

**END-USER DRIVEN SERVICE COMPOSITION FOR
CONSTRUCTING PERSONALIZED SERVICE ORIENTED
APPLICATIONS**

by

HUA XIAO

A thesis submitted to the School of Computing
In conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
(September, 2011)

Copyright ©Hua Xiao, 2011

Abstract

Service composition integrates existing services to fulfill specific tasks using a set of standards and tools. Existing service composition techniques and tools are mainly designed for SOA professionals. The business processes used in the service composition systems are primarily designed by experienced business analysts who have extensive process knowledge. Process knowledge is the information about a process, including the tasks in a process, the control flow and data flow among tasks. It is challenging for end-users without sufficient service composition skills and process knowledge to find desired services then compose services to perform their daily activities, such as planning a trip. Context-aware techniques provide a promising way to help end-users find services using the context of end-users. However, existing context-aware techniques have limited support for dynamic adapting to new context types (*e.g.*, location, time and activity) and context values (*e.g.*, “New York City”).

To shelter end-users from the complexity of service composition, we present our techniques that assist non-IT professional end-users in service composition by dynamically composing and recommending services to meet their requirements. To acquire the desired process knowledge for service composition, we propose an approach to automatically extract process knowledge from existing commercial applications on the Web. By analyzing the context of end-users, our techniques can dynamically adapt to new context types or values and provide personalized service recommendation for end-users. Instead of requiring end-users to specify detailed steps for service composition, the end-users only need to describe their goals using a few keywords. Our approach expands the meaning of an end-user's goal using process knowledge then derives a group of tasks to help the end-user fulfill the goal. The effectiveness of our proposed techniques is demonstrated through a set of case studies.

Co-Authorship

The content of this thesis has been published in the following papers. More specifically, paper [1] and [2] are based on chapter 3. Paper [3] is based on Chapter 4 and Section 7.1. Paper [4] is based on Chapter 5 and Section 7.2. Paper [5] and the extended journal version paper [6] are based on Chapter 6 and Section 7.3.

- [1] H. Xiao, Y. Zou, J. Ng, L. Nigul, “Personalized Service Discovery and Composition,” *Proc. Smart Internet Technologies Working Conference (SITCON)*, Markham, Ontario, Canada, November 2, 2009
- [2] H. Xiao, Y. Zou, R. Tang, J. Ng, L. Nigul, “A framework for Automatically Supporting End-Users in Service Composition,” *In book “The Smart Internet”, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Vol. 6400, 2010 .*
- [3] H. Xiao, B. Upadhyaya, F. Khomh, Y. Zou, J. Ng and A. Lau, “An Automatic Approach for Extracting Process Knowledge from the Web,” *Proc. The 9th International Conference on Web Services (ICWS) 2011, Washington DC, USA, July 4-9, 2010*
- [4] H. Xiao, Y. Zou, J. Ng, L. Nigul, “An Approach for Context-aware Service Discovery and Recommendation,” *Proc. The 8th International Conference on Web Services (ICWS 2010)*, Miami, Florida, USA, July 5-10, 2010
- [5] H. Xiao, Y. Zou, R. Tang, J. Ng, L. Nigul, “An Automatic Approach for Ontology-Driven Service Composition,” *Proc. IEEE International Conference on Service-Oriented Computing and Applications 2009*, Taipei, Taiwan, December, 2009, pages: 17-24
- [6] H. Xiao, Y. Zou, R. Tang, J. Ng, and L. Nigul, “Ontology-Driven Service Composition for End-Users”, *In journal Service Oriented Computing and Applications (to appear)*, 2011

The aforementioned papers resulting from this thesis were co-authored with my supervisor Dr. Ying Zou, researchers (Ms. Joanna Ng, Mr. Leho Nigul and Mr. Alex Lau) at IBM Toronto Lab, and students (Mr. Ran Tang, Mr. Bipin Upadhyaya and Dr. Foutse Khomh) in our lab. In all cases, I am the primary author. More specifically, Dr. Ying Zou supervised all the research related to those papers and polished those papers. Ms. Joanna Ng, Mr. Leho Nigul, Mr. Alex Lau and Mr. Bipin Upadhyaya participated the discussion meeting of the research projects related the papers which they are co-authors and gave feedback as well as suggestions to improve the research. In the paper “A framework for Automatically Supporting End-Users in Service Composition”, Mr. Ran Tang contributed the content related to business process mining which is not a part of this thesis. In the paper “An Automatic Approach for Ontology-Driven Service Composition”, Mr. Ran Tang collected the experiment data for the case studies. Dr. Foutse Khomh polished the paper “An Automatic Approach for Extracting Process Knowledge from the Web”.

Acknowledgements

Ph.D. study is a long journey but I am not alone. There are many people who have helped and supported me all the way through this journey. Firstly, I would like to take this opportunity to thank my supervisor Dr. Ying Zou whose expertise, encouragement, understanding, kindness, and patience have been extremely helpful in supporting and guiding my Ph.D. study. She is actively involved in the work of all her students. She is always accessible and willing to help her students. I appreciate her efforts to improve the quality of this thesis. I really feel lucky to work with her.

I would also like to thank the rest of the members in my supervisory committee: Dr. James R. Cordy, and Dr. Thomas R. Dean. I am grateful to Dr. Cordy and Dr. Dean for their generous help on providing me insightful advices and guidelines on my research. I also want to thank the other members of my thesis examining committee, Dr. Chung-Horng Lung and Dr. Scott Knight, for their time and comments on my thesis.

It was a huge help to my research to work with the people at IBM Canada laboratory. I would like to thank Ms. Joanna Ng, Mr. Alex Lau, Mr. Leho Nigul, Mr. Jay W Benayon, Mr. Bill O’Farell, Ms. Elena Litani and Ms. Jen Hawkins for their ideas, feedback and support.

The Software Reengineering Research Group is like a big family. We help and support each other both in study and personal life. I would like to thanks all the members in the Software Reengineering Research Group: Dr. Ying Zou, Dr. Xulin Zhao, Ms. Jin Guo, Mr. Brian Chan, Mr. Ran Tang, Mr. Bipin Upadhyaya, Dr. Foutse Khomh, Mr. Lionel Marks, Mr. Derek Foo, Ms. Liliane Barbour, Mr. Dwaipayan Sinha, Mr. Tejinder Dhaliwal, and Mr. Maokeng Hung for their supports, opinions and friendship. Especially, I am very thankful to Ms. Jin Guo, Mr. Brian Chan, Mr. Ran Tang, Mr. Bipin Upadhyaya and Dr. Foutse Khomh for their collaboration and contributions to the projects that I participated in during my Ph.D. study.

I am grateful to the faculty, staff and students in School of Computing at Queen's University for making my Ph.D. study colorful and assisting me in many different ways. Especially, I would like to thank Ms. Debby Robertson for her assistance in organizing the defense and many other matters related to the defense procedure.

This research would not have been possible without the financial support of Queen's University, IBM Centers for Advanced Studies (CAS), and the Natural Sciences and Engineering Research Council of Canada. I would like to express my gratitude to those organizations.

Finally, I would like to thank my wife, my parents and family for their endless love and unconditional support. They have encouraged, supported, understood, and loved me at every moment. None of this would happen without them.

Statement of Originality

I hereby certify that research presented in this dissertation is my own, conducted under the supervision of Dr. Ying Zou. Any published (or unpublished) ideas and/or techniques from the work of others are fully acknowledged in accordance with the standard referencing practices.

(Hua Xiao)

(September, 2011)

Table of Contents

Abstract.....	ii
Co-Authorship	iii
Acknowledgements.....	v
Statement of Originality.....	vii
Chapter 1 Introduction	1
1.1 Major Steps in Service Composition	2
1.2 Challenges for Supporting End-Users in Service Composition.....	5
1.3 Thesis Objectives	7
1.4 Research Statement.....	9
1.5 Outline of the Thesis.....	9
Chapter 2 Background and Related Work	11
2.1 Ontologies.....	11
2.2 Introduction on Service Oriented Architecture	14
2.3 Service Description Models	17
2.3.1 Web Services Description Models.....	17
2.3.2 Semantic Web Service Description Models.....	20
2.3.3 Summary of Service Description Models.....	22
2.4 Service Discovery and Recommendation	24
2.4.1 Techniques to Collect Service Description.....	24
2.4.2 Service Matching Mechanisms	26
2.4.3 Context-Aware Service Discovery and Recommendation.....	30
2.4.3.1 Context Modeling and Context-aware Systems.....	30
2.4.3.2 Discovering and Recommending Services using Context	32
2.5 Service Composition.....	34
2.5.1 Model Driven Service Composition	34
2.5.2 Goal Driven Service Composition	37
2.5.2.1 Hierarchical Task Network Planning.....	38
2.5.2.2 Situation Calculus	39
2.5.2.3 Interface Matching and Backward-chain Reasoning	40
2.5.2.4 Comparison.....	41

2.5.3 Context-Aware Service Composition	43
2.5.4 Comparison of Service Composition Techniques.....	43
2.5.5 Supporting End-Users in Service Composition	45
2.5.6 Process Knowledge Acquisition for Service Composition	46
2.6 Summary	49
Chapter 3 Overview of a Framework for Supporting End-Users in Service Composition.....	50
3.1 An Ad-hoc Process Model	50
3.2 An Overview of a Framework for End-User Driven Service Composition	52
3.3 Summary	58
Chapter 4 Process Knowledge Extraction.....	59
4.1 A Meta-model for Describing Websites	59
4.2 Steps for Extracting Process Knowledge from the Web.....	61
4.3 Algorithm for Identifying a Menu from a Website.....	62
4.4 Extracting Ontologies with Process Knowledge.....	65
4.4.1 Selecting Websites with Process Knowledge	65
4.4.2 An Algorithm for Extracting Process Knowledge from a Website.....	66
4.4.3 Extracting Properties and Tasks from Associated WebPages.....	69
4.5 Integrating Process Knowledge Extracted from Different Websites	70
4.6 Summary	73
Chapter 5 Context-Aware Service Discovery and Recommendation	75
5.1 Overview of an Approach for Context-Aware Service Discovery and Recommendation... 76	
5.2 Searching for Matching Ontologies	78
5.3 Identifying Context Relations	79
5.3.1 Identifying Relations of Two Context Values	79
5.3.1.1 Similarity of Entities in Ontologies	79
5.3.1.2 User-Defined Relations Using Domain Knowledge	81
5.3.1.3 Relations between Two Context Values	82
5.3.1.4 Inferring Relations among Multiple Context Values	85
5.4 Generating Service Searching Criteria.....	87
5.4.1 Identifying End-user's Requirements in Given Context Scenarios	87
5.4.2 Generating Service Searching Criteria.....	89
5.5 Summary	92

Chapter 6 Ontology Driven Service Composition	93
6.1 An Overview of an Approach for Composing Ad-Hoc Processes.....	94
6.2 Tag-based Service Description	95
6.2.1 Schema of the Tag-based Service Description.....	96
6.2.2 Management of Tags.....	97
6.3 Searching for Ontology from Ontology Database	99
6.4 Searching for Services	101
6.5 Generating Ad-Hoc Processes	104
6.5.1 Identifying Tasks	104
6.5.2 Identifying the Relations among Tasks.....	108
6.5.3 Merging Tasks	113
6.6 Overview of our Prototype.....	114
6.7 Summary	116
Chapter 7 Case Studies	117
7.1 Evaluation of Extracting Process Knowledge from the Web.....	117
7.1.1 Experiment Setup.....	117
7.1.2 Evaluation Methods	118
7.1.3 Experimental Procedure.....	119
7.1.4 Result Analysis	121
7.2 Evaluation of Context-aware Service Discovery and Recommendation	122
7.2.1 Experimental Setup.....	123
7.2.2 Evaluation Methods	124
7.2.3 Experimental Procedure.....	124
7.2.4 Result Analysis	125
7.3 Evaluation for Ontology Driven Service Composition.....	129
7.3.1 Experiment Setup.....	130
7.3.2 Evaluation Methods	131
7.3.3 Experimental Procedure.....	132
7.3.3.1 Evaluating the Generated Ad-hoc Processes.....	132
7.3.3.2 Evaluating the Performance of Service Discovery	133
7.3.3.3 Evaluating User Experience of the Service Composition	134
7.3.4 Results Analysis.....	135

7.3.4.1 Results of Evaluating the Generated Ad-hoc Processes	135
7.3.4.2 Results of Evaluating Service Discovery	135
7.3.4.3 Results of Evaluating User Experience	137
7.4 Summary	140
Chapter 8 Conclusions and Future Work	141
8.1 Thesis Contributions	141
8.2 Future Research Directions	143
Bibliography	145

List of Figures

Figure 1-1 An overview of service composition (edited from [122]).....	3
Figure 1-2 An example business process (from [159]).....	4
Figure 2-1 An example ontology	12
Figure 2-2 Components of ontology definition model.....	13
Figure 2-3 The basic service oriented architecture (edit from [45] and [70]).....	15
Figure 2-4 The life-cycle of SOA development (from [101]).....	16
Figure 2-5 The relations of UDDI core types (from [106])	19
Figure 2-6 Top level of the service ontology (from [94]).....	21
Figure 2-7 The goal of AI planning	38
Figure 3-1 Description of an ad-hoc process	52
Figure 3-2 An overview of our framework.....	53
Figure 3-3 Architecture of our framework.....	54
Figure 4-1 A meta-model for describing websites.....	60
Figure 4-2 An example of a Web page	61
Figure 4-3 An overview of our approach.....	62
Figure 4-4 Algorithm to identify menu items	63
Figure 4-5 An example to identify menu items	64
Figure 4-6 An algorithm for extracting the ontology from a website	68
Figure 4-7 Algorithm to integrate process knowledge.....	71
Figure 4-8 An example of integrating ontologies	73
Figure 5-1 Steps for context-aware service recommendation.....	76
Figure 5-2 An example of extending context value using ontology	77
Figure 5-3 Main structure of defining a link specification statement.....	82
Figure 5-4 An example link specification.....	82
Figure 5-5 Examples of relations between two context values.....	83
Figure 5-6 An example of integrated E-R diagram.....	86
Figure 6-1 Steps for composing ad-hoc processes.....	94
Figure 6-2 Schema for tag-based service description	95
Figure 6-3 An example of the service description in WSDL.....	97
Figure 6-4 Algorithm for searching for ontologies.....	100

Figure 6-5 Algorithm for searching for services.....	103
Figure 6-6 Algorithm of identifying task list.....	105
Figure 6-7 An example of generating task list.....	107
Figure 6-8 An example of a generated ad-hoc process.....	112
Figure 6-9 Annotated screenshot for our prototype.....	114
Figure 6-10 Ordered ad-hoc process in the Mashup page.....	115
Figure 7-1 Top-k precision.....	136
Figure 7-2 Recall vs. precision curves.....	137

List of Tables

Table 2-1 Summary of service description models.....	23
Table 2-2 Comparison of service registries and service search engines	25
Table 2-3 Comparison of service matching approaches	30
Table 2-4 Comparison of approaches used in goal driven service composition.....	42
Table 2-5 Comparison of service composition technologies	44
Table 4-1 Map form elements to class properties	70
Table 4-2 Operations to add a child class	72
Table 5-1 Generic rules to derive end-user's requirements	88
Table 5-2 Mapping ontology entities in potential task set to WSDL query and webpage query...	91
Table 6-1 Convert relations from ontology to ad-hoc process.....	110
Table 7-1 List of goals	118
Table 7-2 Recall and precision for ontologies extracted from each website	121
Table 7-3 Recall and precision for integrated ontologies	122
Table 7-4 Context types used in our case study.....	123
Table 7-5 Recall and precision for detecting potential tasks	125
Table 7-6 Evaluation results of service recommendation	127
Table 7-7 Characteristics of generated ad-hoc processes	130
Table 7-8 Characteristics of the six ad-hoc processes	134
Table 7-9 Recall and R-precision comparison	135
Table 7-10 Results of satisfaction evaluation	138

Chapter 1

Introduction

Web services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked over the Web. Service-Oriented Architecture (SOA) [111][113] uses loosely coupled Web services as basic units to build more complex systems in a flexible and rapid way. In particular, a single service generally cannot fulfill the functionality required by organizations. A significant amount of efforts from industry and academia intend to provide infrastructure, languages and tools to compose services using well-defined business processes to streamline business operations. Such SOA systems tackle complex business requirements across organizations. However, little attention has been paid to allow non-IT professional end-users to compose services to fulfill their daily activities.

In today's on-line experience, end-users, who are not familiar with Web service standards and tools, frequently re-visit websites and use on-line services to perform daily activities, such as planning a trip. The end-users potentially compose an ad-hoc process to fulfill their needs. Such an ad-hoc process is characterized by a set of tasks performed by end-users without a strict pre-defined plan. For example, planning a trip is an ad-hoc process for many end-users. It involves several tasks, such as searching for transportation, booking accommodation, and checking the weather at the destination. These tasks can be performed in any order to achieve the goal of trip planning.

Specialized service mediators, such as expedia.com, can be used to search for accommodation, flight and train information. However, the services provided by service mediators are pre-defined and limited. For example, the end-user has to visit other websites to check the reviews about a

hotel or a restaurant, and check if the airline suggested by expedia.com is in part of the same alliance so they can collect their frequent flyer points. The end-user has to repeat the same step when planning the next trip. Moreover, service mediators cannot automatically provide services that are personalized to a particular end-user. For instance, consider a scenario that a person plans a trip to Vancouver in two contexts: (1) John, a business man, lives in Toronto and travels to Vancouver for a business meeting. He would like to take Air Canada, as a reliable transportation vehicle, to stay in Marriott Hotel since he is a valued member of Marriott Hotel, and to find a formal seafood restaurant for an important business dinner. (2) Emily, who is a student at the University of Calgary, wants to take a vacation in Vancouver. She chooses to plan her trip at the lowest cost. She will take Greyhound bus, search for a motel in Vancouver and look for fast food chains, such as Wendy's and MacDonald's. In above contexts, John and Emily have to manually browse different services offered by websites and Web services to gather information for performing the same tasks in the trip planning. It is often challenging for end-users to know all the relevant Web services or websites that help them to make a decision. It would be ideal if an end-user can compose a travel planning service using the SOA techniques. Then the composed service automatically gathers the needed information and presents it to the end-user based on their preferences.

1.1 Major Steps in Service Composition

A service can fulfill a basic task through simple interactions between a customer and the service. However, a single service is not enough to perform complex tasks. Therefore, it is necessary to combine multiple services. The process of integrating existing services to create new service (i.e., composite services) is known as service composition. Figure 1-1 gives an overview

of service composition. A service composition framework involves the following seven steps [122]:

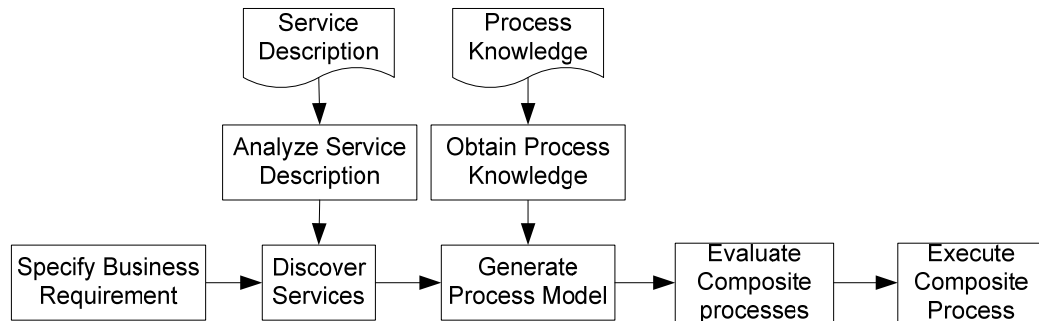


Figure 1-1 An overview of service composition (edited from [122])

(1) Specify business requirements: A requirement (e.g., “buy a book online”) for composing services could be a declarative expression using formal or informal languages, an abstract process model (e.g., a business process model), or a design model (e.g., Unified Modeling Language (UML) diagrams).

(2) Analyze service description: Service descriptions specify the capabilities of services, such as inputs, outputs, exceptions, functional and non-functional description. There are different models and standards [32] [94] to describe services. The service composition system generally analyzes the service description and represents them as a uniform internal specification for further processing (e.g., selecting services and composing services).

(3) Discover relevant services: Service discovery is the process of locating a service that meets certain searching criteria [56]. For example, we search for an appropriate service that offers a weather forecasting service. The searching criteria for the service discovery can be specified by keywords, such as “weather forecast”.

(4) Obtain process knowledge: Process knowledge, such as tasks involved in a process and the control flow and data flow among tasks, is critical for service composition systems to generate business processes or ad-hoc processes. The process knowledge mainly comes from the business analysts, service descriptions, business requirements and process description documents.

Business processes are a set of logically related tasks that are performed to achieve business objectives. A process defines tasks, connections, roles and resources. Tasks define the operations for achieving business objectives. Connections include control flows and data flows among tasks. A role performs a set of designated tasks. Resources are necessities for executing tasks. An example of a business process could be a “purchase order processing”, shown in Figure 1-2. The “purchase order processing” contains five tasks, such as “receive purchase order”, “select a shipper”, and “schedule production”. The data, “purchase order”, flow from “receive purchase order” task to “select a shipper” tasks [160].

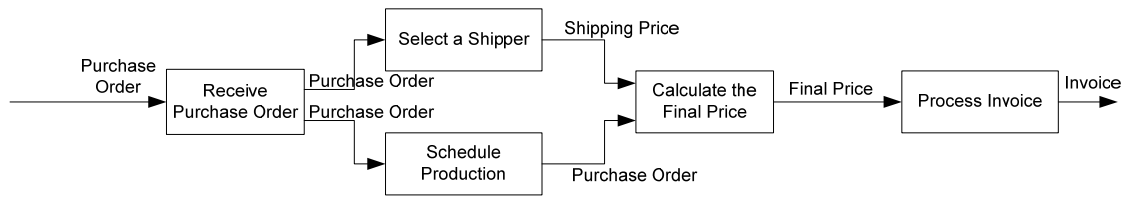


Figure 1-2 An example business process (from [160])

(5) Generate composition process model: A business requirement usually cannot be fulfilled by a single service. We need to integrate various services using process models. A process model contains a set of selected services, as well as control flows and data flows among those services [122]. Standards, such as Business Process Execution Language (BPEL) [139], are used to formally specify process models.

(6) Evaluate composition results: It is quite common that many returned services have the same or similar functionalities. It leads to more than one composition result to fulfill the business requirements. We need to evaluate different composition results and select the best one that meets the functional and non-functional requirements.

(7) Execute the composite process: After a unique composite process is determined, the framework executes the process on an execution engine and returns the result to the service requester.

1.2 Challenges for Supporting End-Users in Service Composition

In the current state of practices, composing services requires a large number of professionals (e.g., business analysts, system integrators and service developers) with strong SOA background. The composition process involves various technical tools (e.g., IBM WebSphere Business Modeler (WBM) [73] and IBM WebSphere Integration Developer (WID) [74]) and languages (e.g., BPEL) to specify, compose, and deploy services. To produce a composite service, the professionals in different roles and tools must interact in harmony. Unfortunately, non-IT professional end-users do not have the knowledge of these tools and SOA standards. In short, involving end-users in service composition has the following three major challenges:

- **Difficult to obtain process knowledge.** Process knowledge provides the information for service composition systems to create processes. In the current state of practices on service composition, business processes are primarily designed by experienced business analysts who have extensive process knowledge. It is challenging for non-IT professional end-users to orchestrate a process due to the lack of process knowledge.

Research efforts have been devoted to sharing knowledge in public. Several knowledge bases are designed to allow machines to retrieve and process the knowledge stored in the

knowledge bases. Ontologies are widely used for knowledge representation and sharing in knowledge bases. Ontologies use a formal way to represent knowledge as a set of concepts and relationships among the concepts. For example, the ontology of “travel” lists relevant concepts, such as “booking flight tickets”, “hotel reservation”, and “weather forecast”. DBpedia [40] and Freebase [54] are examples of knowledge bases that extract the knowledge from Wikipedia [142] and store it as ontologies. Information extraction tools, such as Text2Onto [35], are used to extract ontologies from textual resources, such as text files or Web pages. However, existing knowledge bases and extraction tools focus on explaining high level concepts using more concrete concepts. Such knowledge description is lack of a stepwise description on how to complete a collection of tasks to achieve a goal. To enable non-IT professional end-users to compose services, we need an effective approach to extract the process knowledge and construct a process knowledge database.

- **Difficult to find appropriate services.** Given the large amount of existing services and the diversified end-user's requirements nowadays, it is time-consuming for end-users to find appropriate services for a specific requirement from the large number of services. Context-aware techniques provide a promising way to help end-users obtain their desired services by automatically analyzing an end-user's context and recommending desired services for the end-user. A context characterizes the situation of a person, place or the interactions between humans, applications and the environment [20]. However, most existing context-aware techniques require system designers to manually define reactions (e.g., recommend services) to contexts based on context types (e.g., location) and context values (e.g., Toronto). Those context-aware techniques have limited support for dynamic adaption to new context types and values. Due to the diversity of end-user's environments, the available context types and

potential context values are changing overtime. It is challenging to anticipate a complete set of context types with various potential context values to provide corresponding service recommendation. We need an effective way to provide services recommendation using the context of end-users.

