9-5-2014 12:00 AM

Intents-based Service Discovery and Integration

Cheng Zheng
The University of Western Ontario

Supervisor
Weiming Shen
The University of Western Ontario Joint Supervisor
Hamada Ghenniwa
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy
© Cheng Zheng 2014

INTENTS-BASED SERVICE DISCOVERY AND INTEGRATION

(Thesis format: Monograph)

by

Cheng Zheng

Graduate Program in Electrical and Computer Engineering
Department of Electrical and Computer Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

With the proliferation of the Web and Web services, when a new application is being developed, it makes sense to seek and leverage some existing Web services rather than implementing the corresponding components from scratch. As a result, significant research efforts have been devoted to the techniques for service discovery and integration. However, most of the existing techniques are based on the ternary participant classification of the Web service architecture which only takes into consideration the involvement of service providers, service brokers, and application developers. The activities of application end users are usually ignored.

This thesis presents an Intents-based service discovery and integration approach at the conceptual level which is inspired by two industrial protocols: Android Intents and Web Intents. The proposed approach is characterized by allowing application end users to participate in the process of service seeking. Instead of directly binding with remote services, application developers can set an intent which semantically represents their service goal in applications. When applications are running, an Intents user agent will resolve their intents and generate candidate service lists. Then application end users can choose a service from the candidate lists to complete their application tasks. The intents in this work are classified into explicit intents, authoritative intents, and naïve intents. This thesis examines in depth the issue of naïve intent resolution analytically and empirically. Based on the empirical analysis, an adaptive intent resolution approach is devised. For validation purposes, this thesis studies two cases to show the advantages of Intents. In addition, a design for the Intents user agent is presented and its proof-of-concept prototype is demonstrated. Finally, Intents and the Intents user agent are applied to integrate Web applications and native applications on mobile devices.

Compared with the traditional techniques for service discovery and integration, the Intents-based approach is innovative and opens up new promising directions in this area. However, Intents is still a newborn framework, and it still has a lot of room for improvement and requires further research and development efforts.

Keywords

Web Services, Intents, Service Discovery and Integration, Assistant Agent, Semantic Integration, Information Retrieval, Mobile Computing

Acknowledgments

First, I would like to express my sincere gratitude to my supervisors Dr. Weiming Shen and Dr. Hamada H. Ghenniwa for their continuous support for my PhD study and research. Their patience, experience, and knowledge helped me in all of my research and writing of this thesis. I learned a lot from them and have progressed rapidly in the past 4 years. I could not imagine what my doctorate life would be without their assistance.

Besides my supervisors, I would like to thank the other members of my thesis committee for their encouragement, insightful comments, and challenges.

My sincere thanks also go to my colleagues in the same lab: Fujun Yang, Afshan Samani, Dr. Yunjiao Xue, Dr. Wafa Ghonaim, Ali Hussain, and Islam Ali for their priceless help. Their enthusiasm on the road of science and technology makes me work hard every day.

I also want to express my great appreciation to Henry Xue and Qi Hao who provided me with a happy life as a volunteer in NRC. They made every effort to teach me how to participate in a project and collaborate with other team members.

Last but not least, I would like to thank my parents for supporting me spiritually throughout my life.

Table of Contents

List of Tables

List of Figures

List of Acronyms

CRUD	Create, Retrieve, Update, and Delete
CSS	Cascading Style Sheets
DAML	DARPA Agent Markup Language
EM	Expectation-Maximization
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDF	Inverse Document Frequency
IR	Information Retrieval
JSON	JavaScript Object Notation
MAP	Mean Average Precision
MLE	Maximum Likelihood Estimation
MRR	Mean Reciprocal Rank
OS	Operating System
OWL	Web Ontology Language
PC	Personal Computer
QoS	Quality of Service
RDF	Resource Description Framework
REST	REpresentational State Transfer
SOAP	Simple Object Access Protocol

TF	Term Frequency
UDDI	Universal Description, Discovery and Integration
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
W3C	The World Wide Web Consortium
WSDL	Web Services Description Language

Nomenclature

d	Text document
$\|d\|$	Document length
q	Text query
t	Index term
$sim(q,d)$	The similarity between query q and document d
N	The number of all documents
$df(t)$	Document frequency for term t
$f(t,d)$	The count of term t in document d
I	Intent
A	Intents advertisement
S	Similarity model for naïve intent resolution
C	The set of services having passed intent type checking and data type matching
$f_{common}(C)$	The common fields of C
M	IR model set
m	IR model
$\Im$	Similarity model template set
s	Similarity model template
$< s,m >$	A pair of a similarity model template and an IR model which determines a specific S

E	Evaluation measure
Γ	All intents used to search for services
$\Re_e(I)$	All the effective top services for intent I
p_k	Precision at rank k
$rank_I$	Rank of the first relevant service for intent I

Chapter 1

1 Introduction

In the early 1990s, Tim Berners-Lee [Berners-Lee, 1992] wrote a proposal which articulated the idea of the World Wide Web (later also named the Web). Within just a few years of its birth, the Web had achieved unbelievable success and won substantial fame. *WorldWideWebSize.com*[1] estimates that there exist at least 1.63 billion pages currently on the Web. The fast growth of the Web along with its vast amount of information including text, audio and video has created an era of information explosion. On the other hand, the Web initiative was to establish a global man-knowledge sharing system with hypertext to link pieces of the content in text or other media which mimics the human association of ideas. In addition to such ambition, after decades of evolution, the functionalities of the Web have been far beyond simple knowledge sharing with the development of transaction processing systems [Gray and Reuter, 1992], code on demand [Fuggetta et al., 1998], and the representational state transfer (REST) architectural style [Fielding, 2000; Fielding and Taylor, 2002]. Today's Web has been an aggregation of social networking sites, e-business services, blogs, wikis and online games which influences every corner of society.

With the proliferation of the Web, Web services emerged as a communication method between two devices over the Web. A Web service is a software component, placed on the Web, that exposes its access through a programming interface and adopts common protocols such as Extensible Markup Language (XML) for communications [Chappell and Jewell, 2002; Manes, 2003; Newcomer, 2002]. *W3C*[2] defines the Web services architecture with a series of protocols including WSDL, SOAP and UDDI. Web Services Description Language (WSDL) is used for describing the functionality offered by a Web service and encompasses the information such as operations and their parameters in the

[1] "The size of the World Wide Web," accessed Jan 10, 2014, http://www.worldwidewebsize.com/

[2] The World Wide Web Consortium

form of XML. Simple Object Access Protocol (SOAP) is a protocol for exchanging messages between service consumer clients and Web services. Universal Description, Discovery and Integration (UDDI) defines the standard for constructing platform-independent service registries to provide a mechanism to register and locate Web services. These standards constitute the core elements of *SOAP Web services* which is one major family of Web services. Compared with the SOAP Web services, in recent years, another family of Web services, named *REST Web services*, which are created directly on Hypertext Transfer Protocol (HTTP) and uniform resource locators (URL) have achieved wide popularity as well.

1.1 Research Issue

Over the past decade, the scale of Web services has surged significantly. In 2010, Zheng et al. [Zheng et al., 2010] claimed that over 28,500 public Web services exist on the Web under their monitoring. With such a scale of Web services, it is an attractive and sensible option to seek suitable existing Web services and integrate them instead of implementing the corresponding components from scratch in a new product development. In order to achieve this vision, the techniques on seeking and integrating a best suitable service for a system requirement, i.e., service discovery and integration, should be developed.

Most of the state-of-the-art research efforts on service discovery and integration are based on the currently famous Web services ternary participant role classification. In the classification, participant roles revolving around Web services are divided into three categories: *service providers, service brokers* and *service consumers* [Al-Jaroodi and Mohamed, 2012]. Service providers take the responsibility for designing and developing a Web service. Service brokers collect the available services and advocate them to the rest of the world through mechanisms such as UDDI or Web services search engines. Service consumers find and locate the desired services and create their products depending on these services. Based on this classification, if the product is assumed to be a software application, the process of discovering and integrating a desired service is usually as follows:

> The developers of the software application manually create queries representing their functional and non-functional service needs and search the service registries provisioned by service brokers. Once they discover a suitable service, a piece of binding script for the service is hardcoded into the application. When *end users* of the application use it, the binding script will directly communicate with the remote service.

In the above service discovery and integration process, the application developers take on almost all the work for seeking and binding the desired remote service while at the same time the end users, who indirectly use the service are ruled out of the steps for determining the working service. The very limited end user participation in service discovery and integration may cause serious problems. For example, application developers may choose service A for their released product. However, while using the product, some end users may prefer service B and others have interests in service C. Even worse, if A is blocked in a network, the product may be out of order to the end users in the network. As a result, the product marketing will be seriously affected.

1.2 Proposed Solution

In order to address the issues incurred by limited end user participation, an Intents-based approach for service discovery and integration is proposed in this thesis. The approach is inspired by two industrial protocols: Web Intents and Android Intents. The difference between Intents and the existing techniques is that Intents introduces the involvement of application end users into the service seeking process, i.e., application end users instead of application developers ultimately decide which service is selected to complete the given application task.

In Intents, if an application developer wants to leave the right of selecting a working service to the end users of an application, he/she can create a corresponding *intent* in the application. The intent is a data structure which semantically represents the operation of the service desired by the application developer. When an end user uses the application and triggers the intent, a message enclosing the intent is created and sent to a *user agent*. The user agent resolves the intent and generates a list of candidate services to the end user according to the content of the intent. Then the end user selects a service from the candidate service list to continue the application.

The term “Intents” means the proposed approach for service discovery and integration revolves around a collection of various intents. Basically, this work classifies intents into *explicit intents*, *authoritative intents*, and *naïve intents*. Explicit intents specifically point to a desired service. If an application developer uses the explicit intent, it means the developer has determined the desired service and wants to rule application end users out of the service seeking process. Authoritative and naïve intents are together called *implicit intents*. It is only by implicit intents that end users have the right of taking a role in service discovery. The difference between authoritative and naïve intents is that the former category asks for third-party or authority participation. The specification of an authoritative intent should be made up by an authoritative organization so that the authoritative intent is more reliable, robust and effective in service discovery. In contrast, naïve intents sacrifice the involvement of authoritative organizations for flexibility.

A Web service capable of accepting an intent is called *Intents service*. When a message enclosing an intent arrives at the service, it can extract the data from the intent for further internal data processing. An Intents service should be marked with an *Intents advertisement* which can be leveraged by the user agent in creating candidate services. This work formally defines the concept of Intents advertisements and comprehensively discusses the ways of publishing an Intents advertisement to be captured by the user agent. Since currently SOAP and REST Web services have together dominated the Web services world, this work also presents how to wrap a SOAP or REST Web service to create an Intents service.

The process of resolving an intent by user agents is complicated. This work presents the process systematically including the utilization of different levels of Intents advertisement registries for the generation of candidate services. In addition, this work formally defines the process of resolving naïve intents as an optimization problem and applies information retrieval (IR) models to the problem. However, this problem is difficult to solve by analytic methodologies. Instead, this work conducts a set of experiments on an Intents dataset acquired from the real world for an empirical study of the problem. Based on the empirical study, the work presents an adaptive approach for intent resolution.

The user agent plays a pivotal role in intent resolution. It takes the responsibility of storing user-collected Intents advertisements, rendering the user interfaces of Web applications and Intents services, resolving intents for the generation of candidate services, and communicating with remote services. This work shows a design of the user agent and implements a prototype based on the design. The prototype demonstrates the advantage of applying Intents to the case of text sharing.

With the development of mobile devices, implementing a mobile application in the form of Web applications or native applications is a hard choice. On the other hand, integrating Web applications and native applications to make the most of their advantages is an attractive direction. Since Intents originates from Android Intents which is used to communicate Android native components, this work applies Intents to integrate Web applications and Android applications to show the benefits and advantages of Intents in this direction.

1.3 Outline of the Thesis

The rest of the thesis is organized as follows. Chapter 2 reviews the background and related research literature. Chapter 3 formally and systematically describes the proposed Intents-based approach for service discovery and integration in terms of the architecture, intent data structure, Intents services, and the intent resolution mechanism. In addition, two cases are applied to demonstrate the benefits and advantages of Intents in service discovery and integration. Chapter 4 defines the intent resolution problem as an optimization problem and conducts an empirical study to address the problem. Based on the empirical study, an adaptive intent resolution approach is presented. Chapter 5 presents a design and implementation of user agents. In addition, this chapter also presents that Intents and the proposed Intents user agent is leveraged to address the problem of integrating Web applications and native applications on mobile devices. Chapter 6 provides the conclusion and discusses the future work.

Chapter 2

2 Background and Literature Review

This work aims to employ Intents to address the problem of service discovery and integration. Thus the existing efforts which have been devoted to the field will be investigated in this chapter. On the other hand, since information retrieval (IR) techniques are applied in the naïve intent resolution process, which is one of the key steps for the proposed approach, we will also review their related literature.

2.1 Web Service Discovery and Integration

The resources used for service discovery and integration include service functionality and quality. In addition, some researchers tried to utilize other auxiliary information such as service related Web pages, user history log, and peer data. With the development of the semantic Web, ontologies are also applied to this field.

Android Intents and Web Intents are two industrial protocols which introduce application end users into the process of seeking and integrating Android and Web applications, respectively. The two protocols also provided inspiration for this work.

2.1.1 Service Functionalities and Qualities

One early attempt for Web service discovery and integration was the UDDI initiative. A business can register its related information with a UDDI registry for the services it provides [Jewell and Chappell, 2002]. UDDI specifies three types of information: white pages, yellow pages, and green pages. A white page contains basic contact information and identifiers about a service provider, including business name, address, contact information, and its unique identifiers. Yellow pages have information that describes the taxonomy of Web services. Service consumers are able to browse a UDDI registry for desired services by such information. Green pages are used to describe how to access a Web service such as service binding information. On the other hand, UDDI implementations usually have user-friendly interfaces through which service consumers may search for their desired services. However, service searching in UDDI is still keyword-based. With the development of Web services, keyword-based searching is far from satisfying user requirements. As a result, a great deal of effort is devoted to explore innovative techniques for service discovery and integration.

User requirements for Web services can be divided into functional and non-functional categories. Functional requirements indicate if the functionalities of a service satisfy user demands. Non-functional requirements indicate if service quality properties such as price, reputation and response time are appropriate for users. A lot of previous research efforts for service discovery and integration are inspired by the two types of user requirements.

The Woogle project [Dong et al., 2004] attempted to compute functional similarities based on input parameters, output parameters and operation names in Web services. Their approach clusters input and output parameters into concept groups and the concept groups are exploited in the computation of input and output similarities. In addition to a keyword search, the proposed approach also uses template search and composition search. In a template search, users are able to represent a query in the form of service structures including input parameters, output parameters, and operation names. Composition search means that if any single service operation could not fulfill user requirements but a composition of some service operations can, the operation composition should be returned.

Wang and Stroulia [Wang and Stroulia, 2003] presented a set of similarity assessment methods based on WSDL documents and the WordNet lexical database [Miller, 1995]. For each WSDL document, in addition to the original words in the document which constitute a word vector, another two vectors are proposed. One is the vector of synonyms for all word senses. The other is the vector of direct hypernyms, hyponyms, and siblings for all word senses. If a user inputs a natural language query for the desired service, the vector space model in IR is applied for each vector and three similarity scores are obtained. Then an overall score which indicates the service relevance is calculated as a linearly weighted combination of the three similarities.

Hatzi et al. [Hatzi et al., 2012] designed and developed a specialized search engine for Web services. It captures Web service descriptions from the Web, parses them and constructs an index. The search engine adopts TF-IDF and cosine to create similarities between service descriptive documents and user queries. If a user inputs a multiple-field query to describe his/her desired service, a linearly weighted combination of all the similarities in each field will be applied.

Plebani and Pernici [Plebani and Pernici, 2009] attempted to address the similarity problem in Web services through graph theory. They organized the operation names, parameter names, and parameter data types of a Web service into a tree structure. Inspired by the assignment problems in bipartite graphs [Wang et al., 2005; Wolsey, 1998], Plebani and Pernici treated operation name terms and parameter name terms in two Web services as the two separate sets in a bipartite graph. Based on the bipartite graph, the closeness of any two terms is modeled by their edge weight. Then operation similarities and parameter similarities can be obtained by maximizing the average weight over all the matching assignments in the bipartite graph. They also defined parameter data type similarities and applied WordNet in their computation. Then all the similarities for the operation names, parameter names, and parameter data types are combined according to the Web service tree structure. Liu et al. [Liu et al., 2010] later improved this method by taking account of term relations within each set in bipartite graphs. They demonstrated the effectiveness of their improvement by a set of experiments.

The approaches mentioned above mainly focus on employing user functional requirements and service functionalities to address the problem of service discovery and integration. With the growth of Web services, there could be the case that many services provide similar functionalities. As a result, more and more researchers began to apply other service attributes such as service qualities in seeking desired services.

Al-Masri and Mahmoud [Al-Masri and Mahmoud, 2007] proposed a set of service quality attributes such as response time, throughput, availability, accessibility, interoperability, and cost. They also defined how to compute these attributes and attempted to use a linear combination of these attributes to construct a relevance function. In order to demonstrate the significance of service qualities in service discovery, Al-Masri and Mahmoud [Al-Masri and Mahmoud, 2009] conducted a survey and discussed the impact of service quality on service discovery and integration. They divided user queries for service goals into exploratory and informational categories. Exploratory queries have no service quality requirements while informational queries do. They made statistics of user queries for the two categories. The results show that more than 80 percent of user queries are informational which demonstrate that service qualities are significant in service discovery and integration.

Ran [Ran, 2003] also noticed the importance of service qualities and suggested an extended model for the UDDI architecture. In the new model, a UDDI service registry should have both the functionality information and quality attributes of registered services. The quality attributes come from service providers. In addition, Ran introduced a Web service QoS (Quality of Service) certifier which takes the responsibility for verifying any service quality attribute from service suppliers. With the extended UDDI architecture, user requesters are able to search for the desired services by queries with constraints on service qualities.

Canali et al. [Canali et al., 2013] divided service qualities into static and dynamic qualities. Static qualities (e.g., service provider security or reputation) remain the same or change very slowly over time. In contrast, dynamic qualities (e.g., response time or throughput) may change on a per-invocation basis. They claimed that most techniques treated dynamic qualities as static and may lead to very poor performance in realistic scenarios. So they proposed a set of algorithms for selecting Web services by satisfying both static and dynamic requirements.

Hang et al. [Hang et al., 2012] employed trust as an assessment for service qualities. They proposed a model for the trust of a service based on both positive and negative evidence for the service. The beta distribution is applied in the model. In addition, they also treated composite services as a statistical mixture of beta distributions, each for a constituent service. When constituent services behind a composite service cannot be fully observable, the trust of each constituent service may be estimated based on its contribution to the composite service.

Mehdi et al. [Mehdi et al., 2012] improved Hang et al.'s method by extending the positive and negative evidence classes to a set of more than two quality classes so that more evidence degrees can be considered. As a result, they adopted the multinomial Dirichlet distribution [Bouguila, 2008] to model the trust for each single Web service in which the Dirichlet distribution [Bishop, 2006] is a multivariate generalization of the beta distribution. As for composed services, they used two ways (Bayesian networks [Jordan, 1998] and a mixture of multinomial Dirichlet distributions) to model their trust.

Mobedpour and Ding [Mobedpour and Ding, 2013] noted the significance of assisting users in the formulation of QoS-based queries. Thus their work mainly focuses on user interfaces for service query formulation in three aspects. First, a tool is integrated to help non-expert users gain a perception of QoS value ranges by browsing through available services. The tool is designed because ordinary users have little idea of QoS values. Then user service requests are divided into exact and fuzzy classes. The former is for service qualities which users know clearly and the latter is for those when users have only vague requirements. In addition, if no service is returned for a service request, some service constraints in the request will be relaxed because they are too strict to find a service. Thus a QoS attribute preference order is defined in service requests for users to relax the constraints. Moreover, their approach classifies service discovery results into full and partial matching classes to meet different user requirements.

Yau and Yin [Yau and Yin, 2011] proposed selecting the service that best satisfies user service quality requirements instead of the service with best service qualities which may be overqualified. In order to achieve this objective, they defined a service quality requirement specification which enables users to specify the expected upper and lower bounds, weight, and confidence for each quality attribute in their service requirements. On the other hand, their approach divides service quality attributes into two types. One is the utility type which users want to maximize their values. The other is the cost type which users want to minimize their values. They also designed different normalization methods for the two types. Their service quality satisfactory score for each service is modeled based on prospect theory [Kahneman and Tversky, 1979; Tversky and Kahneman, 1992].

Shi et al. [Shi et al., 2012] argued that experienced users and novice users should be treated separately in service quality computation. Based on this stand, they proposed improvements on the three main steps in the service quality computation of Yau and Yin's work: property normalization, satisfaction calculation, and the aggregation of multiple properties. Experienced users have the freedom to set the parameters for each step because they are familiar with the parameters in practice. On the other hand, novice users are only permitted to use default parameter values.

Xu et al. [Xu et al., 2011] designed and implemented a domain specific Web service management system for bioinformatics research. The system adopts a skyline-based algorithm [Kossmann et al., 2002; Papadias et al., 2003] for Web service recommendation. Each service is described by a quality vector. The skyline algorithm could find the service which is not dominated by any others. The algorithm's feature is in asking no weight input for service qualities.

2.1.2 Collaboration of Auxiliary Information

In addition to service functionalities and qualities, service related Web pages, user history logs, and peer data can also be exploited to collaboratively address the issues in service discovery and integration.

Chan et al. [Chan et al., 2012] argued that user history data on Web service usage play significant roles in service recommendation. They applied collaborative filtering [Chen and Mcleod, 2006; Herlocker et al., 2004] on usage data and created four algorithms for service discovery: operation-operation filtering, user-user filtering, combination filtering, and priorities-assignment strategy. The operation-operation filtering algorithm aims at finding the closest Web service operations for an operation. The user-user filtering algorithm is to find the most relevant users for a user. The two algorithms attempt to seek the similarities for operations and users, respectively. Based on the two filtering algorithms, the combination filtering algorithm was proposed to improve the accuracy of service recommendation. In addition to the above three algorithms, the priorities-assignment strategy algorithm was designed to address the "new item ram-up" problem in applying history data, i.e., the most often used items are easily recommended while at the same time other items are never considered for service recommendation.

Li et al. [Li et al., 2011] developed a Web service search engine named CoWS. The search engine collects Web-related online pages and refines their content to extract a service functional description. Then the functional description is combined with the content from Web service WSDL files to compute Web service functional similarities. On the other hand, the search engine also collects user experience feedback such as Web service ratings and comments. The user feedback is employed to calculate service reputation. Service reputation and other service qualities collaboratively constitute Web service non-functional similarities. The search engine ultimately ranks Web services based on their functional and non-functional similarities.

Yao et al. [Yao et al., 2012] presented a collaborative filtering method based on both user history data and Web service content. They adapted a three-way model [Popescul et al., 2001] to make it applicable to service recommendation. Their new model includes a set of users, a set of Web services, and a set of semantic descriptions for the Web services. The three sets collaboratively imply a set of latent topics which represent user preferences. They adopted the expectation-maximization (EM) algorithm [Dempster et al., 1977] to obtain the parameters for their model from training data.

2.1.3 Semantic Web

With the emergence of the semantic Web, many researchers attempt to address the problem of service discovery and integration by taking advantage of the progress in the semantic Web such as ontologies. An ontology is a formal, explicit specification of a shared conceptualization [Guarino et al., 2009]. As an innovative mechanism for information organization, ontologies are able to represent complex entities and their relationships.

Paolucci et al. [Paolucci et al., 2002] proposed an approach based on DAML-S to augment the search capability of UDDI. DAML-S is a DAML-based (DARPA Agent Markup Language) language for service description. A DAML-S advertisement for Web services consists of three fields: a service profile field, a service model field, and a service grounding field. However, Paolucci et al.'s method only considers the service profile field. Later Bansal and Vidal [Bansal and Vidal, 2003] made an improvement and designed an algorithm by bringing in the service model field.

Si et al. [Si et al., 2013] proposed a service matchmaking approach by considering service input and output parameters. Each parameter is denoted by an ordered pair and the ordered pair consists of a parameter type and its value. As a result, each input or output can be represented as a set of ordered pairs. On the other hand, user queries are also represented as a set of input and output parameter types. In order to obtain the closeness between two parameter types, a directed tree structure named ordered concept tree is constructed from Web service related ontologies. Each node of the tree represents a concept and its directed edge points to a super concept. Equivalent concept nodes are merged to remove duplicate concepts. Based on the ordered concept tree structure, the closeness between two parameter types can be reflected by the distance of their corresponding tree nodes.

Vaculin et al. [Vaculin et al., 2008] devised a service discovery strategy specifically for data providing services. Data providing services provide access to data sources with structured data. The local schema of each data source behind data providing services is represented as a set of RDF (Resource Description Framework) views [Chen et al., 2006] over a shared mediated schema [Halevy, 2001] which is composed of the concepts from a shared OWL (Web Ontology Language) ontology. On the other hand, service requests are represented as input and output tuples with RDF constraints. Their algorithm leverages the two semantic representations to make matches between services and service requests.

2.1.4 Android Intents and Web Intents

Although the techniques in the above subsections have contributed a lot to service discovery and integration, they are mainly built on the ternary participant role classification paradigm. Very little of them takes into consideration further dividing the service consumer role.

Android Intents[1] and Web Intents[2] are two industrial protocols which initiate a new paradigm for application discovery and integration. In this paradigm, if an application is dependent on other applications, its developers are allowed to leave the right of determining the working application to its end users. An application developer just needs to specify his/her service goal in an intent data structure. However, the two protocols are just specific protocols for Android applications and Web applications, respectively. They are designed to address domain issues but not a concept-level or generic method. In addition, their underlying service discovery and integration mechanisms only employ the exactly matching strategy which is too simple and may rule out users' desired services.

Even though the idea of this work comes from Android Intents and Web Intents, the proposed approach is a systematic and extended version and is presented at a higher level for generic uses. The proposed approach in this work also addresses the issues in the two protocols.

2.2 Information Retrieval

Information retrieval aims at finding material of an unstructured nature that satisfies an information need from within large collections [Manning et al., 2008]. In the context of text material, each item in the searched collection is a text document and information needs are represented by text queries. This section will examine IR techniques in terms of two aspects. Since IR models are applied in the optimization problem of naïve intent resolution, mainstream retrieval models will be discussed. On the other hand, each intent has a field named *action* usually composed of short text. Therefore, the techniques for short text document retrieval will also be examined in this section.

