other hand. Table 4.2 shows the experiment design for users and properties. The detail of the experiment design for all of the users is available in Appendix B.

Table 4.2- Experiment design preferred

Number of Users (Out of 60)	Number of preferred Properties (Out of 10)	Number of “<i>no preferences</i>” (Out of 10)
10	1	9
10	2	8
10	4	6
10	6	4
10	8	2
10	10	0

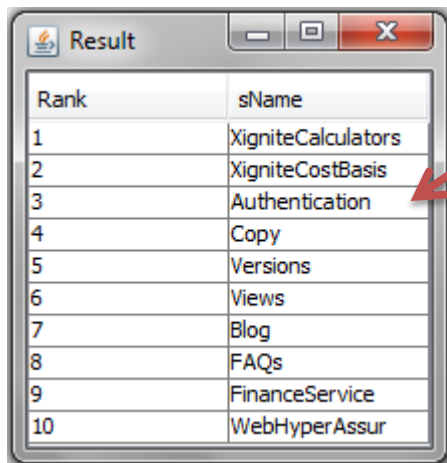
4.3.2 Sample Query and Results

We simulated user profiles randomly with possible combinations of different non-functional properties, and also we had a list of functional queries. In this part we will explain a sample query on the user *test* profile that we made in the previous section, and will show that the personalized ranking results can promote good results to top positions. The summary of user preferences for user *test* sample is shown in Table 4.3.

Table 4.3- *test* user profile

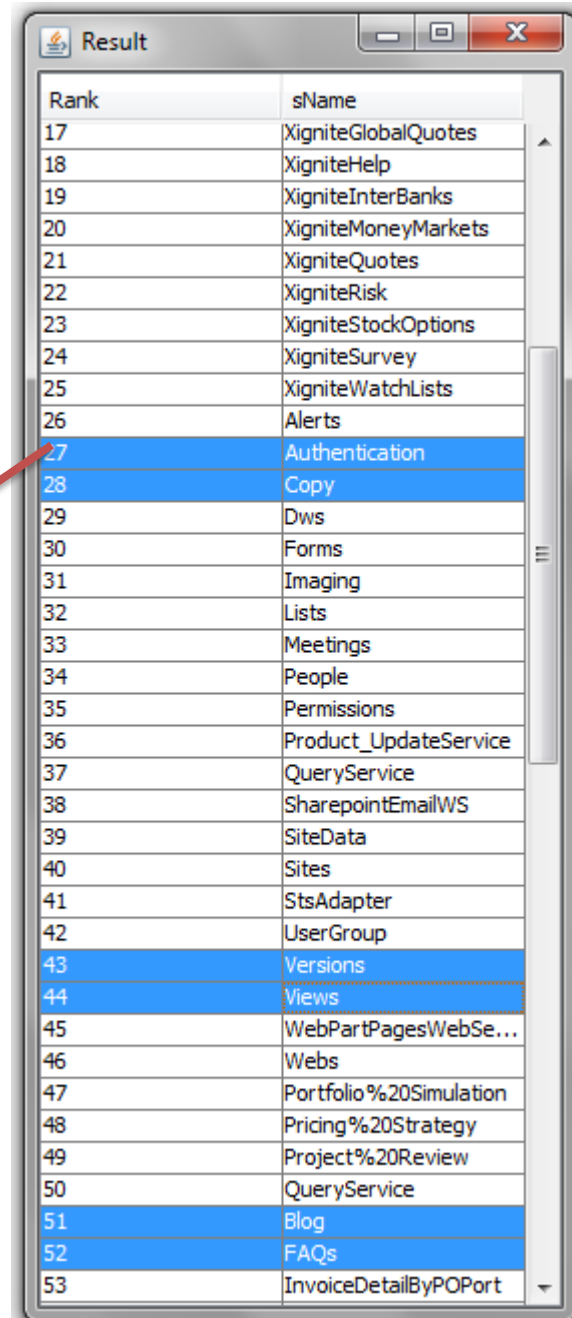
<i>Non-functional Property</i>	<i>Value</i>
pName	<i>no preference</i>
pLocation	<i>North America, United states Europe, France</i>
pHistory	<i>Long</i>
sLanguage	<i>English, French</i>
sHistory	<i>no preference</i>
sFreshness	<i>no preference</i>
sRaiting	<i>no preference</i>
sDocumentation	<i>good</i>
sResponseTime	<i>no preference</i>
sAvailability	<i>no preference</i>

The ranking top 10 results for the user *test* on query “finance” with the defined preferences as in Table 4.3 are shown in Figure 4.9. By comparing the ranking of the results with baseline (*no preference*), show that by using our personalized ranking algorithm we can offer better results to the user. Also by asking user’s preferences explicitly we can make the results closer to user’s taste. Baseline results are shown in Figure 4.10. For instance, service with the ranking order 27 was promoted to the rank 3 after considering the user’s preference by using our User Modeling system.



Rank	sName
1	XigniteCalculators
2	XigniteCostBasis
3	Authentication
4	Copy
5	Versions
6	Views
7	Blog
8	FAQs
9	FinanceService
10	WebHyperAssur

Figure 4.9- Top 10 result on “finance” for user *test*



Rank	sName
17	XigniteGlobalQuotes
18	XigniteHelp
19	XigniteInterBanks
20	XigniteMoneyMarkets
21	XigniteQuotes
22	XigniteRisk
23	XigniteStockOptions
24	XigniteSurvey
25	XigniteWatchLists
26	Alerts
27	Authentication
28	Copy
29	Dws
30	Forms
31	Imaging
32	Lists
33	Meetings
34	People
35	Permissions
36	Product_UpdateService
37	QueryService
38	SharepointEmailWS
39	SiteData
40	Sites
41	StsAdapter
42	UserGroup
43	Versions
44	Views
45	WebPartPagesWebSe...
46	Webs
47	Portfolio%20Simulation
48	Pricing%20Strategy
49	Project%20Review
50	QueryService
51	Blog
52	FAQs
53	InvoiceDetailByPOPort

Figure 4.10- Baseline result for “finance”

4.4 Evaluation and Analysis of User Modeling System

Since our personalized ranking algorithm adds user’s preferences on top of the original search engine, we would like to check the accuracy of the system. Suppose our algorithm gets

personalized results to top positions, we could use *MAP* (Mean Average Precision) of these top 10 results in the original baseline run to measure the improvement of the accuracy from our algorithm.