- **Limited support for composing services on-the-fly for end-users.** A system integrator can specify BPEL processes to compose Web services using tools, such as IBM WID. After deploying a Web service, the composition logic can hardly be customized to accommodate changes to consumer's requirements, as this involves a long lifecycle from design, development and testing to deployment. Moreover, it is also infeasible to expect an end-user to specify the details of each task and orchestrate a well-defined process in BPEL. In the current practices, there is very limited support for end-users to compose ad-hoc processes which involve the dynamic integration of various services (e.g., Web services, and websites) on-the-fly.

1.3 Thesis Objectives

To address the aforementioned challenges, we focus on the following objectives:

- **Extracting process knowledge from the Web.** Websites, such as on-line stores, and travel agencies provide specialized services to end-users. Such websites capture domain specific process knowledge which can be used to assist end-users in achieving a user's goal. For example, expedia.com provides interactive end-user interfaces to allow end-users to complete various tasks in a trip planning process such as "buying flight tickets", "booking hotels" and "purchasing travel insurance". To leverage the domain knowledge embedded in specialized websites and obtain the process knowledge for service composition, we propose techniques to extract the process knowledge from various websites.

- **Personalizing service recommendation using contexts.** Effective personalized service recommendation requires the ability to detect and analyze an end-user's context at the run-time environment. The context characterizes the conditions for an end-user to fulfill a goal, such as end-user's profiles, computation resources, scheduled tasks, and end-user's preferences. To recommend personalized services, we design and develop techniques that capture an end-user's dynamically changing context and formulate searching criteria to discover the desired services. Different from existing approaches which depend on context models to know the relations among context types and values, then use predefined rules to infer user's needs, we seek an automatic approach to recognize the relations between context values and a user's needs. For example, luxury hotel and limited budget are two potential user's needs in conflict. Therefore, the services for booking luxury hotels are automatically filtered when a user has a limited budget. We expect that such relations can be used to express more accurate searching criteria which better reflect a user's context.
- **Supporting end-users to compose services on-the-fly.** In the current state of practices, SOA practitioners manually describe the details of each task and the interactions among tasks using BPEL. In an end-user environment, most of the activities, such as planning a trip, are spontaneously prompted from an end-user. An end-user may not have a clear plan on achieving a goal (e.g., planning a trip). To guide an end-user to achieve their goal, we devise techniques that dynamically search and compose services on-the-fly to assist an end-user in fulfilling their desired goals (such as planning a trip). We aim to provide an approach to shelter the end-user from complex programming issues.

1.4 Research Statement

In this thesis, we propose a generic framework for supporting non-IT professional end-users to compose services. The framework enables end-users, who do not have SOA background, to compose services easily. To shelter non-IT professional end-users from the complexity of service composition, the framework extracts the process knowledge from the Web instead of requiring end-users to provide the process knowledge, automatically recommends services based on the context of end-users, and generates ad-hoc processes on-the-fly using the extracted process knowledge.

1.5 Outline of the Thesis

Chapter 2 reviews the background and related work for supporting end-users in service composition.

Chapter 3 overviews our proposed framework that automatically uses contextual data to discover and recommend services, then composes services on-the-fly. This chapter introduces the major components of our framework and the general steps to help end-users composition services. We further provide the definition of the ad-hoc process which is used to describe the process for fulfilling the daily activities of end-users.

Chapter 4 presents our approach that extracts process knowledge from various websites and merges the extracted process knowledge to generate an integrated ontology with rich process knowledge.

Chapter 5 provides an approach for context-aware service discovery and recommendation. We use ontologies to enhance the meaning of context values and automatically identify the

relations among different context values. Based on the relations among context values, we recommend the potential service which the end-user might need.

Chapter 6 presents an ontology-driven service composition approach which can automatically generate ad-hoc processes to help end-users fulfill their daily activities.

Chapter 7 presents three case studies to evaluate the approaches for process knowledge extraction, context-aware service recommendation and ontology-driven service composition respectively.

Chapter 8 concludes the thesis by illustrating the major contributions and discussing the future research directions.

Chapter 2

Background and Related Work

This section presents the background and surveys of the related work. Section 2.1 introduces the background of ontologies which are used to extend the semantics of contexts and describe the process knowledge in the thesis. Section 2.2 covers the background of SOA. Section 2.3 discusses different service description models which characterize the capabilities of services. The accuracy of service discovery increases the success in service composition and eventually enhances the satisfaction of the composition requirement. Section 2.4 compares different service discovery and recommendation approaches. Service composition systems take the composition requirements and available services as inputs. Then composition systems produce executable process models (e.g., BPEL processes) automatically or with the help of service integrators. Section 2.5 discusses the research related to process model generation. Finally, Section 2.6 summarizes the chapter.

2.1 Ontologies

An ontology models a domain of knowledge or discourse as a set of concepts (*e.g.*, people, travel and weather), and the relations between these concepts [61]. An ontology can be visualized as a graph that contains nodes representing concepts and edges representing relations among the concepts. Figure 2-1 illustrates an example ontology for defining the concept “Travel”. More specifically, the concept “Travel” is related to four more specific concepts: “Transportation”, “Accommodation”, “Tourist Attraction” and “Car Rental”.

Ontologies are defined using various ontology specification languages, such as Web Ontology Language (OWL) [129], Resource Definition Framework (RDF) [16] and DARPA Agent Markup Language + Ontology Inference Layer (DAML+OIL) [37]. To capture the entities and general

structures specified in various ontology specification languages, we summarize the commonality of different ontology specification languages using an ontology definition model. Figure 2-2 illustrates the main components of our ontology definition model and their relations.

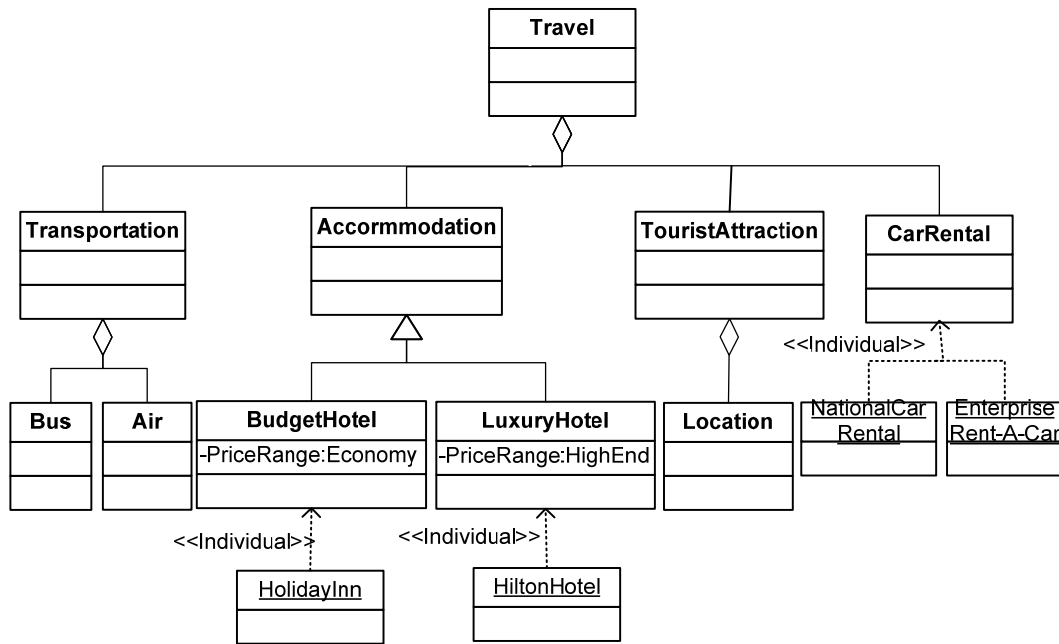


Figure 2-1 An example ontology

- *Class*: is an abstraction description of a group of resources with similar characteristics. For example, “Accommodation” is a *class* which describes the common characteristics of different types of accommodation.
- *Individual*: is an instance of a *class*. *Individuals* are the objects and a *class* describes a set of *individuals*.
- *Property*: is an attribute that a *class* and an *individual* can have. The properties defined in a class are applied to the individuals.

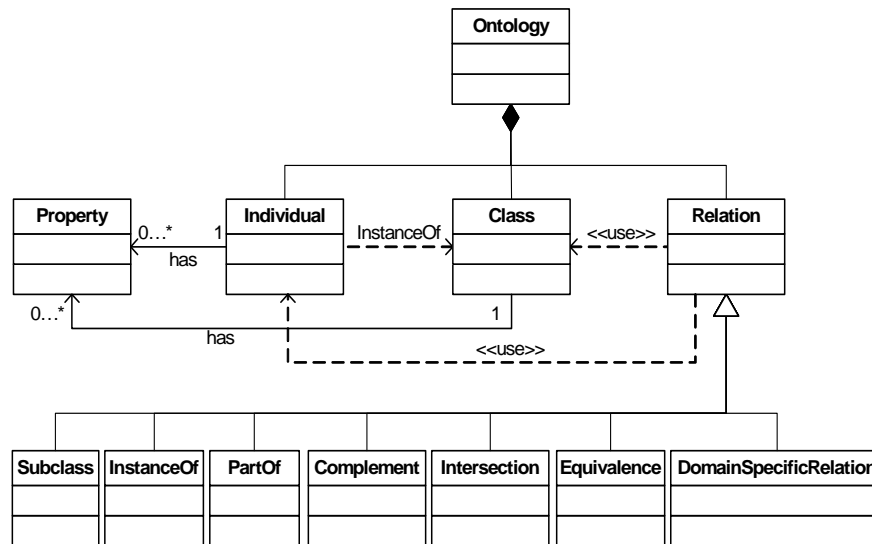


Figure 2-2 Components of ontology definition model

- *Relation*: defines various ways that *classes* or *individuals* can be related to one another. For example, entity “BudgetHotel” is a subclass of “Accommodation”. The relations among classes are applied to the corresponding individuals. Figure 2-2 lists the common relations defined in various ontology specification languages. We summarize seven types of relations among classes and individuals.
 - *Subclass*: extends an abstract *class* to convey more concrete knowledge. The *subclass* relation describes the parent and children relations among the connected *classes*. The *subclasses* can inherit the attributes in its parent class. As shown in Figure 2-1, “Budget Hotel” and “Luxury Hotel” are the subclasses of “Accommodation”.
 - *InstanceOf*: describes that an *individual* belongs to a *class*. For example, individual “Hilton Hotel” is an instance of class “Luxury Hotel”.
 - *PartOf*: indicates that a class is a part of another class or an individual is a part of another individual. For example, class “Accommodation” is a part of class “Travel”. In some

ontology specification languages, such as OWL, a class could be defined as a union of several classes, i.e., a class contains distinct member classes. For example, “Weather” is a union of “Forecast”, “Wind”, “Temperature” and “Precision”. To simplify the ontology definition model, we treat the union relation as a *PartOf* relation since the classes in a union relation are a part of the united class. For example class “Forecast”, “Wind”, “Temperature” and “Precision” is in a *PartOf* relation with class “Weather”.

- *Complement*: selects all the *classes* from the domain that do not belong to a certain *class*. For instance, class “Budget Hotel” could be defined as a complement of class “Luxury Hotel”.
- *Intersection*: defines a *class* which has the common *individuals* of several classes. For example, “Luxury Hotel” is the intersection of “Hotel” and “Expensive Consumption”.
- *Equivalence*: declares that two classes contain exactly the same individuals. For example, we can define that class “Nation” is equivalent to class “Country”.
- *DomainSpecificRelation*: specifies domain specific relations among classes. For example, airline company “Air Canada” is the sponsor of “2010 Winter Olympics”. Therefore, the relation between “Air Canada” and “2010 Winter Olympics” is “Sponsorship”. The ontology specification languages, such as OWL, use properties to describe domain specific relations. To distinguish properties from relations, we convert such a type of property description to *DomainSpecificRelation* in our ontology definition model.

2.2 Introduction on Service Oriented Architecture

Today’s organizations face a rapidly changing market and global competition. These situations require the IT infrastructure of organizations to respond quickly in support of new business models and requirements. Integration is the key element of an on demand IT

infrastructure. Integration can optimize operations across organizations and enable them to interoperate seamlessly through the efficient and flexible combination of resources.

Service Oriented Architecture (SOA) [111][113] provides a logical way of designing a software system to support integration. SOA wraps applications as services. New applications can be assembled by a set of existing distributed services across organizations and platforms. SOA increases reuse and cooperation among services. Therefore, applications can be built in a rapid way with relatively low cost.

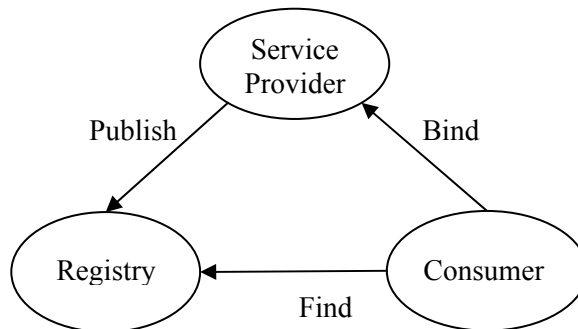


Figure 2-3 The basic service oriented architecture (edit from [45] and [70])

In SOA, services are the primary units. A service is a software component designed to support machine-to-machine interaction over a network [130]. A service could be as simple as a query, or as complicated as a sophisticated business process. Any component of an application can be wrapped as a service and published to the network for other applications to reuse. SOA consists of three major stakeholders: the service provider, the service registry, and the service consumer (client), shown in Figure 2-3 [45][70]. Service providers deliver services and are responsible for publishing descriptions of the delivered services. Service consumers search for and invoke services to achieve their goals. A service registry organizes services and facilitates the service searching for service consumers. In a typical scenario, service providers set up an environment to

hold a service and publish the service to a service registry. A service consumer finds the service from the service registry, and integrates the service with their existing applications. To execute a service, a service consumer needs to bind the abstract properties obtained from the service registry to a concrete service, and then remotely invoke the service.

The life-cycle for developing an SOA system has four steps, shown in Figure 2-4 [101]: 1) Business processes are modeled to reflect the business objectives. 2) We search for services to accomplish the tasks defined in business processes. Returned services are assembled and bound to executable business processes. 3) We deploy the executable business processes and execute them. 4) After the processes are deployed, we manage processes by considering their business functions and technical issues. Monitoring the execution of processes and collecting execution data are the most important aspects of business processes management. When a problem which impacts the performance or execution of a business process is identified, we refine the business process and repeat the cycle.

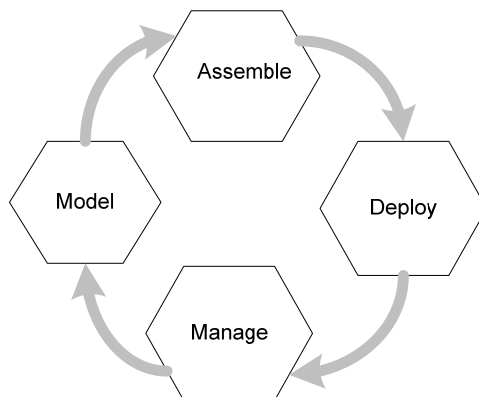


Figure 2-4 The life-cycle of SOA development (from [101])

2.3 Service Description Models

When publishing a service, service providers hide the implementation details of a service. In the perspective of end-users, services are black boxes. To enable potential consumers to find and execute services successfully, it is necessary to provide service description models to support the description and discovery of services.

Service description models specify the capabilities of services, which usually includes input and output parameters, exceptions, functional and non-functional description. For example, Web Services Description Language (WSDL) [32] uses Extensible Markup Language (XML) to describe the operations offered by a service, the input and output messages of operations, and the address to access a Web service. OWL-S [94] uses ontology to describe the semantics of Web services, especially the functional description.

Service composition depends on service description models to find required services for processes. There are a variety of service description models. This section gives an introduction of different service description models and illustrates their pros and cons through comparison.

2.3.1 Web Services Description Models

WSDL is an XML based language to specify how a potential client can access a service [32][33]. WSDL is a well accepted industry standard. WSDL describes Web services as a set of operations. An operation is an interaction between the service and a service consumer. Each interaction contains a set of messages exchanged between a service and a service consumer. The messages themselves are described abstractly and are bound to a concrete network protocol and message format [32].

The Universal Description, Discovery, and Integration (UDDI) [36] framework is defined on top of WSDL. UDDI is a platform-independent framework for describing, publishing and discovering services. A UDDI registration is a directory service to allow service providers and service consumers to register and search for Web services. A UDDI model is composed of the following core types [36][106][135]. Relations among these types are described in Figure 2-5.

- **BusinessEntity:** A BusinessEntity describes a provider of Web services. The BusinessEntity includes company information, contact information, industry categories, and business identifiers.
- **Business Service:** A BusinessService describes a list of services provided by a BusinessEntity. Each businessService is the child of a BusinessEntity.
- **BindingTemplate:** A BindingTemplate represents an individual Web service. The BindingTemplate provides the technical information of a Web service which helps other application to bind and interact with the described service. The BindingTemplate includes the access point of a service or an indirection mechanism that leads to the access point. An example BindingTemplate is a WSDL file that describes the programming interface of a Web service.
- **tModel (Technical Model):** tModels are used in UDDI to represent concepts or constructs which allow reuse and standardization within a system. To describe a Web service that conforms to a particular set of specifications and protocols, we use BindingTemplate to refer to tModels which define the specifications and protocols. In such a way, a tModel could be used by multiple BindingTemplatees.

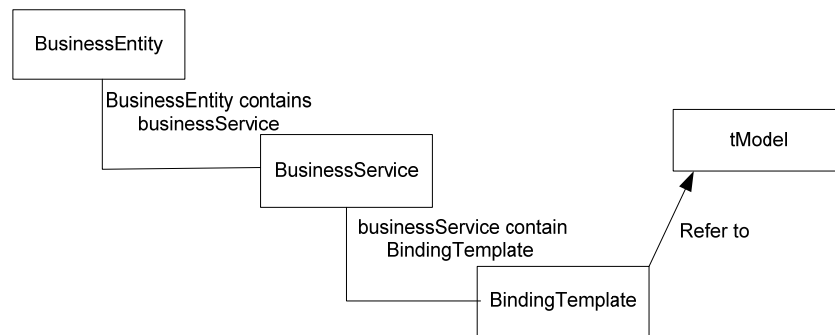


Figure 2-5 The relations of UDDI core types (from [106])

Before accessing a service, a service client usually needs to know the precondition of using the Web service. The Web Services Policy [136] provides a general purpose model and the corresponding syntax to describe the policies of entities in a Web services-based system. Policies describe conditions, capabilities or other properties of a behavior (e.g., an operation). For example, a policy could declare that a specific message body must be encrypted in order to protect the context of the message. In Web Service Policy specification, a policy is associated with entities (e.g., messages, operations and resources). Therefore, we can use policy to enhance the description of entities defined in service description models, such as WSDL and UDDI.

Neither WSDL nor UDDI describes the quality of services. To represent the performance characteristics of Web services (e.g., response time, throughput, utilization), D’Ambrogio [38][39] proposes a Performance-enabled WSDL (P-WSDL) meta-model. P-WSDL extends WSDL with performance descriptions: PStep, PHostingEnv, PNetwork, PWorkload, PClosedWorkload, POpenWorkload, and PData. PStep describes the attributes of an operation provided by a service, such as execution time and response time. PHostingEnv presents the hosting environment which executes the service. PNetwork represents the characteristics of the

network, such as network speed. PWorkload, PCloseWorkload and POpenWorkload specify the workload of a specific operation. PData describes the size of operation messages.

2.3.2 Semantic Web Service Description Models

Conventional Web service description models specify the syntax of the data, but cannot describe the semantics of the data. Without semantic description, service consumers and service integrators need additional background or additional documents to understand and access a service. The lack of semantic description makes automatic service composition difficult. To add semantics to Web services, various semantic Web service description models [5] [50] [94] [120] [127] are proposed by researchers. Semantic Web service description models are generally built on accepted standards to exchange semantic data. Semantic data decrease the difficulty for service consumers and machines to understand the service description.

As an industry standard, WSDL does not include semantic description. To enhance the WSDL model, there are some specifications and research adding semantic models to WSDL. Semantic Annotation for WSDL and XML Schema (SAWSDL) [50] is a World Wide Web Consortium (W3C) recommendation to add semantics to WSDL and XML. The specification provides mechanisms to associate semantic models (e.g., Ontologies) to WSDL and XML schema components. The semantic models are defined outside the WSDL document. SAWSDL does not denote any specific language for representing the semantic models. Sivashanmugam et al. [127] add semantics to WSDL using the extensibility supported by WSDL specification. They also enable end-users to search for Web services using UDDI by annotating semantic descriptions to UDDI. Rajasekaran et al. [5][120] create a language called WSDL-S to extend WSDL with semantic description. In their approach, Rajasekaran et al. assume that the semantic models

already exist. The semantic models are maintained outside of WSDL documents and are referenced from the WSDL document through WSDL extensible elements.

Instead of extending WSDL with semantics, many researchers create a full framework for semantic Web services. Ankolekar et al. [9] use a DAML+OIL based ontology, named DAML-S, to add semantic service description on top of WSDL. OWL-S is developed based on DAML-S. OWL-S provides an OWL-based framework for describing semantic Web services. The OWL-S ontology is written in Ontology Web Language (OWL). In OWL-S, the class “Service” provides an organizational point of references for a declared Web service. Each published service is mapped to an instance of “Service”. Class “Service” contains ServiceProfile, ServiceModel and ServiceGrounding shown in Figure 2-6.

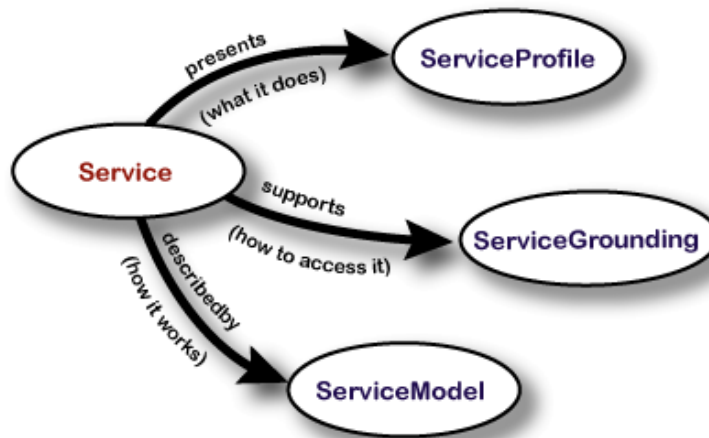


Figure 2-6 Top level of the service ontology (from [94])

- 1) ServiceProfile provides information needed for an agent to discover a service, i.e., what the service does. ServiceProfile includes a description of what is accomplished by the service, limitations and quality of service, and preconditions for invoking the service.

- 2) ServiceModel describes how to use the service. ServiceModel provides the semantic content of requests, the conditions of some particular outcome, and the detailed process.
- 3) ServiceGrounding provides information about how a service client can access the service, such as communication protocol and message formats. Using ServiceGrounding, the OWL-S description can be bound to a WSDL description or other service description models. For example, if a service is described by both OWL-S and WSDL, we can use ServiceGrounding to specify the mapping from the inputs and outputs defined in OWL-S to the corresponding inputs and outputs in the WSDL description.

2.3.3 Summary of Service Description Models

To summarize the aforementioned service description models, we create a table to show the characteristics of different service description models. As shown in Table 2-1, each service description model addresses different perspectives of services. With the cooperation of different models, we can describe different perspectives of services.

To interact with Web services, the programming interface of Web services is a necessary part of a service description model. Since WSDL is a well-accepted industrial standard to describe the programming interface of services, many service description models are constructed on top of WSDL. Based on the compatibility of WSDL, we can classify the service description models listed in Table 2-1 into two types.

1) *Service description models built on top of WSDL.* Such models reuse WSDL schema to describe the programming interface of Web services. UDDI, Web service Policy, SAWSDL, OWL-S, and WSDL-S are in this category. By keeping the WSDL description, it is easy to adapt those models by extending existing WSDL Web services.

2) *Service description models constructed by extending the WSDL schema*, (e.g., P-WSDL in [38][39]). This type of models might be simpler than the other type since it does not have to consider the compatibility issue between the new model and the WSDL schema. However, changing the schema of WSDL may impact other components of a SOA system. For example, the business process execution engine depends on the schema of WSDL to invoke Web services. The lack of compatibility may hinder the adoption of this type of models.