2.2.1 Information Retrieval Models

In the past few decades, a variety of approaches for modeling the similarity between a query and a document from a collection have been proposed and developed. These

[1] http://developer.android.com/guide/components/intents-filters.html

[2] http://webintents.org/

models are called IR models. The following paragraphs will present some classic IR models including the Boolean model, the vector space model, the probabilistic model, the language model, and the axiomatic model in sequence.

The Boolean model [Baeza-Yates and Ribeiro-Neto, 1999; Manning et al., 2008] is a simple IR model based on set theory and Boolean algebra. The model judges document relevance by checking the relationship between document representations and query Boolean expressions. Unfortunately, it is difficult for the Boolean model to compute a similarity score for documents to measure their relevance degrees. Thus the application of the Boolean model in information retrieval is very trivial.

The vector space model [Salton et al., 1975; Salton and McGill, 1983] represents documents and queries as vectors in a high-dimensional space. Each vector is a tuple of index term weights. The weighting scheme for index terms may vary greatly in practice. One of the basic weighting schemes is the TF-IDF scheme. For any index term, term frequency (TF) is proportional to its number of occurrences in a document or query, and inverse document frequency (IDF) is inversely proportional to the number of documents containing the index term. A good index term should have a high IDF value to discriminate between documents. The ultimate term weight for each index term is computed by utilizing the product of its TF and IDF values. Given index term weights, the cosine value of the angle between a document vector and a query vector can be treated as their similarity. However, applying only the cosine-based similarity is not enough. For one thing, cosine is not a proper mathematical distance or metric [Munkres, 2000]. It does not have the triangle inequality property and it violates the coincidence axiom. For another, the cosine similarity has a tendency to retrieve more short documents than long documents [Singhal et al., 1996]. Therefore, many implementations of the vector space model modify the cosine similarity to make it more effective in practice. Apache Lucene [McCandless et al., 2010] has a modified built-in implementation achieving wide success. This implementation will be applied in this work.

The probabilistic model tries to address the document-query relevance problem by probability theory [Robertson and Jones, 1976; Manning et al., 2008]. Formally, given a document d and a query q, their relevance can be modeled by the probability $P(R=1|d,q)$, where $R=1$ means d is a relevant document for q. According to the Bayes' law, we have:

$$P(R=1|d,q)=\frac{P(d\mid R=1,q)P(R=1|q)}{P(d\mid q)}$$

Using odds to replace the probability $P(R=1|d,q)$, item $P(d\mid q)$ can be removed. So:

$$O(R\mid d,q)=O(R\mid q)\frac{P(d\mid R=1,q)}{P(d\mid R=0,q)}$$

$O(R\mid q)$ which means the odds of relevant documents for q is a constant over all documents. Assuming the index terms in d are independent, removing the constant, and applying logarithms to transform products into sums, the probability model gives the similarity between a document and a query as follows:

$$\begin{aligned} sim(q,d) &= \log\prod_{t\in q\cap d}\frac{P(t\mid R=1,q)(1-P(t\mid R=0,q))}{P(t\mid R=0,q)(1-P(t\mid R=1,q))} \\ &= \sum_{t\in q\cap d}\log(\frac{P(t\mid R=1,q)(1-P(t\mid R=0,q))}{P(t\mid R=0,q)(1-P(t\mid R=1,q))}) \end{aligned}$$

The probability items in the similarity formula can be estimated by

$$P(t\mid R=1,q)=\frac{r+0.5}{R-r+0.5}$$

and

$$P(t\mid R=0,q)=\frac{df(t)-r+0.5}{N-df(t)-R+r+0.5}$$

where $df(t)$ is the document frequency for index term t, N the number of all the documents, R the number of relevant documents for query q, and r the number of documents in R having index term t.

The initial similarity scheme derived from the probability model only contains document frequency which performs poorly in practice. Robertson and his group made a series of revisions to the similarity scheme by introducing parameters such as term frequency and document length. The revised similarities are applied in the Okapi information retrieval system and have achieved good results [Robertson et al., 1995; Robertson et al., 1999]. One of the successful revisions, the Okapi BM25 similarity scheme, will be applied in this work.

The language model [Ponte and Croft, 1998] also creates a probability to measure the relevance between queries and documents. Different from the probability model, given a query q and a document d, the language model tries to estimate the probability $P(q \mid d)$, i.e., the probability of generating the query from the document. The language model also assumes the terms in a query are independent and transforms $P(q \mid d)$ into:

$$P(q \mid d) = \prod_{t \in q \cap d} P(t \mid d)$$

The probability $P(t \mid d)$ can be estimated by the fraction of term t in document d which comes from the maximum-likelihood estimation (MLE) approach. However, the language model has the disadvantage of assigning zero probabilities to the terms unseen in documents. As a result, smoothing methods are needed to assign a non-zero probability to each term unseen in documents and discount the probabilities for the terms occurring in documents. Zhai and Lafferty [Zhai and Lafferty, 2001a] made a study of three efficient smoothing methods for the language model including *Jelinek-Mercer*, *absolute discount*, and *Dirichlet*. The Dirichlet smoothing method employs Baysian analysis [Casella and Berger, 2001] with the Dirichlet distribution as its prior distribution. A Dirichlet-based language model implementation will be applied in this work.

The axiomatic model derives IR relevance similarities from a set of axioms [Fang, 2007; Fang et al., 2004]. The axioms are formal expressions of the IR heuristics which have been applied in existing IR models. Table 2.1 lists the axiom set.

Table 2.1: Axioms from IR heuristics

Name	Contents
TFC1	Let q be a query and d be a document. If term $t_1 \in q$ and $t_2 \notin q$, then $sim(q, d \cup t_1) > sim(q, d \cup t_2)$
TFC2	Let q be a query and d be a document. If term $t_1 \in q$ and $t_2 \notin q$, then $sim(q, d \cup t_1 \cup t_2) - sim(q, d \cup t_2 \cup t_2) > sim(q, d \cup t_1 \cup t_1) - sim(q, d \cup t_1 \cup t_2)$
TFC3	Let q be a query and d be a document. If term $t_1 \in q$, $t_2 \in q$ and $td(t_1) = td(t_2)$ ($td()$ is a term discrimination function, e.g., IDF), then $sim(q, d \cup t_1 \cup t_2) > sim(q, d \cup t_1 \cup t_1)$
TDC	Let d be a document and $q = < t_1, t_2 >$ be a query. Assume there are two documents d_1 and d_2, where $\|d_1\| = \|d_2\|$. d_1 contains only t_1 and d_2 contains only the same number of t_2. If $td(t_1) > td(t_2)$ ($td()$ is a term discrimination function, e.g., IDF), then $sim(q, d \cup d_1) > sim(q, d \cup d_2)$
LNC1	Let q be a query and d be a document. If for some term $t \notin q$, then $sim(q, d) \geq sim(q, d \cup t)$
LNC2	Let q be a query and d be a document. If $d \cap q = \Phi$ and d' is formed by concatenate q with itself k times, then $sim(q, d') \geq sim(q, d)$
TF-LNC	Let q be a query and d be a document. If for some term $t \in q$, then $sim(q, d \cup t) \geq sim(q, d)$

TFC1, LNC1, and TF-LNC come from the heuristic that if a document has more occurrences for a query term, its relevance similarity should be larger. TFC2 is inspired by the law of diminishing marginal utility in economics [Rittenberg and Tregarthen, 2009] which means the first term occurrence yields more relevance increase than subsequent terms, with a continuing reduction for more terms. TFC3 means a good relevance model should favor documents that contain various query terms than more occurrences for just one query term. TDC indicates that a term with a stronger discrimination capability (e.g., a term with a larger IDF value) should yield more relevance for a document. LNC2 to some extent discloses the essence of long documents. A long document is generated from the mixture of two conditions. One is by incorporating more different topics. The other is by duplicating one topic many times. LNC2 means that if a long document is generated by duplicating one topic many times it should be more relevant than the original document before duplicating.

The axiom model develops a set of relevance formulae from the axioms. One of them will be applied in this work.

2.2.2 Short Text Document Retrieval

The effectiveness of standard IR techniques is weak when they are directly applied to the problem of short text document retrieval. Because most standard IR techniques depend on the common terms occurring in both queries and documents and it is difficult for short text documents to achieve such requirement, the similarity scores for short text documents are very low. As a result, very few of the relevant documents can be retrieved and the recall for short text document retrieval is usually not satisfying. On the other hand, the feature of polysemy that a word may have multiple meanings exacerbates the problem. For instance, there is a query “Apple computer” and two documents “MacBook” and “apple pie”. The standard IR techniques usually develop a lower similarity score between “Apple computer” and “MacBook” than “apple pie”.

Most of the short text retrieval techniques are based on query expansion [Buckley et al., 1994; Mitra et al., 1998] which has been studied for years in the IR community. Thesaurus looking-up is a straightforward and effective method to expand short text with semantically similar or related words. A thesaurus is a reference work that lists words grouped together according to their semantic similarities. Thesauri can be automatically generated or manually created. WordNet [Miller, 1995] is a popular thesaurus which was created by the Cognitive Science Laboratory of Princeton University. It provides abundant resources for query expansion. Voorhees [Voorhees, 1994] presented an automatic query expansion method by adding synonyms and descendents from WordNet.

In addition to manually created thesauri, there are also many techniques for automatically constructing a thesaurus from documents.

Crouch and Yang [Crouch and Yang, 1992] presented an automatic thesaurus generation method from a document corpus. Their approach employs a hierarchical clustering technique [Voorhees, 1985] to create document clusters. Then the low frequency terms in each cluster are selected to form a thesaurus class. Low frequency terms are terms whose document frequency compared to the whole document corpus is less than 1 percent. Such terms have a strong capacity in discriminating between documents.

Qiu and Frei [Qiu and Frei, 1993] represented a term as a feature vector. Each vector dimension is computed by a function of the term and the whole document set. After the feature vector for each term is constructed, the similarities between feature vectors are computed by the cosine value of vector angles. If the vectors for two terms are similar, they will be put into the same thesaurus synonym class.

Schütze and Pedersen [Schütze and Pedersen, 1997] created a term-document matrix. Then they computed word co-occurrences from the matrix. Their approach constructs thesauri through the co-occurrence relationship.

In the past few years, short text based applications such as microblog and image searching have achieved unbelievable success. As a result, many innovative techniques have been invented specifically for short text retrieval. These techniques attempt to employ external resources like commercial Web search engines instead of thesauri.

Sahami and Heilman [Sahami and Heilman, 2006] treated short text documents as a query to Web search engines so that a set of relevant regular length documents can be retrieved. Then the TF-IDF term weighting scheme is applied to the returned documents. For each returned document, only the highest ranked terms are kept and their term vector is normalized. Finally, the centroid of all the normalized vectors for the returned documents is selected as the context vector for the original short text document. In essence, Sahami and Heilman’s method transforms short text documents into their context vectors which are regular in length. Their approach employs context vector inner products to construct a semantic similarity kernel function [Cristianini and Shawe-Taylor, 2000] for short text documents.

Metzler et al. [Meek et al., 2007] also employed commercial search engines to enrich and expand both short text queries and documents. Similarly, queries and documents are fed to a commercial Web search engine. Then the titles and snippets of the top 200 results for each query or document are extracted as their expanded representations. As for similarity, they developed a hybrid model based on exact matching and a language model on the Kullback-Leibler (KL) divergence measure [Lafferty and Zhai, 2001; Zhai and Lafferty, 2001b].

The techniques discussed in this section indicate that adding extra information such as similar words or words from relevant documents are crucial to short text retrieval. This hinted to us that depending only on the action field in intent resolution may not be enough and leveraging other service related information may bring in benefits.

2.3 Summary

This chapter reviews the techniques for service discovery and integration. Most of them fail to take into account end user participation in the problem. On the other hand, Android Intents and Web Intents are two innovative industrial protocols which introduce end user participation and let them determine working services. This thesis which is inspired by the two protocols is a concept-level approach for generic uses in service discovery and integration.

The proposed approach has a step named naïve intent resolution which requires IR techniques. As a result, this chapter also reviews some classic IR models and their implementations. In addition, since the intent action field is a short text field, the techniques for short text retrieval are also discussed in this chapter.

Chapter 3

3 Intents-based Service Discovery and Integration

This chapter will present the proposed Intents-based service discovery and integration approach which aims to address the issues of existing techniques based on the current Web service architecture. The approach is articulated in terms of the Intents architecture, intent data structure, Intents services, and the intent resolution process. In addition, two cases are studied to show the potential benefits and value of Intents.

3.1 Intents Architectures

Figure 3.1 shows the classic Web service architecture. There are three participants in the architecture: service providers, service brokers and service consumers. Service providers create Web services and publish their descriptive information on service functionalities, service qualities, and service addressing methods. Service brokers construct service registries, collect published service descriptive information, and provide querying interfaces to the external world. If a service consumer needs a Web service, he/she sends queries to service brokers to find a desired Web service and obtain its descriptive information. Then the service consumer directly communicates with the service through the addressing methods in its descriptive information.

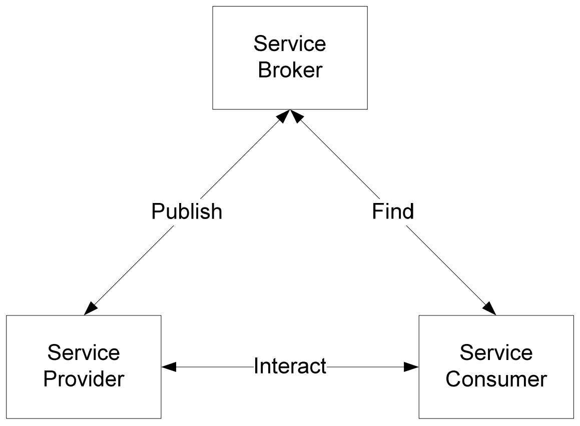


Figure 3.1: Web service architecture

Based on this architecture, if a new product such as a Web application is in development and its developers plan to use external Web services in the application, Figure 3.2 shows the process of searching for a desired service and its interactive activities with the Web application.

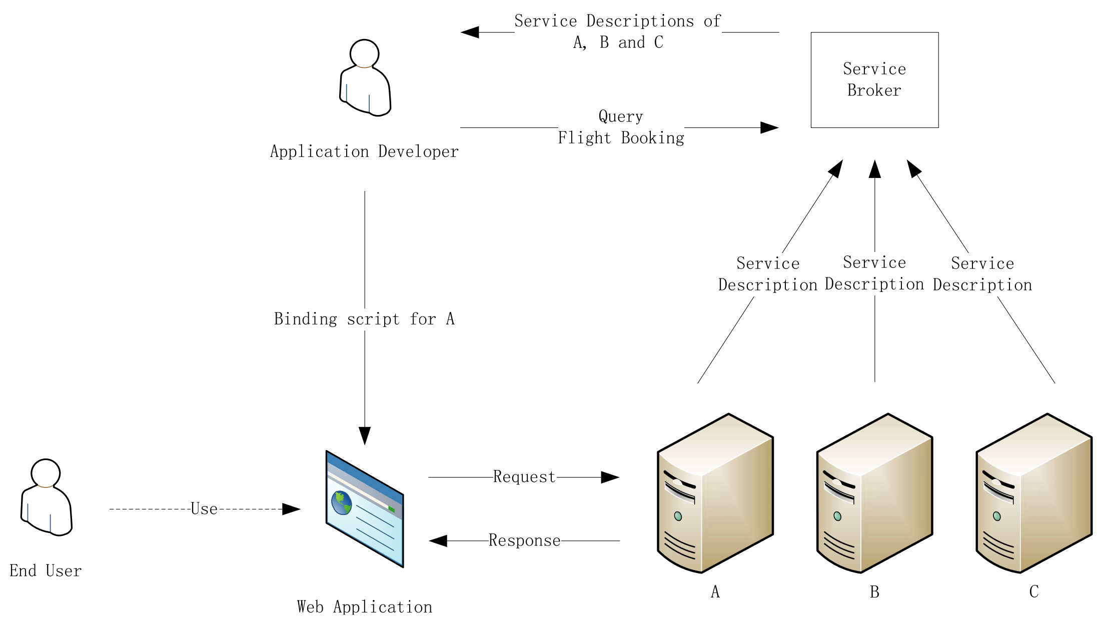


Figure 3.2: The process of service discovery and integration in the current Web service architecture

Figure 3.2 takes the flight booking scenario for instance and assumes a set of flight booking services separately running on servers supported by different service providers (e.g., Expedia[1], Google Flight[2] and Priceline[3]) which are denoted by A, B and C. Their service description documents are registered with a service broker. If an application needs to use one of the flight booking services, its application developers first search the broker by sending a query "Flight booking". Then the broker returns a list of relevant services including A, B and C. If the developers decide to choose A as the desired service,

[1] http://www.expedia.ca/

[2] https://www.google.ca/flights/

[3] http://www.priceline.com/

they will embed a piece of scripting code for communicating with A into the application. When the application is released and an end user tries to use it for a flight booking task, the application will communicate directly with A and complete the task.

In the above process, application developers and end users both belong to the category of service consumers in the Web service architecture. Application developers create service-dependent applications for end users. However, end users play almost no role in the process of service discovery and integration. This scheme may cause serious problems in the following two situations:

- **Service A runs ineffectively or is totally blocked in end user networks.** It is possible that A performs the best when application developers test it in their own networks in terms of reliability, responsiveness and other service quality attributes. However, as a result of the heterogeneity of computer networks, end user environments may be totally different and it is highly possible that A becomes inferior to other services. Even worse, if A is blocked by the gateway of an end user network, the corresponding functionalities supported by A will also be out of work.
- **End users have a preference for other services rather than A.** For instance, some end users may have a B membership card which offers a discount. However, it is impossible for application developers to have such information for all prospective end users. As a result, the end users who have a B membership card are forced to use A by the application. Thus they may choose other applications which are dependent on B and as a result the application depending on A is devalued.

In order to address these issues, Figure 3.3 provides a modified design for service discovery and integration.

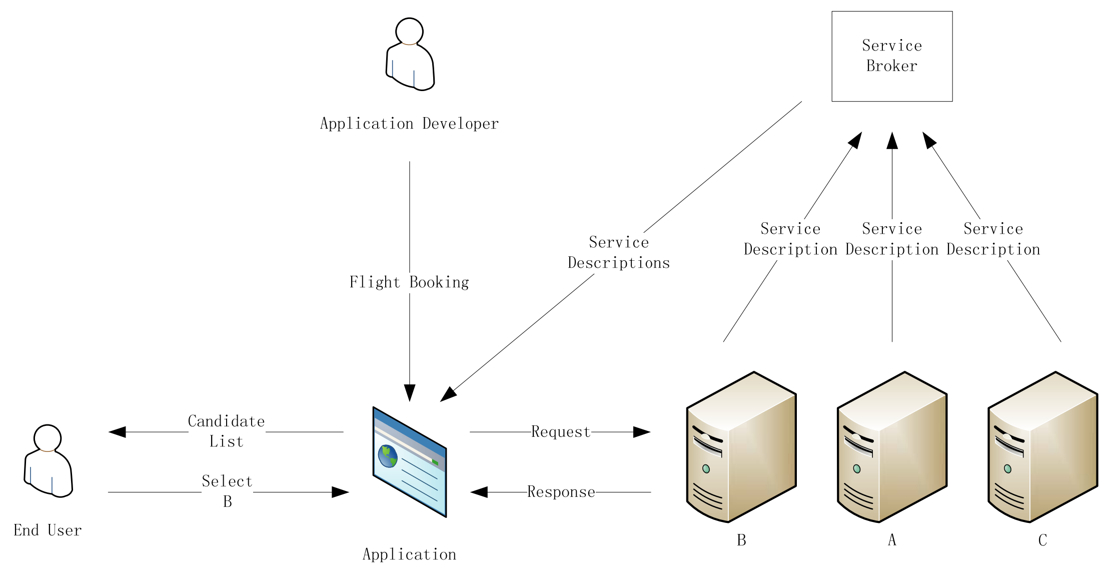


Figure 3.3: Modified design for service discovery and integration

In Figure 3.3, if application developers want to take advantage of external Web services, they can just specify a semantic service goal (e.g., “Flight Booking”) in their application instead of statically binding it to a specific service. While an end user is using the application, it retrieves the service description from the service brokers and generates a list of candidate services to the end user as per the semantic goal specified by the developers. Then the end user takes the responsibility for selecting a working service such as B in Figure 3.3. After that, the application communicates with the selected service and completes the end user’s task. This scheme is capable of addressing the aforementioned issues as follows:

- End users can dynamically choose the services which are valid and that perform well in terms of reliability, responsiveness or other service quality attributes in their own computer networks.
- End users are capable of choosing any services for which they have a preference. For instance, if one user has a membership card for service A and another user has a card for service B, they can apply their own favorite service separately while using the same application.

Even though the design in Figure 3.3 is an improvement compared to Figure 3.2, it is non-trivial and tedious for application developers to implement the functionality of generating candidate service lists and communicating with selected services in every application. Thus each end user should have a user agent which takes the above responsibilities for applications.

On the other hand, service providers should be allowed to publish their service descriptive information directly to the Web. For instance, service providers may create hypertext references to their Web services descriptive information files and put them into relevant Web pages. At the same time, both service brokers and service consumers are capable of acquiring these files directly from the Web.

3.1.1 Implicit Mode

Based on the above discussion, and motivated by the two industrial protocols of Android Intents and Web Intents, an Intents architecture in compliance with the idea in Figure 3.3 is presented in Figure 3.4.

In Figure 3.4, if an application is dependent on some external services, its developers can specify their service goal by a construct named *intent* and embed it into the application. Once an end user executes the application and triggers the intent, a message with the intent is created as a service request. On the other hand, services in the architecture are *Intents services* which can accept and process intents. When an Intents service is published, part of its service description is wrapped into an *Intents advertisement* which may be registered with service brokers or put directly on the Web. At the same time, service brokers are able to capture Intents advertisements by searching the Web. The process is analogous to that when Web spiders capture Web pages from the Web. Each end user has a user agent. It contains a private service registry which collects Intents advertisements from both service brokers and the Web under the control of its owner. End users may choose to add the Intents advertisements of interest, or remove the Intents advertisements which are not needed. Intents advertisement management in user agents looks like software management on PCs or mobile devices. The above mentioned intent message is first sent to the end user’s user agent. Then the user agent may generate a list of candidates according to both the intent content and its private service registry. The candidate services are returned to the end user. After the end user makes a choice, the user agent forwards the intent to the selected service. If the service produces some results, the user agent receives them and sends them back to the application.

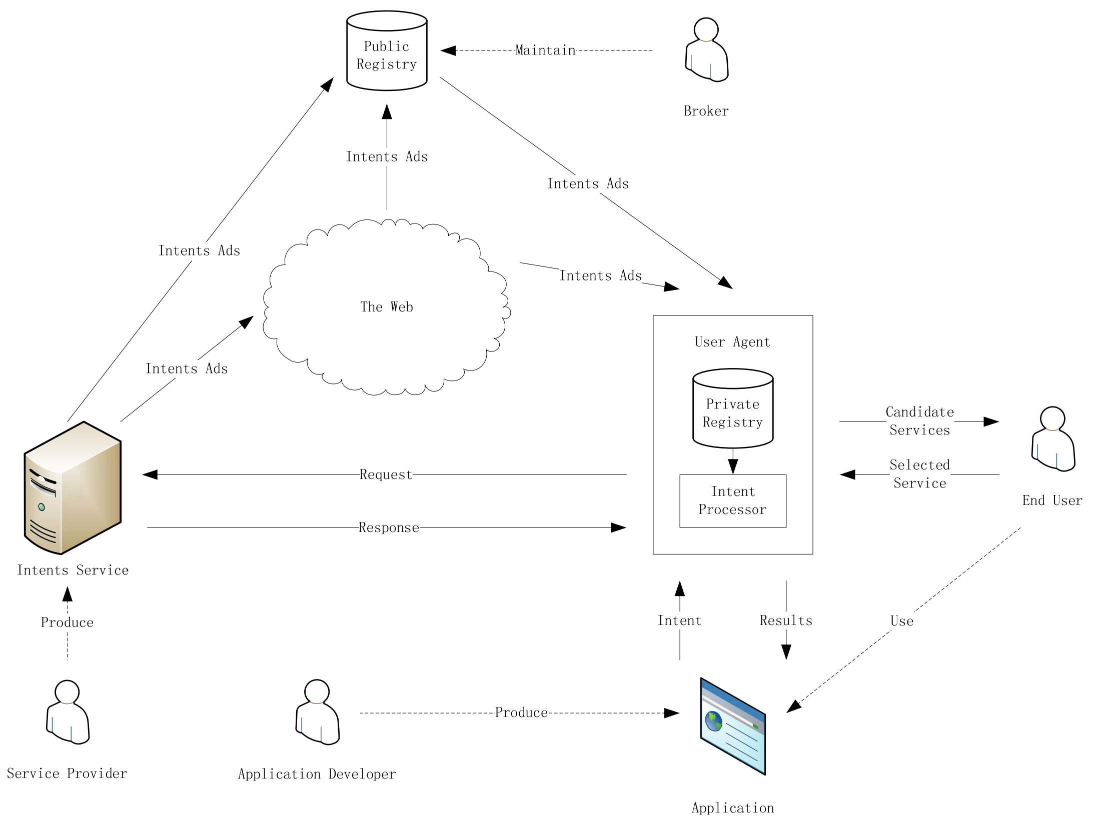


Figure 3.4: Implicit mode of the Intents architecture

The architecture shown in Figure 3.4 is the *implicit mode* of the Intents architecture. The intents in this mode only represent a semantic service goal instead of a specific service. In addition, end user participation and interaction is required in this mode for service discovery and integration. The ultimate working services are determined dynamically at run time.

3.1.2 Explicit Mode

Sometimes, application developers know which specific services should be used in their applications. They may not need end user participation and this is the currently used paradigm for service discovery and integration as in Figure 3.2. Thus in addition to the implicit mode, an *explicit mode* of the Intents architecture is also designed in Figure 3.5 to be compatible with the current paradigm in Figure 3.2. The explicit mode of the Intents architecture permits directly binding between applications and specific services without end user interference.