Mean Average Precision (*MAP*) is one of the ways to evaluate the performance of the information retrieval systems and the measure needs a set of queries [68]. *MAP* has more discriminative power in compare with the other metrics in information retrieval [68]. *AP* (Average Precision) is the average precision value after each relevant result is retrieved. *MAP* is used for a set of queries to measure the mean of the *APs* for each query. In our case, using *MAP* can show that personalized ranking can promote good results to top positions. Formula 4.1 shows *AP* as follows:

$$AP = \frac{\sum_{r=1}^N (P(r) \times rel(r))}{number\ of\ relevant\ services} \quad (4.1)$$

Where *AP* is the average precision of a single query which is the mean of the precision scores after each relevant service is retrieved, $P(r)$ is the precision at cut-off r in the list. Here, r is the rank list of services, N is the number of retrieved services by using our ranking algorithm (which is 10 in our experiment design) and $rel()$ is the relevancy of each service, which is an indicator function equal to 1 if the service at rank r is relevant, otherwise is zero [68,69]. Formula 4.2 shows the *MAP* for a set of queries, where Q is the number of queries.

$$MAP = \frac{\sum_{q=1}^Q AP(q)}{Q} \quad (4.2)$$

In our case, we get the result set when considering user preferences, take the top k (i.e. 10) results as the relevant result set, and then check the top k results from the baseline run when user

preferences are not considered, to get its MAP value. When the *MAP* value is less, it means that the baseline result is worse than our personalized ranking result. We have calculated *MAP* in different aspects.

Table 4.4 shows the results for different keywords with different number of non-functional preferences defined.

Table 4.4- *MAP* calculation on each keyword

	Charter flight	Telecommunication	Traffic	Travel	Government	Finance	University	Tourism	Bioinformatics
1 preferred	1.000000	0.655843825	0.516257653	0.327105512	0.22300576	0.271859828	0.142051042	0.211911011	0.152547815
2 preferred	1.000000	0.644020962	0.36759549	0.244404361	0.197175335	0.169476703	0.216093295	0.111420863	0.078058293
4 preferred	1.000000	0.619150869	0.306778718	0.153982517	0.189835893	0.193277768	0.163855861	0.070880691	0.066629246
6 preferred	1.000000	0.467197663	0.292959909	0.247937825	0.175946697	0.086297201	0.110002429	0.07994943	0.123158745
8 preferred	1.000000	0.586770647	0.340984625	0.180187836	0.24431011	0.130214806	0.118819951	0.081314985	0.058700476
10 preferred	1.000000	0.457261	0.334913197	0.250624048	0.196057294	0.14466184	0.10823554	0.079228121	0.064612085

By analyzing the results in the above table and comparing with the base condition (*no preference*), we found that when the number of preferred properties and the number of services increase, the *MAP* value generally decreases. The result of comparison is shown in Figure 4.11 and 4.12.

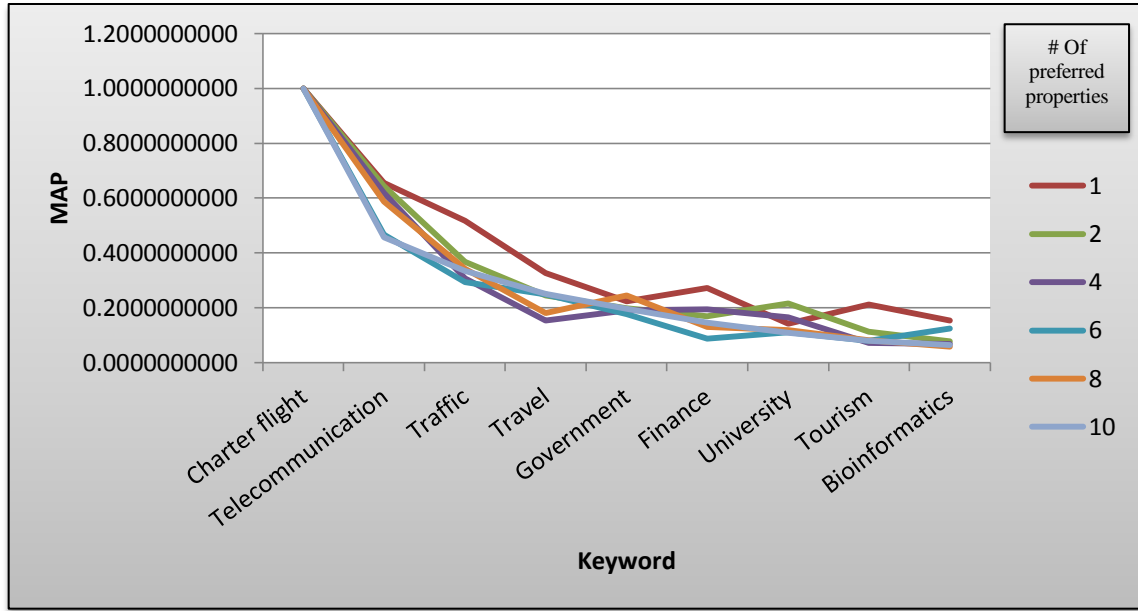


Figure 4.11- MAP calculation on different keywords

Also the interesting point is that when the number of services is less than k (k , in top k), which is 10 in our calculations, the MAP remains 1. For example, in our data base the maximum number of services on “charter flight” was seven, as a result the MAP equals to one for all the users (Figure 4.12).