Table 2-1 Summary of service description models

Description model name	Programming Interface	Service Provider info.	Support Semantic Description	Requirements (pre-conditions) for invoking services	Describe processes inside a composite Web services	XML-based	QoS	References
WSDL	√					√		[33][32]
P-WSDL	√					√	√	[38][39]
UDDI		√				√		[36] [106][135]
Web Services Policy				√		√		[136]
SAWSDL	√		√			√		[50]
WSDL-S	√		√			√		[120][5]
OWL-S (developed from DAML-S)	√		√	√	√	√		[9][94]

However, the service description models listed in Table 2-1 are designed for programmers or machines to read. It is difficult for non-IT professional end-users to understand these service description models. In addition, due to the complexity of applying semantic descriptions for Web services in practice, it is challenging for service developers to provide formalized semantic

descriptions. In our approach, we use descriptive tags to enhance the description of WSDL, which can be easily understood by end-users. Instead of requiring the formal semantic descriptions from service developers, we use structured tags to enhance the descriptions of service descriptions. By simplifying the services description using tags and encouraging the participation of end-users, our approach can use the knowledge from end-users to enhance the existing service description model (i.e., WSDL) with a certain level of semantic meanings.

2.4 Service Discovery and Recommendation

The accuracy of service discovery increases the success in service composition and eventually enhances the satisfaction of the service composition requirements. This section discusses different approaches to collect service descriptions from service providers and introduces different algorithms to match services descriptions with the requirements from the service consumer.

2.4.1 Techniques to Collect Service Description

Service providers advertise their services in the service registry and enable service consumers to search for services from the registry. A service registry creates a bridge between service providers and service consumers and enables service consumers to access wide range of Web services that match specific searching criteria. UDDI service registry is a representative service registry which can be used to register and search for services. The first public UDDI business registry (UBR) nodes were run by IBM, Microsoft, and SAP [123].

Centralized registries [110][34] provide an efficient way to manage service registrations. However, centralized registries suffer from problems of single point of failure, bottleneck, and scalability.

Service registries can also be organized by maintaining multiple distributed registries. The distributed UDDI registries are connected together as a registry federation. Many decentralized solutions have been proposed to create registry federations. For example, Sivashanmugam et al. [45] provide a discovery mechanism for publishing Web services on a federated registry environment.

In addition to service repository, service search engines, such as Google [59], Yahoo [154], and Bing [18], are another source for searching for Web services. Service engines find the service description documents such as WSDL. Moreover, some service search engines, such as Seekda [125] and Woogle[143], are particularly designed to retrieve Web services. Service search engines use a focused crawler [17][81] to collect Web service information and index service description data in a database. Service consumers can use keywords or some specific query languages to find services using these search engines.

Table 2-2 Comparison of service registries and service search engines

Approach of discovering services	Way to find a service	Service description	Number of services	References
Service Registry	Passive: assume that service providers would voluntarily registry their services.	Service providers can give structured and detailed description of services, such as UDDI	Depends on the service providers	[45] [110][34]
Search Engine	Active: automatically crawl the Web to search for service description documents (e.g., WSDL)	Mainly are WSDL	Easy to collect service description; Better coverage comparing with service registries	[17][125] [143] [81]

Table 2-2 compares service registries and service search engines. The key difference is the way of obtaining service descriptions. The service registry approach requests service providers to manually publish the services to the service registry. On the contrary, the service search engine automatically finds services by crawling the Web. In addition, as discussed in [12], service engines have a better coverage than service registries. Service registries are not well accepted by the public. In January 2006, IBM, Microsoft and SAP closed their public UDDI hosts [123].

2.4.2 Service Matching Mechanisms

After a service registry or a service search engine collects service descriptions, the service discovery system needs to find proper services based on the requests from potential service consumers. The service matching mechanisms can be classified into two categories: information retrieval approaches and semantic matching approaches.

Information Retrieval Approaches: Document matching and classification are the typical problem in the field of information retrieval (IR). IR techniques are used to find desired services.

- *Keyword based matching:* Keyword matching is the most common form of text search in the field of IR. It retrieves the document by matching the keywords provided by the end-user and the keywords extracted from the text.

UDDI [36] natively supports simple keyword searching. The textual descriptions of services are indexed and stored in the database. Given a search query (i.e., keywords), the service discovery systems search the textual descriptions and return a candidate answer set. End-users can select the appropriate one from the answer set. However, simple keyword matching cannot use the semantics represented by structured documents. It is difficult for keyword matching approaches to distinguish the differences of semantic meanings for a word

which has different meanings. To use the structure information of documents, WSXplorer [64] analyzes the structure of input and output parameters of services and converts the input and output structured data into trees. Given the description of input and output data for the desired service, WSXplorer finds services by analyzing the structures and nodes of input and output data trees. If end-users give a service, WSXplorer uses the tree matching algorithm to find similarity services. In addition, WSXplorer identifies the association relations between services by matching the output data of a service with the input data of another service. A set of services are associated if they potentially contribute to an end-user's service composition, i.e., is able to compose together to fulfill a task.

- *Classification*: Services are manually or automatically classified into different categories (or domains). Given a query for searching services, the system identifies the category that the query belongs to then returns the services which are in the same category as the query. For example, Woogle [43] supports similarity search for Web services. Woogle uses clustering techniques to group operation names, parameter names of inputs and outputs into meaningful concepts. Given a service, Woogle can return the similar services. Arabshian et al. [10] use service classification ontology to classify services into different classes. When an end-user gives a query to search for services, the service classification ontology identifies the class of the query, and guides end-users to provide more detailed data to trace the service classification ontology from high-level class to concrete services.
- *Link Ranking*: If there is more than one service matching the required functionality, it is necessary to rank those matching services and select the best one. Service rank is a quantitative metric which shows the importance of a service within a Web service network. Service rank can be used to select services [57]. PageRank[21] is a link analysis algorithm

used by Google Internet search engine to evaluate the importance of Web pages. Similar to pageRank algorithm, Gekas al et. [57][58] propose an approach to rank services using dynamic data flows among Web service operations.

Semantic Matching Approaches: A limitation of traditional information retrieval approaches is the lack of semantic matching [110]. Service providers and service consumers have very different perspectives and knowledge about the same service. It is unrealistic to expect that service providers and service consumers would give the same description of services. Furthermore, the service providers would not provide exactly the service that a service consumer needs. For example, a service provider may describe a service as a financial news provider. While the service consumer might need a service to obtain updated stock information which is a part of the financial news. We cannot match the service request with the service description using the traditional information retrieval approaches.

To improve the quality of service searching, research efforts are focused on semantic matching approaches. Paolucci et al. [110] propose a semantic matching approach for DAML-S service description model. Given a searching query, i.e., the input and output of a service, the approach uses ontologies to examine the relations between input and output of the request and the input and output of the service description. Then a matching degree is calculated based on relations of the request and the service description. The matching algorithm defines four levels of matching between two concepts (i.e., concepts that describe inputs and outputs): a requested concept, C_{Req} , and an advertised concept, C_{Adv} .

-- Exact: if $C_{Req} = C_{Adv}$ or C_{Req} is a direct subclass of C_{Adv}

-- Plug in: if C_{Adv} subsumes C_{Req} , in other word, C_{Adv} could be used in the place of C_{Req} .

-- Subsume: if C_{Req} subsumes C_{Adv} , in this case, the service does not completely fulfill the request. Therefore, another service may be needed to satisfy the rest of the expected data.

-- Fail: failure occurs when no subsumption relation is identified between the advertised concept and the requested one.

Wang and Stroulia [138] synthesize the issues of UDDI related to service discovery by combining the similar textual description of Web services with semantic structure similarity. The textual description similarity is calculated using WordNet [116]. WordNet is a lexical database which groups words into sets of synonyms and connects words to each other via semantic relations. The semantic structural similarity depends on the similarity of data types used in WSDL files.

Benatallah et al. [17] transform service searching requests and service description into description logics (DLs). DLs use a formal way to define the structure and semantics of concepts in ontologies. The authors treat the service matching as the best covering problem. Benatallah et al. provide an approach to automatically transform service description model DAML-S into DLs description. When service searching request (Q) is also specified using DLs, the process of service matching is to find services which have the best cover of Q. The descriptions of matching services should contain as much common information with Q as possible and as little extra information with Q as possible.

Table 2-3 summarizes different service matching approaches discussed in this section. Keyword searching is the basic searching approach and supported by most of existing service searching engines, such as UDDI registry and Seekda. To refine the keyword matching approach, WSXploer combines the keyword searching with structures matching to provide better searching

results. However, WSXplorer requires the user to provide structure description of the input and output data of services. Woogle focuses on finding similar services for a given service by comparing different service descriptions. Link-ranking is the complement of service searching which helps the system to refine the searching result and select the best services.

Table 2-3 Comparison of service matching approaches

Service matching model	Content of Service Request	Matching method	Service description model	References
UDDI registry	keywords	Searching by categories (classified by service providers) and keywords matching	UDDI and WSDL	[36]
WSXplorer	Input and output data description	Matching the structures and keywords of input and output parameters	WSDL	[64]
Woogle	A given service	Cluster services according to parameter names of inputs, outputs and operations names	WSDL	[43]
Link Ranking	A set of available services	Ranking servers using the dynamic data flow among Web service operations	OWL-S or DAML-S	[58][57]
Paolucci's model	The input and output of services	Ontology based matching	DAML-S	[110]
Wang and Stroulia's model	A textual description of the desired service	Calculate the similarity between request and UDDI & WSDL description	UDDI and WSDL	[138]
Benatallah's model	Provided using description logics (DLs)	Treat the service matching as a best covering problem in description logics area	DAML-S	[17]

2.4.3 Context-Aware Service Discovery and Recommendation

2.4.3.1 Context Modeling and Context-aware Systems

In general, context is the information that characterizes the situation of a person, place or the interactions between humans, applications and the environment [41]. In Web services, the term “context” can be differentiated into two parts: the context of service consumers, and the context of Web services [157]. The context of service consumers includes the surroundings information about the consumers. The surroundings information could be utilized by a Web service to adjust its execution and output to provide the consumers with a customized and personalized behavior. The surroundings information can help service consumers discover and access services. Examples of a service consumer’s context could be consumers’ preferences, locations, and activities. The context of services includes common data about the current status of a service and the capability of collaborating with other services, such as network protocols, platforms and computing devices for services execution.

Several context models and context-aware systems are proposed in the literature [13][28][131] [131]. Strang and Linnhoff-Popien [131] survey existing context models and classify them into different types based on the data structures. The context models are classified into 6 types: key-value models, markup scheme models, graphical models, object oriented models, logic based models, and ontology based models. The context models are evaluated using six requirements. Ontologies are the most expressive model that can fulfill most of the requirements according to their evaluation. Chen and Kotz [28] investigate the research on context-aware mobile computing. Chen and Kotz discuss the types of context used, the ways of using context, the system level support on collecting context, and approaches to adapt to the changing context. Baldauf et al. [13] present a layered conceptual design framework to describe the common architecture principles of context-aware systems. Based on their proposed design framework, Baldauf et al. compare

different context-aware systems on various issues: the context sensing, context models, context processing, resource discovery, historical context data, security and privacy.

2.4.3.2 Discovering and Recommending Services using Context

Applying context-aware techniques to discover and recommend services has gained lots of attention. Yang et al. [29][157] design an event-driven rule based system to recommend services according to people's context. Yang et al. define an ontology-based context model to represent a context. Requester ontology and service ontology are developed for specifying the context of requesters and services respectively. Using context inference rules, further contextual information can be inferred from the current contextual information. For example, an end-user's activity at a given time can be derived by examining the time and calendar. When searching for Web services, Yang et al. firstly identify the similarities of inputs and outputs between requests and published services using capability matching. If there is no matched service, a semantic matching component decomposes the request into sub-requests based on requester's contextual information and searches for services for each sub-request. Balke and Wagner [14] propose an algorithm to select a Web service based on end-user's preferences. The algorithm starts with a general query. If there are too many results, the algorithm expands the service query using end-user's preferences. The algorithm expands the query with loose constraints extracted from end-user's preferences. If too many results are retrieved, it extends the query with restrict constraints and searches for Web services again. The algorithm adds constraints step by step to narrow down the number of service searching results to a small value.

Chen et. al [31] use collaborative filtering technique to recommend services based on the Quality of services. Their approach clusters users into several regions using the physical locations and historical QoS data of users. When the system needs to recommend a Web service from a set

of candidate Web services, their approach uses the historical data of the user to find the regions that the user belongs to, then predicts the QoS of the candidate Web services. Based on the predication, the service with the best predicated QoS is recommended to the user. Qi et al. [118] combine UDDI and OWL-S to describe the context of Web services. In OWL-S, class “process: local” allows the users of OWL-S to define local parameters in terms of their needs. Qi et al. use “process: local” to describe context information. Qi et al. define 6 types of contexts: load of server, performance of server, response time of service, geographical position of client, geographical position of server, and distance between client end and server. Dynamic context can be updated on time. After finding services using semantic matching, Qi et al. use context data to evaluate the quality of services and rank the matching services. Mosefaoui et al. [154] present a Context-Based Service Composition (CB-SeC) service description model. Mosefaoui et al. define an optional part called the context function In the CB-SeC service description model. The context function represents the context of the service (e.g., the current workload of the service) and is shipped with other service description. The context function is used to select the best services from the matching Web service list if there is more than one matching Web service. The value of context function is not known in advance. It needs to be calculated during run time when it is needed.

Abbar et al. [2] provide an approach to recommend services using the logs of an end-user and the current context of the end-user. To select and recommend services, the proposed approach requires historical data which are usually not available in the practice. Blake et al. [19] use an agent to detect the execution of applications and the behavior of human end-users, such as browsing the Internet. Then the agent extracts the context data from applications and end-users’ behaviors. Based on the contextual data, the agent generates a query to search for available Web

services. The agents recommend services by matching the similarity of input and output and the operation name of Web services with the contextual information extracted by the agent.

However, to select and recommend services, those approaches either require historical data which are usually not available in practice, or need to predefine the specific reactions on context using rules which are hard to provide due to the diversity of context types and values in the real world. Our approach can automatically recommend services based on the semantics of context scenarios without needing the historical data or requiring the designer of context-aware systems to provide specific rules. In addition, Chen, Qi, and Mosefaoui's approaches focus on using contexts to select services with high QoS, our approach is intended to detect the functional requirements of end-users and recommend services.

2.5 Service Composition

Programming models for service composition provide the basic framework and composition strategies required to compose services. The goal of research on programming models for service composition is to increase the automation of service composition and improve the quality of composition results.

In this sub-section, we identify the methods used for service composition in 2 categories: 1) Model-driven service composition; and 2) Goal-driven service composition. In addition, we introduce the current research on using context-awareness techniques to support end-users in service composition. Finally, we introduce different techniques in acquiring process knowledge to build business processes for service composition.

2.5.1 Model Driven Service Composition

Model driven service composition uses design data, such as UML diagrams and workflows, as the start point to transform the design data to executable business processes such as BPEL.

IBM WebSphere Business Modeler (WBM) [73] and WebSphere Integration Developer (WID) [74] are representative products of model driven service composition. In the design stage, business analysts use WBM to capture business requirements and model business processes. Business analysts can verify the requirements by simulating the business process. Then business processes are transformed into WID as abstract BPEL processes since BPEL is an industrial standard and is easy for SOA developers to implement the business processes. In the integration stage, SOA developers use WID to add details to BPEL processes, search services and bind services to BPEL processes.

Existing software products such as IBM WBM and WID provide a systematic way to develop SOA systems and generate high quality business processes using model-driven service composition techniques. However the existing products need a lot of manual work and also requires experienced designers as well as SOA developers. To reduce the manual work, research efforts have aimed at increasing the automation of model driven service composition. Given a composition requirement, an abstract BPEL process and relevant services, Pistore et al. [114][115] propose an approach to automatically generate an executable BPEL process. Their approach automatically translates the abstract BPEL process and a set of relevant services into state transition systems. Then Pistore's approach formalizes the requirements of the composite service. Using the state transition systems and the formalized requirements, Pistore et al. can automatically generate a state transition system to satisfy the requirements. Finally, the generated state transition system is automatically translated into an executable BPEL process.

Orriens et al. [107][108] propose a model driven approach to compose services. Orriens et al. define a set of information models (e.g., activity, condition, event, flow, message) to represent the basic building blocks of service composition. They also specify five types of rules to describe relations among information models: structure related rules, role related rules, message related rules, event related rules, and constraint related rules. To compose services, Orriens et al. use the following steps: define abstract composition, schedule composition, construct composition, and map constructed composition to executable composition. In the phase of defining abstract composition, the system receives requests from users and determines the relevant activities based on requested functionalities. Business rules are defined to constrain activities and messages among activities. In the scheduling phase, Orriens' approach derives relations among relevant activities and messages using business rules and the input and output of activities. The approach produces different composition schedules and asks the application developers to select the best composition schedules. The composition construction phase constructs a concrete process and then assigns concrete services to activities. Finally, the concrete process is mapped to an executable process and is executed on a process engine.

To consider the QoS in service composition, Gao et al. [55] propose a method to dynamically compose Web service with high QoS using Markov Decision Process. A Markov Decision Process treats the decision problem as discrete time stochastic control process. A Markov Decision Process includes:

- A finite set of states: S
- A finite set of actions: A
- Transition function: $\Phi: S \times A \rightarrow \pi(S)$. The transition function maps each action in a state to a probability distribution over S for the possible resulting state.

- $R(s, a, s')$ is the reward when the system transits from s to s' under the action a .

The solution to a Markov Decision Process (MDP) is expressed as a policy which maps each state to an action. The goal of MDP is to find a policy that maximizes the total expected reward. In Gao's work, composite Web services are defined as an abstract representation of a business process. The task nodes in a business process are independent of any concrete Web services. A QoS evaluation function is defined and treated as the reward in the Markov Decision Process. The QoS evaluation function considers the cost of invoking services, response time, reliability, and availability. Markov Decision Process is used to select the appropriate Web service for each task based on the overall QoS in the entire business process.

2.5.2 Goal Driven Service Composition

In the goal driven service composition, the requirements of service composition are provided as a goal. The goal is expressed in a declarative way which could use both formal and informal languages. Process models are created on-the-fly to realize the goal.

Most approaches in this category treat service composition as an Artificial Intelligence (AI) planning problem [122]. A planning problem can be described as a quintuple $\langle S, S_0, G, A, \Gamma \rangle$, where S is a set of all possible states; $S_0 \subset S$ represents the initial state of the world; $G \subset S$ represents the goal state of the world which the AI planning system tries to achieve; A is a set of actions that the AI planning system can perform; and $\Gamma \subseteq S \times A \times S$ defines the precondition and effects for the execution of each action. For example, $S_1 \times A_1 \times S_2$ means that after executing action A_1 , the system transits from the state S_1 to state S_2 . In service composition, the target state is an end-user's goal; actions are available services and Γ is the effect of executing services. The goal

of AI planning techniques is to find a path from the initial state to the target state as illustrated in Figure 2-7.

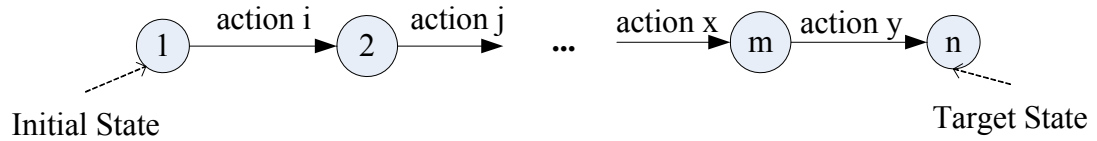


Figure 2-7 The goal of AI planning

2.5.2.1 Hierarchical Task Network Planning

Hierarchical Task Network (HTN) planning is a type of AI planning techniques that creates a plan by decomposing a task (i.e., goal). HTN planning decomposes a task into smaller sub-tasks until all the tasks can be achieved by the primitive tasks (i.e., existing services). Three types of tasks are defined in HTN planning: goal task which is the desired properties of the final state; primitive tasks that can be achieved by invoking available actions (i.e., existing services); and compound tasks that can be fulfilled by composing of a set of primitive tasks and other compound tasks. The compound tasks enable planning systems to add domain knowledge for decomposing complex tasks into simple tasks.

Wu et al. [144] use SHOP2, which is an HTN planner, to automatically compose DAML-S Web services. SHOP2 plans for the tasks in the same order as the tasks later be executed, which ease the transition from SHOP2 to processes. SHOP2 uses operators, methods and various non-action related facts and axioms to capture the domain knowledge. A SHOP2 operator describes inputs, preconditions, effects of executing a primitive task. A SHOP2 method describes the way to decompose a compound task into partially ordered subtasks. Wu et al. translate DAML-S models into the SHOP2 domains, and use SHOP2 to compose services. The domain knowledge

of SHOP2 is converted from atomic processes and composite processes defined in DAML-S models.

Paolucci et al. [95] use the RESTSINA planner to create autonomous Web services which can automatically discover and interact with other Web services. Similar to SHOP2, RESTSINA is a planner based on the HTN planning paradigm. The RESTINA planner allows the Web service to execute before a plan is completely generated. Therefore, the planner can react to the real time situation. For example, if a Web service does not give any response, the planner is able to look for an alternative Web service. However, Paolucci et al. do not reveal the details of how the HTN planning is used in the system.

2.5.2.2 Situation Calculus

In the situation calculus, the state of the world is described by functions. Situation calculus has three basic elements: action, situation and fluent. An action defines a task that can be performed in the world. For example, *call-taxi(me, home)* is an action which means “I (*me*) perform the task *call-taxi* at *home*”. A situation represents a history of actions performed from a given situation. A fluent is a state variable and the value of a fluent is subject to change over time. For example, *do(pay-driver(A), do(ride(A, s), do(call-taxi(A, s₀)))* represents the sequence of actions: [*Call-taxi(A), ride(A), pay-driver(A)*]. There are also a set of formulae in the situation calculus to describe action preconditions, action effects, and state transit axioms.

GoLog is a programming language built on the situation calculus. McIlraith et al. [96] [97][102] extend GoLog to compose services automatically. They convert Web services into actions based on the semantic service description model DAML-S. The process model defined in DAML-S are used to transform a complex action to a set of primitive actions. By extending GoLog, individuals can customize the GoLog program by specifying personal constraints. The

extension enables the service composer to add additional constraints to the system and customize the composition results.

2.5.2.3 Interface Matching and Backward-chain Reasoning

A conversation is an exchange of messages between participants involved in joint services. By examining the conversation among services, we can compose services together to form a composite service to satisfy the required inputs and outputs.

Carlson et al. [24] provide an approach to progressively compose services based on the interface matching. Given a service, they extract semantic description of the output interface from the service, and use the output description to match the inputs of existing services in the repository. The next potential component can be identified by matching the input and output descriptions. A user can finally compose an executable business process based on the recommended components.

Arpinar et al. [11] use inputs and outputs matching to compose a business process. In the approach, DAML-S Web service ontology and process ontology are needed to describe the interfaces of services and the relationships among services. The procedure of service composition is to find the best path which can convert a set of inputs provided by end-users to a set of outputs desired by end-users. A service is treated as a node to convert the input of the service into the output of the service.

Conversation-driven service composition only considers the interfaces and input and output data of services. However, the input and output data of services cannot always reflect the full functions of services. Thus, this type of approach might ignore the functions of some services. To improve the conversation-driven service compositions, some approaches adopt both the interface matching and the semantic descriptions of Web services to compose services.

Sheshagiri et al. [126] present a backward-chain planner that composes services described in DAML-S into a composite service. The action (i.e., service) consists of inputs, preconditions, outputs and effects of services [25]. The planner starts from the final goal and repeats the following two steps using backward chaining to trace the required actions (i.e., services): (1) Find services that fulfill the existing goal and save the service in a set, and (2) Convert the inputs and preconditions of all the services in the set into new goals, and go to step (1) until all the inputs and preconditions are satisfied or provided by the initial state.

Similar to Sheshagiri's work, Ma [90] uses an ontology to decompose goals into sub-goals. Ma searches service repository to find matching services for each sub-goal. The relations of those matching services are automatically found using partial-order planning which is a backward reasoning AI planning technique. Then, the relations are matched with workflow patterns and matching services are composed as a business process using the workflow patterns. Finally, the business process is converted into a BPEL process and is executed in a process engine.

2.5.2.4 Comparison

Table 2-4 summarizes and compares different service composition approaches used in goal-driven service composition. HTN planners (e.g., SHOP2 and RESTSINA) and situation calculus use the top-down approach to compose services. HTN planners and situation calculus start from the overall goal and keep decomposing the goal into small tasks until the planner finds existing services to achieve the tasks. By importing the domain knowledge to tell the planner how to decompose tasks, HTN planners and situation calculus can handle the composition requirement in different domains. Backward-chain reasoning uses the preconditions, input, output and effects of Web services to compose a process. Comparing with HTN planners and situation calculus, purely

backward-chain reasoning approaches are difficult to use domain knowledge during service composition.