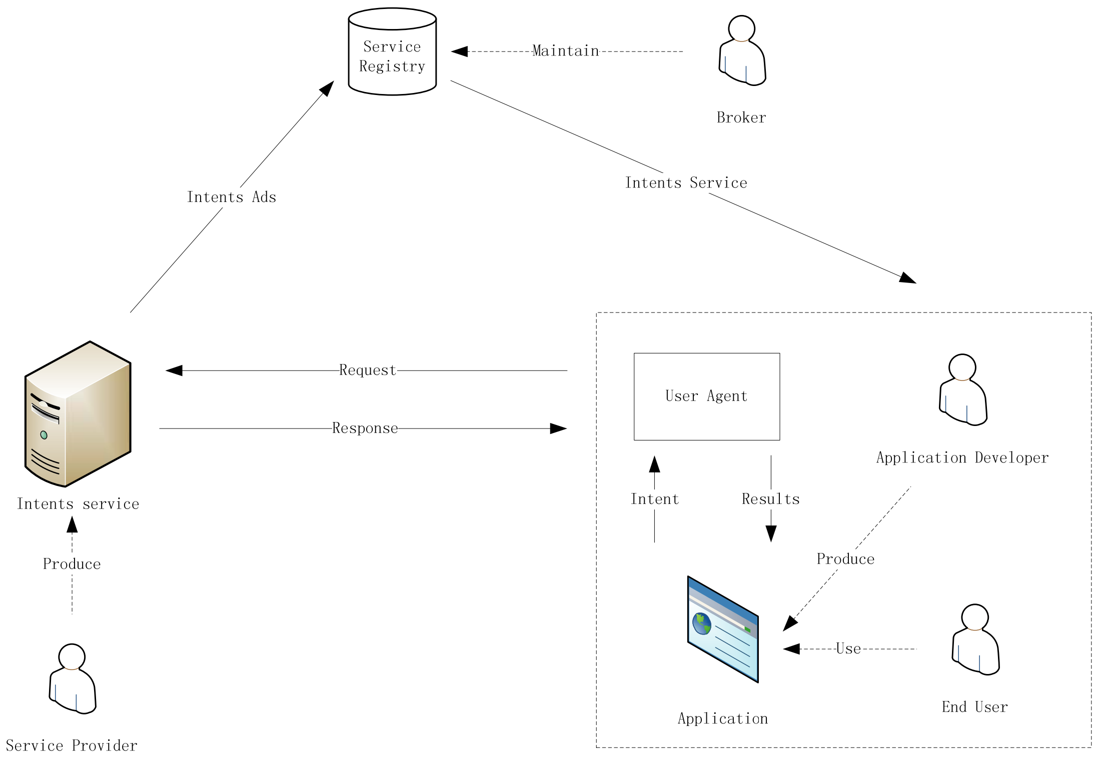


Figure 3.5: Explicit mode of the Intents architecture

In the explicit mode, the developers of an application search service brokers for a desired service. After they find the service, an intent is specified in the application as a binding between the application and the service. The intent has the necessary addressing information which can be used to locate the service. Once an end user executes the application and triggers the intent, a message enclosing the intent is created and sent to the end user’s user agent. The user agent resolves the intent and extracts its service addressing information. Then the intent is directly transferred to the specified Intents service. If the service produces some results, the user agent receives them and sends them back to the application.

The implicit and explicit modes of the Intents architecture are characterized by two different paradigms of service discovery and integration whose main difference is end user participation and interaction. The explicit mode keeps the current widely applied paradigm in which only application developers make a decision on the ultimate working services. Its existence shows that Intents is compatible with current techniques. However, only applying the explicit mode is not enough to meet all requirements. Thus the implicit mode is created to bring in end user participation and interaction in service discovery and integration which complements the defects in the explicit mode.

The intents applied in the two modes are explicit intents and implicit intents, respectively. Their content is different. The next section will present them in detail.

3.2 Intent Data Structure

The aforementioned intents are a data structure which represents a service need or goal. An intent contains an intended operation and the data prepared for the operation. Application developers specify intents in their applications. When an end user executes an application task which triggers an intent, a message with the intent will be created and sent to the end user’s user agent. Then the user agent resolves the intent and assists the end user in seeking an appropriate service to complete the task.

Formally an intent is a tuple $I = (t, a, dt, dv)$, where t stands for intent type, a intent action, dt intent data type, and dv intent data value. We design three types of intents: *explicit intents*, *authoritative intents* and *naïve intents*. Explicit intents are designed for the explicit mode of the Intents architecture where end users are not involved in service discovery and selection. Authoritative intents and naïve intents are designed for the implicit mode of the Intents architecture where end users participate in the process of determining ultimate working services. Therefore, authoritative intents and naïve intents are together in the implicit intent category. Figure 3.6 illustrates the hierarchy of intent classification.

The action field indicates an intended operation. For implicit intents, the action field is a text string which semantically represents the intended operation name. On the other hand, if it is an explicit intent, the action field is the identifier of a desired Intents service which can be used to identify and locate the service directly.

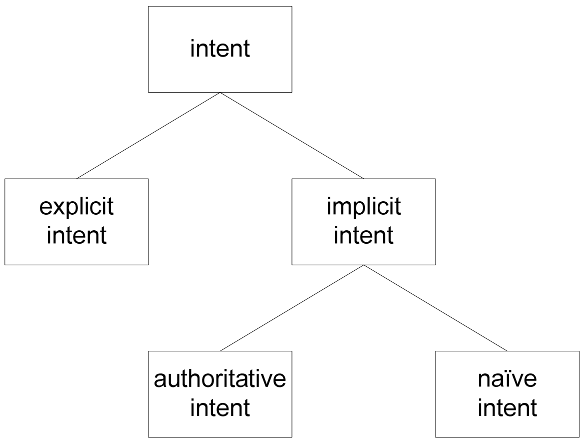


Figure 3.6: The hierarchy of intent classification

The data type field and data value field are input parameter types and values, respectively. They are prepared for the intended operation. The data type field adopts the internet media type [Bray, 2002] which is a two-level format composed of a type and a subtype, and the data value field obeys the format specified by the selected internet media type.

In the following subsections, authoritative intents, naïve intents, explicit intents, and a comparison among them will be presented.

3.2.1 Authoritative Intent

Authoritative intents are implicit intents whose action, data type, and the format of data value comply with a specification designed by an authoritative organization. On the other hand, service providers who choose to support the authoritative intent specification should implement their services to enable them to accept and process any intent in conformity with the specification. If the authoritative specification is thought of as an interface in the object-oriented programming language, the Intents services which support the specification are the implemented classes of the interface. Then any authoritative intent of the specification is the statement to invoke one of the class functions.

For instance, W3C as an authoritative organization may publish an authoritative intent specification for one-way flight searching. It specifies the intent fields as listed in Table 3.1.

Table 3.1: Authoritative intent specification sample

Intent Field	Remark	
Intent Type	authoritative	
Action	www.w3.org/intent/one-way_flight_search	
Data Type	application/json	
Data Value Format	**Requested parameters**	
	departure_location	The location where customers depart
	arrival_location	The location where customers arrive
	departure_date	The date when customers depart

An application developer who wants to use services complying with the specification may set up an authoritative intent in his/her applications as listed in Table 3.2.

Table 3.2: Authoritative intent sample

Intent Field	Value
Intent Type	authoritative
Action	www.w3.org/intent/one-way_flight_search
Data Type	application/json
Data Value	{ “departure_location” : “London, London Int'l, Ontario (YXU)”, “arrival_location” : “Toronto, Pearson Int'l, Ontario (YYZ)”, “departure_date” : “2013-11-25” }

Authoritative intents of the same specification should share the same action which is unique compared to authoritative intents from another specification. Thus it is suggested that the domain name of the authoritative organization which creates the specification be added into the action field.

In addition to intent fields, specifications for authoritative intents may also contain service outcome formats. Services supporting a specification should produce results according to the outcome formats of the specification so that the applications depending on the services are able to leverage their results.

Authoritative intent specifications may be included in technical online documents to which both service providers and application developers have access. On the service provider side, services are implemented to accept any valid authoritative intent and produce outcomes in conformity with an authoritative intent specification. On the other hand, application developers make their applications trigger intents according to the same specification. Thus to some extent, authoritative intent specifications are a well-designed contract made by third-party organizations for service providers and application developers. When end users use an application which creates authoritative intents according to an authoritative intent specification, any service conforming to the specification will have a chance to be selected as the ultimate working service.

The concept of authoritative intents has been currently applied in Web Intents and Android Intents. Web Intents defined a suite of public intents as in Table 3.3 which can be thought of as a kind of authoritative intent. More details about their specifications can be found on Web Intents' official website[1].

Android Intents also defined a set of standard activity and broadcast actions in the class "`android.content.Intent`"[2] which also reflect the idea of authoritative intents in this work. When android application developers create an object of the class with one of the standard actions, they actually construct an authoritative intent.

[1] http://webintents.org/

[2] http://developer.android.com/reference/android/content/Intent.html

Table 3.3: Web Intents public intents

Intent Name	Action	Description
Share	http://webintents.org/share	The share intent is designed to give applications the ability to offer a simple mechanism for sharing data from the current page.
Edit	http://webintents.org/edit	The edit intent is designed to give applications the ability to offer a simple mechanism to edit data from the current page.
View	http://webintents.org/view	The view intent is designed to give applications the ability to offer a simple mechanism to view data in their application.
Pick	http://webintents.org/pick	The pick intent is designed to give services the ability to allow their users to pick files from their service for use in a client application.
Subscribe	http://webintents.org/subscribe	The subscribe intent is designed to give applications the ability to offer a simple mechanism for subscribing to data from the current page.
Save	http://webintents.org/save	The save intent is designed to give applications the ability to offer a simple mechanism to save data in their application.

Authoritative intents are not perfect. They may have drawbacks in the following three situations:

- **There is no specification of authoritative intents defined in the domain of application developers and service providers.** It takes time for authoritative organizations to propose, draft, refine, and finalize an authoritative intent specification. As a result, before a satisfied authoritative intent specification is created, the contract between service providers and application developers cannot be constructed. On the one hand, service providers could not make their developed services follow any authoritative intent specification. On the other hand, application developers are not able to make their applications create corresponding authoritative intents. For instance, an application developer wants his/her application to trigger an image editing intent to seek a desired service, but there is no related authoritative intent specification available. Therefore, the application developer cannot create the required authoritative intent for his/her service goal.

- **Application developers or service providers cannot strictly comply with authoritative intent specifications.** Even though there exist authoritative intent specifications in the domain of application developers and service providers, they may not be satisfied by any of them. Since authoritative intent specifications must be fully obeyed in intent creation and service implementation, any additional modifications by application developers or service developers are not permitted. Take the aforementioned flight searching intent, for instance. If the service provisioned by a service provider only accepts XML-based input for technical reasons, it cannot support JSON-based authoritative intents. On the other hand, if an application developer has his/her application to generate intents with airlines input in addition to the data value format specified in the authoritative intent specification, its end users may be returned no candidate services as a result of input mismatching.
- **Authoritative intents require the exact matchmaking scheme in all intent fields including the action field in intent resolution.** When user agents receive messages with an enclosed intent from any application, it needs to analyze the intent to determine its intent type, and extract its field content for further processing like generating candidate services for implicit intents. Such a process is called *intent resolution*. Once service providers implemented an Intents service, its Intents advertisement should be created with information including supported action and data type. In the process of resolving an intent, user agents search for the services whose supported action and data type match the corresponding fields in the intent. For an authoritative intent, its action should be exactly matched with the supported action of a service so that the service can be selected as a candidate. However, the exact matchmaking scheme for the action field will cause issues in certain situations. For instance, organization A and B separately design and publish an authoritative intent specification for file storage. All the fields of the two intent specifications are compatible except for the action field. Organization A uses “A/file_upload” while B adopts “B/file_upload”. The two actions mean

semantically the same. However, if an application developer chooses to support A’s intents, his/her end users will miss the services supporting B and vice versa.

In order to address the above problems for authoritative intents, naïve intents are designed in the next subsection.

3.2.2 Naïve Intent

Compared with the strictness of authoritative intents, naïve intents are designed as a loose or relaxed contract between service providers and application developers. For application developers, they may have applications to trigger an intent without an authoritative specification. On the other hand, service providers are allowed to claim non-authoritative intent support in their service Intents advertisements. It is the responsibility of user agents to judge if services and naïve intents are matched according to its built-in similarity models in intent resolution. Taking the image editing case for instance, if application developers fail to find a satisfied authoritative intent specification, their application may generate a naïve intent as in Table 3.4.

Table 3.4: Naïve intent example

Intent Field	Value
Type	naive
Action	image edit
Data Type	image/jpeg
Data Value	the image file

At the same time, service providers who cannot find a satisfied authoritative specification may create their services and mark them with actions like “picture editor” or “photo editing” in the Intents advertisement. Then user agents may adopt a similarity model so that relevant services can be put onto candidate lists. Naïve intents are characterized by its extension to non-exact matching schemes compared with authoritative intents in intent resolution.

Web Intents and Android Intents can also create naïve-similar intents. Applications and intents in the two protocols can choose any action content without having to comply with public contracts. However, just like their authoritative-similar intents, only the exact matching scheme which is the simplest similarity model is applied in resolving these intents.

3.2.3 Explicit Intent

When application developers design a product and want to use some external services, they may have found the desired specific service and refuse end user participation in service discovery and integration like the explicit mode of the Intents architecture. Thus explicit intents are employed to allow application developers to create direct communication between their products and remote services. The major difference between implicit intents and explicit intents is that the action field of an explicit intent is able to directly locate the desired remote service. As a result, it is suggested that a URL be used in their action fields. Table 3.5 is an explicit intent example for image editing services.

Table 3.5: Explicit intent example

Intent Field	Value
Type	explicit
Action	http://202.117.0.119/intents_services/image_edit
Data Type	image/jpeg
Data Value	the image file

Explicit intents are designed to make Intents compatible with the currently used service discovery and integration paradigm, i.e. application developers take the full load in service seeking. Application developers who need the use of direct service binding may choose explicit intents in their products. When a user agent receives an explicit intent, it directly transfers the explicit intent to the service specified in the intent. If the service produces some results, the user agent receives them and sends them back to the application triggering the intent.

3.2.4 Comparison and Discussion on Intent Types

Authoritative intents, naïve intents and explicit intents are designed to complement each other in service discovery and integration. Explicit intents keep the traditional characteristics of service discovery and integration in the explicit mode of the Intents architecture. Authoritative intents compensate for explicit intents by introducing end user participation. However, authoritative intents adopt a strict action matching scheme in intent resolution which may cause problems in some cases. Thus naïve intents are designed as a relaxed version of authoritative intents. The three intent types collaboratively meet various requirements of service providers, application developers, and end users.

Table 3.6 lists a comparison of the three intent types in terms of third-party and end user participation, action matching scheme, positive false error in candidate services, flexibility, and reliability.

Table 3.6: Comparison of explicit, authoritative, and naïve intents

	Explicit	**Authoritative**	**Naïve**
Third Party Participation	No	Yes	No
End User Participation	No	Yes	Yes
Action Matching	Exact Matching	Exact Matching	Exact and Non-exact Matching
False Positive Error	No	No	Yes
Flexibility	Weak	Medium	Strong
Reliability	Strong	Strong	Weak

Authoritative intents require third-parity participation to make up the specifications including action, data type, data value format, service outcome format, and other details on technical information. On the contrary, explicit and naïve intents do not have that requirement. Neither of them needs a predefined specification to obey when service providers create a service or application developers specify an intent for their application.

Authoritative and naïve intents work for the implicit mode of the Intents architecture where user agents generate candidate services to end users for service selection. Thus the two intent types require end user participation. However, when user agents receive an explicit intent, they will directly transfer the intent to the remote service, ruling out any end user participation.

In intent resolution, action matching is a critical step. Since the action field of explicit intents can locate the required service, exact action matching applies to explicit intents. Authoritative intents also adopt the exact action matching scheme which is more accurate in generating candidate service lists. The exact action matching scheme applied in explicit intents and authoritative intents guarantees no false positive error in candidate services. On the contrary, naïve intents may relax the exact action matching scheme and introduce non-exact action matching schemes. This may put any services whose functionality is relevant onto candidate lists without their action fields having to be the same as the resolved intent. As a result of the applied similarity model, naïve intents may cause false positive errors in service discovery and integration. Thus a tradeoff between coverage and accuracy should be taken into careful consideration when a non-exact action matching scheme is being designed for naïve intents.

Explicit intents permit no end user participation in service discovery and integration, therefore they have weak flexibility. In contrast, though authoritative intents introduce end user participation, they may rule out some qualified services as a result of the exact action matching scheme. Naïve intents are most flexible among the three intent types. They permit end user participation in service selection, do not involve authoritative organization, and bring in non-exact action matching in the generation of candidate services. However, resolving a naïve intent depends too much on user agent built-in similarity models and they may result in false positive errors, hence reliability is weak in naïve intents compared to the other two intent types.

3.3 Intents Services

An Intents service is a Web service which is able to accept and process intents as service input. Each Intents service is identified by a unique address to which user agents send their received intents. After receiving an intent, Intents services extract data from the intent, process the data, and return results.

If Intents services are constructed in reference to an authoritative intent specification, they should be able to accept and process any authoritative intent in compliance with the specification. In addition, they can also accept any naïve intent whose data fields (data types and values) coincidently comply with the authoritative intent specification. On the other hand, if service providers cannot find a satisfied authoritative intent specification, they may also create Intents services based on their service designs and mark them with appropriate Intents advertisements. These Intents advertisements should semantically reflect corresponding service functionalities (operation names and parameters).

An Intents service should be marked with only one Intents advertisement as part of its service description. Intents advertisements are compact and can be embedded into Web pages in the form of tags as well as being registered with service brokers. User agents collect Intents advertisements under the control of end users and use them in the process of implicit intent resolution.

Intents services can be created from current mainstream Web services. This section will present the Intents advertisement concept which is closely related to Intents services and the method of creating Intents services from two mainstream Web services: SOAP and REST Web services.

3.3.1 Intents Advertisement

An Intents advertisement is a tuple $A = (id, a, dt)$ where id is its Intents service identifier, a intent action and dt data type. More details on the three fields are presented as follows:

- **Identifier.** An Intents service identifier is a text string. It is used to uniquely identify and locate the Intents service. With the identifier, Intents user agents are able to forward received intents to the specified Intents service. For instance, a URL (Universal Resource Locator) can be used as a service identifier.
- **Action.** This field corresponds to the action field in intents. The action field in the Intents advertisement of an Intents service specifies the operation supported by the Intents service. If the Intents service wants to be advertised for the authoritative intents of an authoritative intent specification and implemented according to the specification, its action field should be the same as that in the specification. Otherwise, Intents user agents will not be able to discover the service for the authoritative intents of the authoritative intent specification. On the other hand, if an Intents service is not developed for any authoritative intent specification, its Intents advertisements should be defined by its developers to semantically reflect the functionalities of the service.
- **Data type:** This field corresponds to the data type field in intents. The data type field in the Intents advertisement of an Intents service specifies the input parameter data types accepted by the Intents service. Similarly, if an Intents

service is implemented in accordance with an authoritative intent specification, the data type field of its Intents advertisements should comply with the specification. Otherwise, Intents user agents will not be able to discover the service for the authoritative intents of the specification. On the other hand, if an Intents service is not developed for any authoritative intent specification, its Intents advertisements may be defined by its developers according to the data types accepted by the service. The data type field also adopts the Internet media type. However, since an Intents service may support multiple data types, the data type field in Intents advertisements is slightly different from that in intents. For instance, an image editing service may only accept one image file, but the image can be in the form of a gif, jpeg or png. In this situation, the data type field should be “image/gif, image/jpeg, image/jpeg”. The Internet media types are separated by commas. If the image editing service supports all image formats, it can also use a global or generic type with the wildcard character (*), e.g., “image/*”.

Intents advertisements can be thought of as a kind of service description. They specify where an Intents service is and what action and data type it supports. It is compact and machine readable which user agents are able to leverage in intent resolution. Intents advertisements are created by service providers and can be published in the following two ways:

- **To the Web.** An Intents advertisement can be created in the form of HTML tags which can be embedded into any online Web page. Intents advertisement tags are not rendered with Web page content but they can be detected by Intents user agents. Besides, it is suggested that the enclosing Web pages be relevant to their advertised Intents services with introductory information. As a result, Web page content may be utilized in intent resolution. Figure 3.7 is an Intents advertisement sample for link sharing services.

- **To service brokers.** Intents advertisements can also be registered with service brokers similar to registering WSDL files with UDDI registries. Service brokers should provide user interfaces for service providers to register the Intents advertisements of their Intents services. Meanwhile, service brokers can also capture and collect Intents advertisements from their enclosing Web pages.

```
<intent
  id=https://twitter.com/intent/tweet
  action=”share to twitter”
  type=”text/uri-list” />
```

Figure 3.7: An Intents advertisement example for link sharing services

Figure 3.8 shows the two ways for publishing Intents advertisements and the relationship between the Web, service brokers, user agents, and service providers.

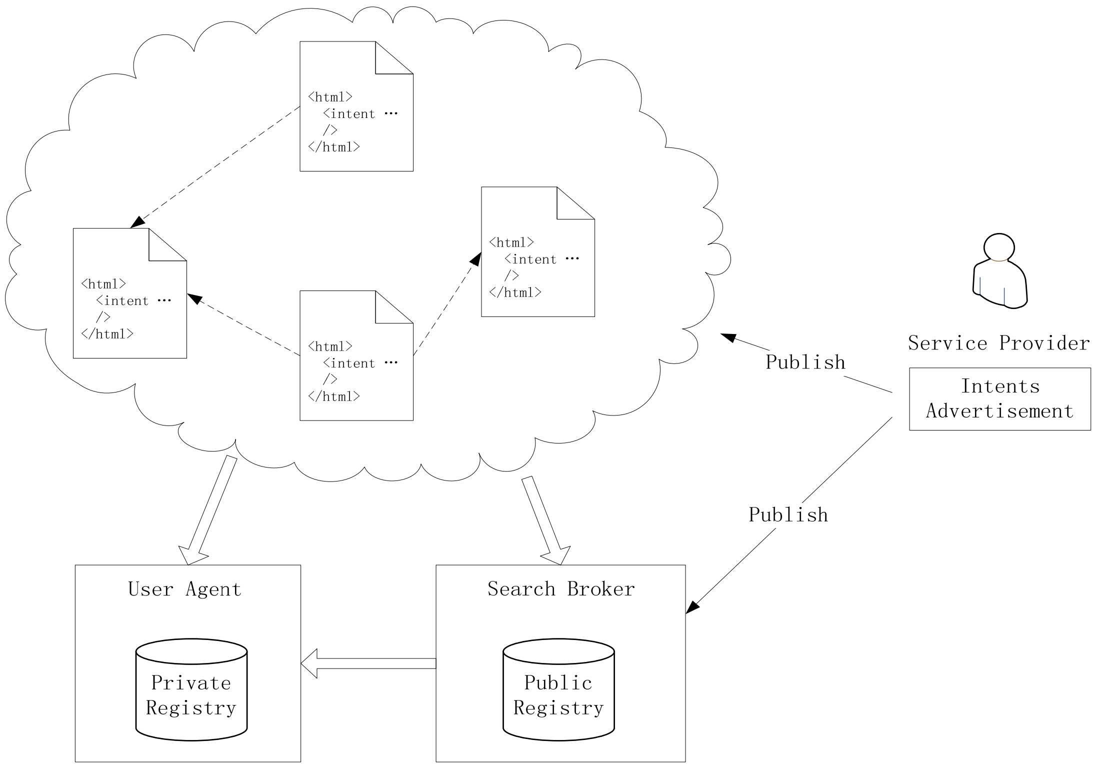


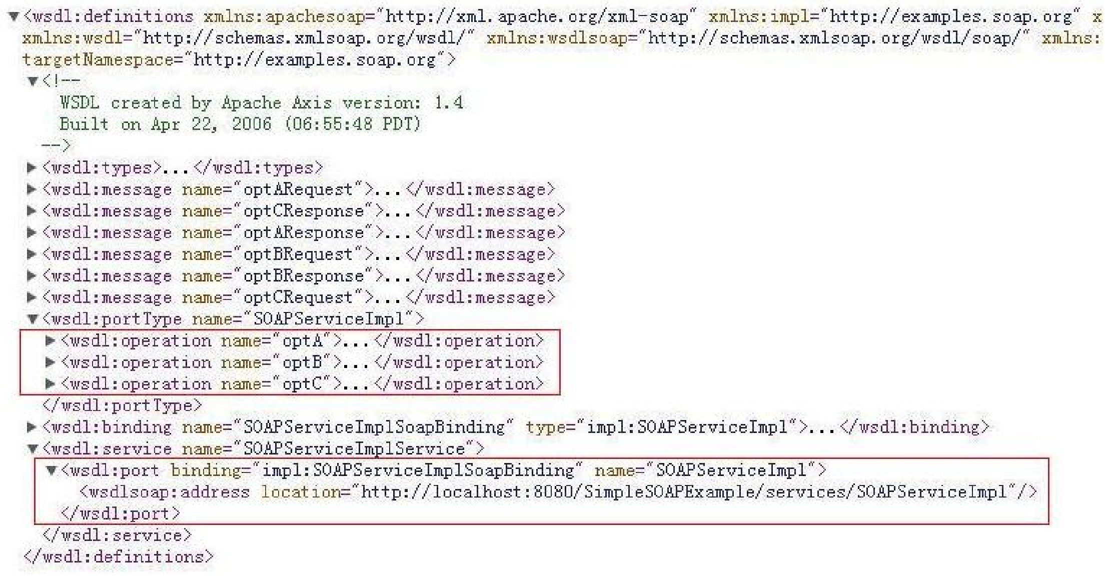
Figure 3.8: Intents advertisement publishing ways

As shown in Figure 3.8, user agents and service brokers maintain separate registries for Intents advertisements. Service providers publish their Intents advertisements to the Web as well as to service brokers. Intents advertisements on the Web also connect with each other through the hypertext references in their enclosing Web pages, which are illustrated by dashed arrows in the figure. Service brokers are also able to capture and collect Intents advertisements from the Web. For a user agent, it can collect Intents advertisements from the Web or service brokers under the control of its owner.

3.3.2 Intents Services from SOAP Web Services

SOAP Web services are one category of mainstream Web services which are constructed on public protocols including SOAP, WSDL, UDDI, and XML. Each SOAP Web service is usually created and published along with a WSDL document which is composed of the elements for invoking the SOAP Web service (e.g., parameters, operation names, and service address). On the other hand, service consumers can use toolkits for SOAP Web services to generate a client stub from WSDL documents. The stub takes the responsibility for converting application objects into SOAP messages. Then the communication between service consumers and service providers is completed through the SOAP protocol. If results are returned, the stub converts the results back to application readable objects.