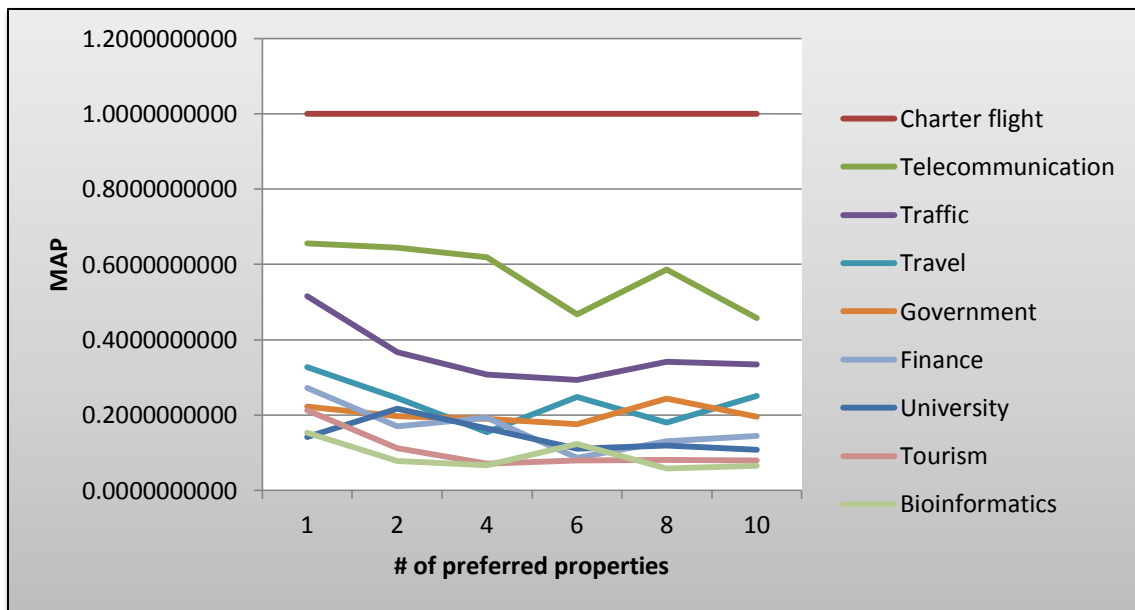


Figure 4.12- MAP comparison on different number of preferences

All of the numbers and calculations for *AP* and *MAP* on different users and groups are available on Appendix C.

4.5 Summary

In this chapter, we explained our implementation part with all of experiment designs. Also we used *MAP* formula to calculate the accuracy of our work by measuring the precision at all the queries. By investigating all of the above results we showed that using a personalized service ranking algorithm can offer the users a list of better results.

CHAPTER 5

CONCLUSION

5.1 Conclusion

In this thesis, we reviewed different research about User Modeling and web services. According to our research, there is no comprehensive User Modeling system on non-functional preferences for web services. This finding motivated us to propose a User Modeling system on user general preferences and use it for personalized service selection and ranking. The model can infer implicitly on user preferences based on previous history and invocation data. By using fuzzy range values to represent user preferences both explicitly and implicitly, the system would be easier to follow for all the users in any level. Also the matching could cover more potential useful services which would partially solve the data sparsity problem for many recommender systems. Our user model is flexible and can be expanded to include more non-functional properties when their information is available in the source data. Also, the user can add as many preferences as needed on different properties and the system could support it. To check whether personalized ranking could improve the selection accuracy, we compare the results with a well-known service search engine. This search engine does not consider user preferences in its selection algorithm and is the data source for our testing dataset. The result showed the accuracy of our algorithm would be better than the original. The result proves that the effectiveness of our personalized service ranking algorithm and indirectly proves the soundness of our User Modeling approach.

5.2 Main Contributions

The major contributions of our work are as follows:

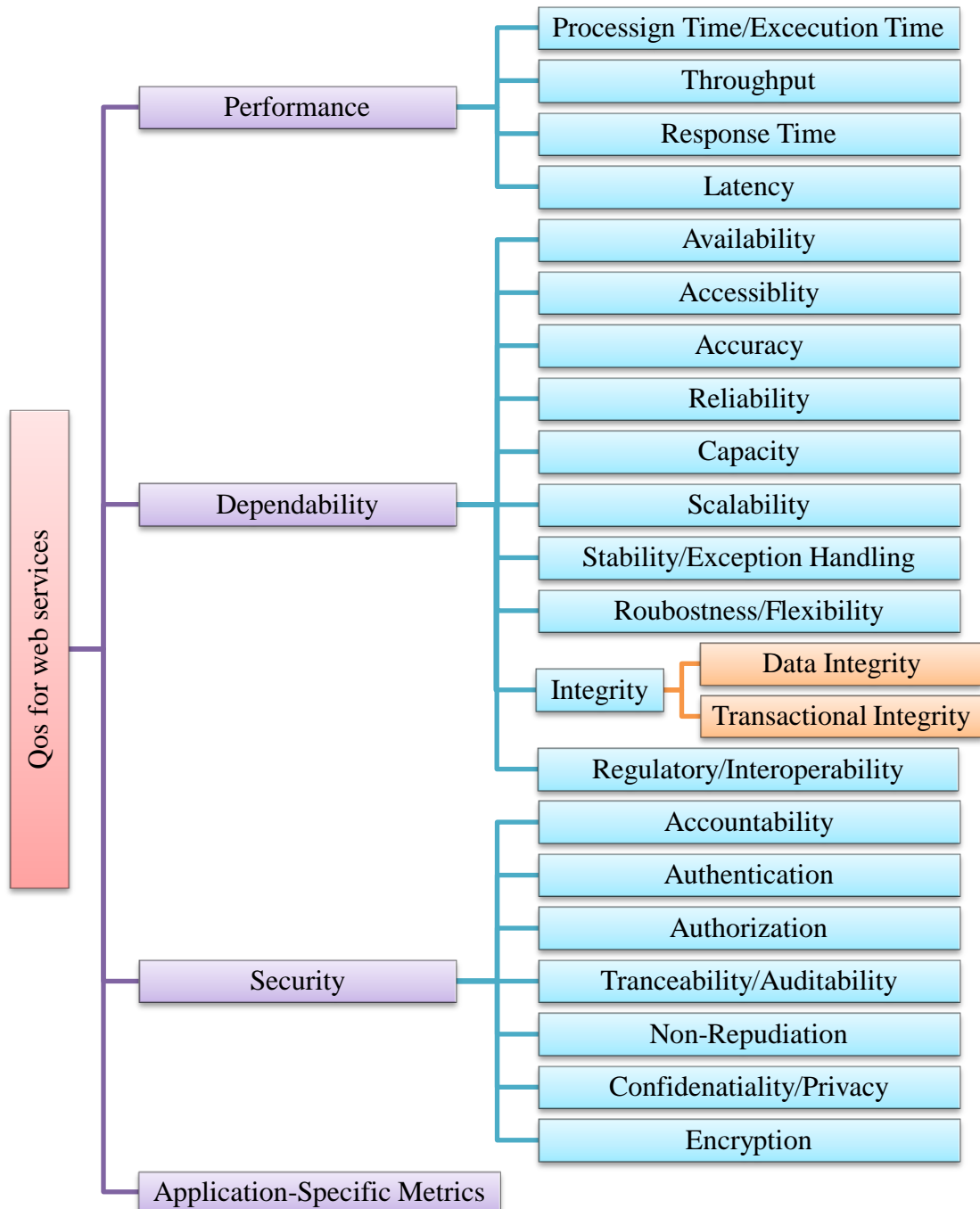
- To the best of our knowledge, it is a novel idea to propose a User Modeling system on user's non-functional preferences on web services. Those are some general preferences which are not restricted to a particular domain.
- We defined which non-functional preferences are important for the users to select services. The list of properties and the formulas of how to calculate use preferences on them from implicit user feedback was explained. The system also supports explicit preference definition from the user.
- We discussed a sample application about how a user model can be used, which is personalized service selection and ranking in our implementation. The result of the User Modeling could also be used in the other facets of web service selection.