Most existing goal driven service composition techniques use semantic services description models and assume that the service providers provide the required semantic descriptions of services. However, the formal semantic descriptions are not generally available on the Internet and the majority of available services on the Internet are used WSDL. The approaches listed in Table 2-4 have limited support for dynamic services composition when there is no formal semantic description available. Instead of requiring semantic descriptions of Web services, in this thesis, we present an approach that is designed based on the existing industrial standard WSDL services which do not have formal semantic descriptions. Our approach uses the knowledge in ontologies to dynamically identify required tasks and generates ad-hoc processes.

Table 2-4 Comparison of approaches used in goal driven service composition

Authors or tool name	Involved techniques	Dynamic process generation	Based on semantic services description	Generate BPEL Process	Automated service discovery	Failure recovery	References
Wu et al.	SHOP2	√	√	×	*	-	[144]
Vukovic and Robinson	SHOP2 and Context-aware	√	√	√	*	-	[137]
Paolucci et al.	RESTSINA planner	√	√	√	√	√	[95]
McIlraith et al.	situation calculus	√	√	×	×	-	[96][97][102]
Sheshagiri et al.	backward-chain planner	√	√	√	*	-	[126]
Ma	Backward-chain reasoning	√	√	√	*	-	[90]

Legend: × means not support; * means partial or proposed support, √ means full support, - means unknown.

2.5.3 Context-Aware Service Composition

Context-aware service composition uses contextual data to improve the quality of composition results. Most approaches in context-aware service composition are designed by extending the model-driven, goal-driven or data-driven approaches.

Vukovic and Robinson [137] employ SHOP2 to compose web services which are aware of the context changing. Vukovic and Robinson use the SHOP2 planner to define different SHOP plans corresponding to different context scenarios. Given a context scenario, the SHOP2 planner can provide a plan which reflects the requirement of the context. The SHOP2 plans are then translated into BPEL processes.

Hesselman et al. [68] present a platform to dynamically discover and compose services in response to context changes in pervasive computing environments. The platform consists of a composition component, a context-aware discovery component and a basic discovery component. The composite service is not a business process. It collects a set of services and allows clients to subsequently access these services. To discover services, the platform defines a registration interface to register services, and a discovery interface to find services by matching the discovery requests with the description of registered services. The discovery interface supports two ways to find services: active discovery and passive discovery. In active discovery, clients request the platform to search for a specific service. In passive discovery, clients wait for the platform to push services to them.

2.5.4 Comparison of Service Composition Techniques

In this sub-section, we classify different service composition approaches into three categories shown in Table 2-5 based on the form of requirement descriptions.

Model-driven service composition uses abstract business processes to express the requirements of service composition, most approaches in model-driven service composition cannot fully automatically compose services.

Goal-driven service composition describes the requirement in a declarative way. It makes the requirement description easily and can automatically compose services. However, except the interface matching approach which uses the input and output data of services to find the relevant services and then compose services, goal-driven service composition usually does not enable the interaction of users. The composition result highly depends on the knowledge defined in the system.

Context-aware service composition takes the contextual data into consideration while composing services. Usually, context-aware service composition is achieved by enhancing an approach belonging to the above two categories.

Table 2-5 Comparison of service composition technologies

Composition technologies	Requirement description	Require semantic service descriptions	Automatically generate business process	Interface mediation
Model-Driven	Abstract business process	Depends on the specific approaches	no	Manually
Goal-driven	in a declarative way	Most approaches require	Fully automatic or semi-automatic	Interface matching or do not mention
Context-Aware	Depends	Need contextual data of services	depends	Depends

2.5.5 Supporting End-Users in Service Composition

Mashups are browser-based applications. End-users can easily access Mashup applications using Web browsers (*e.g.*, Internet Explorer and Firefox) without installing any software on the client side. Several products, such as Yahoo! Pipe [155] and IBM Mashup center [72], provide an end-user friendly environments for end-users to manually connect Web resources into one Web page. Such environments are easy for non-IT professional end-users to learn and to manually compose services. However, those products require end-users to manually identify all the services to form an ad-hoc process. Our approach reduces the workload of end-users by automatically generating ad-hoc processes for end-users. Liu *et al.* [85] propose a Mashup architecture which extends the SOA model with Mashups to facilitate service composition. As discussed in section 2.5.2.3, Carlson *et al.* [24] provide an approach for end-users to progressively compose services based on the interface matching.

The project Ubiquitous service composition for all end-users (*i.e.*, UbiCompForAll) [134] provides support for non-IT professionals to compose services. UbiCompForAll conducts an initial experiment to evaluate a service composition tool which creates mobile tourist services for end-users. UbiCompForAll uses different case scenarios relevant to the city guide to develop and validate the end-user interfaces of the service composition tool. However, no details and concrete results on providing support for end-users have been revealed by UbiCompForAll. Obrenovic *et al.* [104] provide a spreadsheet-based tool to help end-users compose services. A spreadsheet (*e.g.*, Microsoft Excel) is a software application that uses rectangular tables to display information. In a spreadsheet, the content is specified in the cells of the table, and the relations among the cells are defined by formulas. Obrenovic *et al.* enable the spreadsheet to exchange messages with services and support different composition patterns.

Different from above approaches, our work enhances service Mashups by providing guidance to end-users through the automatic composition of services. We generate ad-hoc processes to help end-users compose services and fulfill the goals of daily activities.

2.5.6 Process Knowledge Acquisition for Service Composition

Process knowledge provides the required data to transform business process requirements into business processes and compose services. In model-driven service composition, process knowledge is primarily provided by experienced business analysts who have extensive process knowledge. In goal-driven service composition, most approaches are built on semantic Web services which contain process knowledge (e.g., pre-conditions and post-conditions) to reason business processes. However, there is no well-accepted semantic service description model. For example, the two most popular semantic service description models, OWL-S and Web Service Modeling Ontology (WSMO) [140], are not recommended by W3C. In the practice, WSDL is the default service description language. WSDL does not have semantic description and does not have the ability to describe process knowledge. It is challenging for novice business analysts and nonprofessional end-users to identify a complete set of services to orchestrate a well-defined business process due to the lack of process knowledge. Therefore, it is necessary for service composition systems, especially for the systems which are not built based on semantic Web services, to have the required knowledge to compose services. There are three major research areas involved in capturing process knowledge.

Process mining is a technique to extract business process information from event logs recorded by information systems. Agrawal et al. [5] present an approach to construct process models from the log. The approach can generate a process model with the control flow of the business process from the logs of unstructured executions of a process. Francescomarino et al. [53] trace the Web

system executions and analyze the application Graphical User Interface (i.e., the forms and their fields on the client Web page) during the execution to infer processes. Liang et al. [82] provide an approach to mine service association rules from service transactional data. Aalst et al. [1] use Petri nets to model processes and discuss the class of processes which can be discovered from logs, then they propose a mining algorithm to discover business processes. However, the business processes mined by the aforementioned approaches are formally defined and are not used to handle daily activities. Moreover, the event logs for daily activities are distributed on many servers. It is hard to collect the event logs from different places, especially when it involves personal data. In our work, we extract the process knowledge from the publicly available websites. The extracted process knowledge is the information related to daily activities instead of the formal defined business processes.

Business Process Recovery is a technique to extract business processes from the source code of business applications. Zou et al. [159] presents an approach to automatically recover business process definitions from business applications. However, business process recovery techniques need to analyze the source code of business applications running on the server. The source code of business applications is confidential data and generally not available. The approach proposed in this thesis extracts process knowledge from publicly available Web pages without requiring the source code of business applications or event logs.

Information extraction systems transform unstructured documents or semi-structured documents into structured data such as a relational database. Cimiano and Völker [35] develop a framework to learn ontologies from text documents. An ontology learning tool named Text2Onto is developed by Cimiano and Völker in their paper. Chang et al. [27] summarize and compare the existing Web information extraction systems which extract information from semi-structured

documents (e.g., HTML documents). Chang et al. compare the major Web data extraction approaches based on the task of the IE systems (i.e., the types of input documents and the extraction targets), the automation degree, and the techniques used (e.g., extraction rules, approach to assemble the extracted values). Yoshida et al. [158] provide an unsupervised learning method to extract ontologies from tables shown the Web pages. However, the aforementioned approaches are designed to extract general information from documents instead of the process knowledge, which makes us difficult to use the extracted information to generate processes and compose services. Liu and Agah [88][89] develop a process-based search engine to search process knowledge from the Web. The results of the search engine are text description. Their process-based search engine can retrieve the processes explicitly published on the Web, such as www.eHow.com, www.wikiHow.com and www.howtodothings.com. However, their approach can only extract the process knowledge from the Web pages with explicit process information. Meanwhile, the output of Liu and Agah's work is unstructured text description about a process. It is difficult to be further processed by machines. Hoxha et al. [69] provide an approach to extract semantic descriptions of processes in the Web. Hoxha's approach fills the forms on Websites, and recovers the process following the submission buttons step by step dynamically. However, to automate Hoxha's approach, it requires automatic approaches to fill online forms, which are difficult to achieve nowadays and may impact the execution of the online applications. Our approach analyzes the static Web pages and does not require the generation of input data. By extracting the process knowledge from the online applications, we can collect the practical process knowledge from the Web.

2.6 Summary

This chapter investigates and compares the research related to service composition. We summarize different service description models and discuss the approaches used in service discovery. Different programming models for service composition and approaches for acquiring process knowledge are also presented and compared. In addition, we highlight our contributions in these areas by comparing our approaches with existing work.

Chapter 3

Overview of a Framework for Supporting End-Users in Service

Composition

To shelter end-users from the complexity of service composition, we propose a framework that supports non-IT professional end-users to dynamically compose services and recommends services to meet their situational needs. Section 3.1 presents an ad-hoc process model. Section 3.2 describes our framework. Finally, Section 3.3 summarizes this chapter.

3.1 An Ad-hoc Process Model

Service composition languages, such as Business Process Execution Language (BPEL) [139], are designed for SOA professionals to assemble services to form well-defined business processes. Service composition languages require very formal descriptions in different perspectives, such as variables, control flow, fault handling, and service binding. Although BPEL process modeling tools are provided to visualize service composition languages, such as IBM WebSphere Integration Developer (WID) [74], Oracle BPEL Process Manager [105] and ActiveVOS Designer [3], those tools are designed for SOA professionals instead of non-IT professional end-users. For example, the tools require end-users to understand different components of BPEL before the end-users can design a BPEL process. Moreover, the ad-hoc processes needed by end-users for daily activities generally compose services in a loose way without strict execution order which cannot be described by existing service composition languages. For example, when planning a trip, the end-user can buy the flight ticket first then book a hotel, and vice versa.

To describe the loosely coupled services in ad-hoc processes, we propose an ad-hoc process model. The ad-hoc process model describes the tasks performed by end-users to fulfill their needs. Each task in the ad-hoc process is associated with a set of services which have similar functionality. To ease the end-users to navigate tasks in an ad-hoc process, we also suggest the possible control flows to reflect the navigational relations among tasks. Figure 3-1 shows the model for representing an ad-hoc process. A task can be associated with more than one service with the similar functions. For example, the task, “Renting a Car”, can be fulfilled by different car rental companies. The end-users can choose their favorite services to fulfill the task. The services fulfilling a task are either directly discovered from a service repository or composed by other ad-hoc processes (i.e., sub-processes). In some cases, a few tasks have to be performed in a particular order. For example, two tasks, “Selecting a Product” and “Adding to a Shopping Cart”, must be performed before the task “Checking out”. To help an end-user perform tasks, we define two basic relations among tasks in our ad-hoc process model:

- *And* relation indicates that all the tasks have to be executed. Some *and* relations can be further specialized as *Sequence* and *Parallel* relations.
 - *Sequence* defines a set of tasks to be executed in a sequential order. For example, the task “Selecting Flight Tickets” needs to be performed before processing the payment for the flight tickets.
 - *Parallel* describes independence of tasks. Tasks in a parallel relation can be executed in any order or at the same time. For example, an end-user can perform “Book a Flight Ticket” and “Check the Weather Forecast” in parallel.
- *Or* relation means that end-users only need to execute one task from a given set of tasks. An *or* relation is further specialized to *Alternative* and *Choice* relations.

- *Alternative* allows end-users to select one task among two tasks. For example, the end-user can select a favorite transportation vehicle from “Car” and “Train”.
- *Choice* defines that end-users can select one task from more than two tasks.

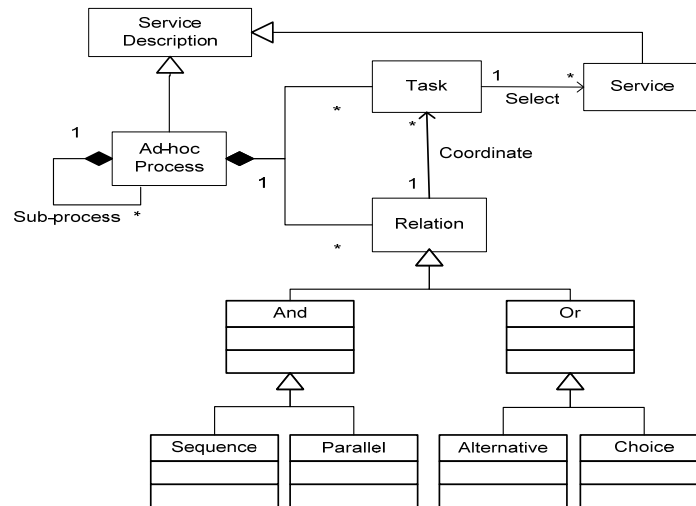


Figure 3-1 Description of an ad-hoc process

3.2 An Overview of a Framework for End-User Driven Service Composition

Figure 3-2 provides an overview of our proposed framework for generating ad-hoc processes and recommending services. To compose an ad-hoc process, an end-user simply describes a desired goal using keywords. Our framework uses keywords to search for matching ontologies with the desired process knowledge. We search for the ontologies from ontology databases, such as Freebase [54]. However, the ontology databases may not contain the matching ontology for the given goal. In our work, we also obtain ontologies by extracting process knowledge from the Web. In this thesis, we use the ontology language OWL to represent ontologies and the extracted

process knowledge is represented as ontologies and is passed to the ad-hoc process generation component to generate ad-hoc processes for end-users.

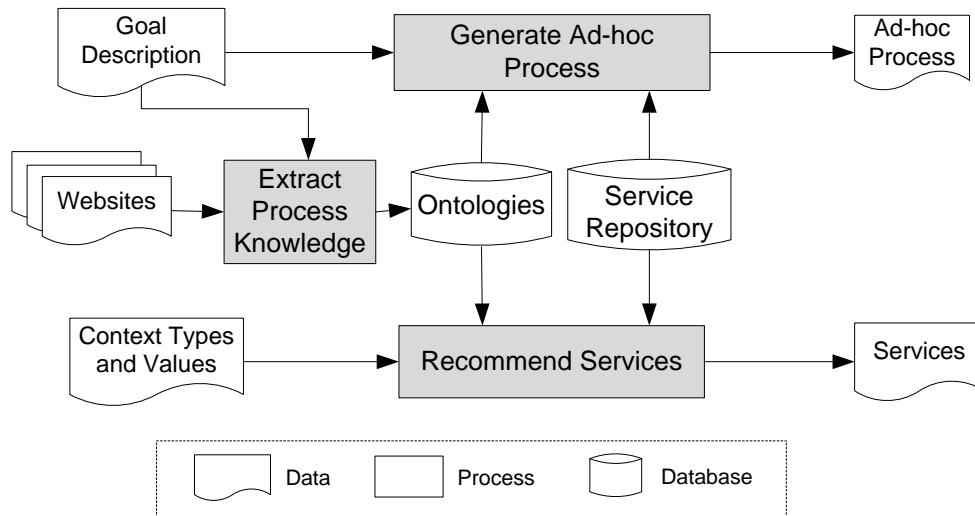


Figure 3-2 An overview of our framework

In an ontology, the semantic of a high-level goal is expanded into more concrete concepts. We use the concepts as keywords to search for services in a service repository. To facilitate the reuse of the services when the same goal re-occurs, we abstract the discovered services into tasks and aggregate tasks into an ad-hoc process. The ad-hoc processes are stored and shared among multiple end-users.

The generated ad-hoc process may not reflect all the needs of end-users since a goal description is usually not able to specify all the situational needs of an end-user. To refine the generated ad-hoc process, we analyze the context of end-users and formulate search criteria to discover and recommend services for end-users. The generated ad-process and the recommended services are displayed in a Mashup page so end-users can modify the process and select the recommended services.

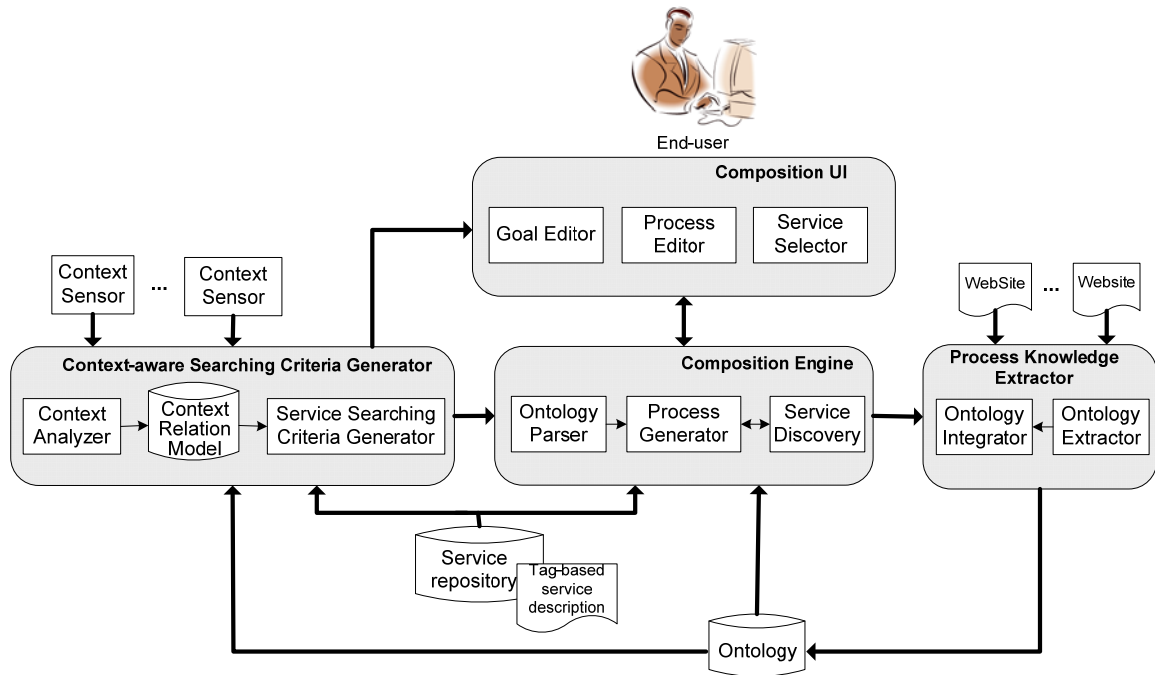


Figure 3-3 Architecture of our framework

Figure 3-3 shows the architecture of our framework. The framework is built using client/server architecture. On the client side, end-users interact with the service composition system through a composition user interface (UI) which is built using Mashup pages. The composition UI provides a user interface to enable end-users to specify the goal, navigate through the generated ad-hoc process, edit the process, and select services. To capture an end-user's context, context sensors are developed to monitor the end-user's activities in their computing environment. We deploy context sensors as plug-ins into various applications, such as Web browsers and on-line calendars.

The server contains three major components that extract process knowledge from the Web, analyze the contextual information gathered from the client, and generate an ad-hoc process. The components are described as follows:

Process knowledge extractor extracts process knowledge from the Web. Such process knowledge is used as the information to understand the goal descriptions from end-users and to automatically generate ad-hoc processes for end-users. More specifically, the process knowledge extractor analyzes the navigation information in a website to identify the tasks needed for completing an ad-hoc process. To provide comprehensive process knowledge for achieving a goal from end-users, the process knowledge extractor merges the process knowledge extracted from multiple websites that serve for the same goal (*e.g.*, travel planning). WordNet groups words into sets of synonyms and connected words via semantic relations. The process knowledge extractor uses WordNet as a global knowledge database to integrate the process knowledge extracted from different websites. Generally, the extracted process knowledge in our framework is the knowledge of ad-hoc processes instead of the well-defined business processes. Business process specification languages, such as BPEL [139] and BPMN [23], are designed to describe business processes with complete information of the process and are not suitable for representing the extracted process knowledge. In our work, we use ontologies to represent the extracted process knowledge.

Context-aware searching criteria generator captures and analyzes the changing context scenarios of an end-user. This component automatically formulates searching criteria to discover the desired services in the service repositories.

A context can be described as a set of pairs of context types and context values. A context type describes a characteristic of the context. A context type is associated with a specific context

value. For example, the context types for an end-user include location, identity, and time. “New York” is a context value for the context type “location”. Furthermore, a context scenario is the combination of different context types with specific values to reflect a user’s situation. To recommend services for a context scenario, we propose an approach which can analyze dynamically changing context types and values, and then formulate search criteria to search for and recommend services for end-users. Different from existing approaches which depend on static context models to know the relations among context types or values and use predefined rules to infer end-user’s needs, we seek an automatic approach to recognize the relations between context values and an end-user’s needs. For example, luxury hotel and budget hotel are two user’s potential needs in conflict. Therefore, the services for booking luxury hotels are automatically filtered when the context shows that an end-user has limited budget. We expect that such relations can be used to express more accurate searching criteria which better reflect an end-user’s context. When a new value for an end-user is detected, our approach automatically computes the relations between the new context value and other context values. Instead of manually defining if-then rules using specific context types or values as the traditional context-aware systems [1], our approach automatically identifies the relations among context values to infer end-user’s needs. Then we generate service searching criteria based on an end-user’s needs in order to discover and recommend services.

Composition engine receives the goal of service composition from end-users and automatically composes an ad-hoc process. We develop an approach for the composition engine to dynamically compose ad-hoc processes. In our approach, instead of specifying the complete tasks for fulfilling a goal, an end-user is required to describe a high level goal using keywords. For example, for planning a trip, an end-user can specify the keywords, such as “travel to Vancouver”. To

understand the semantic meaning of an end-user's goal, we use ontologies to expand the semantic meanings of the keywords. Our approach uses the concepts in the ontology as keywords to search services from service repositories. The service repository allows service providers to advertise their services and provide interfaces for automatic service discovery. To enable end-users and composition tools to understand the properties of Web services, we use descriptive tags (i.e., keywords) to describe services. The detailed information of the tag-based service description schema is described in Chapter 6.

In our framework, the aforementioned three components are collaborated together to help end-users compose services and fulfill the goals of their daily activities. For example, if an end-user wants to plan a trip to Los Angeles, the end-user submits the keywords "Travel" or "Plan a trip" to the composition UI. The composition engine uses the keywords from the end-user to search for the matching ontologies from an ontology database (e.g., Freebase). If the composition engine cannot find a matching ontology of "Travel" from the ontology database, the composition engine invokes the process knowledge extractor which extracts the ontology of "Travel" from the Web by analyzing the websites related to "Travel". The details of extracting process knowledge from the Web are described in Chapter 4. Then the composition engine generates an ad-hoc process based on the ontology of "Travel". The ad-hoc process includes tasks such as "Flight Ticket Reservation", "Hotel Reservation" and "Tourism Attractions". Chapter 6 presents the algorithm to generate ad-hoc processes using ontologies. In addition to the ad-hoc process of "Travel", the end-user may need other services to satisfy the specific needs of traveling to Los Angeles. For example, if the end-user likes National Basketball Association (NBA) games, she or he might hope to know the information (e.g., game schedule and location) related to "Los Angeles Lakers" which is a NBA team in Los Angeles. The context-aware searching criteria generator in our

framework can analyze the context of the end-user and recommend services (e.g., game schedule and location of Los Angeles Lakers) based on the context of the end-user. Chapter 5 describes the details of our approach to recommend services based on the context of end-users.

3.3 Summary

In this chapter, we give an overview of our framework that dynamically generates an ad-hoc process and recommend services to meet the situational needs of end-users. Our framework automatically extracts the process knowledge from the Web to gather the required knowledge for service composition. To compose an ad-hoc process, an end-user only needs to use a few keywords to describe her/his goal and our framework can automatically generate the ad-hoc process for the end-user. In addition, our framework uses ontologies to extend the semantic meanings of context values and identifies an end-user's needs hidden in the context values to recommend the desired services. To describe the loosely coupled task relations for fulfilling the goals of daily activities, an ad-hoc process model is also proposed in this chapter.

Chapter 4

Process Knowledge Extraction

To leverage the domain knowledge embedded in specialized websites, we propose an approach to extract the process knowledge from such websites. Our approach attempts to make the process knowledge available for end-users to use in service composition. More specifically, we analyze the navigation information in a website to identify the tasks needed for completing an embedded process. To provide comprehensive process knowledge for achieving a goal, our approach merges the process knowledge extracted from multiple websites that serve for the same goal (*e.g.*, travel planning).