A SOAP Web service is capable of accommodating multiple operations while at the same time only one end point is exposed to the external world for communicating with service consumers. Figure 3.9 demonstrates a sample WSDL document for SOAP Web services. The document represents a SOAP Web service with three operations but only one end point.



```
▼<wsdl:definitions xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://examples.soap.org" x
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:
 targetNamespace="http://examples.soap.org">
  ▼<!--
     WSDL created by Apache Axis version: 1.4
     Built on Apr 22, 2006 (06:55:48 PDT)
   -->
  ▶<wsdl:types>...</wsdl:types>
  ▶<wsdl:message name="optARequest">...</wsdl:message>
  ▶<wsdl:message name="optCResponse">...</wsdl:message>
  ▶<wsdl:message name="optAResponse">...</wsdl:message>
  ▶<wsdl:message name="optBRequest">...</wsdl:message>
  ▶<wsdl:message name="optBResponse">...</wsdl:message>
  ▶<wsdl:message name="optCRequest">...</wsdl:message>
  ▼<wsdl:portType name="SOAPServiceImpl">
    ▶<wsdl:operation name="optA">...</wsdl:operation>
    ▶<wsdl:operation name="optB">...</wsdl:operation>
    ▶<wsdl:operation name="optC">...</wsdl:operation>
   </wsdl:portType>
  ▶<wsdl:binding name="SOAPServiceImplSoapBinding" type="impl:SOAPServiceImpl">...</wsdl:binding>
  ▼<wsdl:service name="SOAPServiceImplService">
    ▼<wsdl:port binding="impl:SOAPServiceImplSoapBinding" name="SOAPServiceImpl">
       <wsdlsoap:address location="http://localhost:8080/SimpleSOAPExample/services/SOAPServiceImpl"/>
     </wsdl:port>
   </wsdl:service>
 </wsdl:definitions>
```

Figure 3.9: The WSDL document of a SOAP Web service sample

When creating Intents services from the SOAP Web service, each operation should be mapped to a separate Intents service as shown in Figure 3.10.

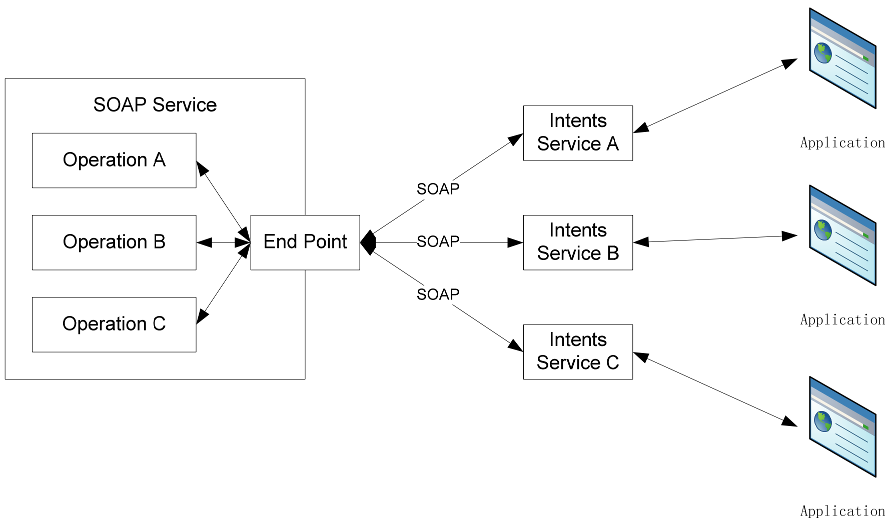


Figure 3.10: Creating Intents services from SOAP Web services

Figure 3.10 presents the SOAP Web service with three operations and one end point. Three separate Intents services are created for three operations. If an Intents service receives an Intents-based message, it will be converted into a SOAP message and transferred to the SOAP service end point by the Intents service. The end point dispatches the SOAP message to the corresponding operation module. When the operation is completed, results are returned to the Intents service in SOAP formats. Then the results are converted to messages in accordance with the specification of the Intents service and sent back to the invoking application.

3.3.3 Intents Services from REST Web Services

REST Web services are another type of mainstream Web service. Richardson [Richardson and Ruby, 2007] divided REST Web services into RESTful and REST-RPC Web services. RESTful Web services comply strictly with the principles and constraints specified in Fielding’s articles [Fielding, 2000; Fielding and Taylor, 2002]. A RESTful Web service only exploits the methods (GET, POST, PUT, DELETE, etc.) specified in Hypertext Transfer Protocol (HTTP). In other words, the communications between a RESTFul Web service and its clients are built directly on top of raw HTTP methods. REST-RPC Web services also employ HTTP messages as communication envelopes but may create new methods instead of only using those given by HTTP. These new methods and their parameters are often embedded into service URLs or other fields of HTTP request messages. For instance, a people profile querying service at the path “/rest_services/get_profile” of host “www.sample.com” can be designed into RESTful or REST-RPC style, respectively, as in Table 3.7.

Table 3.7: RESTful and REST-RPC samples

	HTTP Request
RESTful	POST /rest_services/get_profile HTTP 1.1 Host: www.sample.com name=value
REST-RPC	GET /rest_services/get_profile?name=value HTTP 1.1 Host: www.sample.com

Either RESTful or REST-RPC Web services are each identified with a URL which is also exploited to address the service. This characteristic is similar to Intents services. Therefore, while wrapping a REST Web service, only one Intents service needs to be created.

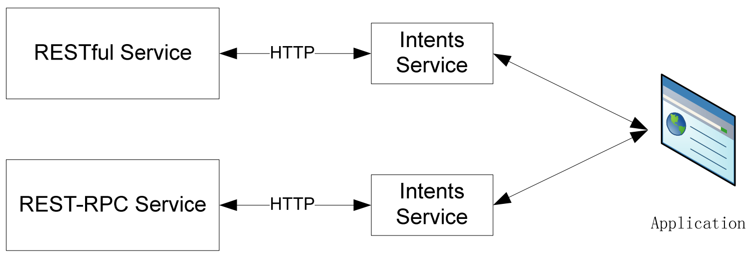


Figure 3.11: Creating Intents services from RESTful and REST-RPC Web services

Figure 3.11 shows how to wrap RESTful and REST-RPC Web services to create Intents services. Once an Intents service receives an Intents-based message, the Intents service converts it into a HTTP request message and transfers it to the corresponding REST Web service. If the REST Web service produces some results, the Intents service transforms it back to the format adhering to the specification of the Intents service and sends it back to the invoking application.

3.4 Intent Resolution

When a user agent receives an intent, it will need to analyze the intent to determine its intent type, and extract its fields for further processing. If it is an explicit intent, the user agent directly transfers the intent to the corresponding service specified by its action field. On the other hand, if it is an implicit intent, the user agent needs to generate a candidate service list based on an Intents advertisement registry and the content of the intent. The whole process described above is called *intent resolution* which will be presented in this section.

Table 3.8: Field comparison between intents and Intents advertisements

Field	**Intent**	**Intents Advertisement**
Service ID	Explicit Intent (Yes) Implicit Intent (No)	Yes
Action	Yes	Yes
Data type	Yes	Yes
Data Value	Yes	No
Intent Type	Yes	No

Table 3.8 lists all the fields of intents and Intents advertisements. Explicit intents are directly transferred to remote services, thus user agents do not have too much work for

them in intent resolution. As for implicit intents, it can be found that only the action field and the data type field appear in both intents and Intents advertisements. Thus implicit intent resolution should be discussed in terms of the two fields.

- **Date type:** The data type field describes the data type for service input. Both of the fields in intents and Intents advertisements adopt the Internet media type. The only difference is that the data type field in Intents advertisements supports multiple and generic types. Thus the intent data type should be exactly matched with one of the non-generic types, or its first level type should be the same as one of the generic types. Table 3.9 lists some instances for data type matching between Intents advertisements and intents.

Table 3.9: Data type matching instances

Intents advertisement data type	Matched intent data type	Unmatched intent data type
"image/jpeg, image/gif"	"image/jpeg" "image/gif"	"image/png"
"image/*"	"image/jpeg" "image/gif" "image/png"	"text/plain"

- **Action:** The action field of an authoritative intent is usually well designed by authoritative organizations in the specification for the intent. Therefore, for authoritative intents, the exact matching scheme should apply for the action field, i.e., only those services which support the same action in their Intents advertisements can be put onto candidate lists. On the other hand, naïve intents are a relaxed version of authoritative intents. Thus a similarity model is required to measure the relevance between the naïve intent action field and the Intents advertisement action field. Then services are scored according to the similarity model and ranked in descending order, following which, the top ranked services above a threshold are returned as candidate services.

Figure 3.12 shows the process of intent resolution in user agents. When an intent comes, user agents first determine its intent type. If it is an explicit intent, it is directly transferred to remote services. If it is an implicit intent, an Intents advertisement registry is needed to generate candidate services. User agents first dispose of the services whose data type fails to pass the data type matching process. Then the intent is checked to see if it is an authoritative intent or naïve intent. For an authoritative intent, exact action matching is employed and only those services which support the same action can be selected as candidate services. On the other hand, for a naïve intent, user agents need to calculate a similarity score for each service. Then services are sorted by the score in descending order and the top ranked services above a threshold are returned to the user as candidate services. After the end user makes a choice, the implicit intent will be sent to the selected service.

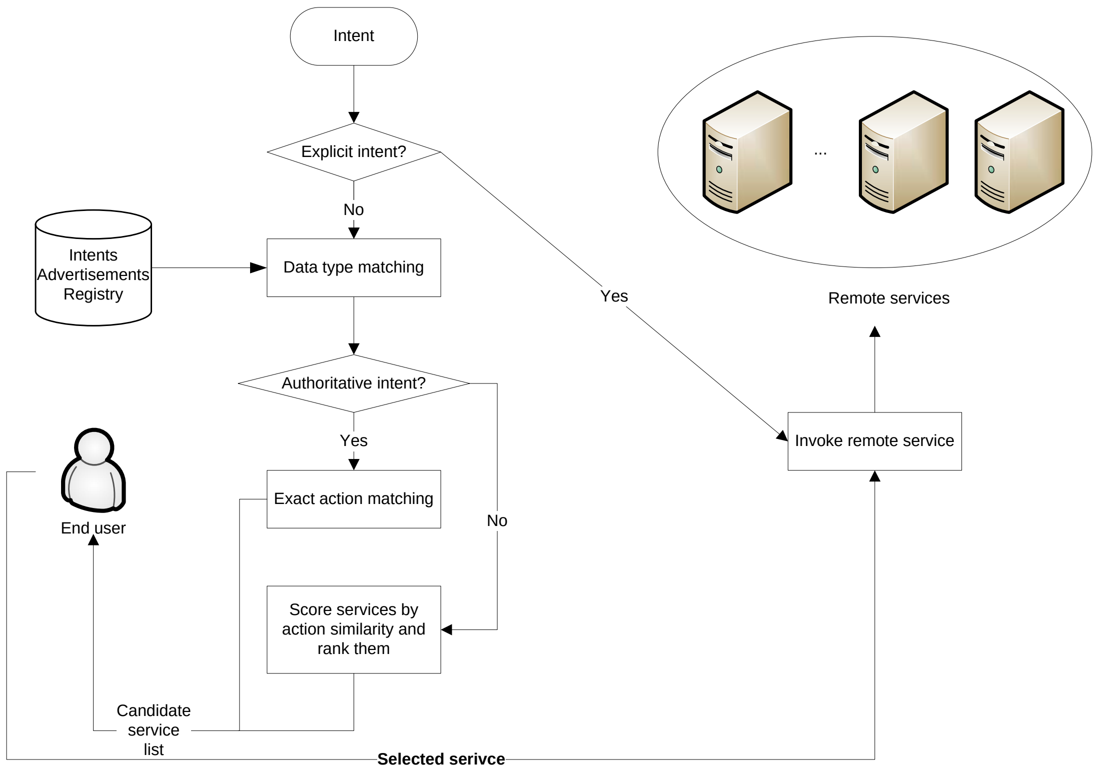


Figure 3.12: Intent resolution process

As for the Intents advertisement registry used for generating candidate service lists, there could be three levels for them: the personal level, the group level, and the public level, as shown in Figure 3.13.

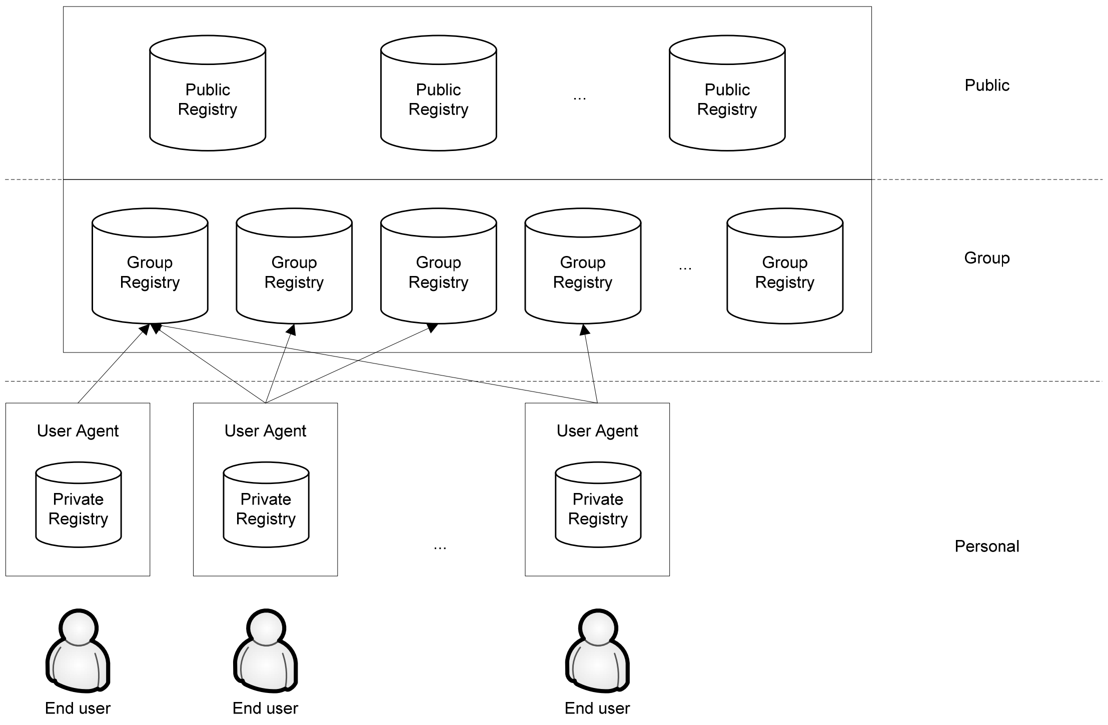


Figure 3.13: Intents advertisement registry levels

- **Personal level:** Each user agent keeps a private Intents advertisement registry and works on behalf of its owner, i.e., an end user. End users add and remove Intents advertisements according to their own interests. As a result, their registries are populated with separate and personalized service sets, and when a user agent generates candidate service lists from its private registry, the lists are personalized and different from user to user.
- **Group level:** Several end users may create an interest group. Then the union of their private Intents advertisement registries constitutes a group Intents advertisement registry. As a result, the candidate services generated from the group registry could reflect a collaborative service recommendation. For instance, user A is using an application which depends on a travel agency service. However, A's user agent has no such service in its internal advertisement registry. Fortunately, the travel agency service E is popular in A's friend circle. Then A's user agent can acquire service E from the group registry as a candidate service to

A. If A is satisfied with E, he/she can add the service to his/her own Intents advertisement registry for later use.

- **Public level:** A service broker may build a public service registry working as a service market. On the one hand, when neither private registry nor group registry is able to assist user agents in generating candidate services, public registries provide the last resort with their most populous storage of Intents advertisements. On the other hand, public registries also provide a facility through which end users can populate their own private registries.

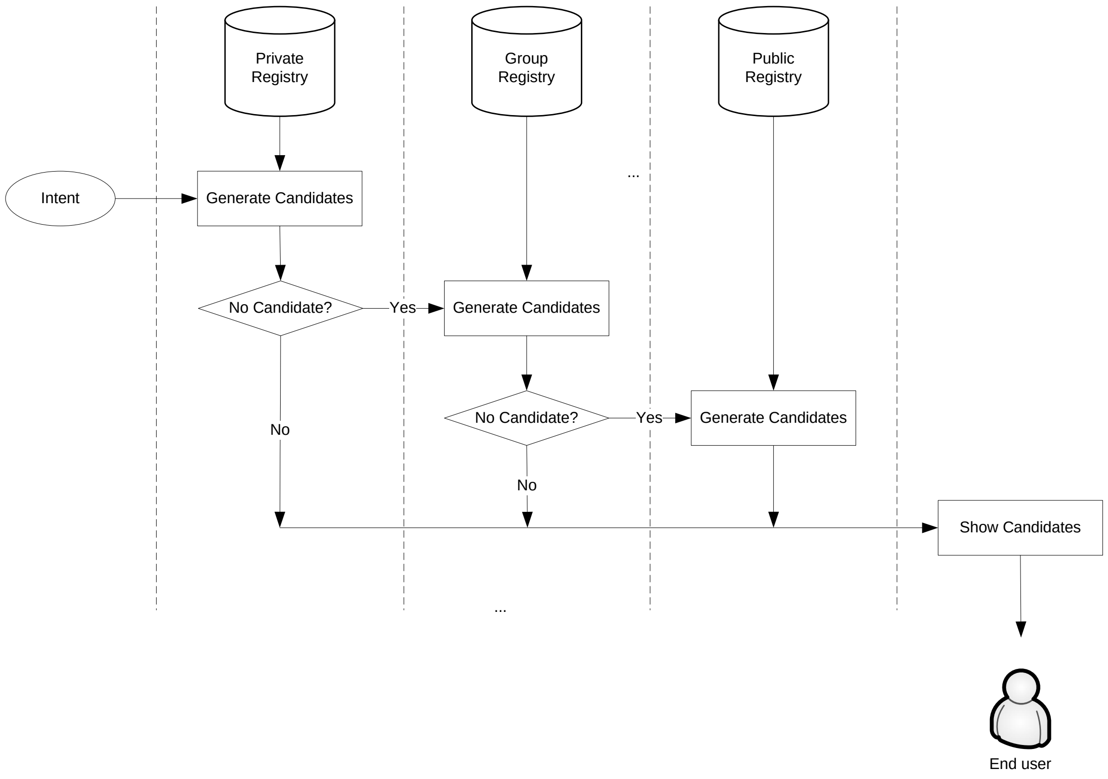


Figure 3.14: Registry upgrading in intent resolution

From private registries to public registries, the sizes and varieties of Intents advertisements increase. Thus the possibilities of finding out candidate services also increase. Figure 3.14 presents the idea that user agents may turn to higher level registries when they fail to generate candidate services from the current registry. When an intent comes, user agents first use their private registry. If there is no candidate service generated, they can turn to a group registry. If neither the private registry nor the group registry is able to assist user agents in creating candidate services, they will resort to a public Intents advertisement registry.

The three levels of registries also provide flexibility to meet different user requirements. For instance, an end user may not want to maintain a private Intents advertisement registry; he/she can just use a group or set a public registry for intent resolution.

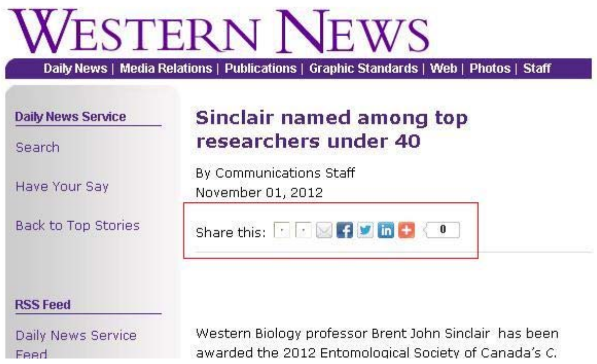
This section only articulates the basic procedure for intent resolution. Chapter 4 will present more details on intent resolution, especially the problem of constructing similarity models.

3.5 Use Cases Study

Intents is able to address some issues prevailing in current application development and provides innovative solutions. This section will present two scenarios to demonstrate the benefits and value of Intents.

3.5.1 Sharing Button

Sharing buttons are widely adopted in current Web applications for users to share links, files, text, images, audio and videos. Figure 3.15 shows a Web application with sharing buttons.



Sinclair named among top researchers under 40

By Communications Staff
November 01, 2012

Western Biology professor Brent John Sinclair has been awarded the 2012 Entomological Society of Canada's C.

Figure 3.15: Sharing an article link with sharing buttons

Figure 3.15 is an application with sharing buttons to share this article through Facebook, Twitter, and LinkedIn. If the three sharing services cannot meet user needs, a plus button follows which can pop up a list with more sharing services.

Like Figure 3.15, current applications usually attach a list of sharing buttons, each for one remote service. However, this design has the following issues:

- If there are too few sharing buttons, they may not meet user sharing requirements. With the rapid growth of social media Web sites, people have many and various preferences on using different platforms for content sharing and these platforms are not limited to just a few such as Facebook, Twitter and Gmail. Besides, not all buttons are applicable everywhere. The gates of regional networks may block the services for some buttons. If all the sharing buttons in an application are blocked, the functionality of the application which depends on sharing buttons will be affected.
- If there are too many sharing buttons, they may degrade application performance because each sharing button is composed in JavaScript code which is interpreted at runtime and needs to communicate with its service provider. In addition, too many sharing buttons may cripple the user's decision making ability. It is difficult for users to quickly find out the desired button when they are faced with too many options.

Figure 3.16 demonstrates how Intents addresses the above issues. Intents applies only one sharing button instead of creating one for each service. When the sharing button is triggered by end users, an implicit intent is created and sent to user agents. User agents resolve the intent and generate candidate service lists based on their own private registry. Since each user agent has a personalized registry which is maintained according to its owner's preference, candidate service lists are totally personalized and moderate in size. In addition, when some services are blocked in a regional network, the end users in the network can maintain a registry of services which are operational in the network. As a result, his/her generated candidate lists will contain only valid services.

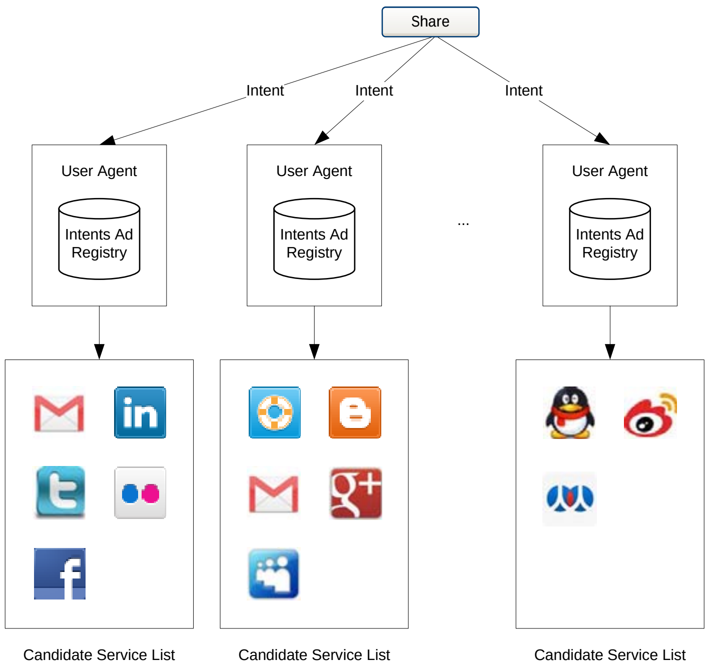


Figure 3.16: The sharing button in Intents

3.5.2 Weaving Services

Intents can also be used in service recommendation. Assuming a case which considers three actions (edit, upload and share) revolving around an image, if they are denoted by A, B and C respectively, some of their possible workflows are listed in Table 3.10.

Table 3.10: Workflow samples for weaving services

	Work Flow	Remark
1	$start \to A \to end$	Edit the image
2	$start \to B \to end$	Upload the image to a cloud drive
3	$start \to C \to end$	Share the image to some social media Web site
4	$start \to A \to B \to end$	First edit the image, then upload it to a cloud drive.
5	$start \to A \to C \to end$	First edit the image, then share it to some media Web site

At the same time, each action has a set of possible services as in Table 3.11.

Table 3.11: Possible services for each action

Action	Possible Services
A	a_1, a_2
B	b_1, b_2
C	c_1, c_2, c_3

Each user selects a service for each action in Intents. For instance, one user with workflow of $start \rightarrow A \rightarrow end$ may select service a_2, and another user with workflow of $start \rightarrow A \rightarrow B \rightarrow end$ may select a_2 and b_2 for each action. Table 3.12 lists a set of end users and their service selections.

Table 3.12: Service selection for each user

User	Work Flow	Service Selection
1	$start \rightarrow C \rightarrow end$	$start \rightarrow c_1 \rightarrow end$
2	$start \rightarrow A \rightarrow C \rightarrow end$	$start \rightarrow a_1 \rightarrow c_2 \rightarrow end$
3	$start \rightarrow B \rightarrow end$	$start \rightarrow b_1 \rightarrow end$
4	$start \rightarrow A \rightarrow end$	$start \rightarrow a_2 \rightarrow end$
5	$start \rightarrow A \rightarrow B \rightarrow end$	$start \rightarrow a_2 \rightarrow b_2 \rightarrow end$

If the flow from one service to another by an end user is considered as a link, all user activities and selected services can be woven together to constitute a graph, as shown in Figure 3.17.

With this graph when a new user comes and takes a picture, his/her Intents user agent is capable of guiding him/her on the follow-up actions and recommending services for each action. For instance, the user agent may suggest editing the picture and recommend a list of services for the action according to the information in the graph.

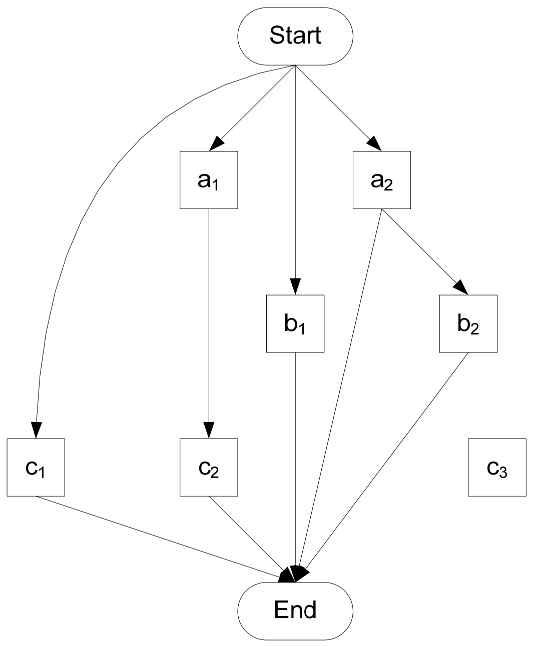


Figure 3.17: User selected services graph

3.6 Summary

This chapter presents a new Intents-based service discovery and integration approach. The work starts off with a discussion on the current Web service architecture and its issues in service discovery and integration. In order to address the issues, the implicit mode of the Intents architecture, which is characterized by end user participation is introduced. Meanwhile, the explicit mode of the Intents architecture keeps the characteristics of the current service discovery and integration techniques.