5.3 Future Works

There are a few directions that we would like to add to our system. Firstly, due to the time constraints and the lack of real usage data, we didn't conduct experiments on proving the accuracy of the implicit User Modeling part. Therefore, in the future we would like to either find a real dataset or use some simulation data to test our User Modeling algorithm.

Secondly, we may expand our work to combine with different users' history as proposed in [38]. Finally, we would like to implement other applications based on the generated user model, such as service recommendation, user clustering, etc.

APPENDIX A – Hierarchical list of all Quality of Service (QoS)[58]



APPENDIX B – Experiment design for all users

Property/ User	pName	pLocation	pHistory	sLanguage	sHistory	sFreshness	sRating	sDocumentation	sRating	sAvailability
			1 preferred properties and rest no preference							
u1		×								
u2										×
u3			×							
u4	×									
u5								×		
u6					×					
u7						×				
u8							×			
u9									×	
u10				×						
			2 preferred properties and rest no preference							
u11			×		×					
u12		×						×		
u13							×		×	
u14			×						×	
u15						×				×
u16	×			×						
u17		×					×			
u18					×			×		
u19	×					×				
u20				×						×
			4 attributes set and rest No preference							
u21	×		×			×		×		
u22		×			×				×	×
u23	×	×		×			×			
u24			×		×			×	×	
u25				×		×	×			×
u26					×		×	×	×	
u27	×			×	×	×				
u28		×				×	×			×
u29	×		×	×					×	
u30		×	×					×		×
			6 preferred properties and rest no preference							
u31		×		×	×	×			×	×
u32	×	×	×	×			×	×		
u33	×		×		×	×	×		×	
u34			×	×	×		×	×	×	
u35	×	×		×			×		×	×
u36		×		×	×	×		×		×
u37	×		×		×			×	×	×
u38			×	×			×	×	×	×
u39	×		×		×		×	×	×	
u40	×	×		×	×			×	×	
			8 preferred properties and rest no preference							
u41	×	×	×	×	×	×			×	×
u42	×	×		×	×	×	×	×	×	

u43		x	x	x	x	x	x	x	x	
u44	x	x	x	x	x	x		x		x
u45	x	x	x		x	x	x		x	x
u46	x	x		x		x	x	x	x	x
u47		x	x	x	x		x	x	x	x
u48	x		x		x	x	x	x	x	x
u49	x	x	x	x			x	x	x	x
u50	x		x	x	x	x	x	x		x
10(all) preferred properties										
u51	x	x	x	x	x	x	x	x	x	x
u52	x	x	x	x	x	x	x	x	x	x
u53	x	x	x	x	x	x	x	x	x	x
u54	x	x	x	x	x	x	x	x	x	x
u55	x	x	x	x	x	x	x	x	x	x
u56	x	x	x	x	x	x	x	x	x	x
u57	x	x	x	x	x	x	x	x	x	x
u58	x	x	x	x	x	x	x	x	x	x
u59	x	x	x	x	x	x	x	x	x	x
u60	x	x	x	x	x	x	x	x	x	x

APPENDIX C – Results

This table shows the precision measurements on our personalized ranking system. The results are the Average Precision (*AP*) values for 9 single-word queries submitted by different users. Users are grouped into sections. For each section of users, the bottom row shows the Mean Average Precision (*MAP*) value for each query. Also the rightmost column shows the *MAP* for each user. At the end of the next page you can find the legend of the table.