This chapter is organized as follows. Section 4.1 presents a meta-model for websites which summarizes the common structures of websites. Such a meta-model captures the data related to process knowledge regardless of the diversity in the design and implementation of various websites. Section 4.2 discusses the steps of our approach that extracts process knowledge from multiple websites. Section 4.3 and Section 4.4 provide the algorithms to extract process knowledge from a website. Section 4.5 discusses our approach for integrating process knowledge extracted from different websites. Finally, Section 4.6 summarizes the chapter.

4.1 A Meta-model for Describing Websites

Due to the diversity in the design and implementation of various websites, the appearance of the same information can be presented differently in various websites. For example, the navigation information (*i.e.*, the menu) of a website can be described as a HTML table [119], a HTML list [119], or parallel paragraphs with different fonts and sizes. To deal with such diversity

in the website design and implementation, we summarize the common structures of websites using the meta-model shown in Figure 4-1.

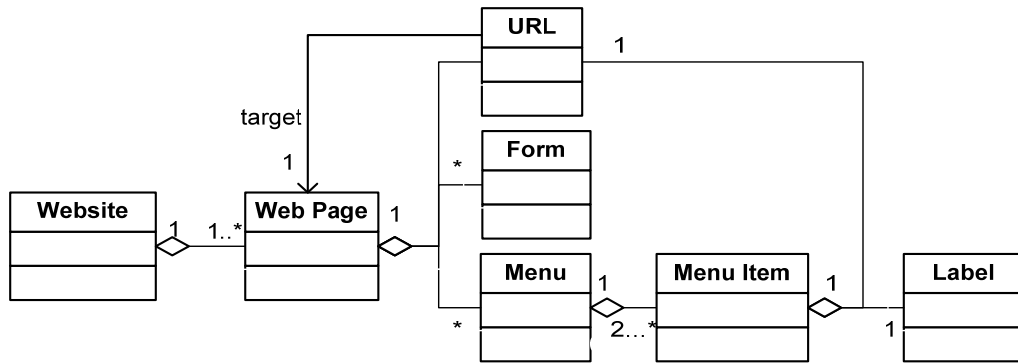


Figure 4-1 A meta-model for describing websites

Generally, a website contains a collection of related Web pages. Each Web page has a Uniform Resource Locator (URL) to indicate the address of the Web page on the Internet. Forms in a Web page are used to collect inputs from end-users. A menu is intended to guide end-users to navigate through different Web pages in the website. A menu contains a group of menu items that link to different Web pages where an end-user can conduct tasks, such as selecting a product. Each menu item contains a label and a URL. The label shows the name of a menu item. The URL is a link to a Web page. Menus can be implemented in different ways, such as HTML table, HTML list or a set of sequential HTML hyperlinks. Figure 4-2 illustrates an example of a website with a menu. The menu is represented using HTML list tags, *i.e.*, ` `. Essentially, the menu items indicate a set of tasks that an end-user needs to perform in order to complete one or more processes. The process knowledge can be captured in the menus.

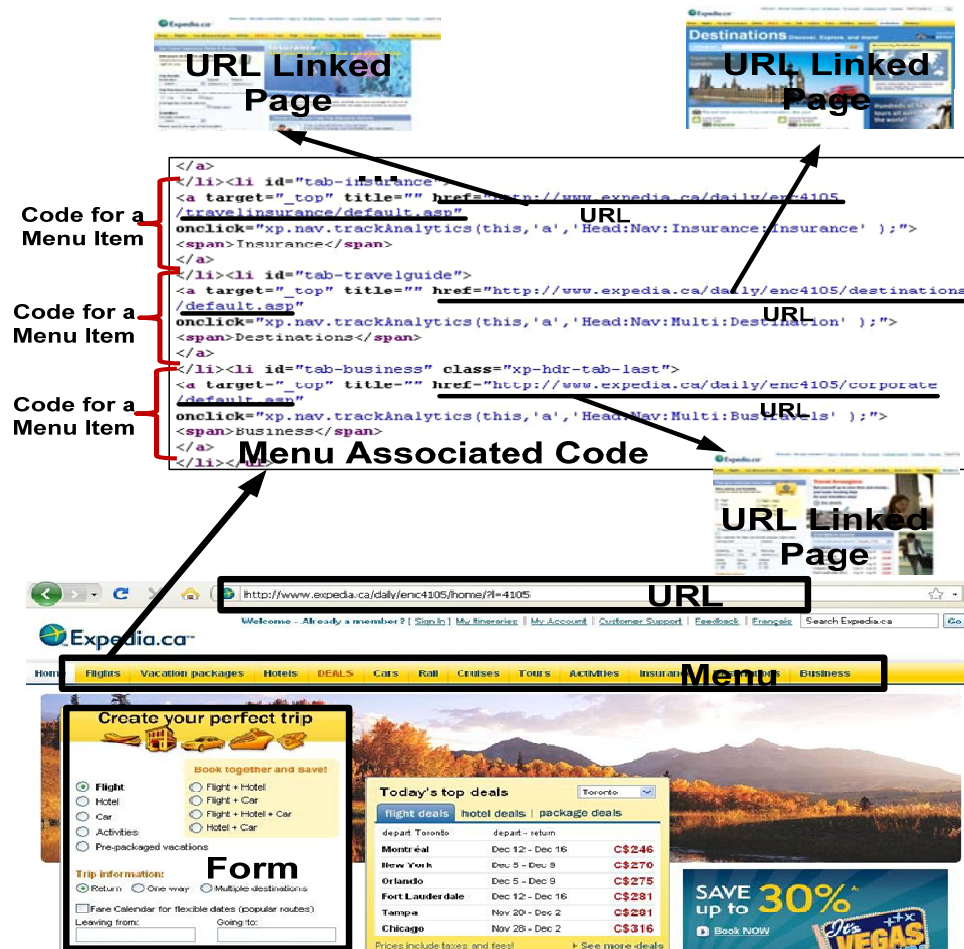


Figure 4-2 An example of a Web page

4.2 Steps for Extracting Process Knowledge from the Web

Figure 4-3 illustrates the steps that extract process knowledge from Websites. As shown in Figure 4-3, the goal of an ad-hoc process can be described using a phrase or a set of keywords, such as “travel” and “apply credit cards”. We submit the goal to an existing Web search engine, such as Google, to search for relevant websites. However, not all the websites retrieved from a search engine encode rich process knowledge. In our approach, we analyze the semantics of menus in websites and use the semantics of menus to select the websites that provide the required

process knowledge. Then we recover tasks and sub-processes from the menus in the selected websites. We also capture the tasks and properties of the sub-processes. Finally, we integrate the process knowledge extracted from different websites to form a more comprehensive ontology that elaborates process knowledge of a given goal.

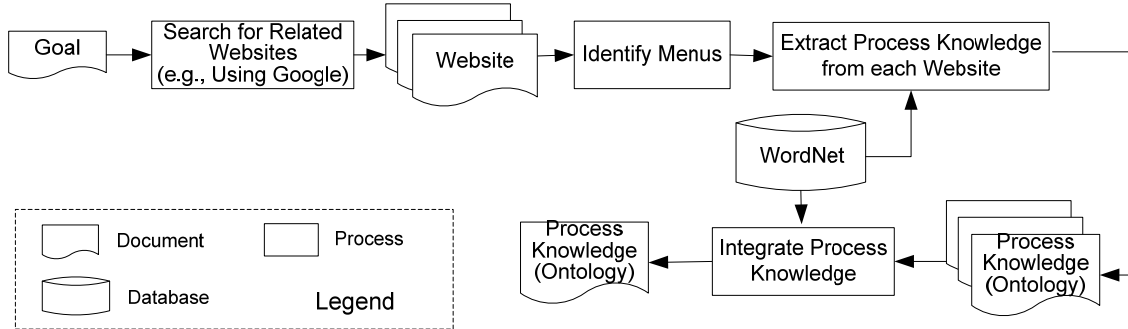


Figure 4-3 An overview of our approach

4.3 Algorithm for Identifying a Menu from a Website

We propose an algorithm as shown in Figure 4-4 to describe the steps for identifying menus in a website. A Web page often contains advertisements irrelevant to the objective of the website. To filter out such noise, we apply the approach proposed by Gupta *et al.* [62] by manually registering the URLs of the well-known advertisement service providers. We parse the HTML document of a Web page to analyze the values of “src” and “href” attributes in a HTML node. We remove the HTML nodes if the source or hyperlink (*i.e.*, “src” or “href”) attributes of the HTML nodes refer to common advertisement servers. From line 4 to line 17 in Figure 4-4, our algorithm traverses the tree structure of an HTML document from the root node “<HTML>” using breadth-first tree traversal algorithm. The queue data structure is used as a temporary storage to facilitate the tree traversal. Initially, the root of a HTML document (*i.e.*, <HTML> tag)

is pushed into the queue. When the queue is not empty, a node is de-queued for further analysis.

If the node is not a menu, we push the children of the node into the queue.

```
Function: identify_menu
Input: HTML code of a Web page
Output: a list of identified menus
1. Remove advertisements;
2. curr_node = root node of the input HTML;
3. queue = empty;
4. queue.push(curr_node);
5. while(queue is not empty){
6.   curr_node = queue.pop();
7.   if (curr_node has children){
8.     child_list = children node of curr_node;
9.   }
10.  else { continue;}
11.  if(child_list satisfies the features of menu items){
12.    add children to the identified menu list;
13.  }
14.  else {
15.    add the nodes in child_list to queue;
16.  }
17.} //End the while loop
18. Output the identified menu list;
```

Figure 4-4 Algorithm to identify menu items

We identify a menu from an HTML node if the child nodes of the HTML node are identified as menu items. In particular, the child nodes of the HTML node satisfy the following features:

- Menu items are sibling HTML nodes with identical HTML structures. For example as shown in Figure 4-2, each menu item is an element of the HTML list tag, *i.e.*,
- A set of menu items are encapsulated by the same parent HTML tag. For the example shown in Figure 4-2, the menu items are encapsulated by the parent HTML tag
- A menu item contains a URL with a short descriptive text (*i.e.*, label) displayed on a Web page.

As shown in Figure 4-2, the URL of a menu item is represented as a HTML href attribute

which links to another Web page. The label of the menu items in the example is surrounded by the HTML tag `...`.

- Identical menu items exist in other Web pages linked by the URLs of menu items. In the example shown in Figure 4-2, the target Web page contains the same navigation menu items.

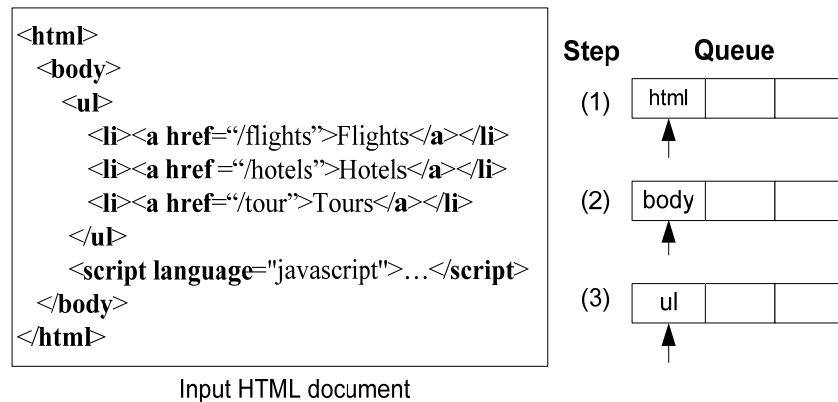


Figure 4-5 An example to identify menu items

Figure 4-5 is a simplified example to illustrate the major steps for identifying menus. In step 1, the algorithm pushes the root node “<html>” into the queue, and then pops the first node in the queue to check if it is a menu. The child node “<body>” does not have sibling nodes. Node “<html>” cannot be identified as a menu since the child of the node “<html>” does not satisfy the features of menu items. Therefore, we push the child nodes of “<html>” into the queue for further analysis. In step 2, we pop the next node (*i.e.*, node “<body>”) from the queue to check whether it is a menu. Similarly, the node “<body>” is not a menu since its child node “” does not have sibling nodes. We add the child node “” to the queue. In step 3, we pop the node “” from the queue. Assuming that this list node “” appears in all the three linked Web pages (*i.e.*, the Web pages with URLs of “/flights”, “/hotels”, “/tour” which contain the same node

“”). The children of the list node satisfy the features of menu items. Therefore, the algorithm identifies the children as menu items and recognizes the node “” as a menu. The algorithm is terminated once the queue is empty.

4.4 Extracting Ontologies with Process Knowledge

In this section, we discuss our approach that analyzes the semantics between the goal and the menus of a website in order to select the websites with the desired process knowledge. Then we present our algorithm for extracting process knowledge from each selected website.

4.4.1 Selecting Websites with Process Knowledge

Websites usually use a menu to guide an end-user through each step of an ad-hoc process. Quite often, a website without menus provides only simple services without detailed process knowledge. Such a simple website is filtered. Moreover, a website may contain more than one menu. Some menus are used to represent general information instead of the desired process knowledge to meet the goal. For example, a menu which contains menu items, such as “Home”, “Contact”, “About”, and “Login” are used by many websites. This menu does not reveal any process knowledge relevant to the goal. In our approach, we select the Websites that have at least one menu in a website relevant to the goal. To identify the relatedness between a menu and a goal described by keywords, we propose a metric, *average Semantic similarity degree*, to measure the semantic similarity between a goal and a menu. We apply the approach proposed by Wu and Palmer [145] to calculate the *similarity degree* of words which are the basic elements of a menu and an end-user’s goal. Wu and Palmer’s approach calculates the similarity degree based on the path length (i.e., the number of words connecting one word to another) between words according to the word relations defined in WordNet. The similarity degree is standardized into a value

between 0 and 1 (including 0 and 1). A short path length between two words means a high *similarity degree* since short path length indicates that these two words are closely connected to each other. For example, the similarity degree between word “Travel” and word “Travel” are 1 since these two words are exactly the same and the path length of these two words is 0. *Average semantic similarity degree* measures the average value of the *semantic similarity degrees* between the label of each menu item and the goal, as defined in Eq. (4-1):

$$Average_Sim = \frac{\sum_{i=1}^n sim(goal, label_i)}{n} \quad (4-1)$$

Where n is the total number of menu items in the menu; and $label_i$ represents the label of the i -th menu item.

To ensure that one of the menus in the website is relevant to the goal, we sort the identified menus based on the *average semantic similarity degree* from high to low. If the highest average semantic similarity degree is greater than a threshold, such a website is relevant to the goal. Otherwise, the menus identified from a website are not related to the goal. Therefore, such a website is filtered.

4.4.2 An Algorithm for Extracting Process Knowledge from a Website

We propose an algorithm for extracting process knowledge from the selected website. As listed in Figure 4-6, the input of the algorithm is a goal description, a website (*i.e.*, a collection of Web pages in a website) which contains process knowledge, and a set of menus identified from the website. The extracted process knowledge is represented as an ontology. The goal description is created as a root class for the ontology as shown in line 2 in Figure 4-6. The menu with the highest average semantic similarity degree is converted to a set of classes. Each menu item in the

menu is converted to a class. The converted classes are added to the ontology as child classes of the root class (as shown in lines 3 and 4). The remainder menus with lower *average similarity degree* may describe the details of a class in the ontology. For the example of a “travel planning” website, the menu with the highest average similarity degree may contain a menu item “flight”. Another menu in the website may contain menu items such as “business class”, “economy class” and “airport lounge”. The latter menu (i.e., “business class”, “economy class” and “airport lounge”) provides the detailed information for the menu item “flight”. By comparing the remainder menu items with the existing classes in the ontology using the word relations provided by WordNet, we can find the relations between the remainder menu items and the classes in the ontology. If our algorithm identifies a “subclass” or “PartOf” relations between a menu item from the remainder menus and a class in the ontology, such a menu item is created as a new class and added into the ontology.

The class relations, such as “subclass” and “PartOf”, are not explicitly specified in the website. We need an approach to identify the class relations when extracting classes from websites. We use WordNet [116] to identify class relations using the following word relations defined in WordNet.

- **Hypernym** represents a “kind of” relation. For example, car is a hypernym of vehicle. The hypernym relation is converted as a subclass relation in the ontology.
- **Hyponym** means that a word is a super name of the other. For example, vehicle is a hyponym of car. Hyponym is the inverse of hypernym. In the ontology, a super class indicates a hyponym relation.
- **Holonym** describes a whole-part (*i.e.*, partOf) relation. For example, a building is a holonym of window.

- **Meronym** is the inverse of holonym and represents a part-whole relation. For example, a window is a meronym of a building. A meronym relation is converted to a PartOf relation in the ontology.

```

Function: extract_process_knowledge
Input: G —goal description
          W —a website with process knowledge
          menu_list — menus identified from website W and
                    sorted based on average semantic similarity
                    degree from high to low
Output: On —an ontology with process knowledge
1. { On = Create an empty ontology;
2.  root_class = Create a root class in On using G;
3.  curr_menu = the first menu in menu_list;
4.  Covert curr_menu to subclasses of root_class;
5.  for each menu item in the remainder menu_list {
6.    new_class = convert the menu item into a class;
7.    Identify relations between new_class and the
        existing classes in On;
8.    if(exist a relation between new_class and the
        classes in On){
9.      add new_class to On based on the relation;
10.   }
11. }//End for loop
12. Extract properties for classes in On;
13. Output ontology On;
14. }

```

Figure 4-6 An algorithm for extracting the ontology from a website

WordNet can be used to identify some semantic related relations. For example, flight is a kind of transportation. Due to the lack of domain knowledge, the relations between two words cannot be recognized by WordNet when the two words are related in ad-hoc processes but do not have strong semantic relations. For example, in the process “travel”, word “hotel” can have a partOf relation with “travel”. However, WordNet cannot recognize such relations. In addition, an

ad-hoc process may use phrases (*i.e.*, more than one word) to describe tasks or the input and output of tasks. For example, “first class”, “business class” and “economy class” could be the input parameters of task “searching for flight tickets”. WordNet is designed as a general lexical database and does not have the capability of recognizing phrases.

As aforementioned, each label in a menu item is associated with a URL which indicates a path for the Web page to be retrieved from the server. A path shows the hierarchical structure for organizing the linked Web pages in different menu items. For example, a “flight” menu item is linked to <http://www.flightcentre.ca/flights> and a “business class” menu item is connected to <http://www.flightcentre.ca/flights/business-class>. The information related to “business class” is stored under the directory of “flights”. The organization of the directory structure suggests *partOf* relations between the two menu items (*i.e.*, “flight” and “business class”). We can infer that entity “business class” is a child of entity “flights” with *PartOf* relation since the URL of “business class” is in the sub-directory of the URL of “Flights”.


4.4.3 Extracting Properties and Tasks from Associated WebPages

In the extracted ontology from a website, a class in the ontology is mapped to a menu item in the website, and therefore a class is associated to a linked Web page by a URL. We further analyze the linked Web pages to recover the properties and children for each class.

HTML forms are often designed to take an end-user's input to provide a service to the end-user. In our approach, we extract the HTML forms from the Web page linked to a class defined in the extracted ontology, and check if the title and content of the HTML form is relevant to the class. More specifically, the label of input fields (*e.g.*, text fields, password fields and radio

buttons) are converted to the properties of a class. Table 4-1 lists the mapping between the elements of forms and the properties of classes defined in an ontology.

Table 4-1 Map form elements to class properties

Form element		Class properties and relations
Name	example	
Label with input area	First name: <input type="text"/>	A class property
Radio Buttons	<input checked="" type="radio"/> Male <input type="radio"/> Female	Class properties with an “OR” relation
Checkboxes	<input type="checkbox"/> I have a bike <input type="checkbox"/> I have a car	class Properties
Select list (drop-down list)	Destination: 	Class properties with an “OR” relation
Title of the form	Search Flight Ticket	If the title or submission button is semantically similar to a class name, we convert the title or submission button as a child of the class with <i>PartOf</i> relation, and put all the extracted properties of the form as the properties of the child class.
Submission Button	<input type="button" value="Search Hotel"/>	

4.5 Integrating Process Knowledge Extracted from Different Websites

Each relevant website contains partial information of the process knowledge since the relevant website is designed for a specific group of people (e.g., graduate students, Canadian Residents). To obtain more complete process knowledge, we integrate the process knowledge (*i.e.*, ontologies) extracted from multiple websites. Figure 4-7 presents our algorithm that integrates process knowledge. As a starting point, we use the goal description to create the root class of the integrated ontology. We gradually add the knowledge (*i.e.*, classes, properties and relations) from

an extracted ontology into the integrated ontology. We use the variable *curr_class* to store a current class that is currently analyzed by the algorithm. As shown from lines 7 to 9 in Figure 4-7, starting from the root of the new ontology, we use the current class (*i.e.*, represented by *curr_class*) to search for the matching classes defined in the input ontologies. Two classes are matched if the names of the classes are the same or synonyms as indicated by WordNet. The properties of the matching classes may vary in different ontologies. We merge the properties of the matching classes to the integrated ontology, so that the *curr_class* in the integrated ontology can include all the properties. As shown in line 10 of Figure 4-7, we add the child classes of the matching classes from different input ontologies to the integrated ontology. We recursively merge

```

Function: integrate_process_knowledge
Input: G —goal description
         onto_list — Ontologies extracted from different
                   related websites
Output: int_onto —an integrated ontology
1. { Create an empty ontology int_onto;
2.   create the root class for int_onto using G;
3.   queue = empty;
4.   queue.push(root class);
5.   while(queue is not empty) {
6.     curr_class = queue.pop();
7.     match_cls_list = find classes that match with
curr_class from the input ontologies;
8.     for each cls in match_cls_list {
9.       Integrate properties from cls to curr_class;
10.      Add children of cls to curr_class;
11.      Push children into the queue;
12.    }
13.  } //End the while loop
14.  output int_onto;
15. }

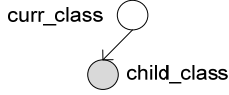
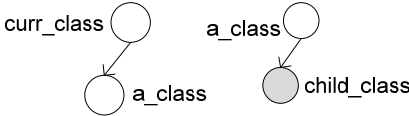
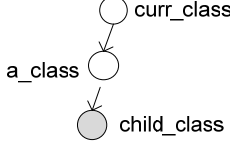
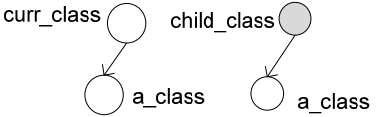
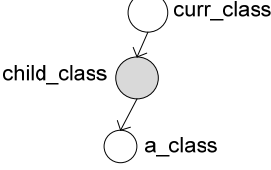
```

Figure 4-7 Algorithm to integrate process knowledge

the child classes of the matching class into the integrated ontology following the conditions listed in Table 4-2. If a child class of the matching class does not exist in the integrated ontology, we

insert the child class into the integrated ontology as a child class of *curr_class* shown in the second row of Table 4-2. If another class in the integrated ontology has a subclass or a partOf relation with the child class of the matching class, we insert the child class into the integrated ontology by adjusting the relations among the three classes (*i.e.*, *curr_class*, *a_class* and *child_class* as shown in Table 4-2). More specifically, as shown in the third row in Table 4-2, if a class is the child of *curr_class* and the parent of *child_class*, we add the class as the child of *curr_class* and as the parent of the *child_class*. As shown in the fourth row in Table 4-2, if a class is the child of both *curr_class* and *child_class*, the class is added as a child of the *child_class*.

Table 4-2 Operations to add a child class

Condition	Operation
The <i>child_class</i> * does not exist in the integrated ontology	
Exist <i>a_class</i> **: 	
Exist <i>a_class</i> **: 	

* *child_class* represents the child class of the *curr_class*;

** *a_class* is in the integrated ontology

Figure 4-8 uses an example to illustrate the main idea of the algorithm that merges two ontologies in a stepwise fashion. In step 1, we create a root class A using the goal description.

Then we find matching classes from the input ontologies (1) and (2) where we identify two matching classes for A. Consequently, we add the properties from the matching classes to A in the integrated ontology. In steps 2 and 3, we find the children (*i.e.*, B and C) of A from ontology 1 and add them into the integrated ontology. In step 4, class C is a child class of A in ontology 2. However, class C exists in the integrated ontology. Instead of adding another class C to the integrated ontology, we merge the properties of C from ontology 2 with the properties of C in the integrated ontology. In step 5, H is a child of A. If WordNet database indicates that H is the parent of B, we add H as a child of A and set B to be the child of H.