An intent represents a user service goal. This chapter presents the intent data structure and three intent types which are designed to meet various user service needs. An Intents service is a Web service which is able to accept and process intents. Intents services should be marked with Intents advertisements as a part of the service description. Intents services can be created directly from scratch or indirectly from current Web services. This chapter demonstrates how to create Intents services from SOAP and REST Web services.

Intent resolution is the process of analyzing an incoming intent and generating candidate services. This chapter presents the basic intent resolution steps and the three levels of Intents advertisement registries which are utilized in intent resolution.

In order to demonstrate the benefits of Intents, two cases are studied. One is the sharing button scenario. Compared with current sharing button mechanisms, Intents only adopts one sharing button and can generate personalized candidate service lists. In the other case, Intents is able to weave services together to constitute a service graph. Based on the graph, Intents may recommend new users with actions and services.

Chapter 4

4 Adaptive Intent Resolution

In intent resolution, a similarity model is needed for naïve intents. This chapter will explore the task of seeking a similarity model by transforming it into an optimization problem. Then the problem is addressed by empirical methods. Based on the empirical analysis, an adaptive intent resolution approach is proposed in this chapter.

4.1 Similarity Model Formulation

As mentioned in the previous chapter, a similarity model is required in the process of naïve intent resolution. The model takes the responsibility for scoring services according to their relevance to the naïve intent. This section will present how to formulate the similarity model.

Formally, if the naïve intent and Intents advertisement are denoted by I and A respectively, their only fields which can be leveraged for calculating the similarity score are I 's action field and A's action field. As a result, the similarity model S should be:

$$S = sim(action_I, action_A) \tag{4.1}$$

Formula 4.1 shows the similarity model as a function of I 's action field and A's action field. However, the action fields are usually short text which may not yield satisfactory results. Inspired by the literature reviewed in Subsection 2.2.2, additional information will be utilized to extend the similarity model.

In addition to the action field in Intents advertisements, each service may have another two fields as its descriptive information: *service title* and *service introduction*. The two fields can be leveraged to help extend Formula 4.1. Service title is a short text field for describing services while at the same time service introduction is a long text field which provides introductory information for services. Compared with the action field, service titles and introductions may provide more descriptive information which assists in calculating service similarity scores. Service titles and introductions are not required in

Intents advertisements but they may be obtained from other service description files such as online documents or Web pages where Intents advertisements are embedded.

With the introduction of the two fields, the similarity model is transformed into:

$$\begin{aligned} S = {} & \omega_1 sim(action_I, action_A) \\ & + \omega_2 sim(action_I, title) \\ & + \omega_3 sim(action_I, introduction) \\ \text{s.t.} \quad & \sum_{i=1}^{3} \omega_i = 1 \end{aligned} \tag{4.2}$$

In Formula 4.2, the $sim(x, y)$ function can be constructed as follows:

Since x comes from a text field of an intent and y from a text field of a set of services, the function $sim(x, y)$ works like for the problem of scoring document d in a document set based on query q. Then $sim(x, y)$ is transformed into

$$sim(q, d) \tag{4.3}$$

which is the classic problem in information retrieval (IR). Thus the similarity model required in naïve intent resolution can be created by employing retrieval models in IR.

In Formula 4.2, S represents a linearly weighted combination of similarities between the intent action field, the Intents advertisement action field, the service title field, and the service introduction field. On the service side, only the Intents advertisement action field always exists (service titles and introductions are optional), therefore the weights in S are subject to meeting a set of constraints in different conditions. In other words, if the set of services having passed intent type checking and data type matching is denoted by C, and $f_{common}(C)$ represents the set of common fields in C, then the weight constraints for S as per $f_{common}(C)$ are listed in Table 4.1.

Table 4.1: Weight constraints for S

Condition	Weight Constraint
$f_{common}(C)=\{action_A\}$	$\begin{cases}\omega_1=1\\ \omega_2=\omega_3=0\end{cases}$
$f_{common}(C)=\{action_A, title\}$	$\begin{cases}\omega_1+\omega_2=1\\ \omega_3=0\end{cases}$
$f_{common}(C)=\{action_A, introduction\}$	$\begin{cases}\omega_1+\omega_3=1\\ \omega_2=0\end{cases}$
$f_{common}(C)=\{action_A, title, introduction\}$	$\sum_{i=1}^{3}\omega_i=1$

Given a set of weighting schemes, there could be a set of similarity model *templates* derived from S. On the other hand, the $sim(x, y)$ function may be selected from a set of IR models, i.e., IR model m from model set M. If the formula template set derived from S is denoted by $\Im$ and s is its member, then each tuple $<s,m>$ determines a specific similarity model which may be used directly in naïve intent resolution.

In order to seek the optimal $<s,m>$, an evaluation measure (denoted by E) which is a function of s and m should be applied. Then the optimization problem for seeking the best $<s,m>$ can be expressed as:

$$\arg\max_{s\in\Im, m\in M}(E(s,m)) \quad \text{s.t.} \quad f_{common}(C) \tag{4.4}$$

Formula 4.4 can help to find out the best $<s,m>$ under a constraint of $f_{common}(C)$. However, analytically solving the formula is almost impossible. In the next section, an empirical approach will be demonstrated to search for the best $<s,m>$ under different conditions.

4.2 Empirical Study on the Similarity Model

In this section, an empirical approach is demonstrated for seeking the best $<s,m>$ pair under different $f_{common}(C)$ conditions. The whole idea of the approach is as follows:

Firstly, a set of templates derived from S and some IR models in M are listed. Then we conduct a series of experiments on a data set to measure the performance of different combinations of the templates and IR models based on a set of selected evaluation measures. According to the results, the best $< s, m >$ pairs are determined for all the $f_{common}(C)$ conditions and selected evaluation measures.

4.2.1 Similarity Model Templates and IR Models

Table 4.2 lists the set of formula templates derived from S which will be applied in the following experiments.

Table 4.2: Set of similarity model templates

Template Formula	Description
$S_1 = S(\omega_1 = 1, \omega_2 = 0, \omega_3 = 0)$	Only consider the Intents advertisement action field
$S_2 = S(\omega_1 = 0, \omega_2 = 1, \omega_3 = 0)$	Only consider the service title field
$S_3 = S(\omega_1 = 0, \omega_2 = 0, \omega_3 = 1)$	Only consider the service introduction field
$S_4 = S(\omega_1 = \frac{1}{2}, \omega_2 = \frac{1}{2}, \omega_3 = 0)$	Equal combination of the Intents advertisement action field and the service title field
$S_5 = S(\omega_1 = \frac{1}{2}, \omega_2 = 0, \omega_3 = \frac{1}{2})$	Equal combination of the Intents advertisement action field and the service introduction field
$S_6 = S(\omega_1 = 0, \omega_2 = \frac{1}{2}, \omega_3 = \frac{1}{2})$	Equal combination of the service title field and the service introduction field
$S_7 = S(\omega_1 = \frac{1}{3}, \omega_2 = \frac{1}{3}, \omega_3 = \frac{1}{3})$	Equal combination of the Intents advertisement action field, the service title field, and the service introduction field

$S_1 - S_7$ cover all the equal combinations of one, two, and three weights. They constitute the $\Im$ set for Formula 4.4.

As for IR models, Apache Lucene default IR model, Okapi BM25, a Dirichlet smoothed language model, and F2-EXP are chosen for the experiments. They are representative implementations for the space vector model, the probabilistic model, the language model, and the axiomatic model, respectively.

Lucene Default Model

Lucene default IR model is an implementation of the vector space model which is empirically successful and widely recognized. Its function can be described as:

$$sim(q,d) = coord(q,d) \times queryNorm(q) \sum_{t \in q \cap d} tf(t,d) idf^2(t) boost(t) norm(t,d) \quad (4.5)$$

More details on the formula are listed in Table 4.3.

Table 4.3: Lucene default IR model items

Item	Calculation
$coord(q,d)$	$\frac{\lvert q \cap d \rvert}{\lvert d \rvert}$
$queryNorm(q)$	Not applicable in the experiments
$tf(t,q)$	$\sqrt{f(t,d)}$
$idf(t)$	$1 + \log \frac{N}{df(t)+1}$
$boost(t)$	Set to 1 in the experiments
$norm(t,d)$	$\frac{1}{\sqrt{\lvert d \rvert}}$

The notation in Table 4.3 is explained in Table 4.4.

Table 4.4: Notation explanation for Table 4.3

Symbol	Explanation
$\lvert q \cap d \rvert$	The number of terms both in query q and document d
$\lvert d \rvert$	The length of document d
$f(t,d)$	The count of term t in document d
N	The number of all indexed documents
$df(t)$	The count of documents containing term t

If only the effective parts are considered, the formula for the Lucene default IR model is reduced to Formula 4.6 which will be applied in the experiments.

$$sim(q,d) = \frac{\lvert q \cap d \rvert}{\lvert d \rvert^{\frac{3}{2}}} \sum_{t \in q \cap d} f^{\frac{1}{2}}(t,d)(1 + \log \frac{N}{df(t)+1})^2 \quad (4.6)$$

Okapi BM25

Okapi BM25 [Robertson et al., 1995; Robertson et al., 1999] is an implementation of the probabilistic model in the Okapi IR system. Its function can be shown as:

$$sim(q,d) = \sum_{t \in q \cap d} \log \frac{N - df(t) + 0.5}{df(t) + 0.5} \times \frac{f(t,d)(k+1)}{f(t,d) + k(1 - b + b\frac{|d|}{avgdl})} \quad (4.7)$$

The notation in Formula 4.7 is explained in Table 4.5.

Table 4.5: Notation explanation for Formula 4.7

Symbol	Explanation
k	Constant, set to 1.25 in the experiments
b	Constant, set to 0.75 in the experiments
$avgdl$	The average length of all the indexed documents
$\|d\|$	The length of document d
$f(t,d)$	The count of term t in document d
N	The number of all indexed documents
$df(t)$	The count of documents containing term t

LM Dirichlet

Bayesian smoothing using the Dirichlet distribution prior to the language model is a technique studied by Zhai and Lafferty [Zhai and Lafferty, 2001a]. Its function can be shown as:

$$sim(q,d) = \sum_{t \in q \cap d} \log(1 + \frac{f(t,d)}{\mu P(t \mid C)}) + |q| \log(\frac{\mu}{|d| + \mu}) \quad (4.8)$$

The notation in Formula 4.8 is explained in Table 4.6.

Table 4.6: Notation explanation for Formula 4.8

Symbol	Explanation
μ	Constant, set to 2000
$\|q\|$	The length of query q
$\|d\|$	The length of document d
$f(t,d)$	The count of term t in document d
$P(t\|C)$	The count of term t in all documents in the collection C

F2-EXP

F2-EXP is an implementation of the axiomatic model. Its function can be shown as:

$$sim(q,d)=\sum_{t\in q\cap d} f(t,d)\times\left(\frac{N}{df(t)}\right)^{k}\times\frac{f(t,d)}{f(t,d)+s+\frac{s|d|}{avgdl}} \tag{4.9}$$

The notation in Formula 4.9 is explained in Table 4.7.

Table 4.7: Notation explanation for Formula 4.9

Symbol	Meaning
k	Constant, set to 0.35
s	Constant, set to 0.5
$avgdl$	The average length of all indexed documents
$\lvert d\rvert$	The length of document d
$f(t,d)$	The count of term t in document d
N	The number of all indexed documents

The four functions are representative implementations for classic IR models. Table 4.8 summarizes them with some brief information. They constitute the M set applied in the experiments in this thesis.

Table 4.8: Summary of the implemented IR models

Implementation Name	Model	$sim(q,d)$
Lucene Default Model	Vector space model	$\frac{\lvert q\cap d\rvert}{\lvert d\rvert^{\frac{3}{2}}}\sum_{t\in q\cap d} f^{\frac{1}{2}}(t,d)(1+\log\frac{N}{df(t)+1})^{2}$
Okapi BM25	Probabilistic model	$\sum_{t\in q\cap d}\log\frac{N-df(t)+0.5}{df(t)+0.5}\times\frac{f(t,d)(k+1)}{f(t,d)+k(1-b+b\frac{\lvert d\rvert}{avgdl})}$
LM Dirichlet	Language model	$\sum_{t\in q\cap d}\log(1+\frac{f(t,d)}{\mu P(t\mid C)})+\lvert q\rvert\log(\frac{\mu}{\lvert d\rvert+\mu})$
F2-EXP	Axiomatic model	$\sum_{t\in q\cap d} f(t,d)\times\left(\frac{N}{df(t)}\right)^{k}\times\frac{f(t,d)}{f(t,d)+s+\frac{s\lvert d\rvert}{avgdl}}$

4.2.2 Evaluation Measures

This subsection will deduce the evaluation measures applied in the experiments. The evaluation measures applied in this work come from five classic evaluation measures commonly applied in IR. They are *recall*, *precision*, *F-measure*, *mean average precision* (MAP), and *mean reciprocal rank* (MRR). However, some of them are slightly changed because of the following concept.

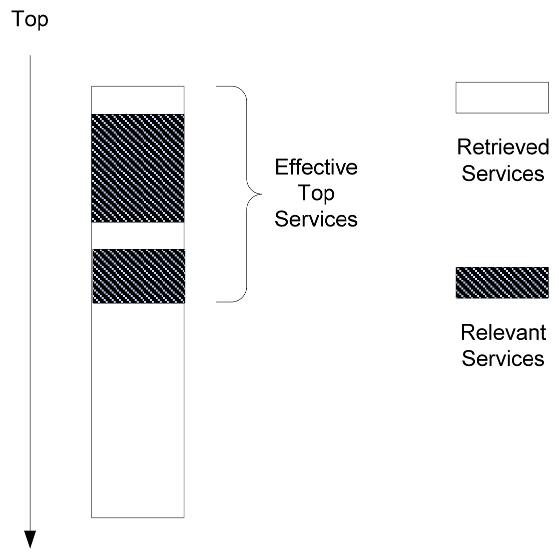


Figure 4.1: Effective top services

Figure 4.1 shows a result of retrieved services. It can be observed that the bottommost part of the retrieved services may be all irrelevant services. When it comes to the last relevant service, the retrieval of extra irrelevant services only produces noise in the retrieved services. As a result, the services before the last retrieved relevant services as shown in Figure 4.1 are named *effective top services* because new retrieved services have a chance to be relevant when their ranks are within effective top services.

The ratio of the effective top services to retrieved services can be treated as a threshold to cut off the irrelevant services in the bottommost part of retrieved services to improve the resulting quality. Since this threshold will be applied in the design of the adaptive intent resolution approach in the next section, the precision, F-measure, and MAP evaluation measures are modified to incorporate the concept of effective top services. The modified evaluation measures are named *precision$_e$*, *F-measure$_e$*, and *MAP$_e$*.

The recall measure is illustrated by Formula 4.10.

$$recall = \frac{|\{relevant\ services\} \cap \{retrieved\ services\}|}{|\{relevant\ services\}|} \tag{4.10}$$

A high recall means that a similarity model returns most of the relevant services.

The *precision$_e$* measure is illustrated by Formula 4.11.

$$precision_e = \frac{|\{relevant\ services\} \cap \{effective\ top\ services\}|}{|\{effective\ top\ services\}|} \tag{4.11}$$

The precision$_e$ measure substitutes effective top services for retrieved services in the ordinary precision definition. However, their underlying rationale is similar. A high precision$_e$ value means that the majority of effective top services are relevant services.

The *F-measure$_e$* measure is illustrated by Formula 4.12.

$$F - measure_e = 2\frac{precision_e \times recall}{precision_e + recall} \tag{4.12}$$

F-measure$_e$ creates a balance between recall and precision$_e$. A high F-measure$_e$ means the sets of effective top services and relevant services are similar.

MAP$_e$ is the mean of average precision over all intents and average precision is the average of the precisions at each rank for all the effective top services. Formally speaking, if Γ is the set of all intents used to search for services, $\Re_e(I)$ is the set of all the effective top services for intent I, and p_k is the precision at rank k then MAP is formalized as Formula 4.13.

$$MAP_e = \frac{1}{|\Gamma|}\sum_{I\in\Gamma}\frac{1}{|\Re_e(I)|}\sum_{k=1}^{|\Re_e(I)|} p_k \tag{4.13}$$

Like precision$_e$ and F-measure$_e$, MAP$_e$ in this thesis is also slightly different from the MAP measure commonly used in IR. We also substitute effective top services for retrieved services. But their underlying rationale is similar. A high MAP$_e$ indicates that the majority of relevant services are placed at the topmost positions in effective top services.

MRR is the average multiplicative inverse of the rank for the first correctly retrieved service, as shown in Formula 4.14.

$$MRR = \frac{1}{|\Gamma|}\sum_{I\in\Gamma}\frac{1}{rank_I} \tag{4.14}$$

where Γ is the set of all intents used to search for services, and $rank_I$ is the first relevant service rank. MRR is high when the first relevant service in effective top services is ranked at the top for most of the intents in question.

Table 4.9 briefly summarizes the five evaluation measures and their corresponding purposes.

Table 4.9: Summary of evaluation measures

Evaluation Measure	Purpose
Recall	Retrieve more relevant services
Precision$_e$	Make the effective top services pure with relevant services
F-measure$_e$	A balance between recall and precision
MAP$_e$	Make the majority of relevant services at the topmost positions
MRR	Make the first relevant service ranked at the top

The five evaluation measures can be applied to meet different user requirements. Thus they will be used to select the best $< s, m >$ for the similarity model in this work.

4.2.3 Experiment set-up and preprocessing

This subsection will present the experimental platform, the dataset and its preprocessing steps.

Platform

Apache Lucene[1] is modified to become the experimental platform for this work. Lucene has inner implemented IR models including a default implementation for the vector space model, an Okapi BM25 for the probability model, and a Dirichlet smoothed implementation for the language model. In addition to the three implementations, we developed an axiomatic IR model (F2-EXP) based on the programming interfaces provided by Lucene.

Dataset

The dataset employed in the experiments was extracted from a public intent registry on OpenIntents[2]. The dataset is composed of Android intent description entries which are registered by Android application developers. Each entry in the dataset consists of an action field, a service title field, an introduction field, and other parts which can be counted as service description. The action field, the service title field and the introduction field are selected and applied in the experiments. The action field and the service title field are short text fields while the introduction field is a long text field and they meet the experimental requirements. An overview of the dataset is shown in Table 4.10.

Table 4.10: Dataset statistics

Name	Statistics
Services	83
Intent actions for querying	88
Relevance judgments	119
Average relevance judgments	1.35

[1] http://lucene.apache.org/

[2] http://www.openintents.org/

Even though this is a comparatively small dataset, all the data are from realistic open Android applications registered by their developers with OpenIntents. Therefore, we believe it can help reveal the true characteristics of intent resolution. A set of 88 intent actions for querying (empirically, more than 50 queries is enough for testing [Manning et al., 2008]). They are created randomly from the service description keywords. Their relevance judgments are generated manually.

Preprocessing

Before the experiments start, some preprocessing steps should be applied to all the involved fields including the intent action field, the intent advertisement action field, the service title field, and the service introduction field. The steps include tokenization, lowercasing, removing stop words, and stemming.

Tokenization is the process of breaking up the text field into words and removing punctuation. Lowercasing is the process of transforming capital letters into their lower case forms. The two steps together generate a sequence of normalized words for each text field.

Stop words are words that appear in most services. They contribute little to discriminate services. Therefore, stop words should be removed from all the service fields in question. Lucene keeps an internal stop word list for ordinary text. In order to make it appropriate to the experimental dataset, “com”, “intent”, and “org” which often occur in the action field, are added to the stop list.

For grammatical reasons, words are used in different forms such as “edit”, “editing”, and “edits”. These words are semantically similar. Thus their separate forms can affect the performance of the experiments. Stemming is the process of reducing words to their stems. For instance, the words “edit”, “editing”, and “edits” are all represented by “edit”. There exist many stemming techniques. In this work, a widely accepted and recognized stemming implementation is adopted, in accordance with the idea by Porter [Porter, 1980].

4.2.4 Results and Analysis

The experiments measured recall precision$_e$, F-measure$_e$, MAP$_e$, and MRR for all the combinations of the seven similarity model templates and the four IR model implementations listed in Table 4.2 and Table 4.8, respectively. This subsection will present the results and analysis from the conducted experiments.

Effective Top Services

Figure 4.2 shows the retrieved services and effective top services for each pair of similarity templates and IR model implementations.

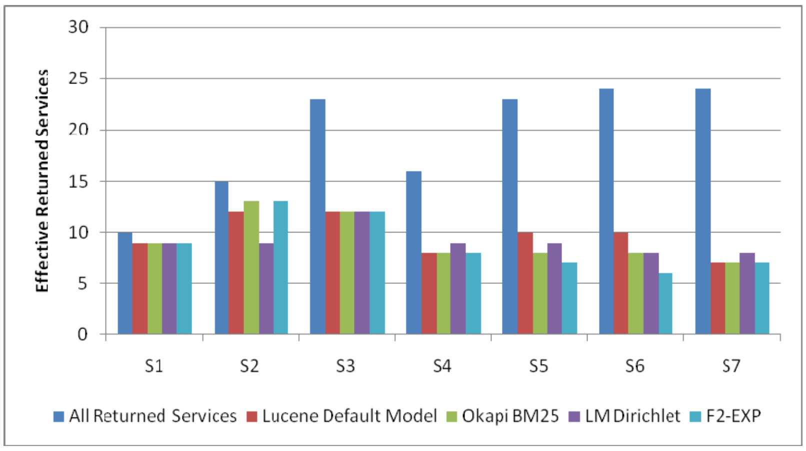


Figure 4.2: Effective top services for the experiments

It can be observed that all the IR model implementations for a similarity template retrieved the same number of services. Since the amount of all the services in C may be very big, retrieving and scoring them is time-consuming. Most IR implementations do the job in two steps. The first step retrieves the services whose fields in question have at least a common word with the intent action field. Then in the second step the retrieved services are scored and sorted in decreasing order. Even though the IR model implementations have separate scoring functions, their process of retrieving services are almost the same, i.e., the services whose fields in question have a common word with the intent action field are retrieved. In essence, for each similarity model template the fields in question are the same. Thus the retrieved services do not change for the same S_i.

$S_1 - S_7$ consider the different combinations of the related fields which cause various retrieved service sets. S_1, S_2 and S_4 retrieve comparatively fewer services. The reason for this is that they only take into account the action field and the service title field which both are short text. S_3, S_5, S_6, and S_7 consider the service introduction field which is a long text field. Thus they retrieve more services. On the other hand, the four similarity templates result in relatively small effective top services compared with their retrieved services. Figure 4.3 shows the effective top services as a fraction of the retrieved services for all the similarity model templates.

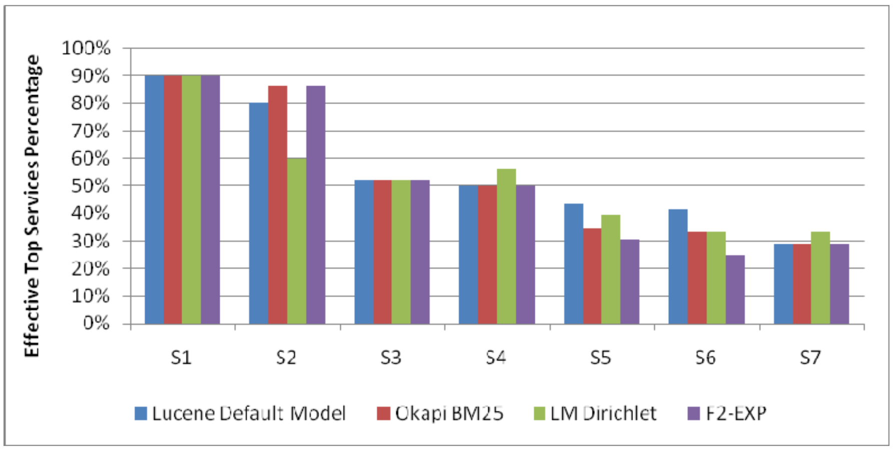


Figure 4.3: Effective top services compared to the retrieved services

It can be observed that under no circumstances the effective top services are equivalent to the retrieved services. Moreover, some similarity model templates only generate a small portion (20%–40%) for their effective top services. Therefore, cutting off the tail of the retrieved services, which are irrelevant, may improve resulting accuracy significantly.

Recall

Table 4.11 shows the recall results for each pair of similarity model templates and IR model implementations. Since all the IR model implementations adopt the same approach in the process of retrieving services, they should have the same set of relevant services. Thus their recalls for the same similarity model template are the same. Table 4.12 shows the best $< s, m >$ pair for each condition under the recall evaluation measure.

Table 4.11: Recall results

Similarity Model Template	Lucene Default Model	Okapi BM25	LM Dirichlet	F2-EXP
S_1	0.891	0.891	0.891	0.891
S_2	0.95	0.95	0.95	0.95
S_3	0.908	0.908	0.908	0.908
S_4	0.958	0.958	0.958	0.958
S_5	0.966	0.966	0.966	0.966
S_6	0.966	0.966	0.966	0.966
S_7	0.966	0.966	0.966	0.966

Table 4.12: Best selection of $< s, m >$ under recall

Condition	Best $< s, m >$ Pair
$f_{common}(C) = \{action_A\}$	$< S_1, Lucene\ Default\ Model >$ $< S_1, Okapi\ BM25 >$ $< S_1, LM\ \ Dirichlet >$ $< S_1, F2 - EXP >$
$f_{common}(C) = \{action_A, title\}$	$< S_4, Lucene\ Default\ Model >$ $< S_4, Okapi\ BM25 >$ $< S_4, LM\ \ Dirichlet >$ $< S_4, F2 - EXP >$
$f_{common}(C) = \{action_A, introduction\}$	$< S_5, Lucene\ Default\ Model >$ $< S_5, Okapi\ BM25 >$ $< S_5, LM\ \ Dirichlet >$ $< S_5, F2 - EXP >$
$f_{common}(C) = \{action_A, title, introduction\}$	$< S_5, Lucene\ Default\ Model >$ $< S_5, Okapi\ BM25 >$ $< S_5, LM\ \ Dirichlet >$ $< S_5, F2 - EXP >$ $< S_6, Lucene\ Default\ Model >$ $< S_6, Okapi\ BM25 >$ $< S_6, LM\ \ Dirichlet >$ $< S_6, F2 - EXP >$ $< S_7, Lucene\ Default\ Model >$ $< S_7, Okapi\ BM25 >$ $< S_7, LM\ \ Dirichlet >$ $< S_7, F2 - EXP >$

From Table 4.12 it can be inferred that recall is not a good measure to help select the best $< s, m >$ pair because it generates too many solutions for each condition.

Precision$_e$

Table 4.13 shows the precision$_e$ results for each pair of similarity model templates and IR model implementations.