Key/user	K1(50) #service(7)	K2(50) #service(21)	K3(50) #service(38)	K4(50-100) #service (70)	K5(50-100) #service(76)	K6(50-100) #service(97)	K7(100) #service(169)	K8(100) #service(140)	K9(100) #service(120)	
	Charter flight	Telecommunication	Traffic	Travel	Government	Finance	University	Tourism	Bioinformatics	
u1	1.0000000000	0.5077893868	0.4952301018	0.5039500502	0.2642577028	0.1849157888	0.3039167657	0.4490200792	0.0737966878	0.4203196181
u2	1.0000000000	0.6635629292	0.4719477855	0.2754639338	0.0831034758	0.1655493835	0.0498372723	0.0750927439	0.3640195292	0.3498418948
u3	1.0000000000	0.7181083837	0.5600312049	0.4604461015	0.2642577028	0.3303396298	0.1721919305	0.3343857672	0.0883412222	0.4364557714
u4	1.0000000000	0.7181083837	0.6003084416	0.3486124738	0.2642577028	0.3303396298	0.1721919305	0.1836329897	0.0889132219	0.4118183082
u5	1.0000000000	0.6747590838	0.4806599551	0.1713922104	0.1412979914	0.1501325374	0.0740091111	0.0658356773	0.0706426168	0.3143032426
u6	1.0000000000	0.7181083837	0.3782353285	0.1044034963	0.2082385493	0.1241773565	0.0742290535	0.0523518500	0.1862511356	0.3162216837
u7	1.0000000000	0.7181083837	0.6003084416	0.3486124738	0.2480850049	0.3303396298	0.1721919305	0.2840270960	0.0953245264	0.4218886096
u8	1.0000000000	0.4550131456	0.4422909035	0.4106419234	0.1606625962	0.4848108606	0.2454796549	0.1967634871	0.1749723276	0.3967372110
u9	1.0000000000	0.6667717903	0.5836720487	0.2989199777	0.3316391681	0.2876538381	0.0863880493	0.2941096980	0.1927450172	0.4157666208
u10	1.0000000000	0.7181083837	0.5498923160	0.3486124738	0.2642577028	0.3303396298	0.1750424475	0.1838907204	0.0855041310	0.4061830894
	1.0000000000	0.6558438254	0.5162576527	0.3271055115	0.2230057597	0.2718598284	0.1525478146	0.2119110109	0.1420510416	
u11	1.0000000000	0.4156549007	0.3830920550	0.1111789726	0.2027746573	0.2129490556	0.0457222186	0.0541735978	0.0922827532	0.2797586901
u12	1.0000000000	0.6721275048	0.3175015842	0.5332722627	0.2176494081	0.0953892721	0.1267165021	0.1387081836	0.2766597852	0.3753360559
u13	1.0000000000	0.8572546898	0.2298654380	0.1094244053	0.1509889398	0.1520094349	0.0575730042	0.1890632679	0.2413976852	0.3319529850
u14	1.0000000000	0.6417717903	0.4386115355	0.1853892307	0.3316391681	0.2477338797	0.0636020878	0.0553973311	0.1664846156	0.3478477377
u15	1.0000000000	0.6538141923	0.4529439285	0.2707744696	0.2520771948	0.1389875248	0.0834893181	0.1695891059	0.1670359765	0.3543013012
u16	1.0000000000	0.7181083837	0.4250464421	0.1307322300	0.1369738938	0.3199502821	0.0652882639	0.2715732931	0.0878294192	0.3506113564
u17	1.0000000000	0.8572546898	0.2459368666	0.1122545238	0.1194184481	0.1049757774	0.0586309280	0.0672552501	0.5338699870	0.3443996079
u18	1.0000000000	0.4156549007	0.4029604978	0.4009659810	0.1860811357	0.0905309890	0.0556554862	0.0520956670	0.2892259228	0.3214633978
u19	1.0000000000	0.5433730159	0.3980920550	0.3065898541	0.2442724054	0.1662750182	0.0537116484	0.0589461949	0.1829375190	0.3282441901
u20	1.0000000000	0.6651955479	0.3819044955	0.2834616833	0.1298781024	0.1659657923	0.1701934757	0.0574067405	0.1232092817	0.3308016799
	1.0000000000	0.6440209616	0.3675954898	0.2444043613	0.1971753354	0.1694767026	0.0780582933	0.1114208632	0.2160932945	
u21	1.0000000000	0.4798412698	0.3830920550	0.1161657100	0.1759016484	0.1747010936	0.0436394100	0.0541272216	0.1798744106	0.2897047577
u22	1.0000000000	0.4156549007	0.2677300674	0.0991950295	0.1294363352	0.1182692772	0.0521958080	0.0577193469	0.1222735149	0.2513860311
u23	1.0000000000	0.8572546898	0.3344259054	0.1749977773	0.1411894859	0.1274236707	0.0763381087	0.0647015906	0.1456780265	0.3246676950
u24	1.0000000000	0.4513763576	0.2515655688	0.1116881714	0.1653422980	0.2824515512	0.0359754139	0.0500425742	0.1165184547	0.2738844878
u25	1.0000000000	0.8572546898	0.2831498443	0.1994574037	0.1779557639	0.1337513939	0.0701009513	0.0758646318	0.1716735878	0.3299120296