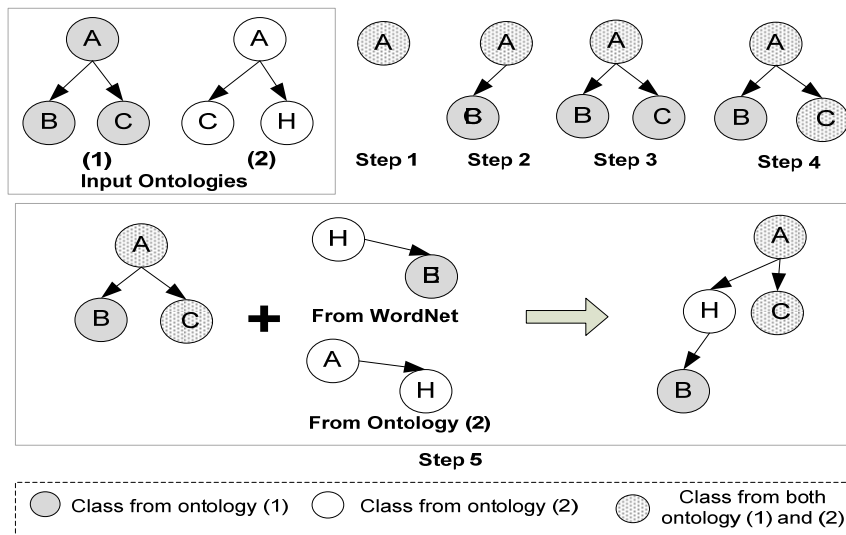


Figure 4-8 An example of integrating ontologies

4.6 Summary

Process knowledge provides the information for service composition systems to generate ad-hoc processes and is essential for service composition. In this chapter, we present an approach to extract process knowledge from the Web. We analyze the content and the structure of relevant websites to extract process knowledge from various websites. Our approach merges process

knowledge extracted from various websites to generate an integrated ontology with rich process knowledge.

Chapter 5

Context-Aware Service Discovery and Recommendation

To recommend services for a context scenario, we propose an approach that captures dynamically changing context scenarios of end-users and formulates searching criteria to discover the desired services. Instead of manually pre-defining if-then rules using specific context types or values as the traditional context-aware systems [13], our approach uses the relations among context values to infer end-user's needs. Then we generate service searching criteria based on end-user's needs to discover and recommend services.

To facilitate the presentation of this chapter, let us consider a travel scenario as an illustrative example throughout this chapter. Tom is a graduate student living in Toronto. Tom is interested in watching Hollywood movies and National Basketball Association (NBA) games. Especially, Tom is a fan of Kobe Bryant who is an American professional basketball player and plays for the NBA team, Los Angeles Lakers. Tom plans to travel to Los Angeles and spend his vacation in Los Angeles next month. When examining the context in this scenario, we find that some contextual information can be helpful for Tom to plan his trip. For example, as a graduate student who has low income, Tom might prefer a budget hotel for the trip. As a fan of NBA, Tom might be glad to know that "Los Angeles Lakers" is in Los Angeles.

This chapter is organized as follows. Section 5.1 gives an overview of our approach. Section 5.2 presents our approach to search for matching ontologies from ontologies databases. Section 5.3 discusses the details of inferring relations among different context values. Section 5.4 presents our approach that identifies end-user's requirements in a given context scenario and generates searching criteria to search for services. Finally, Section 5.5 summarizes this chapter.

5.1 Overview of an Approach for Context-Aware Service Discovery and Recommendation

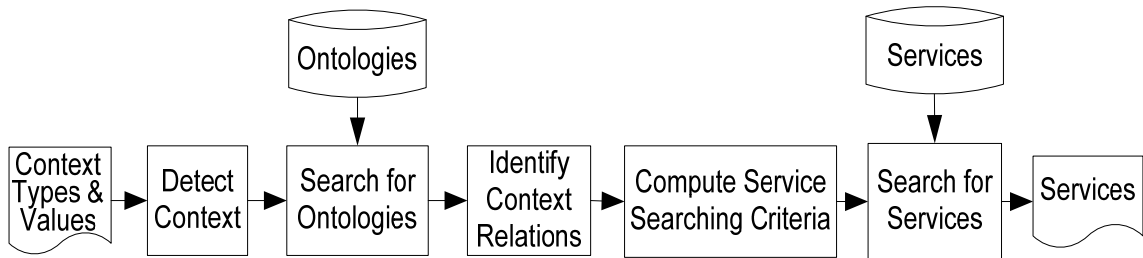


Figure 5-1 Steps for context-aware service recommendation

Figure 5-1 gives an overview of our approach. Context types can be dynamically added and removed to reflect an end-user's situation. The value of a context type can also be changed over time. To correctly model relations among context values, it is critical to understand the semantic meanings of each context value. Ontologies capture the information related to a particular concept using expert knowledge. To identify the semantics of a context value, we search for publicly available ontology databases such as Freebase [54] to extend the meaning of the context value. Figure 5-2 illustrates an example ontology for describing the information about "Los Angeles". In particular, "Los Angeles" is a context value for the context type "Location". The ontology of "Los Angeles" shown in Figure 5-2 expands the semantic meaning of "Los Angeles" with additional information, such as "Geographic Location", "Sports Team", and "Tourist Attraction". When a context value for an end-user is detected, our approach automatically searches for ontologies that expand the semantic meanings of the new context value and compute the relations with other context values.

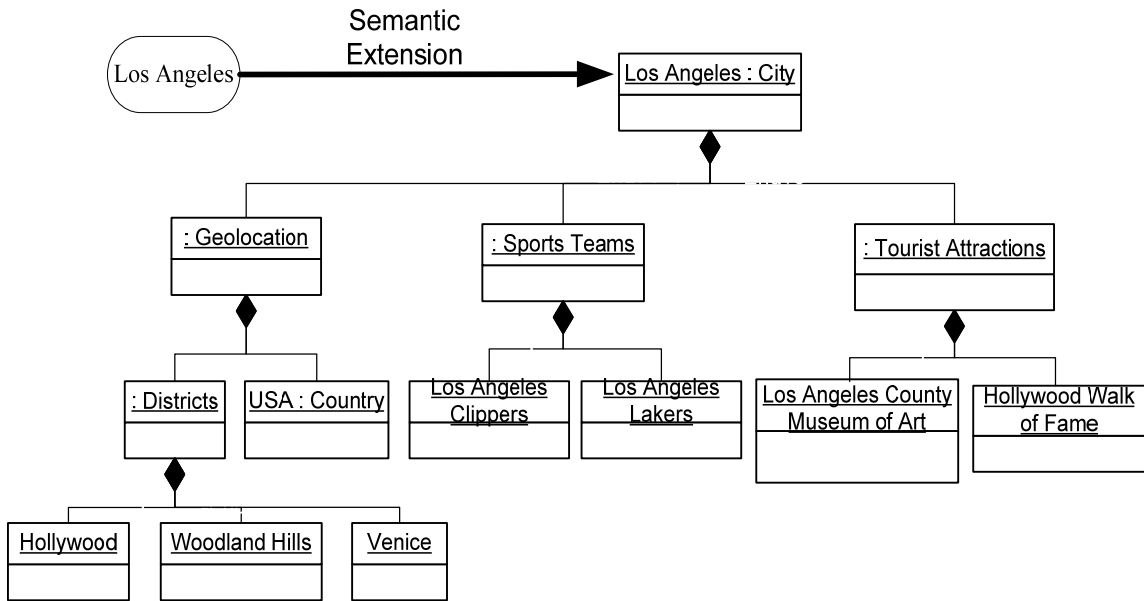


Figure 5-2 An example of extending context value using ontology

We use the identified context relations to discover an end-user’s requirements for a given context scenario and generate the corresponding service searching criteria. For example, when the semantics (i.e., ontologies) of several context values share the same concept, the concept would reflect the potential requirements of the end-user. In the travel scenario, Tom is going to travel to “Los Angeles”, and he is interested in watching NBA games. The ontologies of “Los Angeles” and “NBA” have the same concept “Los Angeles Lakers”. It indicates a high likelihood that Tom would be interested in watching the basketball game played by “Los Angeles Lakers”. Finally, we use the generated service searching criteria to discover services and recommend them to the end-user.

5.2 Searching for Matching Ontologies

Not all the context values can find the matching ontologies. Usually there are no matching ontologies for long phrases. For example, it is difficult to find a matching ontology for the context value “plan a trip to Los Angeles” which is the value of the context type “activity”. We search for relevant ontologies for each context value in the following steps:

- 1) *Search for ontologies using the entire string of the context value.* We treat the context value as a searching string, and use the entire searching string to search for relevant ontologies from ontology databases, such as Freebase [54]. If we can find a matching ontology, we annotate the ontology to the context value.
- 2) *Simplify the string of the context value by removing adjectives and adverbs.* If we cannot find a matching ontology for the searching string, we use an adjective and adverb dictionary (e.g., WordNet) to identify and remove the first adjective or adverb in the searching string. Adjectives and adverbs describe constraints on an entity. For example, in the context value “luxurious travel”, the adjective “luxurious” is a constraint on the “hotel”. Therefore, removing the adjectives and adverbs can keep the important information in the searching string. If the removed adjective or adverb is followed by a stop word (e.g., “to”, “for” and “of”), we remove the stop word since the stop word do not contain important information to be used in search queries. Then we use the remainder of the searching string to search for ontologies.
- 3) *Annotate the matching ontology to the context value and convert the removed adjectives and adverbs to constraints of the context value if we can find a matching ontology.* For example, if we cannot find the ontology for the context value “luxurious travel” but find an ontology

for “travel”, we map the ontology “travel” to the context value “luxurious travel”. We also add an annotation “luxurious” to the context value “travel”.

If we cannot find a matching ontology, repeat Steps 2) and 3) until we find a matching ontology or the string of context value is empty. When we cannot find any relevant ontology using the searching string of a context value, we use synonyms of the context value to search for ontologies and repeat the above steps. If no matching ontology is found, then we create a new ontology and convert the context value to be the root class of the new ontology.

5.3 Identifying Context Relations

The relations among context values can be used to identify an end-user’s requirement in the given context scenario. In our approach, we use two steps to identify the relations among multiple context values.

- 1) *Identify the relations between two context values.* We compare the corresponding ontologies which represent the semantics of context values to identify the relations of two context values. To compare the entities from two different ontologies, we define the term “*similarity*” which can identify similar entities from different ontologies. We also introduce the user-defined relation to link dissimilar entities from different ontologies using domain knowledge.
- 2) *Integrate all the relations of two context values.* To get the relations among multiple context values, we integrate the relations between two context values to construct a relation map that describes the relations of multiple context values.

5.3.1 Identifying Relations of Two Context Values

5.3.1.1 Similarity of Entities in Ontologies

Ontologies on the Internet are defined by various people from different perspectives. The entities (i.e., classes, individuals or properties) defined in two different ontologies may have different names for the same concept. Moreover, the entities of two ontologies can be defined in different granularity, even though both ontologies refer to the same concept. For example, “United States”, “USA” and “America” are different names for the same concept. As shown in Figure 5-5, the class “Tourist Attraction(s)” defined in ontology “Los Angeles” and ontology “Travel” contains different levels of details, although both classes of “Tourist Attraction(s)” refer to places of interest where tourists visit. To identify the same entities defined in different ontologies, we evaluate the semantic *similarity* of two entities in different ontologies in the following cases:

- 1) Two phrases (e.g., entity names and property values) are *similar*, when the words are identical, synonyms or originated from the same stem. In this these, we use WordNet to identify synonyms and stems of words. For the example shown in Figure 5-5, phrases “Tourist Attractions” and “Tourist Attraction” are similar since both phrases are stemmed from the phrase “Tourist Attraction”.
- 2) Entity P_1 and entity P_2 are atomic properties. Property P_1 and property P_2 are *similar* if and only if the property names and property values of property P_1 and property P_2 are *similar*. For example, atomic properties “Price Range: budget” and “Price Range: cheap” are similar since both properties have the same property name “Price Range” and have similar properties values “budget” and “cheap”.
- 3) Entity E_1 and entity E_2 are classes, individuals or non-atomic properties. Entity E_1 and entity E_2 are *similar* if and only if
 - a. The names of entity E_1 and entity E_2 are *similar*; and

b. All the properties defined in entity E_1 exist in entity E_2 ; or all the properties defined in entity E_2 exist in E_2 .

For example, assuming that there is a class “Tourist Attractions” which has a property “location: Los Angeles” and there is another class “Tourist Attraction” which does not have any properties, we say that the two classes are similar since the two classes satisfy the following two conditions. Firstly, the class name “Tourist Attractions” and “Tourist Attraction” are similar. Secondly, the properties (i.e., no properties) defined in the latter class “Tourist Attraction” belong to the former class “Tourist Attractions”.

In case 3), entity E_1 and entity E_2 might have different number of properties. Some ontologies may provide more detailed information of an entity than others due to the different levels of granularity in different ontologies. If the properties of entity E_1 (i.e., a class or an individual) are the subset of the properties of entity E_2 , entity E_1 and entity E_2 are treated as *similar entities*. We use WordNet to identify the synonyms and stem of words. By considering the synonyms and stems of words, we can discover two entities are similar if the entities are not described using the same words

5.3.1.2 User-Defined Relations Using Domain Knowledge

By comparing the similarity of entities, we can discover the semantic relations between context values. However, the similarity of entities cannot identify the relations which require domain knowledge. For the example of the travel scenario, Tom is a graduate student with low income. We can infer that he may prefer a budget hotel instead of a luxury hotel while he is traveling. However, the constraint of “preferring a budget hotel” is not specified in the ontology of “graduate student”. The ontology of “graduate student” may only tell us that graduate students have low income. To overcome this problem, we use LinQL language [65] to specify links

between entities. LinQL is an extension of Structured Query Language (SQL) and defines the conditions that two given entities must satisfy before a link of two entities can be established.

```
Linkspec_stmt= CREATE LINKSPEC linkspec_name  
AS link_method opt_args opt_limit;
```

Figure 5-3 Main structure of defining a link specification statement

Figure 5-3 shows the main structure of defining a link specification statement (linkspec for short) using LinQL. As shown in Figure 5-3, a CREATE LINKSPEC statement defines a new linkspec which specifies the name of the linkspec and a method to establish the link. For example, Figure 5-4 defines that if a person's income is low, then the person would prefer economical consumption style which is defined as terms with a property of economy. The details of defining LinQL are described in [65].

```
CREATE LINKSPEC consumption_style  
AS LINK low_income WITH target  
WHERE synonym(term, economy)  
AND  
target LIKE '%term%'
```

Figure 5-4 An example link specification

5.3.1.3 Relations between Two Context Values

Based on the definitions of the semantic *similarity* of entities and user-defined relations, we identify the following five types of relations between two context values extended by ontologies:

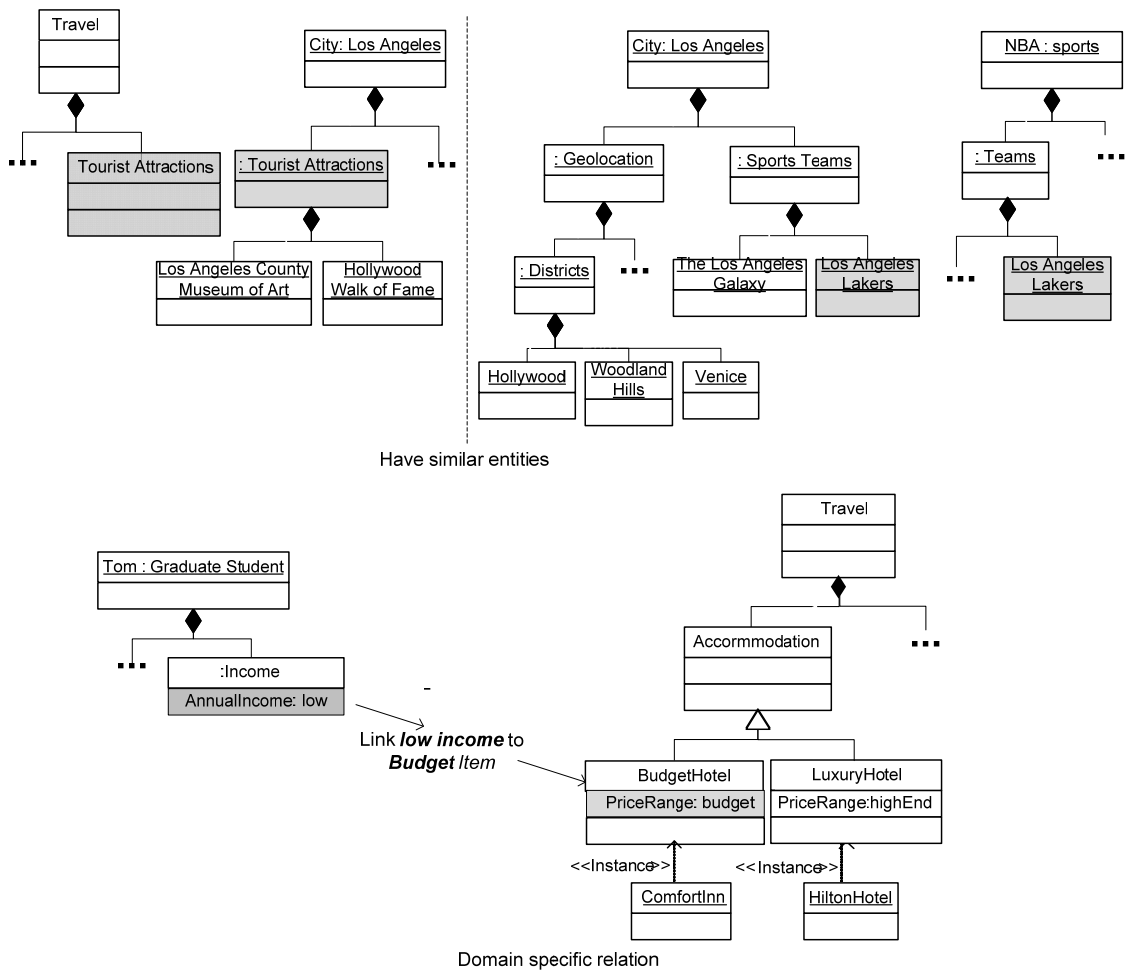


Figure 5-5 Examples of relations between two context values

1) *Intersection* means that the ontologies of two context values contain *similar* entities (i.e., classes or individuals). Figure 5-5 shows three examples of intersection relations. In Figure 5-5, the context value “travel” (i.e., its relevant context type is “activity”) and the context value “Los Angeles” share the same entity “Tourist Attraction”. Context values, “Los Angeles” and “NBA”, contain a common entity “Los Angeles Lakers”. When a context value

is a part of another context value, such context values are in an intersection relation. In the travel example, the ontology “Los Angeles” contains an individual “Hollywood”. Therefore, “Hollywood” is a part of “Los Angeles”. The context values “Hollywood” and “Los Angeles” have an intersection relation.

We use entity names, properties and individuals of ontologies to describe the common entities among two ontologies. The children entities (e.g., sub-classes, individuals of sub-classes) of the common entities are ignored if the children entities are not defined in one of the ontologies. This simplifies the description of common classes, since the children entities contain too many details which could decrease the importance of the information in the common entities.

- 2) *Complement* indicates that all the members of an entity (i.e., classes or individuals) do not belong to another entity, and the two entities together define all the members in the given domain. The *complement* relations can be directly derived from the ontology definitions. For example, context values “Economy Hotel” and “Luxury Hotel” have a complement relation as defined in the ontology of “Travel”.
- 3) *Equivalence* defines that two context values describe the same concept. Equivalence relations should be explicitly defined in one of the ontologies. Explicit defined equal entities are treated as *similar* entities when we compare the entities from two different ontologies.
- 4) *Domain specific relation* means that the corresponding ontologies of two context values contain entities linked by user-defined relations. As shown in Figure 5-5 (3), the domain specific relation is identified by a user-defined relation which links the *low income* to budget items (i.e., *budget hotel* in the travel scenario).
- 5) *Independence* means that two context values do not have any connection.

5.3.1.4 Inferring Relations among Multiple Context Values

We use entity-relationship (E-R) diagrams [30] to create a global view of relations among multiple context values. E-R diagrams provide a formal description for a set of entities and relationships among entities.

For each relation of two context values, we convert the two context values into two entities in the E-R diagram. A relation type (e.g., intersection and complement) is converted into a relationship node in the E-R diagram. A relationship node connects the relevant entities. For an *intersection* relation type, the common entities are converted into the attributes of the *intersection* relationship in the E-R diagram. *Equivalence* relations are used to combine entities in the E-R diagram. To simplify an E-R diagram, *independence* relations are not explicitly described in an E-R diagram. If two entities are not connected by a relation node in the E-R diagram, it indicates that the entities are *independent*. We integrate the relations of two context values into an integrated E-R diagram in the following steps:

- 1) Initialize the integrated E-R diagram as an empty model.
- 2) For each relation in the relation list, we repeat the following steps:
 - a) Convert a relation of two context values into an E-R diagram.
 - b) Add the E-R diagram created in step 2.a into the integrated E-R diagram. If there exist *similarity* or *equivalence* entities, we merge the *similarity* and *equivalence* entities by keeping the entity with richer information in the E-R diagram. If there exist *subclass*, *partOf* or *complement* relations, we add a relationship node in the integrated E-R diagram to indicate the corresponding relation. If two relationship nodes contain the same relation type and relationship attributes, we merge them into one relationship node.

Following the aforementioned steps, all the context values are converted into entities in the integrated E-R diagram. The entities associated to relations of context values are transformed into properties in the E-R diagram. Figure 5-6 shows an example of an integrated E-R diagram for the context values in the travel scenario. In Figure 5-6, context ontologies “Student” and “Travel” have an domain specific relation due to an user-defined relation which links “Income: Low” to “Budget Hotel”. The NBA team “Los Angeles Lakers” is shared by three context values “Los Angeles”, “Kobe Bryant” and “NBA”. Context ontology “Travel” shares the same class “Tourist Attractions” with ontology “Los Angeles”. Class “Tourist Attractions” contains a set of individuals such as “Hollywood Walk of Fame” and “Los Angeles County Museum of Art”. We use the individuals to get specific tourist attractions (i.e., services) in Los Angeles.

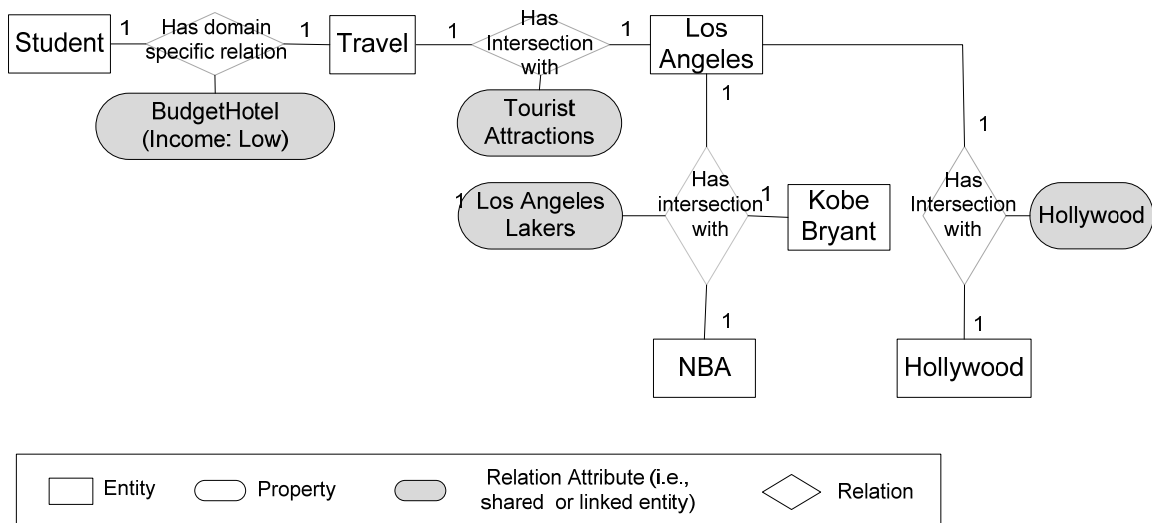


Figure 5-6 An example of integrated E-R diagram

5.4 Generating Service Searching Criteria

To recommend services, we identify end-user's requirements, and generate searching criteria to search for services. An end-user's requirements describe the potential tasks to perform in a given context scenario. We define generic rules to infer end-user's requirements from the E-R diagram. We extract service searching criteria from the description of end-user's requirements to search for services.

5.4.1 Identifying End-user's Requirements in Given Context Scenarios

In our approach, an end-user's requirements in his/her current context scenario are identified based on the relations among different context values. However, some relations among context values exist in all the scenarios of an end-user due to the long-term existence of certain context values. For the example of the travel scenario, Tom's preferences involve "NBA" and "Kobe Bryant". These preferences can be explicitly specified by Tom and generally exist for a long term. Our approach would always identify the common entity "Los Angeles Lakers" since the ontology "NBA" and the ontology "Kobe Bryant" share the same entity "Los Angeles Lakers". As another example, the current "city" (e.g., Toronto) always has a relationship with the current "country" (e.g., Canada). To avoid recommending the redundant information, we ignore the relations among context values when the relations are derived from the context values that exist for a long term or inherently exist in the associated context types.

We design 3 generic rules to derive end-user's requirements from the integrated E-R diagram as shown in Table 5-1. Suppose $E_{c1}, E_{c2}, \dots, E_{cn}$ are entities in the integrated E-R diagram. *Potential Task Set* represents a set of an end-user's requirements.