Table 4.13: Precision$_e$ results

Similarity Model Template	Lucene Default Model	Okapi BM25	LM Dirichlet	F2-EXP
S_1	0.134	0.134	0.134	0.134
S_2	0.107	0.099	0.143	0.099
S_3	0.102	0.102	0.102	0.102
S_4	0.162	0.162	0.144	0.162
S_5	0.131	0.163	0.145	0.187
S_6	0.131	0.163	0.163	0.218
S_7	0.187	0.187	0.163	0.187

Even though all the IR model implementations retrieve the same set of relevant services, they may generate different effective top services because they have separate scoring functions. Thus their results are different under the same similarity template. Table 4.14 shows the best $< s, m >$ pair for each condition under the precision$_e$ evaluation measure.

Table 4.14: Best selection of $< s, m >$ under precision$_e$

Condition	Best $< s, m >$ Pairs
$f_{common}(C) = \{action_A\}$	$< S_1, Lucene\ Default\ Model >$ $< S_1, Okapi\ BM25 >$ $< S_1, LM\ Dirichlet >$ $< S_1, F2-EXP >$
$f_{common}(C) = \{action_A, title\}$	$< S_4, Lucene\ Default\ Model >$ $< S_4, Okapi\ BM25 >$ $< S_4, F2-EXP >$
$f_{common}(C) = \{action_A, introduction\}$	$< S_5, F2-EXP >$
$f_{common}(C) = \{action_A, title, introduction\}$	$< S_6, F2-EXP >$

From Table 4.14 it can be seen that precision$_e$ is much better than recall in selecting the best $< s, m >$ pair. Under two conditions, it has found the best solution.

F-measure$_e$

Table 4.15 shows the F-measure$_e$ results for each pair of similarity model templates and IR model implementations.

Table 4.15: F-measure$_e$ results

Similarity Model Template	Lucene Default Model	Okapi BM25	LM Dirichlet	F2-EXP
S_1	0.233	0.233	0.233	0.233
S_2	0.192	0.179	0.248	0.179
S_3	0.184	0.184	0.184	0.184
S_4	0.277	0.277	0.250	0.277
S_5	0.230	0.279	0.252	0.313
S_6	0.230	0.279	0.279	0.355
S_7	0.313	0.313	0.279	0.313

F-measure$_e$, which is the combination of precision$_e$ and recall should have the advantages of both recall and precision$_e$ in discriminating IR model implementations under the same similarity model template. Table 4.16 shows the best $<s,m>$ pair for each condition under the F-measure$_e$ evaluation measure.

Table 4.16: Best selection of $<s,m>$ under F-measure$_e$

Condition	Best $<s,m>$ Pair
$f_{common}(C)=\{action_A\}$	$<S_1, Lucene\ Default\ Model>$ $<S_1, Okapi\ BM25>$ $<S_1, LM\ \ Dirichlet>$ $<S_1, F2-EXP>$
$f_{common}(C)=\{action_A, title\}$	$<S_4, Lucene\ Default\ Model>$ $<S_4, Okapi\ BM25>$ $<S_4, F2-EXP>$
$f_{common}(C)=\{action_A, introduction\}$	$<S_5, F2-EXP>$
$f_{common}(C)=\{action_A, title, introduction\}$	$<S_6, F2-EXP>$

From Table 4.16 it can be observed that F-measure$_e$ works the same as precision$_e$ in selecting the best $<s,m>$ pair. Under two conditions, it has found the best solution.

MAP$_e$

Table 4.17 shows the MAP$_e$ results for each pair of similarity model templates and IR model implementations.

Table 4.17: MAP$_e$ results

Similarity Model Template	Lucene Default Model	Okapi BM25	LM Dirichlet	F2-EXP
S_1	0.302	0.302	0.301	0.301
S_2	0.281	0.266	0.337	0.267
S_3	0.264	0.260	0.259	0.264
S_4	0.370	0.366	0.343	0.369
S_5	0.325	0.361	0.331	0.392
S_6	0.329	0.376	0.363	0.443
S_7	0.411	0.404	0.366	0.407

Table 4.18 shows the best $< s, m >$ pair for each condition under the MAP$_e$ evaluation measure.

Table 4.18: Best selection of $< s, m >$ under MAP$_e$

Condition	Best $< s, m >$ Pair
$f_{common}(C) = \{action_A\}$	$< S_1, Lucene\ Default\ Model >$ $< S_1, Okapi\ BM25 >$
$f_{common}(C) = \{action_A, title\}$	$< S_4, Lucene\ Default\ Model >$
$f_{common}(C) = \{action_A, introduction\}$	$< S_5, F2-EXP >$
$f_{common}(C) = \{action_A, title, introduction\}$	$< S_6, F2-EXP >$

MAP$_e$ has the strongest capability to discriminate IR model implementations under the same similarity template. It can be observed that three conditions have found the best solution and the solutions for the left one are reduced to just two.

MRR

Table 4.19 shows the MRR results for each pair of similarity model templates and IR model implementations.

MRR is only related to the position of the first returned relevant service, so the distribution of other relevant services has no influence on its value. Table 4.20 shows the best $< s, m >$ pair for each condition under the MRR evaluation measure.

Table 4.19: MRR results

Similarity Model Template	Lucene Default Model	Okapi BM25	LM Dirichlet	F2-EXP
S_1	0.834	0.834	0.832	0.832
S_2	0.950	0.948	0.949	0.949
S_3	0.879	0.858	0.847	0.878
S_4	0.974	0.961	0.962	0.969
S_5	0.973	0.941	0.916	0.949
S_6	0.981	0.981	0.936	0.981
S_7	0.994	0.972	0.946	0.983

Table 4.20: Best selection of $< s, m >$ under MRR

Condition	Best $< s, m >$ Pair
$f_{common}(C) = \{action_A\}$	$< S_1, Lucene\ Default\ Model >$ $< S_1, Okapi\ BM25 >$
$f_{common}(C) = \{action_A, title\}$	$< S_4, Lucene\ Default\ Model >$
$f_{common}(C) = \{action_A, introduction\}$	$< S_5, Lucene\ Default\ Model >$
$f_{common}(C) = \{action_A, title, introduction\}$	$< S_7, Lucene\ Default\ Model >$

MRR also has a strong capability in discriminating between IR model implementations under the same similarity model template. Under three conditions the best solution has been found and the solutions of the left condition are reduced to just two.

4.3 An Adaptive Intent Resolution Approach

In this section, an adaptive intent resolution approach is proposed based on the previous experimental results and analysis.

4.3.1 Empirical Result Review

In the results of seeking the best $< s, m >$ pair in Table 4.12, 4.14, 4.16, and 4.18, it can be observed that

$$\{results\}_{recall} \supset \{results\}_{precision_e} = \{results\}_{F-measure_e} \supset \{results\}_{MAP_e} \qquad (4.15)$$

for each condition. Therefore, the results of MAP$_e$ can also apply to the other three evaluation measures. Table 4.21 shows the selected $< s, m >$ pairs for all the possible conditions under the evaluation measures of recall, precision$_e$, F-measure$_e$, and MAP$_e$.

Table 4.21: Selected $< s, m >$ pairs for recall, precision$_e$, F-measure$_e$, and MAP$_e$

	Introduction	**NOT Introduction**
Title	$< S_6, F2 - EXP >$	$< S_4, Lucene\ Default\ Model >$
NOT Title	$< S_5, F2 - EXP >$	$< S_1, Lucene\ Default\ Model >$

Since the Intents advertisement action field is a required field. Table 4.21 only considers the different conditions of the service title field and the service introduction field. In Table 4.21, “NOT” means such field is absent in C. In the condition that neither the service title field nor the service introduction field occurs, $< S_1, Lucene\ Default\ Model >$ is chosen as the best solution. The reason for disposal of the $< S_1, Okapi\ BM25 >$ pair is that the Lucene default model is also chosen as the best solution in other conditions. So from a global point of view, the Lucene default model works better than the Okapi BM25 model.

The results of MRR are not compatible with the other four, so they are listed in Table 4.22 separately.

Table 4.22: Selected $< s, m >$ pairs for MRR

	Introduction	**NOT Introduction**
Title	$< S_7, Lucene\ Default\ Model >$	$< S_4, Lucene\ Default\ Model >$
NOT Title	$< S_5, Lucene\ Default\ Model >$	$< S_1, Lucene\ Default\ Model >$

Similarly, in the condition that neither the service title field nor the service introduction field occurs, the pair $< S_1, Lucene\ Default\ Model >$ is chosen as the best solution because the Lucene default model is also chosen as the best solution in other conditions. So from a global point of view, the Lucene default model works better than Okapi BM25.

Table 4.21 and 4.22 together constitute the two basic heuristics which can be used to develop our adaptive intent resolution approach.

4.3.2 Adaptive Approach Design

In order to design an adaptive approach, the intent resolution process in Figure 3.12 should be modified to introduce adaptability.

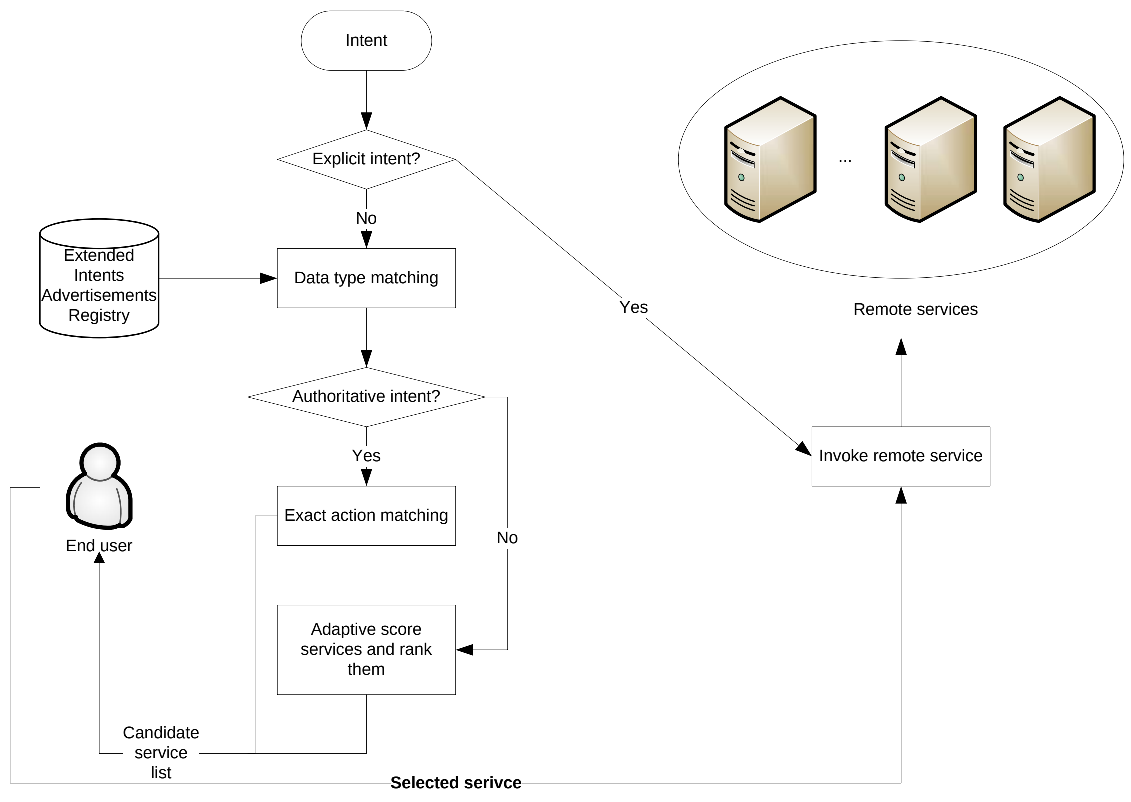


Figure 4.4: Adaptive intent resolution process

Figure 4.4 is a modified intent resolution process for Figure 3.12 to accommodate an adaptive intent resolution capability. The difference between the two figures is that Figure 4.4 introduces an adaptive service scoring and ranking procedure to replace the basic service scoring and ranking procedure. In addition, the Intents advertisement registry is extended to include the service title field and the service introduction field.

Each entry of the extended Intents advertisement registry is composed of a service identifier field, a service title field, a service introduction field, an Intents advertisement action field, and a data type field. Table 4.23 lists an example of the extended Intents advertisement registry.

Table 4.23: Example of the extended Intents advertisement registry

Service ID	Action	Title	Introduction	Data Type
http://202.117.1.119/share	share a link	Facebook link share	NULL	text/url-list
http://202.117.0.200/get-a-weather	local weather	NULL	This is a local weather service.	application/json
http://202.117.1.119/shorten	http://webintents.org/shorten	NULL	NULL	text/uri-list

In the real world, it is impossible for every service to have all of the Intents advertisement action field, the service title field, and the service introduction field. It should be possible that a service lacks the service title field or the service introduction field. Thus the prerequisite of the adaptive scoring process is to detect the common fields of the services which have passed the step of data type matching. As aforementioned, since the Intents advertisement action field is required and the service title field and the service introduction field are optional, the result of common field detection can only be one of four conditions.

The core idea of the adaptive scoring procedure is choosing the best $< s, m >$ to determine the similarity model based on the detected common fields, the predefined evaluation measure, and the two heuristics obtained from the empirical study in Section 4.2. Then the similarity model is used to score and rank the services. Finally, the tail of the ranked services will be cut off according to the effective top service threshold of the $< s, m >$ pair. Table 4.24 lists the effective top service thresholds which are measured in the empirical study in Section 4.2.

Table 4.24: Effective top service thresholds

Similarity Model Template	Lucene Default Model	Okapi BM25	LM Dirichlet	F2-EXP
S_1	90.00%	90.00%	90.00%	90.00%
S_2	80.00%	86.67%	60.00%	86.67%
S_3	52.17%	52.17%	52.17%	52.17%
S_4	50.00%	50.00%	56.25%	50.00%
S_5	43.48%	34.78%	39.13%	30.43%
S_6	41.67%	33.33%	33.33%	25.00%
S_7	29.17%	29.17%	33.33%	29.17%

4.3.3 MAP-based Adaptive Approach

This subsection will present the MAP-based adaptive approach which applies to the evaluation measures of recall, precision$_e$, F-measure$_e$, and MAP$_e$. If one of the four evaluation measures is selected by the user, the MAP-based approach can be used.

Assuming C is the set of services which have passed data type matching and I the naïve intent, Algorithm 1 presents the MAP-based adaptive approach.

Algorithm 1: MAP-based approach for the adaptive service scoring and ranking procedure

```
Input:   Service set $C$
         Naive intent $I$
Output:  A ranked list of candidate services from $C$
  1      Detect the common fields of $C$
  2      if the service introduction field is in the common fields
         then set F2-EXP as the IR model implementation
            if the service title field is in the common fields
            then
               Set the similarity model template to $S_6$
               Set the cut-off threshold to 0.25
            else
               Set the similarity model template to $S_5$
               Set the cut-off threshold to 0.3043
         else Set the Lucene default model as the IR model implementation
            if the service title field is in the common fields
            then
               Set the similarity model template to $S_4$
               Set the cut-off threshold to 0.5
            else
               Set the similarity model template to $S_1$
               Set the cut-off threshold to 0.9
  3      Use the selected similarity model template and IR model implementation to generate a
         ranked list of relevant services from $C$
  4      Use the selected threshold to cut off the tail of the list
  5      Return the resulting candidate service list
```

As demonstrated in Algorithm 1, the procedure begins with detecting the common service fields. The ultimate similarity model template and retrieval model are adaptively set according to the detected common fields and the heuristic in Table 4.21. Then the services are ranked and the tail is cut off according to the adaptively set threshold. Finally, an ordered list of candidate services is generated and returned.

4.3.4 MRR-based Adaptive Approach

This subsection will present the MRR-based adaptive approach which applies if the user selects MRR as the evaluation measure.

Assuming C is the set of services which have passed data type matching and I the naïve intent, Algorithm 2 presents the MRR-based adaptive approach.

Algorithm 2: MRR-based approach for the adaptive service scoring and ranking procedure

```
Input:   Service set $C$
         Naive intent $I$
Output:  A ranked list of candidate services from $C$
   1     Detect the common fields of $C$
   2     Set Lucene's default model implementation as the IR model implementation
   3     if the introduction field is in the common fields
         then
            if the title field is in the common fields
            then
               Set the similarity model template to $S_7$
               Set the cut-off threshold to 0.2917
            else
               Set the similarity model template to $S_5$
               Set the cut-off threshold to 0.4348
         else
            if the title field is in the common fields
            then
               Set the similarity model template to $S_4$
               Set the cut-off threshold to 0.5
            else
               Set the similarity model template to $S_1$
               Set the cut-off threshold to 0.9
   4     Use the selected similarity model template and IR model implementation to generate a
         ranked list of relevant services from $C$
   5     Use the selected threshold to cut off the last part of the list
   6     Return the resulting candidate service list
```

As demonstrated in Algorithm 2, the procedure begins with detecting the common service fields. The ultimate similarity model template and retrieval model are adaptively set according to the detected common fields and the heuristic in Table 4.22. Then the services are ranked and the tail is cut off according to the adaptively set threshold. Finally, an ordered list of candidate services is generated and returned.

4.4 Summary

This chapter begins with formally articulating the similarity model applied in naïve intent resolution. Since the action field is a short text field, we add two other fields (the service title field and the service introduction field) to enrich the service-side information and obtain an extended similarity model determined by a similarity model template and IR model pair. Then we transform the similarity model into an optimization problem.

In order to solve the similarity model problem, an empirical approach is applied. First, we define seven representative similarity model templates, four classic IR model implementations, and five evaluation measures. Then a series of experiments are conducted on a dataset from the real world to measure the performance of each combination of similarity model templates, IR model implementations, and evaluation measures. For each evaluation measure, the best similarity template and IR model pairs are selected under different conditions.

Based on the empirical study and analysis, two heuristics are obtained. Then we revise the intent resolution process to introduce adaptability and devise two adaptive approaches for the process according to the two heuristics, respectively.

Chapter 5

5 User Agent: Design, Implementation, and Application

In the Intents architecture, a user agent exists and works for each end user. It takes the responsibility for resolving intents and generating candidate services for implicit intents. This chapter will present a design for user agents. In addition, a proof-of-concept implementation of the design will be demonstrated.

With the development of mobile devices, integrating Web services and on-device native services is in demand. This chapter will also present employing Intents and the Intents user agent to integrate Web services and native services. The integration will be demonstrated by the implemented user agent.

5.1 User Agent Design

Intents user agents are the most critical component in Intents. A user agent takes the responsibility for collecting and managing Intents advertisements on behalf of its owner, rendering Web applications, resolving intents, generating candidate services, communicating with remote services and service brokers, and managing its owner's interest groups.

Figure 5.1 shows a three-tier architecture design of the user agent at the presentation layer, the application layer, and the data layer. More details on each layer are presented as follows.

- **Data layer.** The data layer of each user agent includes an Intents advertisement registry, a group list, a settings file, and an index for Intents advertisements. The Intents advertisement registry is private where personalized Intents advertisements are stored. The registry is managed under the control of the owner through the interfaces provided by the user agent. The group list keeps a list of interest groups which can refer the user agent to a set of group Intents advertisement registries. The settings file contains broker profiles, evaluation settings, and other configurations which are of great use in intent resolution. The

Intents advertisement index contains the Intents advertisements which are used to generate candidate services. Mature indexing techniques are applied here so that accessing the index by upper layer modules is very fast. The content of the index may not only come from the internal Intents advertisement registry. Broker and group registries can also be sources of the Intents advertisements index.

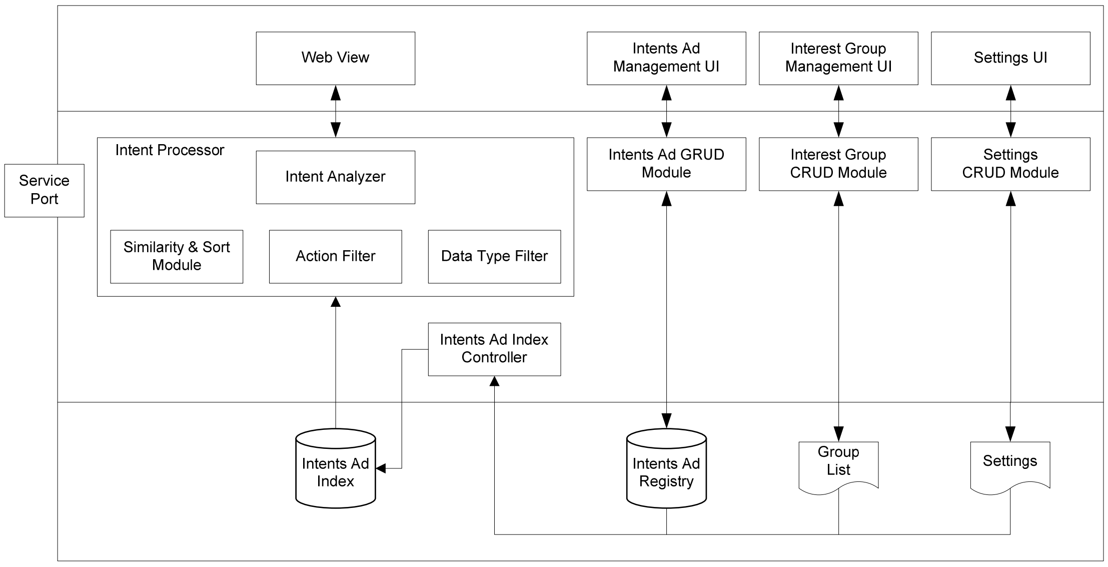


Figure 5.1: User agent design

- **Application layer.** Each user agent should have a service port module to communicate with remote services. Inside the user agent, the port module is only connected to the intent processor module. None of the other user agent modules can communicate with it but through the intent processor. If the port receives a request message from the intent processor, it sends the message directly to the remote service specified in the message. On the other hand, once it receives any response message from remote services, it forwards the message to the intent processor. The intent processor is the critical module of the user agent which is composed of an intent analyzer, a data type filter, an action filter, and a scoring component. The intent analyzer parses incoming intents to extract the content of the intent type field, the action field, and the data type field. Then the fields are distributed to corresponding components according to the intent type field for further processing. If the intent is an explicit intent, it is sent to the specified remote service through the service port directly. If the intent is an authoritative intent, its action field and data type field are sent to the action filter and the data type filter, respectively. If the intent is a naïve intent, its action field and data type field are sent to the scoring component and the data type filter, respectively. The data type filter does the job of data type matching. It filters out services whose data type is not matched with the intent services. The action filter looks for the services whose Intents advertisement actions are the same as the intent action. The scoring component takes the responsibility for scoring and ranking the services based on the adaptive approaches presented in Section 4.3 and the predefined evaluation measure in the settings file. In addition to the intent processor and service port, another four modules are also included in the design. They are the Intents advertisement CRUD (Create, Retrieve, Update, and Delete) module, the group list CRUD module, the settings CRUD module, and the Intents advertisement index controller module. The three CRUD modules are the bridges between their corresponding user interfaces and the underlying data stores. They execute the instructions to create, retrieve, update, and delete the store entries. The Intents advertisement index controller maintains the Intents advertisement index. The controller collects Intents advertisements and other service descriptive information from different sources based on the user configurations in the settings file and creates the index.

- **Presentation layer.** The Web view module renders the user interfaces of Web applications and services including parsing and executing their HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) and JavaScript. During the process of parsing the Web content, the Web view module takes the responsibility for locating the controls in Web applications which are associated with an intent. When an end user executes one of the controls and triggers its intent, the Web view module should capture the intent and send it down to the application layer. In addition to the Web view module, the presentation layer also has three user interface (UI) modules**.** The intent advertisement management UI module helps end users manage their own intent advertisement registry. With the interface, users are able to add, view, and remove intent advertisements for their private registry. Similarly, the interest group management UI module helps users manage their own interest groups. With its help, users are capable of adding, viewing, and removing their own interest groups. Last but not least, the settings UI module is used to configure the settings file.

This section presents only the user agent in design. It can be implemented in various forms. For instance, the user agent may be developed as an extension or plugin to a full-fledged browser such as Chrome or Firefox. It may also be developed as an independent application on a personal computer (PC) or mobile device.

5.2 Prototype

5.2.1 Implementation

We developed a user agent implementation on Android. It is an extension of our previous work presented in [Zheng et al., 2013]. Table 5.1 lists the major development libraries or tools for each layer.

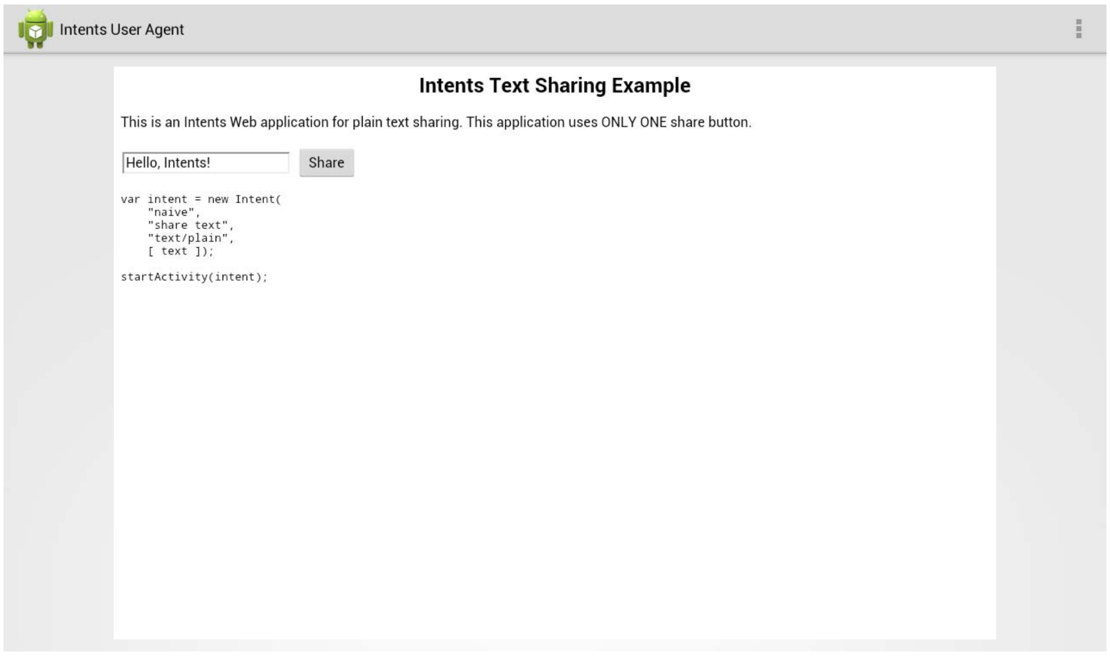
Table 5.1: Development kits for the user agent prototype

Layer	Development Kit
Presentation Layer	Android API
Application Layer	Android API
Data Layer	SQLite, Apache Lucene, file

At the data layer, the SQLite database is used to store private Intents advertisements. Apache Lucene is applied to build an index on the mobile device. Other system settings and the group list are stored in plain files. The application and presentation layer modules are implemented in the API provided by Android Software Development Kit (SDK). The Web view is implemented by the class "`android.webkit.WebView`" in the Android API.