u26	1.0000000000	0.3500131456	0.3572666589	0.1105054844	0.1599196461	0.1324537624	0.0712680968	0.0500413483	0.0710031827	0.2558301472
u27	1.0000000000	0.7859649123	0.3825082308	0.3693863122	0.3398164697	0.3094763326	0.1574448060	0.0786646767	0.3004694774	0.4137479131
u28	1.0000000000	0.8572546898	0.2298654380	0.1311241509	0.1779557639	0.1249228860	0.0734873351	0.1706492778	0.3021804244	0.3408266629
u29	1.0000000000	0.6209066372	0.2384406379	0.1035805519	0.1173704410	0.0769574581	0.0378936836	0.0546723698	0.1056782444	0.2617222249
u30	1.0000000000	0.5159873950	0.3397427708	0.1237245737	0.3134710798	0.4523702530	0.0479488444	0.0523238743	0.1232092817	0.3298642303
	1.0000000000	0.6191508687	0.3067787177	0.1539825165	0.1898358932	0.1932777679	0.0666292458	0.0708806912	0.1638558605	
u31	1.0000000000	0.4437637801	0.2940975850	0.3271580047	0.3855773165	0.0741060278	0.1230058091	0.1409263426	0.1078140505	0.3218276574
u32	1.0000000000	0.4197567354	0.3057378856	0.3007517776	0.1227840984	0.0687668479	0.0380898841	0.0544675432	0.1108219904	0.2690196403
u33	1.0000000000	0.4197567354	0.3910924714	0.1000420887	0.1237565529	0.1083789857	0.0375717398	0.0532412111	0.1085208540	0.2602622932
u34	1.0000000000	0.5911538462	0.3071738364	0.2572904700	0.0858339540	0.0965633739	0.0636096649	0.0834892167	0.1050084055	0.2877914186
u35	1.0000000000	0.3531057371	0.3715799697	0.4256702038	0.0990083271	0.0753592909	0.5638130875	0.0807536945	0.0643367216	0.3370696702
u36	1.0000000000	0.5739652015	0.2600138513	0.3293116663	0.2799774240	0.1329194570	0.0463270747	0.0775909213	0.1523453292	0.3169389917
u37	1.0000000000	0.6169000934	0.2909010510	0.1030649743	0.0785024984	0.0649636550	0.0415050342	0.0498366878	0.1178005060	0.2626082778
u38	1.0000000000	0.4716798123	0.2593116506	0.1477961150	0.0971248508	0.0793349692	0.0548459359	0.0523802682	0.1065611200	0.2521149691
u39	1.0000000000	0.3750131456	0.2228732829	0.1225985979	0.1899747660	0.0680391394	0.0462888914	0.0567237225	0.0795432086	0.2401171949
u40	1.0000000000	0.4068815468	0.2268175058	0.3656943554	0.2969271784	0.0945402584	0.2165303303	0.1500846944	0.1472721067	0.3227497751
	1.0000000000	0.4671976634	0.2929599090	0.2479378254	0.1759466966	0.0862972005	0.1231587452	0.0799494302	0.1100024292	
u41	1.0000000000	0.5579761905	0.2903758415	0.2036919588	0.3957311494	0.1363516387	0.0527724379	0.0608801528	0.1764835413	0.3193625457
u42	1.0000000000	0.5713677989	0.3471107999	0.1682427592	0.3007162390	0.1390810724	0.0562776427	0.0899444998	0.1084991141	0.3090266584
u43	1.0000000000	0.6265414651	0.2934752938	0.1049088562	0.1126296125	0.1429788292	0.0418567616	0.0567391058	0.0922389242	0.2745965387
u44	1.0000000000	0.6391971917	0.3231165549	0.1186602549	0.2558170230	0.1306222493	0.0466559128	0.0595633745	0.1917154369	0.3072608887
u45	1.0000000000	0.6102530803	0.3708587443	0.2707549225	0.0822843066	0.1255346876	0.0554426038	0.0856506681	0.0790463315	0.2977583716
u46	1.0000000000	0.4197567354	0.3838660907	0.3072482364	0.1274711708	0.2186658444	0.0848650307	0.1294951733	0.0630246757	0.3038214397
u47	1.0000000000	0.6531113331	0.3248381891	0.1462739562	0.3387071256	0.1151836364	0.0618494106	0.0681893157	0.0663344455	0.3082763791
u48	1.0000000000	0.6873260073	0.3390733673	0.1423438776	0.3072263135	0.0972951307	0.0662659857	0.0900472878	0.1026710777	0.3146943386
u49	1.0000000000	0.6572652786	0.3327055622	0.1671377063	0.4321874528	0.1245961267	0.0811445556	0.1196436039	0.1180401993	0.3369689428
u50	1.0000000000	0.4449113876	0.4044258047	0.1726158304	0.0903307098	0.0718388474	0.0398744194	0.0529966667	0.1901457652	0.2741266035
	1.0000000000	0.5867706468	0.3409846248	0.1801878359	0.2443101103	0.1302148063	0.0587004761	0.0813149848	0.1188199512	
u51	1.0000000000	0.4197567354	0.3341070492	0.2235516762	0.1003556499	0.1407368903	0.0776829051	0.0736139024	0.0662398029	0.2706716235
u52	1.0000000000	0.4862317748	0.3335098888	0.1111789726	0.3080033703	0.1008685982	0.0482751315	0.0529809669	0.1067341036	0.2830869785
u53	1.0000000000	0.4173611111	0.3219102727	0.2925010463	0.1332679334	0.2692868952	0.0567789064	0.0481323108	0.1352799343	0.2971687122
u54	1.0000000000	0.5337854251	0.3339417651	0.1304976761	0.2177649067	0.1177249966	0.0470767323	0.2391112791	0.1124747231	0.3035975005
u55	1.0000000000	0.4636291486	0.2790626911	0.1116943389	0.1203033706	0.0752041795	0.0497764181	0.0606257062	0.1615689230	0.2579849751
u56	1.0000000000	0.4364377289	0.3168500471	0.1778373641	0.1096086331	0.1825835992	0.0421941046	0.0507343077	0.0708403747	0.2652317955
u57	1.0000000000	0.6071686157	0.3808090597	0.1131967272	0.0945850014	0.0844791985	0.0451169752	0.0516608446	0.1172206924	0.2771374572
u58	1.0000000000	0.3288476444	0.4159919078	0.6179757505	0.3009258419	0.1784855327	0.1399223615	0.0800998365	0.0922845785	0.3505037171
u59	1.0000000000	0.5365445665	0.3692043325	0.1549553051	0.3777802496	0.1619986191	0.0514926387	0.0816406852	0.0921853968	0.3139779771
u60	1.0000000000	0.3428472478	0.2637449601	0.5728516255	0.1979779802	0.1352498938	0.0878046734	0.0536813679	0.1275268685	0.3090760686
	1.0000000000	0.4572609998	0.3349131974	0.2506240482	0.1960572937	0.1446618403	0.0646120847	0.0792281207	0.1082355398	