Rule 1 collects the common entities and properties from the E-R diagram. The common entities in the *Potential Task Set* are contained in two or more ontologies corresponding to the context values. Each entity in the *Potential Task Set* indicates a part of an end-user’s requirements. For example, the context value “Los Angeles” and context value “NBA” have a common entity “Los Angeles Lakers”. The common entity “Los Angeles Lakers ” is a NBA team in Los Angeles, and there is a high chance that the end-user would be interested in the services related to this team.

Table 5-1 Generic rules to derive end-user’s requirements

Rule No.	Relations	Potential Task Set	Description
1	Intersection relations: $E_{Ci1} \cap E_{Ci2} \cap \dots \cap E_{Cim} = \{e_1, e_2, \dots, e_k\} \neq \emptyset$	$\{e_1, e_2, \dots, e_k\}$	$E_{c1}, E_{c2}, \dots, E_{cn}$ are entities in the integrated E-R diagram. e_1, e_2, \dots, e_k are entities or relationship attributes in the integrated E-R diagram.
2	Domain specific relations: E_{C1} is linked to E_{C2} by user-defined relations $\{(e_{11} \rightarrow e_{21}), \dots, (e_{1k} \rightarrow e_{2k})\}$	$\{(e_{11} \rightarrow e_{21}), \dots, (e_{1k} \rightarrow e_{2k})\}$	$E_{c1}, E_{c2}, \dots, E_{cn}$ are entities in the integrated E-R diagram, and $(e_{1i} \rightarrow e_{2i})$ are a pair of linked entities between entity E_{c1} and entity E_{c2} . In the E-R diagram, $(e_{1i} \rightarrow e_{2i})$ represents a property of the user-defined relation.
3	Complement relations: $\overline{E_{c1}} = E_{cj}, e_1 \in E_{c1}, e_2 \in E_{cj}$ and $e_1, e_2 \in \text{potentialTaskSet}$	e_1 and e_2 have an OR relation.	E_{ci} and E_{cj} are entities in the integrated E-R diagram. e_1 and e_2 are entities or relationship attributes in the integrated E-R diagram; and \overline{E} represents the complement of entity E .

A user-defined relation connects two entities from two different ontologies. The linked entities are represented as pairs (e.g., $e_{1i} \rightarrow e_{2i}$) in Table 5-1. If two entities from different context values are linked by a user-defined relation, it means these two entities are different from other entities in the ontologies of context values and the information in these two entities might be interested by the end-users. Therefore, In Rule 2, we extract the linked entities and add them to the *Potential Task Set*. In the example of planning a trip, “Budget Hotel” has a high chance to be interested by Tom since the entity “Budget Hotel” is linked by an attribute of the occupation of the end-user.

Complement relations show that two entities cannot co-exist in the same time. In Rule 3, we use complement relations to split the entities in the *Potential Task Set* and identify them as an “OR” relation. For example, if the *Potential Task Set* contains both the entities “Budget Hotel” and “Luxury Hotel”, we can use the complement relations to identify them as a “OR” relations. Therefore, when an end-user choose one recommendation (e.g., Budget Hotel), we stop to recommend the complement recommendation (e.g., Luxury Hotel) since the user has made a decision between these two types of hotels.

Once the rules are applied on the E-R diagram, we obtain a *Potential Task Set* which contains a set of entities and properties of the entities in the E-R diagram. Some entities in the *Potential Task Set* may describe the same concept at different levels of details. For example, one entity is the subclass of another entity. To reduce the redundancy of service recommendation, we merge the similar end-user’s requirements by classifying the entities in the *Potential Task Set* into different groups. Each group maps to a specific service searching criteria.

5.4.2 Generating Service Searching Criteria

The entities and properties of an end-user’s requirements (e.g., *Potential Task Set*) are described using structured data defined in ontologies. We define mapping rules specified in Table

5-2 to convert the structured data to service searching criteria. The class name in Table 5-2 refers to the name of classes defined in ontologies. $name_{inputPar}$ and $name_{outputPar}$ represent the name of input parameter and the name of output parameters. Furthermore, the generated searching criteria are submitted to existing search engines, such as Google [59].

In Table 5-2, the first column lists the entities contained in the end-user's requirements (i.e., a *Potential Task Set*). The second column shows the associated query to find matching Web services described in WSDL. The third column presents the generated query submitted to a Web search engine. The WSDL query is used to search for available Web services described using WSDL from service repositories (e.g., IBM WebSphere Service Registry and Repository (WSRR) [75]). However, not all the user's requirements have matching Web services to fulfill. For the example of the travel scenario, we might not be able to find a WSDL Web service to provide the tourism attractions in Los Angeles. As an option, we search for the desired information from existing Web Pages using Web search engines since there are a large number of WebPages with rich information on the Internet.

As shown in Table 5-2, a class contains class name and properties. In a WSDL query, a class name is used to match a service name or an operation name in a WSDL document since the class name is the major object name used in a Web service. In most cases, a service name or an operation name in WSDL are not identical to the class name defined in ontologies. For example, an operation used to search for budget hotels can be named as "Get Budget Hotel" or "Book Budget Hotel". The operation names contain the class name "Budget Hotel" with additional verbs (i.e., "Get" or "Book"). Therefore, we check the generated WSDL query if a class name is a substring of the service name or operation name instead of exacting matching the service name or

the operation name with the class name. We apply the same requirements to other ontology entities in the conversion process.

Table 5-2 Mapping ontology entities in potential task set to WSDL query and webpage query

Entities in the <i>PotentialTaskSet</i>		WSDL Query	General Query for WebPages
Type	Involved data		
Class	Class name: $name_{class}$	$(name_{class} \subseteq$ $(name_{operation} \cup name_{service}))$ $\&\&(name_{property_i} \subseteq$ $(name_{inputPar} \cup name_{outputPar}))$ where $name_{property_i} \in$ $(\cup name_{property})$	$involvedContextValue \text{ AND}$ $name_{class} \text{ AND}$ $(property_1 \text{ OR } property_2$ $\dots \text{ OR } property_k)$
	Property names of the class: $\cup name_{property}$		
Individual	Individual name: $name_{individual}$	$(name_{individual} \subseteq$ $(name_{operation} \cup name_{service}))$ $\&\&(name_{property_i} \subseteq$ $(name_{inputPar} \cup name_{outputPar}))$ where $name_{property_i} \in$ $(\cup name_{property})$	$involvedContextValue \text{ AND}$ $name_{individual} \text{ AND}$ $(property_1 \text{ OR } property_2$ $\dots \text{ OR } property_k)$
	Property names of the individual: $\cup name_{property}$		
User-defined relation ($e_1 \rightarrow e_2$)	Name of the class that entity e_2 belongs to: $name_{class}$	$(name_{class} \subseteq$ $(name_{operation} \cup name_{service}))$ $\&\&(name_{property_i} \subseteq$ $(name_{inputPar} \cup name_{outputPar}))$	$involvedContextValue \text{ AND}$ $name_{individual} \text{ AND}$ $(property_1 \text{ OR } property_2$ $\dots \text{ OR } property_k)$
	Properties of entity e_2 : $\cup name_{property}$		

Properties of a class specify the detailed attributes of the class. There is a high chance that the properties of classes are the input or the output to perform an operation in WSDL services. For example, in our travel scenario, the property “Price” in the class “Budget Hotel” becomes a parameter of the operation “Book Budget Hotel”. As listed in Table 5-2, the names of the properties are used to match parameters of operations in a WSDL service. However, a service may not need to use all the properties defined in a class. Therefore, we use an OR relation to

connect all the properties. The searching criteria for Web search engines are focused on using keywords. In the column 3 of Table 5-2, we convert the class name into a keyword and the properties of classes to the optional (i.e., OR relations) keywords in the query since the properties of classes can add constraints to the query but too many constraints may limited the results of queries. For the individuals in the *Potential Task Set*, we use the same way as classes of ontologies to convert them to quires since the data in individuals are similar to classes.

In the user-defined relation $e_1 \rightarrow e_2$ as shown in the fourth row of Table 5-2, entity e_1 is the source entity that is used to search for entity e_2 , and entity e_2 is the target entity that we want to recommend to the end-user. For example, when we link the entity “low income” to all the budget (or economic, cheap) items, the budget items, such as budget hotel in the travel scenario, are the information that we want to recommend to end-users. Therefore, we convert the target entity instead of the source entity to a search query as shown in the fourth row of Table 5-2.

5.5 Summary

In this chapter, we present an approach that dynamically derives context relations from ontologies and automatically recommends services based on specific context values. By identifying the semantic relations among context values, we can infer end-user’s tasks hidden behind the context values and generate searching criteria for service discovery and recommendation.

Chapter 6

Ontology Driven Service Composition

In this chapter, we propose an approach that hides the complexity of Web services standards and tools to help non-IT professional end-users compose services for their daily activities. Our approach can identify services and generate ad-hoc processes to reflect the situational needs of end-users. More specifically, to ease the end-users's difficulty in understanding the functional and non-functional properties of a service, we propose a service description schema that describes service using descriptive tags (*i.e.*, keywords). The end-users can provide feedback on their experience of using the services. Instead of requiring end-users to specify the concrete tasks, the end-users only need to describe the goal that they want to achieve by invoking services using keywords. To have a better understanding of the specified goal (*i.e.*, keywords), we search for existing ontologies that can expand the meaning of a specified goal. Furthermore, we provide a technique that analyzes the identified ontology to automatically discover services and compose an ad-hoc process to achieve the specified goal.

This chapter is organized as follows. Section 6.1 gives an overview of the approach for composing ad-hoc processes. Section 6.2 describes our proposed tag-based service description. Section 6.3 and Section 6.4 present our approaches to search for matching ontologies and services respectively. Section 6.5 discusses an algorithm to generate ad-hoc processes. Section 6.6 introduces our prototype which implements a framework for supporting ontology driven service composition. Finally, Section 6.7 concludes the chapter.

6.1 An Overview of an Approach for Composing Ad-Hoc Processes

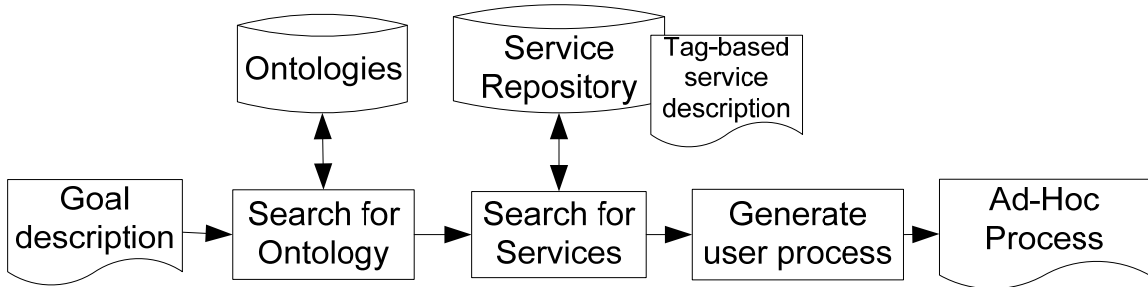


Figure 6-1 Steps for composing ad-hoc processes

Figure 6-1 gives an overview of our approach for supporting end-users to compose ad-hoc processes. To compose an ad-hoc process, an end-user simply describes a desired goal using keywords. Our composition system uses the goal description (i.e., keywords) to find a matching ontology from the ontology database. In an ontology, the semantics of a high-level goal is expanded into more concrete information, such as the attributes of the goal, the objects and actions of the goal. We parse the matching ontology and extract keywords from the matching ontology to search for services in service repositories. The service repository allows service providers to advertise their services and provide interfaces for automatic service discovery. The services in service repositories are described using our tag-based service description schema which uses descriptive tags (i.e., keywords) to simplify the existing service descriptions, such as WSDL. The service description schema can help end-users and composition tools understand the properties of services. End-users can also edit the tags to refine the service description. To help end-users fulfill their goals regardless the types of services, the services in the ad-hoc processes

could be any Web resources, such as WSDL services, Websites and RSS feeds. And end-users can use the service description schema to tags those different types of services.

To facilitate the selection and execution of services, our approach aggregates the discovered services into tasks. We use the relations defined in the ontology to identify the control flow among tasks. The identified tasks and control flows are represented as an ad-hoc process. The ad-hoc processes are stored in the ad-hoc process database and shared among multiple end-users.

6.2 Tag-based Service Description

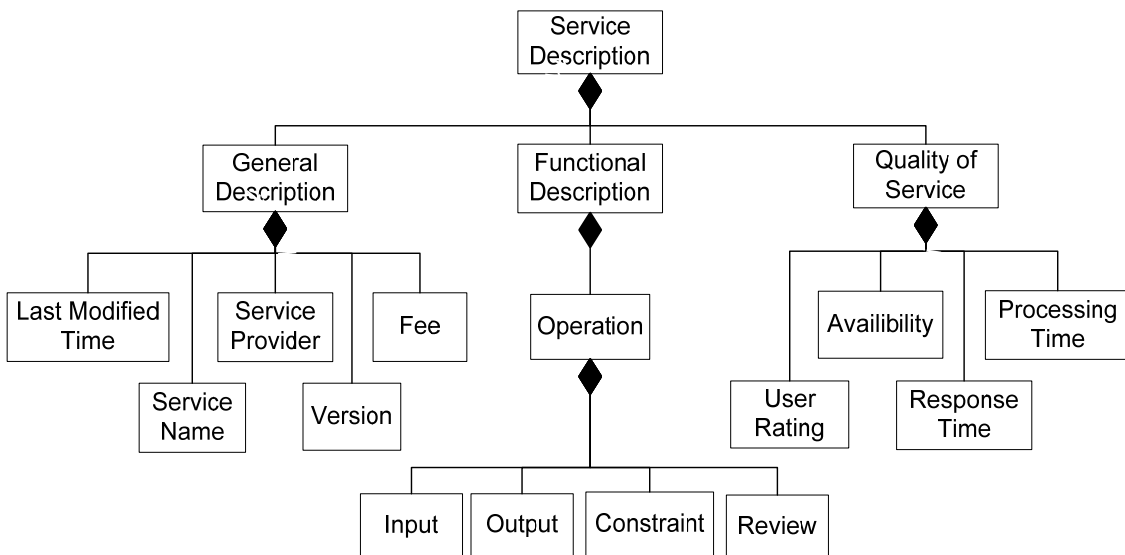


Figure 6-2 Schema for tag-based service description

To generate a meaningful ad-hoc process, it is critical to describe services in an efficient way that allows end-users and composition tools to understand the properties of Web services. In Web 2.0, tags are a popular feature to describe Web resources. For example, Facebook [48] uses tags to describe images and Seekda [125] takes tags to describe Web services. However, those tags are designed for the purpose of classification and searching. The tags are not organized in a structured way to help the end-user to understand the detailed properties of services (*e.g.*, the

operations of a service). Moreover, the tags can be redundant or irrelevant to a service. In our work, we propose a schema for associating Web services with structured descriptive tags which capture various properties of a service provided by both end-users and service providers to reflect the different perceptions of a service. To ensure tags to reflect the functionality of the services, the tags are initially extracted from WSDL. End-users can add new tags.

6.2.1 Schema of the Tag-based Service Description

Figure 6-2 illustrates the schema for our tag-based service description. In the schema, service providers describe the technical specification of services. End-users can provide their feedback on the quality of services and the delivered functionality. In general, we classify the tags into three categories:

General description specifies the basic characteristics of a service, such as version number, the last modified time, and the URL address (i.e., link) for accessing the service. The general description is provided by a service provider.

Functional description is provided by both service providers and end-users to describe the functionality of a service. The functional description is composed by a set of operations. Each operation contains several detailed descriptions, such as input, output, constraints, end-user reviews and provider's description on the functionality of an operation. A service provider publishes the name of a service, the operations and parameters as keywords. For example, an operation name, "*getWeather*", can be represented by a keyword, "*weather*". Such functional description tags are automatically extracted from WSDL. The constraints of each operation are expressed as a set of tags and value pairs (i.e., weather forecast period = 7 days). An end-user can add their own descriptions about each operation using a set of keywords. Many end-users may

submit similar reviews. Similar to the indexing techniques used in existing search engines [21], we extract the meaningful keywords from the reviews and store them as tags.

Quality of services (QoS) specifies the quality attributes either perceived by end-users or measured by service providers. The availability of a service, response time and the processing time are major concerns when invoking a discovered service. The values for these quality attributes are monitored and provided by service providers. Furthermore, the end-users can submit their rating about a service. A set of tag and value pairs (i.e., availability = 99%, and end-user rating = excellent) are used to describe the quality attributes. More quality attributes can be added to extend the tag-based service description. The values for the quality attributes are used to select services when multiple services of equivalent functionality are returned.

The tag-based description for a service is represented in XML and stored in a service repository which links the existing description documents of a service (e.g., WSDL) with the tag-based description.

6.2.2 Management of Tags

```
+ <wsdl:message name="GetWeatherByPlaceNameHttpPostOut"></wsdl:message>
- <wsdl:portType name="WeatherForecastSoap">
  + <wsdl:operation name="GetWeatherByZipCode"></wsdl:operation>
  + <wsdl:operation name="GetWeatherByPlaceName"></wsdl:operation>
  </wsdl:portType>
  :
  :
- <wsdl:service name="WeatherForecast">
  + <documentation></documentation>
  + <wsdl:port name="WeatherForecastSoap" binding="tns:WeatherForecastSoap"></wsdl:port>
```

Figure 6-3 An example of the service description in WSDL

The WSDL description for a service is provided by the service provider. We analyze the WSDL description of a service to extract the tags required from service providers. For example, we extract the service name, operation names, input parameters, output parameters and service access location (i.e., the URL to access the service) from WSDL documents.

Figure 6-3 shows an example WSDL description, which contains two operations, namely, “GetWeatherByZipCode” and “GetWeatherByPlaceName”. The name of the service is “WeatherForecast”. To extract meaningful tags from the end-user’s input and WSDL files, we filter out stop words, such as “the”, “to”, and “by” from the input phrases. When a stop word is placed in the middle of a phrase, we use the stop words to separate the phrase into two tags. For example, the operation name “GetWeatherByZipCode”, is converted to <Get Weather> and <Zip Code >.

Instead of treating derived words (*e.g.*, traveling, and traveled) as different tags, we detect the derived words from the end-user’s input and WSDL documents, and store only the corresponding root form in the description. For example, the derived words, such as “Travelling”, “Traveled”, and “Travels” are stored using the root word “Travel” in the service description. Not all the tags in our schema can be extracted from WSDL documents since WSDL is designed to provide the programming interfaces and does not have the semantic descriptions of functionalities and QoS. We allow end-users and service providers to add more tags by providing either phrases or sentences to describe their reviews of a service. Over time, tags associated with a service would grow considerably large since any end-users can freely add new tags to the service description. Extraneous tags would negatively affect the effectiveness of service discovery. To reduce the redundant tags, we use WordNet [116] to detect the synonyms of a new tag to be added in the

existing tag-based description. We filter out semantically equivalent tags before adding them to the tag-based description document.

6.3 Searching for Ontology from Ontology Database

Figure 6-4 shows our algorithm that uses keywords to search for ontologies. An end-user describes a process as a goal using a collection of keywords (i.e., $keyword(G) = \{k_1, k_2, \dots, k_n\}$, k_i refers to a keyword in the goal description). For example, a travel planning goal can be represented as a set of keywords, i.e., $keyword(plan\ trip) = \{travel, trip\}$. Instead of predicting the possible end-users' goals and predefining the corresponding ontology, we search for relevant ontologies from the Internet. Meanwhile, we also use the ontologies extracted from the Web described in Chapter 4. The availability of ontologies enables an end-user to specify any goals for the ad-hoc processes. To improve the chances of finding a matching ontology, we enhance the keywords provided by end-user with the synonyms of each keyword. The keywords and the associated synonyms are collected as a keyword set for a goal (i.e., $keyword(G)$). As an optional choice, we allow end-users to specify the tasks that they want to perform using keywords. For example, a "Planning Travel" goal contains tasks, such as "CarRental", "HotelReservation", and "Transportation". Similarly, we collect the synonyms for the task descriptions. We denote the task descriptions as $keyword(T) = \{tk_1, tk_2, \dots, tk_m\}$ where tk_i refers to a keyword for describing a task.

An ontology of the specified goal is identified when the root entity of an ontology matches one of the keywords in $keyword(G)$ (i.e., the set of keywords for goal description). A root entity is the entity in the ontology graph which does not have a parent node (e.g., node "Travel" in Figure 2-1). For the example shown in Figure 2-1, when the root class, "Travel", is matched with a "Plan a Trip" goal description, the ontology is returned. If a goal description is matched with an

entity defined within an ontology instead of the root entity, we retrieve the matched entity and its children nodes. For the example ontology shown in Figure 2-1, if the goal is “Accommodation”, we take the node “Accommodation” and all the related children nodes, such as “BudgetHotel” and “LuxuryHotel”. In some cases, more than one ontology can be matched with the goal description. To select an appropriate one, we count the frequency of the keywords of different meanings from both the task description (i.e., *keyword (T)*) and the goal description (i.e., *keyword (G)*) appearing in each ontology. We select the ontology with the highest frequency of the provided keywords.

Input: Goal description; task description (option)
Output: A matching ontology
Procedure searchOnto () {
 1. **Var** ontoSet = null;
 2. goal description = goal description \cup { the synonyms of the goal description };
 3. **For each** keyword k_i in goal description {
 4. Search for ontologies matching with k_i ;
 5. If(find matching ontologies)
 6. { Add the matching ontologies to ontoSet;}
 7. } //end for
 8. If (ontoSet is empty) // Cannot find matching ontoloigies.
 9. { Return null; }
 10. If (the size of ontoSet == 1)
 11. { Return the ontology in ontoSet; }
 12. else{ //ontoSet contains more than one matching ontology;
 13. Sort the selected ontologies from high to low using the frequency of keywords in goal description and task description;
 14. Return the first ontology in the sorted ontology list;
 15. }//end else
 16. }

Figure 6-4 Algorithm for searching for ontologies

If the keywords in a goal description (i.e., *keyword(G)*) are phrases with more than one word, we search for matching ontologies using the entire phrase. However, we may not be able to locate

any ontologies using the phrases in *keyword (G)*. In our experience, adjectives and adverbs are constraints for describing nouns. The essential information is expressed as nouns in the goal description. Therefore, we identify adjectives and adverbs by querying a dictionary and remove the adjectives and adverbs from the phrases. We use the rest of nouns as keywords to search for ontologies. The removed adjectives and adverbs are used to refine the searching results by selecting the ontology which contains the adjectives and adverbs if there are more than one matching ontologies.

To improve the chances for identifying ontologies, WordNet is used to identify the synonyms of the keywords in *keyword (G)*. We combine all the keywords in the goal description and the task description to search for ontologies based on the frequency of provided keywords. Then we ask end-users to inspect the returned ontologies and select one. To help end-users understand the ontology description, ontology visualization tools such as protégé [117] are used by an end-user to visualize ontologies.

6.4 Searching for Services

The identified ontology provides more detailed description about an end-user's goal. Essentially, the classes and individuals defined in ontologies capture the characteristics of the functional requirements for desired services that help achieve an end-user's goal. We use classes and individuals (i.e., entities) as criteria to search for the matching services. However, simply using a single entity in the search criteria may prevent us from discovering services since a single keyword provides limited knowledge. Similar to the search engines, which use the expanded query to search for the relevant documents [4][49], we group the name of the entity, its attributes, and the relevant entities which have direct relations (*e.g.*, *Subclass*, *Partof*, *Equivalent*, and *InstanceOf*) with the entity as a set of keywords. I.e.,

$$entity(e_0) = \{e_0\} \cup \{e_1, e_2, e_3, \dots, e_m\} \cup attr(e_0)$$

$$attr(e_0) = \{a_{e_01}, a_{e_02}, \dots, a_{e_0p}\} \quad (6-1)$$

where e_0 is the name of the entity; e_i , where $i = 1, 2, 3, \dots, m$, refers to an entity which has direct relation with e_0 (i.e., e_i directly inherits from e_0 , is a part of e_0 , is equivalent with e_0 , or is an Instance of e_0); and a_{e_0j} is the j -th attribute of entity e_0 , where $j = 1, 2, \dots, p$.

In Equation 6-1, entities $e_1, e_2, e_3, \dots, e_m$, which are the *Subclass*, *PartOf*, *equivalence*, or *InstanceOf* of entity e_0 , extend the meaning of entity e_0 . Therefore, we use these entities e_1, e_2, e_3, \dots , and e_m to enhance the meaning of entity e_0 . To avoid obscuring the meaning of entity e_0 with too many details, we take the entities which have a direct relation with entity e_0 . For example, the class “Travel” is expanded with a set of its components described as *partOf* relation in Figure 2-1, i.e., $entity(travel) = \{travel\} \cup \{transportation, accommodation, tourist attraction, car rental\}$. However, we do not use entities “Budget Hotel” and “Hilton Hotel” to extend the meaning of “Travel” since the two entities have no direct relations with “Travel”.