5.2.2 Demonstration

Figure 5.2 demonstrates a typical text sharing sample in Intents. It can be compared with Figure 3.15 to see the advantages of Intents in content sharing applications.



This is an Intents Web application for plain text sharing. This application uses ONLY ONE share button.

```
var intent = new Intent(
    "naive",
    "share text",
    "text/plain",
    [ text ]);

startActivity(intent);
```

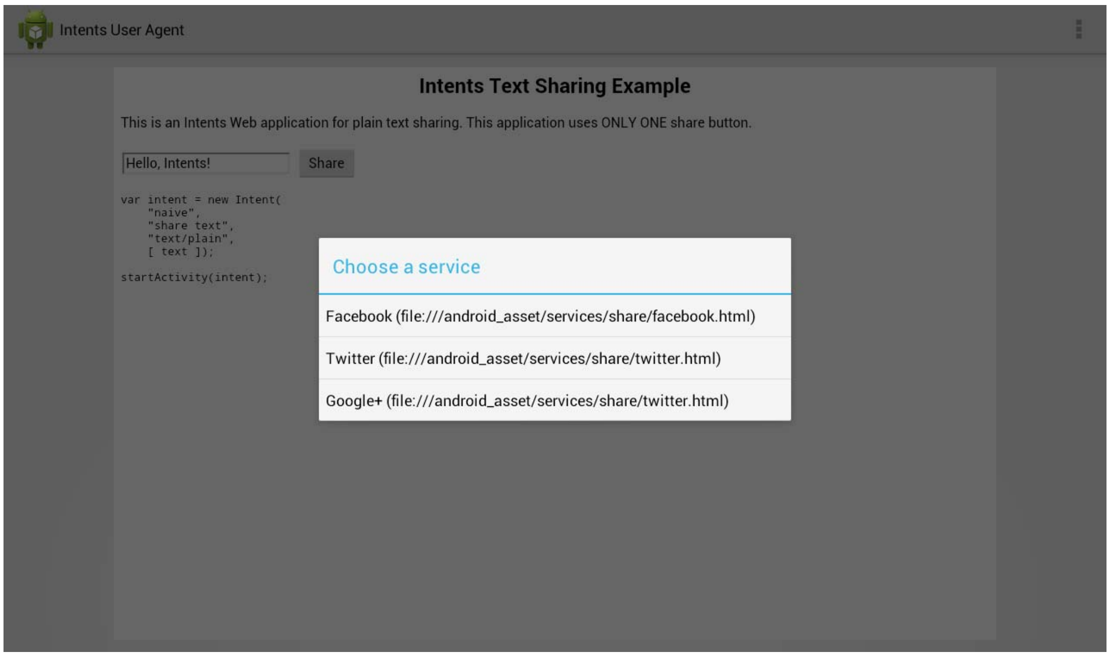
Figure 5.2: Intents text sharing example

In Figure 5.2, the Intents-supported application is rendered in the Web view of the user agent. Instead of adopting a set of sharing buttons, the application uses only one sharing button. The click event of the button is associated with an event listener function written in JavaScript. The function only executes the instructions in Figure 5.3.

```
var intent = new Intent(“naive”,
   “share text”,
   “text/plain”,
   [text]);
startActivity(intent);
```

Figure 5.3: Instructions to trigger an intent in JavaScript

The JavaScript code creates an intent object with its fields of intent type, action, data type, and data value. Then the “`startActivity(intent)`” instruction sends the intent to the user agent. The user agent generates a list of candidate services as in Figure 5.4.


Intents User Agent

Intents Text Sharing Example

This is an Intents Web application for plain text sharing. This application uses ONLY ONE share button.

```
var intent = new Intent(
    "naive",
    "share text",
    "text/plain",
    [ text ]);

startActivity(intent);
```

Choose a service

Facebook (file:///android_asset/services/share/facebook.html)

Twitter (file:///android_asset/services/share/twitter.html)

Google+ (file:///android_asset/services/share/twitter.html)

Figure 5.4: Candidate services for the example in Figure 5.3

Each candidate service entry is composed of a service title and service identifier. Currently in the implementation, all the sample Intents services and applications are embedded into the user agent Android application for demonstration purposes. Thus the identifiers are in the form of file paths.

The candidate services are generated from the private Intents registry of the user agent so they are totally personalized. If a service is selected by end users, for instance, Twitter, it continues the workflow and completes the action as in Figure 5.5.

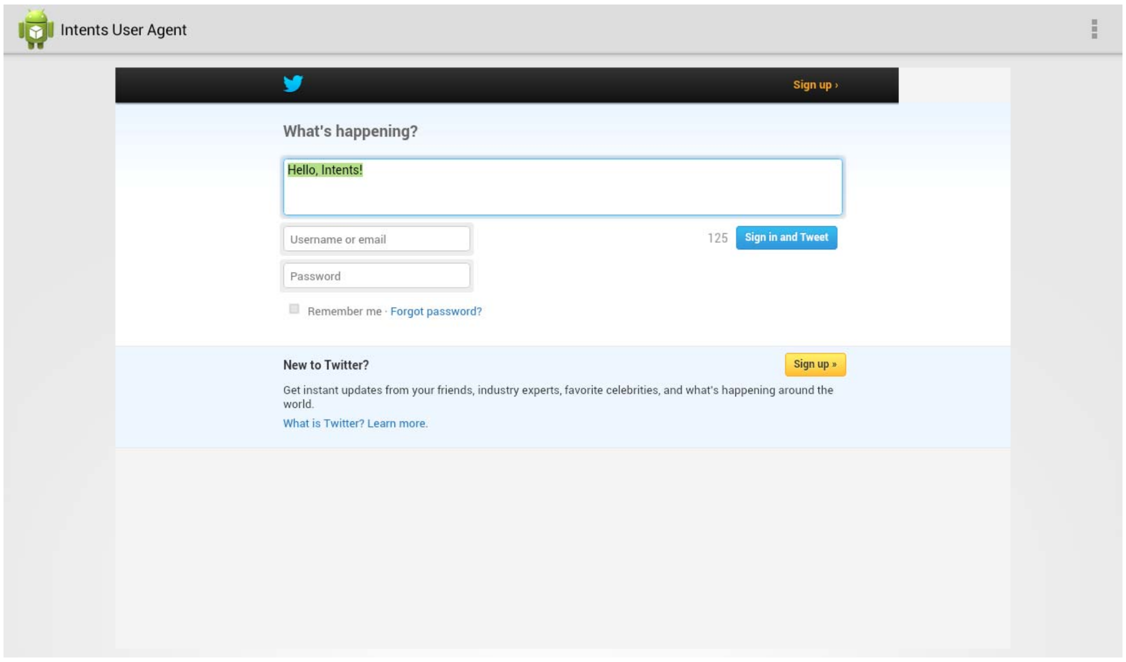

Intents User Agent
Sign up ›
What's happening?
Hello, Intents!
Username or email
125
Sign in and Tweet
Password
Remember me · Forgot password?
New to Twitter?
Sign up »
Get instant updates from your friends, industry experts, favorite celebrities, and what's happening around the world.
What is Twitter? Learn more.

Figure 5.5: Selection of the Twitter application and continue the sharing task

5.3 User Agent Application: Integration of Web and Native Applications on Mobile Devices

In recent years, mobile devices including smart phones and tablets have prevailed over traditional PCs in popularity. Applications developed on mobile devices can be either native or Web-based. A Web application is an application with an UI entry that runs in a Web browser and is in compliance with common Web standards. In contrast, native applications are developed specifically for a platform. Both Web and native applications have their own irreplaceable characteristics. Thus it is of substantial significance to explore the interoperability of the two kinds of applications. This section will present how to use Intents and the Intents user agent to facilitate the integration of Web applications and native applications

5.3.1 Motivation for the Integration

When planning an application on mobile devices, it is often difficult to decide to make the application native or Web-based. Both Web and native applications have advantages and disadvantages in terms of platform independence, maintenance, performance, and device feature exploitation. A comparison of these two kinds of applications can be summarized as follows:

- **Platform independence.** For Web applications, even though they are running within different browsers, most browsers are designed to support almost the same Web technology standards (HTML, CSS and JavaScript). Thus Web application projects can be easily ported to different mobile platforms without too much effort. On the contrary, native applications running on separate mobile platforms encounter problems caused by the many and various mobile operating systems (OS) such as Apple, Android, Blackberry, Windows, and Symbian. Each platform requires a different development skill set which is distressing and troublesome for application developers. Table 5.2 lists the detailed skills by Charland and Leroux [Charland and Leroux, 2011]. Therefore, individual developers or startups may only be able to support their applications in one or two platforms. As a result, a limit is imposed on spreading the application. On the other hand, giant companies are required to invest more to make their applications support most of the existing mobile platforms.

Table 5.2: Mobile platform and their required skill set

Mobile OS	Skill Set
Apple iOS	C, Objective C
Google Android	Java
RIM BlackBerry	Java
Symbian	C, C++, Python, HTML/CSS/JS
Window 7 Phone	.NET
HP Palm webOS	HTML/CSS/JS
MeeGo	C, C++, HTML/CSS/JS
Samsung bada	C++

- **Maintenance.** Native application developers have to publish updates simultaneously into all application stores for the platforms on which their applications are running. This requirement is extremely hard and sometimes they may publish the latest update onto the most popular platforms first. The result of the action is fragmentation with platforms possibly keeping different application versions. In addition, the maintenance of native applications on all platforms is also very expensive and users have to install their updates manually. However, the major maintenance of Web applications only happens on the servers where Web applications are kept. Application developers just need to update application files on the server side and it automatically takes effect when users open the applications again in their browsers.

- **Performance.** Native applications are usually developed in programming languages which are supported by a specific platform. Thus native applications can be compiled and optimized to the platform which is able to achieve a much faster response time. In contrast, Web applications are downloaded and interpreted at runtime which slow their running speed. Even though caching technology is applied and the JavaScript engines on some platforms have been enhanced, the whole performance of Web applications falls far behind compared to native applications, especially for heavily resource-consuming applications such as games and videos.

- **Device features.** Native applications have the advantage of employing device sensors including the accelerometer and the compass to enhance their functionalities. Besides, they may employ platform-dependent UI elements and controls which are capable of providing a good user experience. Web applications, however, are limited in leveraging device-dependent features. Currently, some third-party libraries like PhoneGap[1] enable Web applications to use device units including cameras and sensors. But they still could not match native applications in terms of device features and user experience.

Now that both native and Web applications are advantageous in different aspects, integrating them provides a possibility to make the most of mobile devices. Specifically, Android native applications are developed revolving around a mechanism named *Android Intents* which, in this work, provides inspiration for Intents. Therefore, the nature of

[1] http://phonegap.com/

Intents should be that it can be leveraged to integrate Web applications and Android native applications.

Android Intents indicates the `android.content.Intent` class in Android SDK and the mechanism revolving around it for communication in the Android operating system. Android components communicate with each other through its instance transmission. In order to make a difference between intents in this work and Android `Intent` instances, the latter type of intents are called *Android intents*. Although, in this work, Intents partly originates from Android Intents, their intent data structures are different. Table 5.3 compares the fields of intents and Android intents.

Table 5.3: Comparison between intents and Android intents

Field	Intent	Android Intent
type	Yes	No
action	Yes	Yes
data	Yes	Yes
data type	Yes	Yes
extra	No	Yes

From Table 5.3 it is seen that Android intents have no intent *type* field but they added an *extra* field to accommodate additional information. On the other hand, the Android intents *data* field only supports URI-like data which is totally different from the intents *data* field. Thus integrating Web and Android applications requires a method for the conversion between intents and Android intents.

In the next two subsections, Intents and the Intents user agent are applied to two situations for the integration of Web applications and Android native applications. One is calling Android components from Web applications. The other is calling Web services from Android applications. The conversion between intents and Android intents will be presented in the two situations, respectively.

5.3.2 Web Applications Depending on Android Components

Calling Android components in Web applications requires creating *Android intents* in the form of *intents*. Then the user agent can identify the intent-formed Android intent and extract its fields to create a real Android intent. After that the real Android intent is sent to the underlying Android system for further processing. If the Android intent is explicit, the expected component is executed directly. On the other hand, if the Android intent is an implicit intent, Android may generate a list of candidate components to the end user. Then the end user chooses a component to continue his/her task. Figure 5.6 shows how to map the fields of *Android intents* into the fields of *intents*.

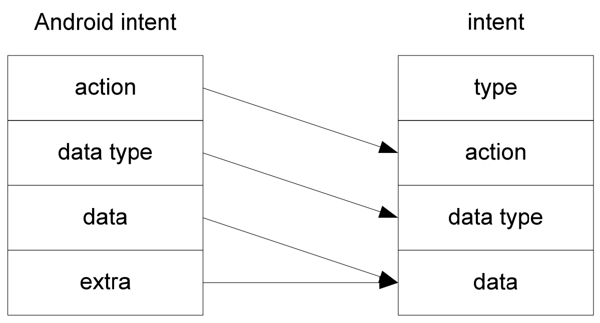


Figure 5.6: Field mapping from Android intents to intents

In Figure 5.6, the Android intent action and data type fields are mapped to the intent action and data type fields, respectively. The Android intent data and extra fields are together put into the intent data field. Moreover, the intent type field is filled with “android” to let the user agent know it is an intent-formed Android intent.

Figure 5.7 shows the scheme to combine the Android intent data and extra fields.

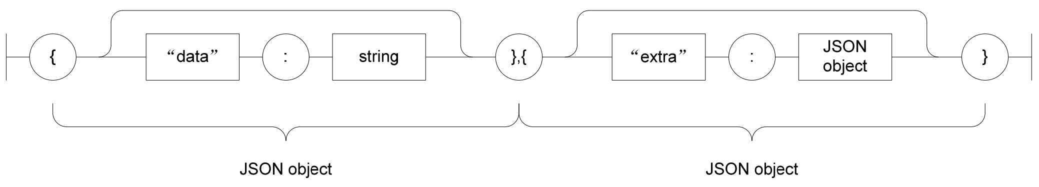
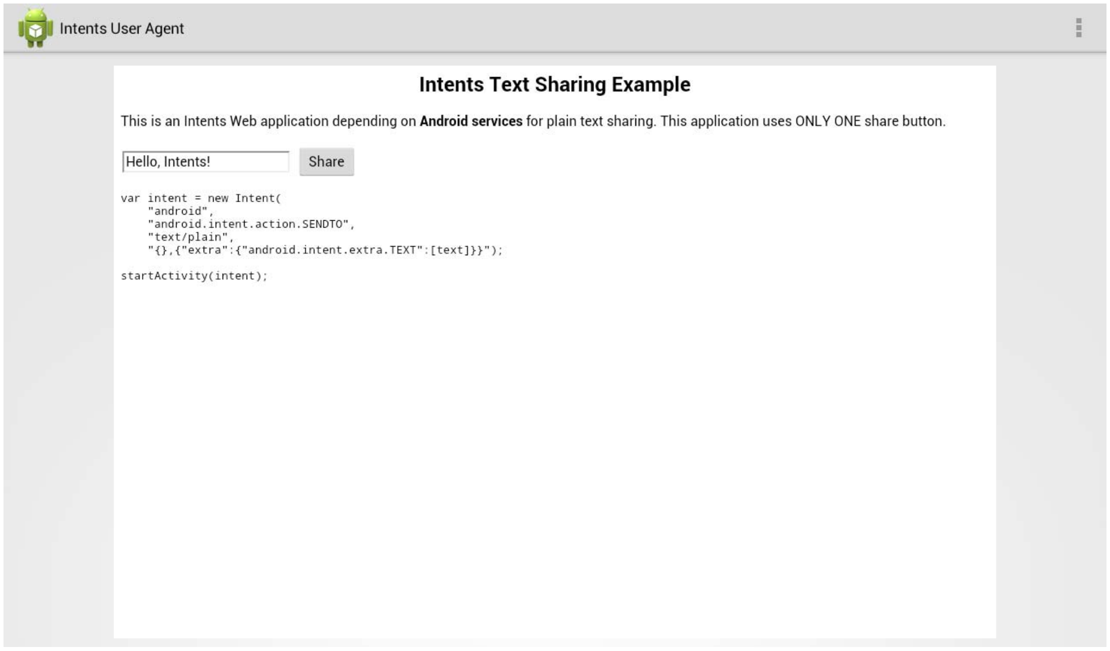


Figure 5.7: Scheme to combine the Android intent data and extra fields

In Figure 5.7, two JSON objects are joined together with a comma as the separator. The first JSON object is for the data field. Its name/value pair has a “data” name and a string value. The latter JSON object is for the extra field. Its name/value pair has an “extra” name and a JSON object which is used for multiple extra settings in Android intents. The upper path of the two JSON objects means both the data field and the extra field are optional and can be set to null.

Figure 5.8 demonstrates an example of calling Android components in Web applications through Intents.



Intents Text Sharing Example

This is an Intents Web application depending on **Android services** for plain text sharing. This application uses ONLY ONE share button.

Hello, Intents!

```
var intent = new Intent(
    "android",
    "android.intent.action.SENDTO",
    "text/plain",
    "{},{"extra":{"android.intent.extra.TEXT":[text]}}");

startActivity(intent);
```

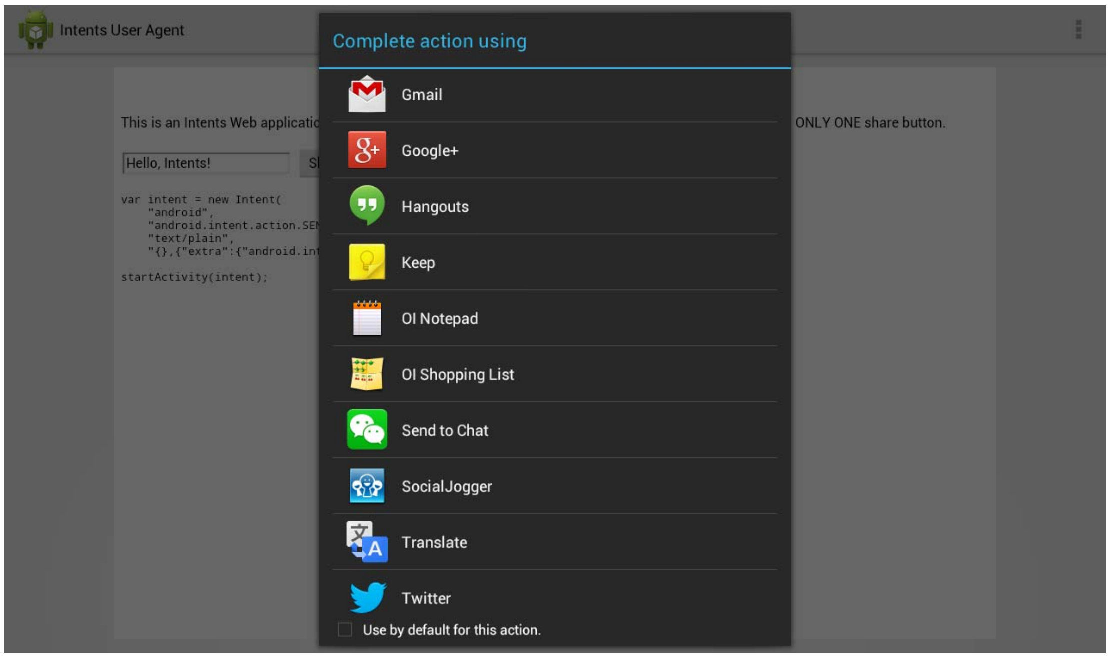
Figure 5.8: Web text sharing example by calling Android components

The example also adopts the text sharing scenario. Its JavaScript code to create Android intents is shown as in Figure 5.9.

```
var intent = new Intent(“android”,
   “android.intent.action.SENDTO”,
   “text/plain”,
   “{},{"extra":{"android.intent.extra.TEXT":[text]}}”);
startActivity(intent);
```

Figure 5.9: Instructions to invoke Android components in the Web application in Figure 5.8

In Figure 5.9, the text to be shared is put into the *extra* field. When a user agent receives the intent, it is first identified as an intent-formed Android intent. Then the user agent extracts the intent fields, creates an `android.content.Intent` instance, and sends the instance to the underlying Android operating system. Because the Android intent is implicit, the operating system generates a candidate list as shown in Figure 5.10.



```
var intent = new Intent(
    "android",
    "android.intent.action.SEN
    "text/plain",
    "{},{"extra":{"android.int

startActivity(intent);
```

Figure 5.10: Candidate Android components for the Web application example in Figure 5.8

Then the end user is able to choose an Android component to complete the text sharing task.

5.3.3 Native Applications Depending on Web Services

Calling Web services in Android applications requires creating intents in the form of Android intents. Then an Intents helper module transforms them into real intents which can be recognized by user agents. After that user agents accept and process the intents. The whole workflow is shown in Figure 5.11.

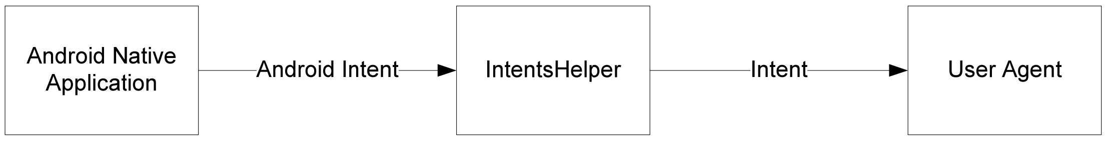


Figure 5.11: Flow of calling Web services in Android applications

Figure 5.12 shows how to map the Android intent fields into the intent fields. The intent action and data type fields are mapped into the Android intent action and data type fields, respectively. The intent type and data fields are together mapped into the Android intent extra field.

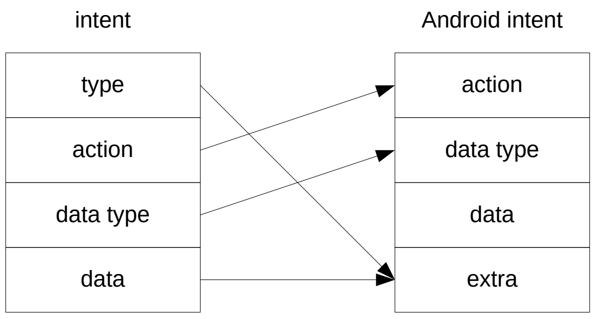
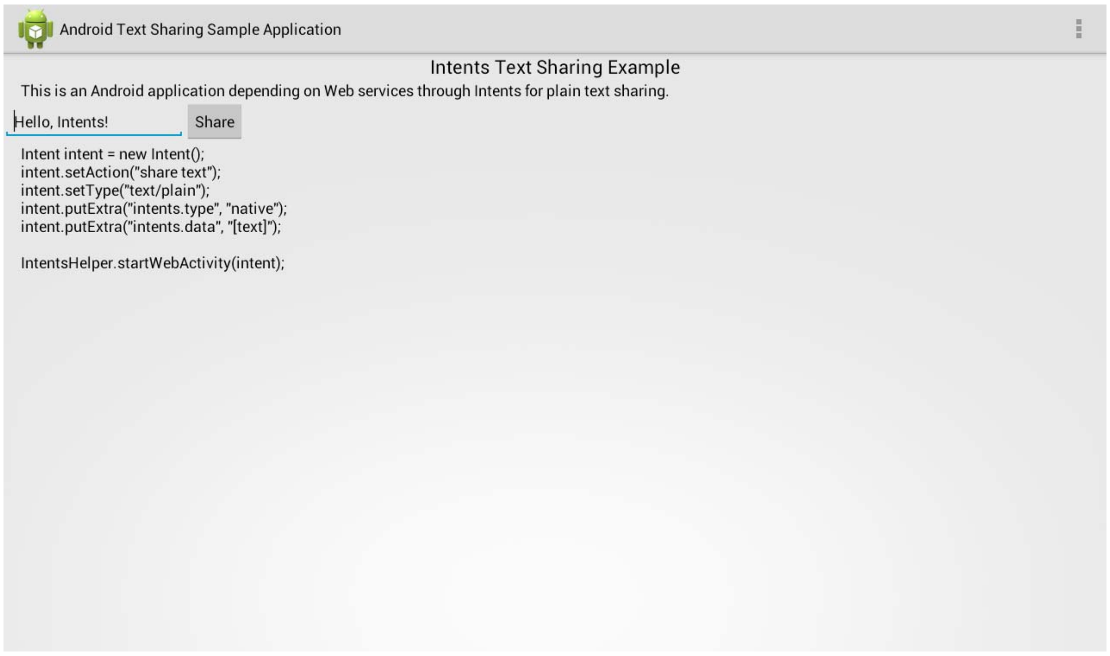


Figure 5.12: Field mapping from intents to Android intents

Because the Android intent data field only supports a URI type, it is not suitable for storing intent data. On the other hand, the extra field is able to accommodate multiple name/value pairs. Thus both the intent type and data fields are mapped into the Android intent extra field.

Figure 5.13 demonstrates an example of calling Web services in Android applications through Intents.



Intents Text Sharing Example

This is an Android application depending on Web services through Intents for plain text sharing.