MAP

MAP

AP

AP

REFERENCES

- [1] G. Fischer, "User Modeling in Human-Computer Interaction," *User Modeling and User-Adapted Interaction*, vol. 11, pp. 65-86, 2001.
- [2] A. Kobsa, "A Component Architecture for Dynamically Managing Privacy Constraints in Personalized Web-Based Systems," in *Privacy Enhancing Technologies*. vol. 2760, R. Dingledine, Ed., ed: Springer Berlin / Heidelberg, pp. 177-188, 2003.
- [3] Y. Wang, "A Framework for Privacy-Enhanced Personalization," *Doctoral Dissertation, University of California, Irvine*, Oct. 2010.
- [4] P. de Vrieze, "Fundaments of adaptive personalization," *PhD thesis, Radboud University Nijmegen*. ISBN-13: 978-90-9021113-8, 2006.
- [5] S.E. Middleton, "Capturing knowledge of user preferences with recommender systems," *Ph.D. thesis, University of Southampton*, May 2003.
- [6] G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, pp. 734-749, 2005.
- [7] A. Kobsa, "Modeling the user's conceptual knowledge in BGP-MS, a user modeling shell system," *Comput. Intell.*, vol. 6, pp. 193-208, 1991.
- [8] A. Kobsa, "Generic User Modeling Systems," *User Modeling and User-Adapted Interaction*, vol. 11, pp. 49-63, 2001.
- [9] J. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative Filtering Recommender Systems," in *The Adaptive Web*. vol. 4321, P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds., ed: Springer Berlin / Heidelberg, 2007, pp. 291-324.
- [10] Y. Sun, H. Li, I. G. Councill, J. Huang, W.-C. Lee, and C. L. Giles, "Personalized ranking for digital libraries based on log analysis," presented at the Proceeding of the 10th ACM workshop on Web information and data management, Napa Valley, California, USA, pp. 133-140, 2008.
- [11] G.-w. You and S.-w. Hwang, "Search structures and algorithms for personalized ranking," *Inf. Sci.*, vol. 178, pp. 3925-3942, 2008.
- [12] Y.-H. Yang, Y.-C. Lin, and H. Chen, "Personalized music emotion recognition," presented at the Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, Boston, MA, USA, pp. 748-749, 2009.
- [13] P. Zigoris and Y. Zhang, "Bayesian adaptive user profiling with explicit \& implicit feedback," *presented at the Proceedings of the 15th ACM international conference on Information and knowledge management*, Arlington, Virginia, USA, pp. 397-404, 2006.
- [14] X. Amatriain, J. M. Pujol, and N. Oliver, "I Like It... I Like It Not: Evaluating User Ratings Noise in Recommender Systems," *presented at the Proceedings of the 17th International Conference on User Modeling, Adaptation, and Personalization: formerly UM and AH*, Trento, Italy, pp. 247-258, 2009.
- [15] A. Kobsa, "Generic user modeling systems," in *The adaptive web*, B. Peter, K. Alfred, and N. Wolfgang, Eds., ed: Springer-Verlag, pp. 136-154, 2007.
- [16] P. R. Cohen and C. R. Perrault, "Elements of a Plan-Based Theory of Speech Acts*," *Cognitive Science*, vol. 3, pp. 177-212, 1979.
- [17] J. F. Allen, "A Plan-Based Approach to Speech Act Recognition," *Dept. of Computer Science, University of Toronto, Canada*, Technical Report 131/79, 1979.

- [18] R. Kass, "Acquiring a Model of the User's Beliefs from a Cooperative Advisory Dialog," in *Dept. of Information and Computer Science*, Philadelphia, PA, University of Pennsylvania, 1988.
- [19] A. Kobsa. *Benutzermodellierung in Dialogsystemen*. Berlin, Heidelberg, Springer Verlag, Spri. 1985.
- [20] T. Finin. "GUMS: A General User Modeling Shell," in *User models in Dialog Systems*, .A. Kobsa, W. Wahlster Berlin and Ed. Heidelberg: Springer-Verlag, 1989, pp. 411-430.
- [21] G. Brajnik and C. Tasso, "A shell for developing non-monotonic user modeling systems," *Int. J. Hum.-Comput. Stud.*, vol. 40, pp. 31-62, 1994.
- [22] H. Vergara. "PROTUM: A prolog based tool for user modeling," *WIS Memo 10*, WG Knowledge-Based Information Systems, Department of Information Science, University of Konstanz, Germany, 1994.
- [23] A. Paiva and J. Self, "TAGUS — A user and learner modeling workbench," *User Modeling and User-Adapted Interaction*, vol. 4, pp. 197-226, 1994.
- [24] J. Kay. "UM: A Toolkit for User Modeling," In: *Second International Workshop on User Modeling*, Honolulu, HI, 1990, pp. 1-11.
- [25] A. Kobsa, D. Müller and A. Nill. "KN-AHS: An Adaptive Hypertext Modeling System BGP-MS," *Proceedings of the Fourth International Conference Modeling*, Hyannis, MA, PP. 99-105, 1994.
- [26] A. Kobsa and W. Pohl. "The BGP-MS User Modeling System," *User Modeling and User-Adapted Interaction: The Journal of Personalization Research 4*, 59-106, DOI: 10.1007/BF01099428, 1995.
- [27] J. Fink. *User Modeling Servers- Requirements, Design, and Evaluation*. Amsterdam, Netherlands: IOS Press, 2004.
- [28] Group Lens, <http://www.grouplens.org/>, last retrieved at Aug. 2011.
- [29] G. Fischer, "User Modeling: the long and winding road," presented at the Proceedings of the seventh international conference on user modeling, Banff, Canada, pp. 349-355, 1999.
- [30] F. Albrecht, N. Koch, and T. Tiller, "SmexWeb: An adaptive web-based hypermedia teaching system," *J. Interact. Learn. Res.*, vol. 11, pp. 367-388, 2000.
- [31] F. Zhang, Z. Song, and H. Zhang, "Web Service Based Architecture and Ontology Based User Model for Cross-System Personalization," presented at the Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, pp. 849-852, 2006.
- [32] F. Qiu and J. Cho, "Automatic identification of user interest for personalized search," presented at the Proceedings of the 15th international conference on World Wide Web, Edinburgh, Scotland, pp. 727-736, 2006.
- [33] A. Micarelli, F. Gaspiretti, F. Sciarrone, and S. Gauch, "Personalized search on the world wide web," in *The adaptive web*, B. Peter, K. Alfred, and N. Wolfgang, Eds., ed: Springer-Verlag, 2007, pp. 195-230.
- [34] S. Kordumova, I. Kostadinovska, M. Barbieri, V. Pronk, and J. Korst, "Personalized Implicit Learning in a Music Recommender System," in *User Modeling, Adaptation, and Personalization*. vol. 6075, P. De Bra, A. Kobsa, and D. Chin, Eds., ed: Springer Berlin / Heidelberg, 2010, pp. 351-362.
- [35] S. Berkovsky, T. Kuflik, and F. Ricci, "Mediation of user models for enhanced personalization in recommender systems," *User Modeling and User-Adapted Interaction*, vol. 18, pp. 245-286, 2008.

- [36] M. Morita and Y. Shinoda, "Information filtering based on user behavior analysis and best match text retrieval," presented at the Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, Dublin, Ireland, pp. 272-281, 1994.
- [37] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: an open architecture for collaborative filtering of netnews," presented at the Proceedings of the 1994 ACM conference on Computer supported cooperative work, Chapel Hill, North Carolina, United States, pp. 175-186, 1994.
- [38] Q. Zhang. "Collaborative Filtering Based Service Ranking with Invocation Histories." Master of Science Thesis (MSc.), Ryerson University, Toronto, Canada, 2011.
- [39] R.J. Mooney, P.N. Bennett, and L. Roy, "Book Recommending Using Text Categorization with Extracted Information," Proc. Recommender Systems Papers from 1998 Workshop, Technical Report WS-98-08, 1998.
- [40] M. Pazzani and D. Billsus, "Learning and Revising User Profiles: The Identification of Interesting Web Sites," *Mach. Learn.*, vol. 27, pp. 313-331, 1997.
- [41] J. B. Schafer, J. A. Konstan, and J. Riedl, "Meta-recommendation systems: user-controlled integration of diverse recommendations," presented at the Proceedings of the eleventh international conference on Information and knowledge management, McLean, Virginia, USA, pp. 43-51, 2002.
- [42] J.S. Breese, D. Heckerman, and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," Proc. 14th Conf. Uncertainty in Artificial Intelligence, pp. 43-52, July 1998.
- [43] D. Austin, A. Barbir, C. Ferris, and S. Garg, "Web services architecture requirements", *W3C Working Group Note*, W3C, 2002. Available at: <http://www.w3.org/TR/wsa-reqs>, last retrieved at Aug. 2011.
- [44] M. Papazoglou, *Web Services: Principle and Technology (1st ed.)*, Harlow: Pearson Education Limited, 2008.
- [45] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, "Deploying and managing Web Services: issues, solutions, and directions," *The VLDB Journal*, vol. 17, pp. 537-572, 2008.
- [46] S. A. McIlraith and D. L. Martin, "Bringing Semantics to Web Services," *IEEE Intelligent Systems*, vol. 18, pp. 90-93, 2003.
- [47] W3C, "Resource Description Framework (RDF)", <http://www.w3.org/RDF/>, last retrieved at Aug. 2011.
- [48] W3C, "RDF Schema (RDF-S)", <http://www.w3.org/TR/rdf-schema/>, last retrieved at Aug. 2011.
- [49] W3C, "Web Ontology Language (OWL)", <http://www.w3.org/TR/owl-features/>, last retrieved at Aug. 2011.
- [50] W3C, "Web Ontology Language for Services (OWL-S)", <http://www.w3.org/Submission/OWL-S/>, last retrieved at Aug. 2011.
- [51] J. Zhou, J. Koivisto, and E. Niemelä, "A Survey on Semantic Web Services and a Case Study", in *Proc. CSCWD*, 2006, pp. 763-769.
- [52] IBM Services Architecture Team, "Understanding WSDL in a UDDI registry, Part 1", <http://www.ibm.com/developerworks/webservices/library/ws-wsdl/>, last retrieved at Aug. 2011.

- [53] J. Wu and Z. Wu, "Similarity-based Web Service Matchmaking," presented at the Proceedings of the 2005 IEEE International Conference on Services Computing - Volume 01, pp. 287-294, 2005.
- [54] G.-R. Xue, H.-J. Zeng, Z. Chen, Y. Yu, W.-Y. Ma, W. Xi, and W. Fan, "Optimizing web search using web click-through data," presented at the Proceedings of the thirteenth ACM international conference on Information and knowledge management, Washington, D.C., USA, pp. 118-126, 2004.
- [55] E. Agichtein, E. Brill, and S. Dumais, "Improving web search ranking by incorporating user behavior information," *presented at the Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, Seattle, Washington, USA, pp. 19-26, 2006.
- [56] W3C Working Group, "QoS for Web Services: Requirements and Possible Approaches", <http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/>, last retrieved at Aug. 2011.
- [57] A. Averbakh, D. Krause, and D. Skoutas, "Recommend me a Service: Personalized Semantic Web Service Matchmaking." *In: 17th Workshop on Adaptivity and User Modeling in Interactive Systems*, 2009.
- [58] Y. Wang and J. Vassileva, "Toward Trust and Reputation Based Web Service Selection: A Survey," *Int'l Trans. Systems Science and Applications*, vol. 3, no. 2, pp. 118-132, 2007.
- [59] WSMO, "Non-functional properties in Web Services", <http://www.wsmo.org/TR/d28/d28.4/v0.1/>, last retrieved at Aug. 2011.
- [60] C. Tziviskou and M. Brambilla, "Semantic personalization of web portal contents," presented at the Proceedings of the 16th international conference on World Wide Web, Banff, Alberta, Canada, pp. 1245-1246, 2007.
- [61] D. Mobedpour, C. Ding, and C.-H. Chi, "A QoS Query Language for User-Centric Web Service Selection," presented at the Proceedings of the 2010 IEEE International Conference on Services Computing, pp. 273-280, 2010.
- [62] D. Bertram, "Likert scales", <http://pages.cpsc.ucalgary.ca/~saul/wiki/uploads/CPSC681/topic-dane-likert.pdf>, last retrieved at Aug. 2011.
- [63] M. Chen and J. P. Singh, "Computing and using reputations for internet ratings," presented at the Proceedings of the 3rd ACM conference on Electronic Commerce, Tampa, Florida, USA, pp. 154-162, 2001.
- [64] H. Wang, C. Lee, and T. Ho, "Combining subjective and objective QoS factors for personalized Web Service selection", presented at *Expert Syst. Appl.*, 2007, pp.571-584.
- [65] Seekda, "UDDI", <http://seekda.com/blog/the-fairytale-of-uddi-registry-and-public-web-services/>, last retrieved at Aug. 2011.
- [66] Web Services search engine, "xMethods", <http://xmethods.com/ve2/index.po>, last retrieved at Aug. 2011.
- [67] E. Al-Masri and Q. H. Mahmoud, "QoS-based Discovery and Ranking of Web Services," in *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, 2007, pp. 529-534.
- [68] A. Dasdan, K. Tsioutsoulis and E. Velipasaoglu, "Web search engine metrics: (direct metrics to measure user satisfaction)", in *Proc. WWW*, 2010, pp.1343-1344.
- [69] A. Turpin and F. Scholer, "User performance versus precision measures for simple search tasks," *presented at the Proceedings of the 29th annual international ACM SIGIR*

conference on Research and development in information retrieval, Seattle, Washington, USA, pp. 11-18, 2006.