Hello, Intents!

```
Intent intent = new Intent();
intent.setAction("share text");
intent.setType("text/plain");
intent.putExtra("intents.type", "native");
intent.putExtra("intents.data", "[text]");

IntentsHelper.startWebActivity(intent);
```

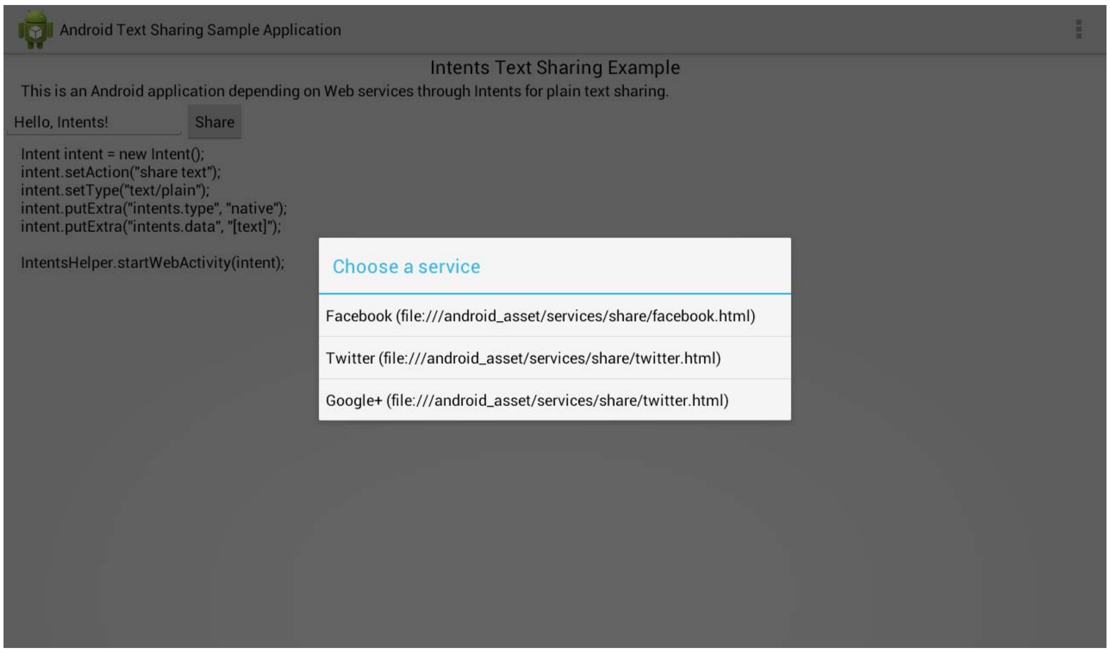
Figure 5.13: Android text sharing example by calling Web services

The example also adopts the text sharing scenario. The code to create intents is written in Java as shown in Figure 5.14.

```
Intent intent = new Intent();
intent.setAction(“share text”);
intent.setType(“text/plain”);
intent.putExtra(“intents.type”, “naive”);
intent.putExtra(“intents.data”, “[text]”)
IntentsHelper.startWebActivity(intent);
```

Figure 5.14: Instructions to invoke Web services in the Android application in Figure 5.13

The Android application creates an intent in the form of Android intents and sends it to the Intents helper. Then the Intents helper extracts its fields and creates a real intent. After that the intent is sent to the user agent for further processing. Because the intent is naïve, the user agent generates a list of candidate services as shown in Figure 5.15.



Intents Text Sharing Example

This is an Android application depending on Web services through Intents for plain text sharing.

Hello, Intents!

```
Intent intent = new Intent();
intent.setAction("share text");
intent.setType("text/plain");
intent.putExtra("intents.type", "native");
intent.putExtra("intents.data", "[text]");

IntentsHelper.startWebActivity(intent);
```

Facebook (file:///android_asset/services/share/facebook.html)

Twitter (file:///android_asset/services/share/twitter.html)

Google+ (file:///android_asset/services/share/twitter.html)

Figure 5.15: Candidate Web services for the Android application example in Figure 5.13

Then the end user is able to choose a Web service to complete the text sharing task.

5.4 Summary

This chapter presents a design for Intents user agents which play an important role in Intents. A proof-of-concept prototype is developed on Android and a text sharing application is demonstrated.

This chapter also explores applying Intents and the proposed Intents user agent to the integration of Web applications and native applications on mobile devices. Two attempts are demonstrated on the developed user agent. One is to call Android components on Web applications. The other is to invoke Web services on Android applications.

Chapter 6

6 Conclusion and Future Work

In this chapter, we will conclude the research work and discuss future research directions.

6.1 Conclusion

Developing an effective and efficient technique for service discovery and integration is a long-standing challenge. Although significant efforts have been devoted to this area, most of them are based on the ternary participant classification for the Web service architecture which only takes into consideration the involvement of service providers, service brokers, and the application developers in service consumers. The application end user participation is usually ignored.

This thesis presents an innovative service discovery and integration approach named Intents which is inspired by two industrial protocols: Android Intents and Web Intents. The approach is characterized by allowing application end users to participate in the process of service seeking and provides a new direction for service discovery and integration. The major contributions of this work can be summarized as follows:

- **Proposed the Intents approach at the conceptual level.** The proposed Intents approach is at the conceptual level so it has a strong capability in addressing generic problems. Our approach not only inherits the innovations of Android Intents and Web Intents but also extends them in terms of the Intents architecture, intent data structure, intent types, Intents advertisements, the intent resolution process, and Intents service creation methods.
- **Examined the process of intent resolution and developed an adaptive intent resolution approach.** Intent resolution is a critical process in Intents and the naïve intent resolution process asks for a similarity model. We formulated a similarity model and constructed an optimization problem for seeking the best similarity model. Then we conducted an empirical study of the problem. Based on

the results and analysis of the empirical study, an adaptive intent resolution approach has been developed.

- **Presented a design and implementation of the Intents user agent and applied Intents and the implemented user agent to the integration of Web applications and native applications on mobile devices.** The Intents user agent is the most significant component in the Intents architecture which takes the responsibilities including collecting and managing Intents advertisements, rendering Web applications, intent resolution, and communicating with remote services. We proposed a design and implementation for user agents in this thesis. In addition, with the development of mobile devices, integrating Web applications and native applications on mobile devices is in great demand. This thesis makes an attempt by applying Intents and the proposed user agent to the integration of Web applications and Android native applications.

6.2 Future Work

Intents is an innovative framework which opens up a new direction in the research area of service discovery and integration. In the research directions related to this thesis, the following future work is envisioned:

- **Enrich Intents services and Intents-based Web applications.** Currently there are only a limited number of Intents services and Intents-based Web applications for research purposes. In the long run, more mature Intents services and Intents-based Web applications should be developed to exploit the advantages of Intents.
- **Continue improving the adaptive intent resolution process.** The dataset applied in the empirical study in this work is comparatively small. More data is needed to examine the intent resolution process. At the same time, more IR model implementations and similarity model templates should be explored for seeking the best similarity model in different conditions. In addition, machine learning techniques can be applied to help formulate the similarity model template set.
- **Apply semantic integration techniques such as ontologies to the organization of data types.** Intents currently adopts the Internet media type for the data type

field. It works well for simple-parameter Web services, i.e., Web services that have one basic input parameter. As for complex parameters, JSON or XML is applied. Thus it requests service providers to provide more information on how to construct input and application developers to form the data in compliance with such information. As a future work, ontologies will be used to organize data types so that each input of data conforms to a concept in a global ontology.

References

[Al-Jaroodi and Mohamed, 2012] J. Al-Jaroodi and N. Mohamed, “Service-oriented Middleware: a Survey,” *Journal of Network and Computer Applications*, Vol. 35, No. 1, pp. 211–220, 2012.

[Al-Masri and Mahmoud, 2007] E. Al-Masri and Q. Mahmoud, “QoS-based Discovery and Ranking of Web Services,” In *Proceedings of 16th International Conference on Digital Object Identifier* (ICCCN 2007), Honolulu, HI, USA, 2007, pp. 529–534.

[Al-Masri and Mahmoud, 2009] E. Al-Masri and Q. Mahmoud, “Understanding Web Service Discovery Goals,” In *Proceedings of 2009 IEEE International Conference on Systems, Man, and Cybernetics* (IEEE SMC 2009), San Antonio, Texas, USA, 2009, pp. 3714–3719.

[Baeza-Yates and Ribeiro-Neto, 1999] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, Addison Wesley, 1999.

[Bansal and Vidal, 2003] S. Bansal and J. Vidal, “Matchmaking of Web Services Based on the DAML-S Service Model,” In *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems* (AAMAS 2003), Melbourne, Australia, 2003, pp. 926–927.

[Berners-Lee, 1992] T. Berners-Lee, “World-Wide Web: The Information Universe,” *Electronic Networking*, Vol. 2, No. 1, pp. 52–58, 1992.

[Bishop, 2006] C. Bishop, Pattern *Recognition and Machine Learning (Information Science and Statistics)*, Springer, 2006.

[Bouguila, 2008] N. Bouguila, “Clustering of Count Data Using Generalized Dirichlet Multinomial Distributions,” *IEEE Transactions on Knowledge and DataEngineering*, Vol. 20, No. 4, pp. 462–474, 2008.

[Bray, 2002] T. Bray, “Internet Media Type Registration, Consistency of Use,” *W3C*, September 4, 2002, Accessed February 8, 2014, http://www.w3.org/2001/tag/2002/0129-mime.

[Buckley et al., 1994] C. Buckley, G. Salton, J. Allan, and A. Singhal, “Automatic Query Expansion Using SMART: TREC 3,” In *Proceedings of the 3rd Text REtrieval Conference* (TREC-3), Gaithersburg, MD, USA, 1994, pp. 69–80.

[Canali et al., 2013] C. Canali, M. Colajanni, and R. Lancellotti, “Algorithms for Web Service Selection with Static and Dynamic Requirements,” *Service Oriented Computing and Applications*, Vol. 7, No. 1, pp. 43–57, 2013.

[Casella and Berger, 2001] G. Casella and R. Berger, *Statistical Inference*, Duxbury Press, 2001

[Chan et al., 2012] N. Chan, W. Gaaloul, and S. Tata, “A Recommender System based on Historical Usage Data for Web Service Discovery,” *Service Oriented Computing and Applications*, Vol. 6, No. 1, pp.

51–63, 2012.

[Charland and Leroux, 2011] A. Charland and B. Leroux, “Mobile Application Development: Web vs. Native,” *Communications of the ACM*, Vol. 54, No. 5, pp. 49–53, 2011.

[Chappell and Jewell, 2002] D. Chappell and T. Jewell, *Java Web Services*, O’Reilly, 2002.

[Chen and Mcleod, 2006] A. Chen and D. McLeod, “Collaborative Filtering for Information Recommendation Systems,” In *Encyclopedia of E-Commerce, E-Government, and Mobile Commerce*, edited by M. Khosrow-Pour and M. Khosrowpour, pp. 118–123, Idea Group Publishing, 2006.

[Chen et al., 2006] H. Chen, Y. Wang, H. Wang, Y. Mao, J. Tang, C. Zhou, A. Yin, and Z. Wu. “Towards a Semantic Web of Relational Databases: A Practical Semantic Toolkit and an In-Use Case from Traditional Chinese Medicine,” In *Proceedings of the 5th International Semantic Web Conference* (ISWC 2006), Athens, GA, USA, 2006, pp. 750–763.

[Cristianini and Shawe-Taylor, 2000] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, 2000.

[Crouch and Yang, 1992] C. Crouch and B. Yang, “Experiments in Automatic Statistical Thesaurus Construction,” In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 1992), Copenhagen, Denmark, 1992, pp. 77–88.

[Dempster et al., 1977] A. Dempster, N. Laird, and D. Rubin, “Maximum Likelihood from Incomplete Data via the EM Algorithm,” *Journal of the Royal Statistical Society, Series B*, Vol. 39, No. 1, pp. 1–38, 1977.

[Dong et al., 2004] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, “Similarity Search for Web Services,” In *Proceedings of the 13th International Conference on Very Large Data Bases* (VLDB 2004), Toronto, Canada, 2004, pp. 372–380.

[Fang, 2007] H. Fang, “An Axiomatic Approach to Information Retrieval,” PhD Dissertation, University of Illinois at Urbana-Champaign, 2007.

[Fang et al., 2004] H. Fang, T. Tao, and C. Zhai, “A Formal Study of Information Retrieval Heuristics,” In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 2004), Sheffield, United Kingdom, 2004, pp. 49–56.

[Fielding, 2000] R. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” PhD Dissertation, University Of California, Irvine, 2000.

[Fielding and Taylor, 2002] R. Fielding and R. Taylor, “Principled Design of the Modern Web Architecture,” *ACM Transactions on Internet Technology*, Vol. 2, No. 2, pp. 115–150, 2002.

[Fuggetta et al., 1998] A. Fuggetta; G. P. Picco, and G. Vigna, “Understanding Code Mobility,” *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 342–361, 1998.

[Guarino et al., 2009] N. Guarino, D. Oberle, and S. Staab, "What Is an Ontology?," In *Handbook on Ontologies, International Handbooks on Information Systems* (2nd ed), edited by S. Staab and R. Studer, pp. 1–17, Springer, 2009.

[Gray and Reuter, 1992] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1992.

[Halevy, 2001] A. Halevy, "Answering Queries Using Views: a Survey," *The VLDB Journal*, Vol. 10, No. 4, pp. 270–294, 2001.

[Hang et al., 2012] C. Hang, A. Kalia, and M. Singh, "Behind the Curtain: Service Selection via Trust in Composite Services," In *Proceedings of the IEEE 19th International Conference on Web Services* (ICWS 2012), Honolulu, Hawaii, USA, 2012, pp. 9–16.

[Hatzi et al., 2012] O. Hatzi, G. Batistatos, M. Nikolaidou, and D. Anagnostopoulos, "A Specialized Search Engine for Web Service Discovery," In *Proceedings of the IEEE 19th International Conference on Web Services* (ICWS 12), Honolulu, Hawaii, USA, 2012, pp. 448–455.

[Herlocker et al., 2004] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl, "Evaluating Collaborative Filtering Recommender Systems," *ACM Transactions on Information Systems*, Vo. 22, No. 1, pp. 5–53, 2004.

[Jewell and Chappell, 2002] T. Jewell and D. Chappell, *Java Web Services*, O'Reilly, 2002.

[Jordan, 1998] M. Jordan, *Learning in Graphical Models (Adaptive Computation and Machine Learning)*, MIT Press, 1998.

[Kahneman and Tversky, 1979] D. Kahneman and A. Tversky, "Prospect Theory: An Analysis of Decision under Risk," *Econometrica*, Vol. 47, No. 2, pp. 263–292, 1979.

[Kossmann et al., 2002] D. Kossmann, F. Ramsak and S. Rost, "Shooting Stars in the Sky: an Online Algorithm for Skyline Queries," In *Proceedings of the 28th International Conference on Very Large Data Bases* (VLDB 2002), Hong Kong, China, 2002, pp. 275–286.

[Lafferty and Zhai, 2001] J. Lafferty and C. Zhai, "Document Language Models, Query models, and Risk Minimization for Information Retrieval," In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 2001), New Orleans, LA, USA, 2001, pp. 111–119.

[Li et al., 2011] M. Li, J. Zhao, L. Wang, S. Cai, and B. Xie, "CoWS: An Internet-Enriched and Quality-Aware Web Services Search Engine," In *Proceedings of the IEEE 18th International Conference on Web Services* (ICWS 2011), Washington, DC, USA, 2011, pp. 419–427.

[Liu et al., 2010] F. Liu, Y. Shi, J. Yu, T. Wang, and J. Wu, "Measuring Similarity of Web Services Based on WSDL," In *Proceedings of the IEEE 17th International Conference on Web Services* (ICWS

2010), Miami, Florida, USA, 2010, pp.155–162.

[Manes, 2003] A. Manes, *Web Services: A Manager's Guide*, Addison Wesley, 2003.

[Manning et al., 2008] C. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*, Cambridge University Press, 2008.

[McCandless et al., 2010] M. McCandless, E. Hatcher, and O. Gospodnetic, *Lucene in Action* (2nd ed.), Manning, 2010.

[Mehdi et al., 2012] M. Mehdi, N. Bouguila, and J. Bentahar, “Trustworthy Web Service Selection Using Probabilistic Models,” In *Proceedings of the IEEE 19th International Conference on Web Services* (ICWS 2012), Honolulu, Hawaii, USA, 2012, pp. 17–24.

[Meek et al., 2007] D. Metzler, S. Dumais, and C. Meek, “Similarity Measures for Short Segments of Text,” In *Proceedings of the 29th European Conference on IR Research* (ECIR 2007), Rome, Italy, 2007, pp. 16–27.

[Miller, 1995] G. Miller, “WordNet: a Lexical Database for English,” *Communications of the ACM*, Vol. 38, No. 11, pp. 39–41, 1995.

[Mitra et al., 1998] M. Mitra, A. Singhal, and C. Buckley, “Improving Automatic Query Expansion,” In *Proceedings of the 21st Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 1998), Melbourne, Australia, 1998, pp. 206–214.

[Mobedpour and Ding, 2013] D. Mobedpour and C. Ding, “User-centered Design of a QoS-based Web Service Selection System,” *Service Oriented Computing and Applications*, Vol. 7, No. 2, pp. 117–127, 2013.

[Munkres, 2000] J. Munkres, *Topology* (2nd ed.), Pearson, 2000.

[Newcomer, 2002] E. Newcomer, *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Addison-Wesley, 2002.

[Paolucci et al., 2002] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara, “Semantic Matching of Web Services Capabilities,” In *Proceedings of the First International Semantic Web Conference on The Semantic Web* (ISWC 2002), Seattle, Washington, USA, 2002, pp. 333–347.

[Papadias et al., 2003] D. Papadias, Y. Tao, G. Fu, and B. Seeger, “An Optimal and Progressive Algorithm for Skyline Queries,” In *Proceedings of the 22th ACM SIGMOD International Conference on Management of Data* (SIGMOD 2003), San Diego, CA, USA, 2003, pp. 467–478.

[Plebani and Pernici, 2009] P. Plebani and B. Pernici, “URBE: Web Service Retrieval Based on Similarity Evaluation,” *IEEE Transactions on Knowledge and Data Engineering*, Vol. 21, No. 11, pp. 1629–1642, 2009.

[Ponte and Croft, 1998] J. Ponte and W. Croft, “A Language Modeling Approach to Information

Retrieval," In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 1998), Melbourne, Australia, 1998, pp. 275–281.

[Popescul et al., 2001] A. Popescul, L. Ungar, D. Pennock, and S. Lawrence, "Probabilistic Models for Unified Collaborative and Content-based Recommendation in Sparse-Data Environments," In *Proceedings of the 17th International Conference on Uncertainty in Artificial Intelligence* (UAI 2001), San Francisco, CA, USA, 2001, pp. 437–444.

[Porter, 1980] M. Porter, "An Algorithm for Suffix Stripping," *Program*, Vol. 14, No. 3, pp. 130–137, 1980.

[Qiu and Frei, 1993] Y. Qiu and H. Frei, "Concept-based Query Expansion", In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 1993), Pittsburgh, PA, USA, 1993, pp. 160–169.

[Ran, 2003] S. Ran, "A Model for Web Services Discovery with QoS," *ACM SIGecom Exchanges*, Vol. 4, No. 1, pp. 1–10, 2003.

[Richardson and Ruby, 2007] L. Richardson and S. Ruby, *Restful Web Services*, O'Reilly Media, 2007.

[Rittenberg and Tregarthen, 2009] L. Rittenberg and T. Tregarthen, *Principles of Macroeconomics*, Flat World Knowledge, 2009.

[Robertson and Jones, 1976] S. Robertson and K. Jones, "Relevance Weighting of Search Terms," *Journal of the American Society for Information Science*, Vol. 27, No. 3, pp. 129–146, 1976.

[Robertson et al., 1995] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford, "Okapi at TREC-3," In *Proceedings of the 3rd Text REtrieval Conference* (TREC-3), Gaithersburg, MD, USA, 1994, pp.109–126.

[Robertson et al., 1999] S. Robertson, S. Walker, and M. Hancock-Beaulieu, "Okapi at TREC-7," In *Proceedings of the 7th Text REtrieval Conference* (TREC-7), Gaithersburg, MD, USA, 1999, pp. 253–264.

[Sahami and Heilman, 2006] M. Sahami and T. Heilman, "A Web-Based Kernel Function for Measuring the Similarity of Short Text Snippets," In *Proceedings of the 15th International Conference on World Wide Web* (WWW 2006), Edinburgh, Scotland, 2006, pp. 377–386.

[Salton et al., 1975] G. Salton, A. Wong, and C. Yang, "A Vector Space Model for Automatic Indexing," *Communications of the ACM*, Vol. 18 No. 11, pp. 613–620, 1975.

[Salton and McGill, 1983] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.

[Schütze and Pedersen, 1997] H. Schütze and J. Pedersen, "A Cooccurrence-based Thesaurus and Two Applications to Information Retrieval," *Information Processing & Management*, Vol. 33, No. 3, pp.

307–318, 1997.

[Shi et al., 2012] C. Shi, D. Lin, and T. Ishida, “User-Centered QoS Computation for Web Service Selection,” In *Proceedings of the IEEE 19th International Conference on Web Services* (ICWS 12), Honolulu, Hawaii, USA, 2012, pp. 448–455.

[Si et al., 2013] H. Si, Z. Chen, Y. Deng, and L. Yu, “Semantic Web Services Publication and OCT-based Discovery in Structured P2P Network,” *Service Oriented Computing and Applications*, Vol. 7, No. 3, pp. 169–180, 2013.

[Singhal et al., 1996] A. Singhal, C. Buckley, and M. Mitra, “Pivoted Document Length Normalization,” In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 1996), Zurich, Switzerland, 1996, pp. 21–29.

[Tversky and Kahneman, 1992] A. Tversky and D. Kahneman, “Advances in Prospect Theory: Cumulative Representation of Uncertainty,” *Journal of Risk and Uncertainty*, Vol. 5, No. 4, pp. 297–323, 1992.

[Vaculin et al., 2008] R. Vaculin, H. Chen, R. Neruda, and K. Sycara, “Modeling and Discovery of Data Providing Services,” In *Proceedings of the IEEE 15th International Conference on Web Services* (ICWS 2008), Beijing, China, 2008, pp. 54–61.

[Voorhees, 1985] E. Voorhees, “Architectural Styles and the Design of Network-based Software Architectures,” PhD Dissertation, Cornell University, 1985.

[Voorhees, 1994] E. Voorhees, “The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval”, In *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 1994), Dublin, Ireland, 1994, pp. 61–69.

[Wang and Stroulia, 2003] Y. Wang and E. Stroulia, “Semantic Structure Matching for Assessing Web Service Similarity,” in *Proceedings of the 1st International Conference on Service Oriented Computing*, Trento, Italy, 2003, pp. 194–207.

[Wang et al., 2005] J. Wang, J. Xiao, C. Lam, and H. Li, “A Bipartite Graph Approach to Generate Optimal Test Sequences for Protocol Conformance Testing Using the Wp-Method,” In *Proceedings of the 12th Asia-Pacific Software Engineering Conference* (APSEC 2005), Taipei, Taiwan, China, 2005, pp. 307–316.

[Wolsey, 1998] L. Wolsey, *Integer Programming*. Wiley, 1998.

[Xu et al., 2011] K. Xu, Q. Yu, Q. Liu, J. Zhang, and A. Bouguettaya, “Web Service Management System for Bioinformatics Research: a Case Study,” *Service Oriented Computing and Applications*, Vol. 5, No. 1, pp. 1–15, 2011.

[Yao et al., 2012] L. Yao, Q. Sheng, A. Segev, and J. Yu, “Recommending Web Services via Combining Collaborative Filtering with Content-based Features,” In *Proceedings of the IEEE 20th International*

Conference on Web Services (ICWS 2013), Santa Clara, CA, USA, 2012, pp. 42–49.

[Yau and Yin, 2011] S. Yau and Y. Yin, "QoS-based Service Ranking and Selection for Service-based Systems," In *Proceedings of the IEEE 8th International Conference on Services Computing* (SCC 2011), Washington DC, USA, 2011, pp. 56–63.

[Zhai and Lafferty, 2001a] C. Zhai and J. Lafferty, "A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval," In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 2001), New Orleans, LA, USA, 2001, pp. 334–342.

[Zhai and Lafferty, 2001b] C. Zhai and J. Lafferty, "Model-based Feedback in the Language Modeling Approach to Information Retrieval," In *Proceedings of the 10th International Conference on Information and Knowledge Management* (CIKM 2001), Atlanta, Georgia, USA, 2001, pp. 403–410.

[Zheng et al., 2010] Z. Zheng, Y. Zhang, and M. Lyu, "Distributed QoS Evaluation for Real-World Web Services," In *Proceedings of the 2010 IEEE International Conference on Web Services* (ICWS 2010), Washington, D.C., USA, 2010, pp. 83–90.

[Zheng et al., 2013] C. Zheng, W. Shen, and H. Ghenniwa, "Design and Implementation of Intents User Agent," In *Proceedings of 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design* (IEEE CSCWD 2013), Whistler, BC, Canada, 2013, pp. 275–280.

Curriculum Vitae

Name:	Cheng Zheng
Post-secondary Education and Degrees:	Xian Jiaotong University Xian, Shaanxi, China 2003-2007 Bachelor's Degree
	Xian Jiaotong University Xian, Shaanxi, China 2007-2010 Master's Degree
	The University of Western Ontario London, Ontario, Canada 2010-2014 Ph.D.
Honours and Awards:	Western Graduate Research Scholarship - Engineering 2010-2014
	Xian Jiaotong University Outstanding Graduate Student Award 2007-2008
	Fuji Xerox Corporation Scholarship 2007-2008
	Xian Jiaotong University Outstanding Undergraduate Scholarship 2003-2007
	Xian Jiaotong University Outstanding Undergraduate Student 2004-2007
Related Work Experience	Research & Teaching Assistant The University of Western Ontario Sep 2010-Aug 2014
	Student Developer GSoC 2012 for Google Inc. May-Aug 2012
	Volunteering Student Worker National Research Council Canada May-Aug 2011
	Research Assistant

Shaanxi Key Lab of Satellite-Terrestrial Network Tech R&D
Dec 2006-June 2010

Publications:

[1]. C. Zheng, W. Shen, H. Ghenniwa, "An Intents-based Approach for Dynamic Service Discovery," *Service Oriented Computing and Applications*, 2014, doi:10.1007/s11761-014-0163-9.

[2]. C. Zheng, W. Shen, H. Ghenniwa, "An Adaptive Intent Resolving Scheme for Service Discovery and Integration," accepted for *Journal of Universal Computer Science,* 2014.

[3]. C. Zheng, W. Shen, H. Ghenniwa, "A Study of Intents Resolving for Service Discovery," *the 18th IEEE International Conference on Computer Supported Cooperative Work in Design* (CSCWD 2014), Hsinchu, Taiwan, China, 2014, pp. 649–654.

[4]. C. Zheng, W. Shen, H. Ghenniwa, "An Intents-based Approach for Service Discovery and Integration," In *Proceedings of the 17th IEEE International Conference on Computer Supported Cooperative Work in Design* (CSCWD 2013), Whistler, BC, Canada, 2013, pp. 207–212.

[5]. C. Zheng, W. Shen, H. Ghenniwa, "Design and Implementation of Intents User Agent," In *Proceedings of the 17th IEEE International Conference on Computer Supported Cooperative Work in Design* (CSCWD 2013), Whistler, BC, Canada, 2013, pp. 275–280.

[6]. C. Zheng, W. Shen, H. Xue, Q. Hao, "A Heterogeneous Sensors Integration Platform for Independent Living Spaces," In *Proceedings of the 16th* IEEE *International Conference on Computer Supported Cooperative Work in Design* (CSCWD 2012), Wuhan, Hubei, China, 2012, pp. 616–621.