Each entity e_i has its own set of synonyms, i.e., $synonym(e_i) = \{s_{i1}, s_{i2}, \dots, s_{in}\}$. For example, the concept, “Travel”, has a set of synonyms, such as trip and journey (i.e., $synonym(travel) = \{trip, journey\}$). To retrieve the relevant Web services from a service repository, we combine the entity set and synonym set into a keyword set (i.e., $entity_keywords(e_0) = entity(e_0) \cup (\cup_{i=0}^m syn(e_i))$) to search for the matching services in a service repository.

$$SIM = \frac{\# \text{ of matched keywords}}{|n|} \quad (6-2)$$

n is the total number of tags in general description and functional description for a service

As discussed in Section 6.2, each service, s_j , in a repository is described by a set of tags that specify the general information and the functionality. Therefore, we have a set of keywords to describe a service using tags, i.e., $ws\text{-keywords}(s_j) = \{t_1, t_2, \dots, t_z\}$ where s_j is a service, and t_i is a tag of the service. To discover a service, we count the number of matched keywords between the entity description keywords (i.e., $entity\text{-keywords}(e_0)$) in the searching criteria and the service description tags (i.e., $ws\text{-keywords}(s_j)$) in the service repository. The similarity degree of the entities and services are defined in equation (6-2). The similarity degree ranges from 0 to 1. A higher value means that more tags in a service description are matched with the supplied concepts. Therefore, a high value indicates a high degree of similarity between the entity and the service. In equation (6-2), the typical value of n is between 10 and 15 since we filter out semantically equivalent tags using WordNet when end-users add new tags to describe services as mentioned in section 6.2.2.

Input: ontology, entity e_0

Output: relevant service list

Procedure searchServices () {

1. Using Equation (6-1) to get the keyword set $entity(e_0)$ of e_0 ;
2. $entity_keywords(e_0) = entity(e_0) \cup (\bigcup_{i=0}^m syn(e_i))$;
3. Search service repository by matching $entity_keywords(e_0)$ with the tag-based service descriptions;
4. Using Equation (6-2) to calculate the similarity degree between the query and service descriptions;
5. Sort the relevant services based on the similarity degree;
6. If (two relevant services have the same similarity degree)
7. { Sort these services based on the QoS description provided by the tag-based service description;}
8. Return sorted relevant service list;
9. }

Figure 6-5 Algorithm for searching for services

As a result of service discovery, we locate the services with the required functionality. When many services are matched, we sort services according to the similarity degree from high to low. When two services have the same similarity degree, they are sorted using the values of QoS description provided in the tag-value pairs specified in the service description. For example, the discovered services are sorted using the values of the processing time given that the discovered services have the same similarity degree. An end-user needs to interpret if the high value of a quality attribute is more desirable for the returned services. Figure 6-5 summarizes the algorithm that searches for services.

6.5 Generating Ad-Hoc Processes

The set of classes, individuals and attributes in the ontology could be used as searching criteria to search for possible services. However, a large number of services could be returned without any logical relations. Returned services may be redundant. It is a tedious job for end-users to manually select desired services. We develop an algorithm to organize the returned services in a logical structure (i.e., ad-hoc process) in the following steps:

- 1) Identify a list of tasks which are associated with one or more functionally similar services;
- 2) Identify the relations among the tasks; and
- 3) Refine the generated ad-hoc process by merging similar tasks which have the same set of associated services and relations.

6.5.1 Identifying Tasks

To group the services with similar functionalities, we design an algorithm to identify tasks for the ad-hoc process. The details of the algorithm are described in Figure 6-6. In this algorithm, we take an ontology which matches with the goal description as the input. More specifically, the

ontology is represented as an ontology graph as shown in Figure 2-1. Classes and individuals defined in ontologies are represented as nodes in the ontology graph. The path length of two nodes is the shortest distance (i.e., the number of edges) along the path from one node to another in the ontology graph. For the example shown in Figure 2-1, the path length of entities “Travel” and “Luxury Hotel” is 2.

Input: Ontology model for the goal

Output: A set of tasks associated with services

Initiate: var E = the entity which matches the goal description (i.e., keywords);

Procedure identifyTask () {

1. Queue q = new Queue();
2. q.push(E);
3. **while**(q is not empty){
4. Entity currentE = q.pop();
5. **If** currentE does not have attributes, subclasses, sub-components (described by PartOf relation), equivalent entities, or instances {
6. continue; //ignore it since it contains too little information to find appropriate services
7. }
8. Use Equation (6-1)and (6-2) described in Section 6.4 to search for services for currentE from service repository*;
9. **If**(the number of matching services > 0){
10. Associate the matching services to currentE, and convert currentE as a task in the ad-hoc process;
11. Output the task of currentE;
12. }
13. Push all the entities which have a direct relation with currentE but not marked as visited to the queue q;
14. Mark all the entities added to queue q as visited;
15. } //end while loop
16. }

* Entity E has matching services if and only if there exist returned services whose similarity degree is greater than a predefined value.

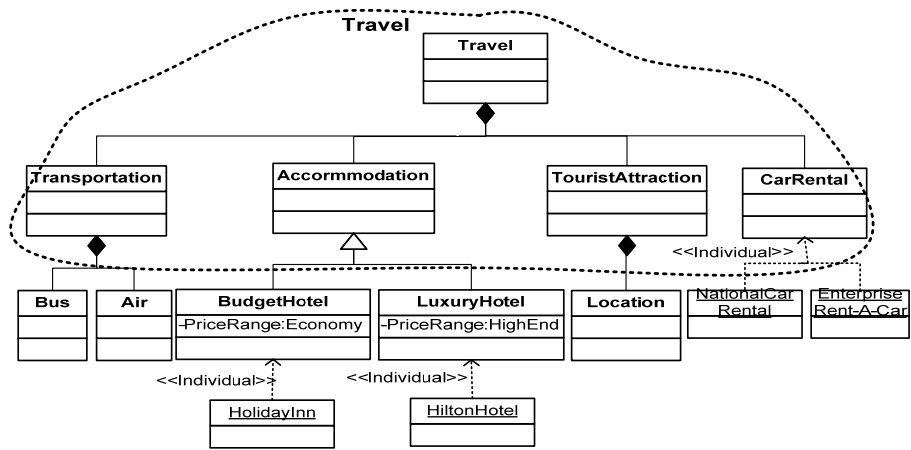
Figure 6-6 Algorithm of identifying task list

In general, our algorithm uses a stepwise approach to discover and organize the tasks according to the level of abstraction. The high level entities in an ontology graph convey more

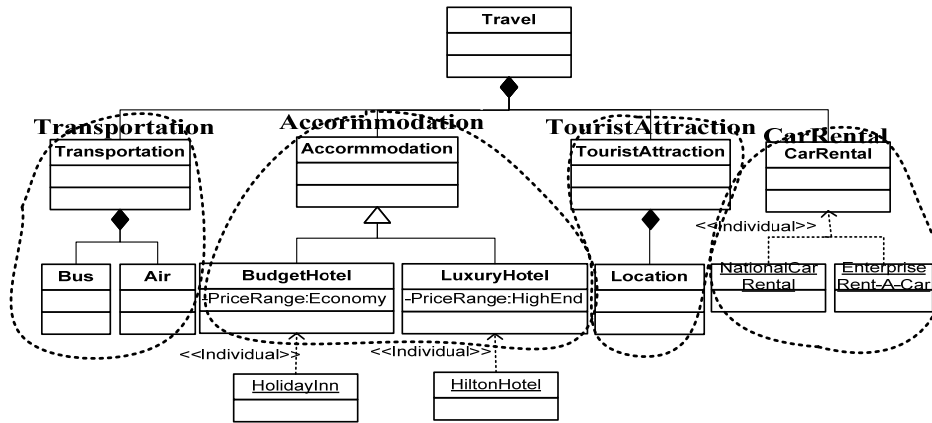
abstract meanings that are suitable for discovering the general purpose tasks. Such tasks allow end-users to receive the desired services (e.g, expedia.com [47]) in one server without having to go through separate servers for different services. The low level entities in an ontology graph provide more specific meanings of the goal and therefore indicate the possibility to find concrete tasks which provide more specialized services. For example, expedia.com provides a general service for planning a trip by providing information on car rental, flight ticket purchasing and hotel reservation. To check into a flight by Air Canada, an end-user has to visit more specialized services by going to Air Canada website to print their boarding passes and check flight status. To satisfy an end-user with varying needs in different levels of specialization, we use the breadth-first search algorithm to scan the ontology graph. We identify general purpose tasks from the top of the graph and the specialized tasks from the low level of entities in the graph.

The algorithm starts from the entity that matches with the high level goal description to find general purpose services. Subsequently, we identify more concrete entities by visiting the entities with the path length of 1 from the start point. Using the breadth-first search algorithm, we can identify a set of entities with different level of specialization by gradually increasing the path length from the starting point. The algorithm converts a visited entity into a task if and only if the entity has at least one *attribute*, *subclass*, *sub-component* (in the *PartOf* relation), *equivalence* entity, or *instance*. The visited entities that cannot meet the condition are ignored since such entities contain too little information to find appropriate services.

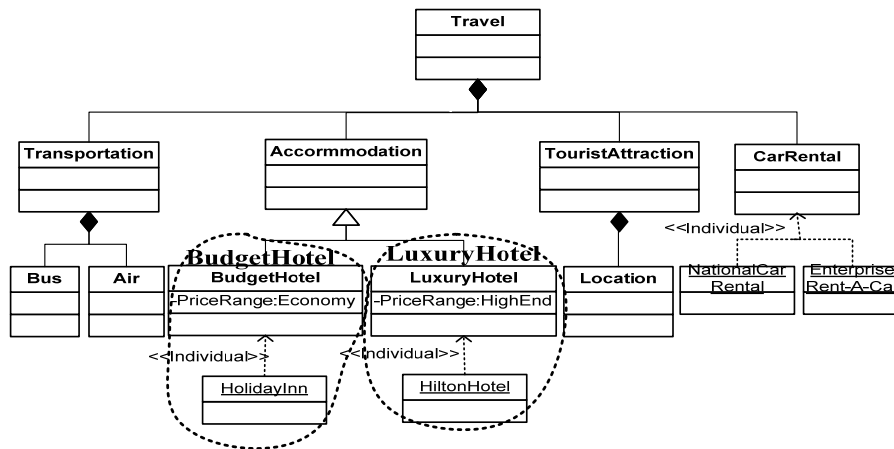
For each identified task, our algorithm extracts the searching criteria (i.e., *entity-keywords(e)*) from the corresponding entity and searches for services. We derive the name of a task from the name of the corresponding entity. When the similarity degree between the keywords in the searching criteria and the tags in a service description is greater than a predefined threshold, the



(1)



(2)



(3)

Figure 6-7 An example of generating task list

task is matched with the service. The task is ignored if no matching services can be found. The discovered services for a task are sorted based on the similarity degree. Finally, the algorithm outputs a list of identified tasks.

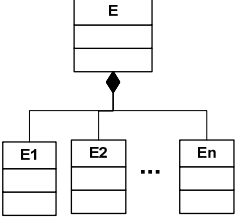
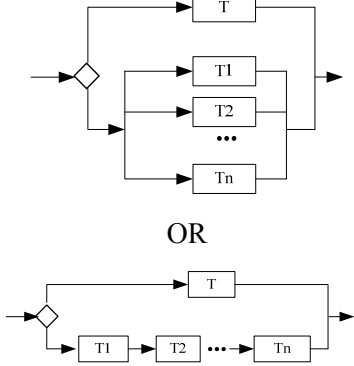
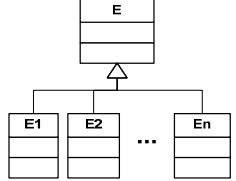
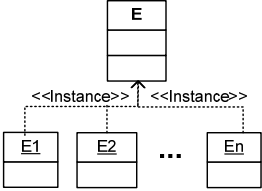
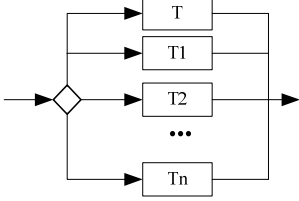
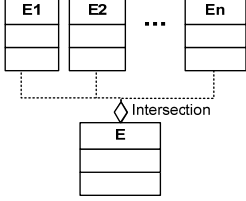
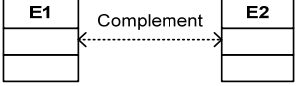
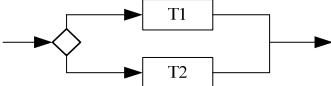
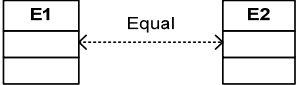
Figure 6-7 gives an example of generating a task list using our algorithm. The algorithm starts from the class “Travel” which is the goal specified by the end-user. The searching criteria for services are formed from the name of class “Travel” as well as the names of entities which have the direct relations (i.e., *PartOf* relation in our example) with class “Travel” (i.e., “Transportation”, “Accommodation”, “TouristAttraction”, and “CarRental”). Each returned Web service is expected to provide a set of comprehensive services (e.g., similar to Expedia.com [47]) for end-users to plan the travel. However, the returned Web services for “Travel” may only provide limited services, such as booking flight and train tickets and renting car, but without bus information. In this case, the end-users may need more specialized services for transportation, accommodation, and tourist attractions. To offer end-users with more options, our algorithm continues to decompose the abstract goal into a set of more concrete tasks as the algorithm traverses deeper in the path. In the second iteration of our algorithm, our algorithm identifies tasks, such as “Transportation”, “Accommodation”, “TouristAttraction” and “CarRental”. In the third iteration, our algorithm further refines the “Accommodation” task to more specific tasks, such as “BudgetHotel” and “LuxuryHotel”. However, the entities, such as class “Bus” and “Air”, do not have sufficient information (e.g., *attributes*, *subclass*, *sub-component*, *equivalence* entities, or *instances*) to discover new services. Therefore, their parent node “Transportation” is not further decomposed to more specialized services.

6.5.2 Identifying the Relations among Tasks

Once we identify a set of initial tasks in an ad-hoc process, we derive relations (i.e., sequence, parallel and choice relations) among tasks by analyzing the relations among the corresponding entities used for identifying tasks. Table 6-1 shows the mapping from entity relations in an ontology to task relations in an ad-hoc process. In Table 6-1, E_i is an entity defined in an ontology, and T_i is the corresponding task identified by entity E_i and its related entities in an ontology graph. As shown in Table 6-1, we convert the relations from the ontology to the ad-hoc process using the following mapping rules:

- 1) *PartOf*: When entities E_1, E_2, \dots, E_n are in *PartOf* relation with entity E , it indicates that entity E is composed by entities E_1, E_2, \dots, E_n . Theoretically, *PartOf* relation cannot guarantee that entity E does not contain the parts other than entities E_1, E_2, \dots, E_n . For example, we define that “Bus” and “Air” are a part of “Transportation” in an ontology, but “Transportation” may also contain other components, such as train and ferry. In our approach, we ignore those parts of entity E since those parts are not defined in the ontology. Similar to the subclass relation, we substitute entity E with entities E_1, E_2, \dots, E_n . Therefore, we define tasks T_1, T_2, \dots, T_n are in a *choice* relation with task T . Moreover, tasks T_1, T_2, \dots, T_n need to be executed together to fulfill task T , i.e., tasks T_1, T_2, \dots, T_n have a *parallel* or *sequence* relation. We discuss how to distinguish *parallel* and *sequence* relation later in this section.
- 2) *Subclass*: If entities E_1, E_2, \dots, E_n are the *subclass* of entity E , it indicates that entities E_1, E_2, \dots, E_n have all the information described in entity E and entities E_1, E_2, \dots, E_n contain more detailed information than entity E . Therefore, we can replace entity E with entity E_1, E_2, \dots, E_n . The tasks T_1, T_2, \dots, T_n corresponding to entities E_1, \dots, E_n have a *choice* relation with task T corresponding to entity E . In addition, tasks T_1, T_2, \dots, T_n are in *choice* relations since they have the similar functionalities conveyed in entity E .

Table 6-1 Convert relations from ontology to ad-hoc process

Relation in ontology definition model	Ontology graph	Control Flows in ad-hoc process
partOf		 <p style="text-align: center;">OR</p>
Subclass		
InstanceOf		
Intersection		
Complement		
Equivalence		

- 3) *InstanceOf* relation is similar to subclass relation in terms of converting relations defined in ontologies to relations defined in control flows. According to the definition of instance, an instance has the all the information defined in the corresponding class. Meanwhile, the instances of the same class have similar information. Therefore, we describe these instances and the corresponding class as a *choice* relation in the control flow since these instances and the class have the similar information.
- 4) *Intersection* relation indicates that class E_i (where $i=1, 2, \dots, n$) has the information defined in class E . Therefore, we can use E_i to replace E , i.e., we can choose either E_i or E to represent the information (i.e., task) that we need. So we convert the intersection relation to a *choice* relation in the control flow.
- 5) *Complement* relation means that two entities cannot co-exist. Therefore, we convert it into *alternative* relation.
- 6) *Equivalence* indicates that two classes have the similar functionality, so we convert the two equivalent classes into *alternative* relation as well.
- 7) *Domain specific relations* vary from domains. The mappings between the domain specific relations and the task relations need to be manually specified.

Figure 6-8 shows an example relations identified from the task lists described in Figure 6-7. The end-user can either choose a general purpose service “Travel” to fulfill the goal of planning a trip or decide to accomplish the goal using more specialized services (i.e., services for transportation, accommodation, tourist attraction and car rental). For the accommodation, the end-user can choose a general service related to accommodation, or select a service offering specific types of hotels (i.e., budget hotel and luxury hotel).

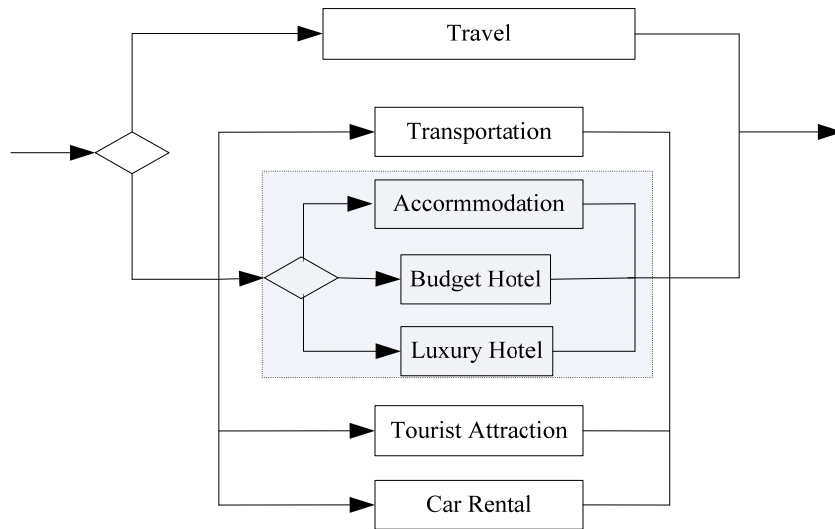


Figure 6-8 An example of a generated ad-hoc process

Sequence indicates that the tasks are executed in sequence. *Parallel* means that the tasks can be executed in any order. To distinguish between *sequence* and *parallel* relations, we compare the interfaces of services associated to tasks. Assume that we have tasks T_1, T_2, \dots, T_n which could be either in *sequence* or *parallel* relations, and we need to distinguish the *sequence* and *parallel* relations. For each task, we collect the input and output parameters from the associated services. We use $Set(input)$ to represent the input parameters from the associated services of a task, and use $Set(output)$ to represent the output parameters from the associated services of a task. Then we match the interfaces of the corresponding services. Enlightened by Paolucci's work [110], we define three levels of matching as follows. Suppose that p_1 and p_2 are the input or output parameters of tasks T_1 and T_2 respectively.

- **Exact:** if $Set(p_1) = Set(p_2)$
- **plug in:** if $Set(p_1)$ is a subset of $Set(p_2)$
- **Subsume:** if $Set(p_1)$ subsumes $Set(p_2)$, in other words, $Set(p_1)$ contains all the parameters in $Set(p_2)$.

If two sets of parameters satisfy one of the three conditions, then the two sets of parameters match. Tasks T_1 and T_2 are in a sequence relation, if and only if the interfaces of the corresponding services can be matched (i.e., $Set(output_{T_1})$ matches with $Set(input_{T_2})$). We treat the tasks which are not identified as a *sequence* relation as a *parallel* relation.

6.5.3 Merging Tasks

In our algorithm, the generated ad-hoc process may contain tasks with very similar functionalities. For example, two tasks which are generated from two *equivalence* classes respectively may have very similar functionalities. Generally, similar tasks are converted into the *choice* control flow to allow end-users to select. However, if two similar tasks are also associated with the same set of services, it becomes meaningless to offer the option to end-users. We use the following two rules to identify and merge similar tasks. Suppose that we have two tasks T_1 and T_2 . $WS-Set(T_1)$ is a set of associated services for task T_1 and $WS-Set(T_2)$ is a set of associated services for task T_2 .

Rule 1: if tasks T_1, T_2 are in a *choice* relation, and the overlap of $WS-Set(T_1)$ and $WS-Set(T_2)$ is greater than a predefined value, we choose the task that has more abstract meaning as defined in the ontology to represent the merged task, and associate services from both tasks to the merged task.

For example, if tasks “Accommodation” and “Hotel” are in a choice relation and associated with over 90% common services, and 90% is greater than the predefined value, we use task “Accommodation” to replace the two tasks and associate the relevant services of these two tasks to the task “Accommodation”.

Rule 2: If $WS-Set(T_1)$ is a subset of $WS-Set(T_2)$ (i.e., $WS-Set(T_1) \in WS-Set(T_2)$), then task T_1 is covered by task T_2 . We remove task T_1 .

6.6 Overview of our Prototype

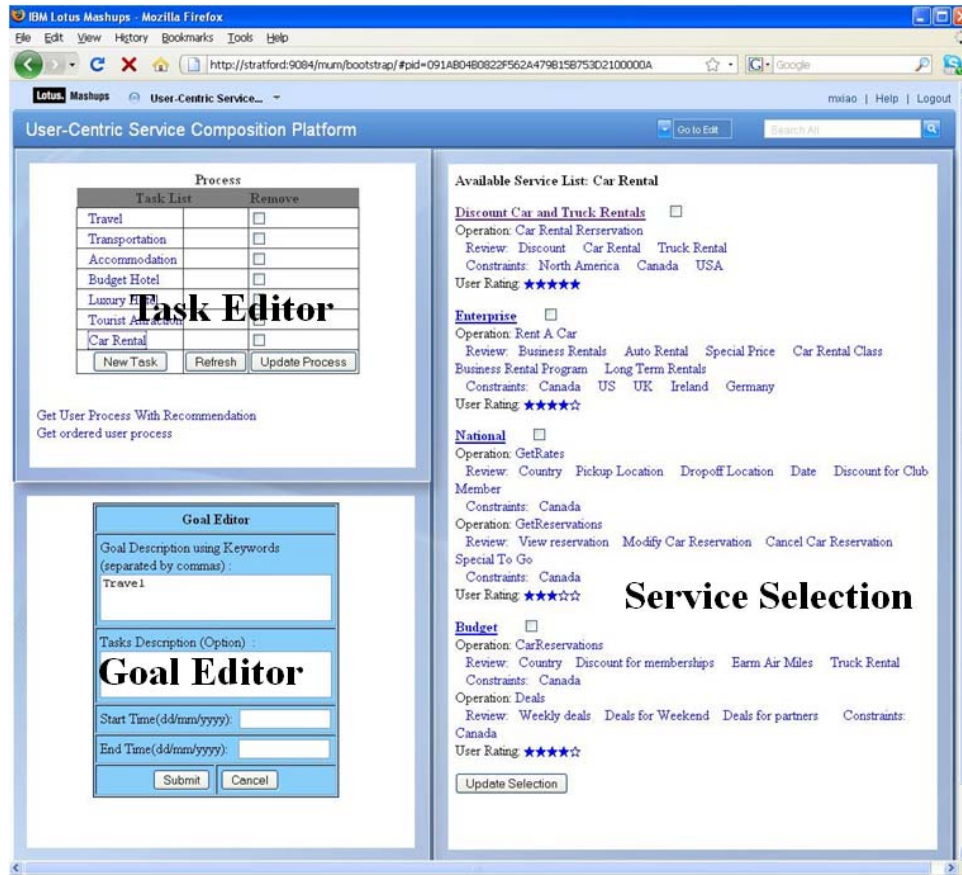


Figure 6-9 Annotated screenshot for our prototype

We design and develop a prototype to help end-users to generate ad-hoc processes based on the goals specified by end-users. We use the IBM WSRR to register and manage Web services. To display the interfaces of selected services and invoke services, we use the IBM Mashup Center [72] as a platform to integrate various Web services.

Figure 6-9 shows an annotated screenshot of our prototype. An end-user can specify their goal (e.g., plan a trip to New York) in the *Goal Editor*. In the current implementation of the prototype, the ontologies are manually searched using Swoogle [132], a search engine for ontologies, and

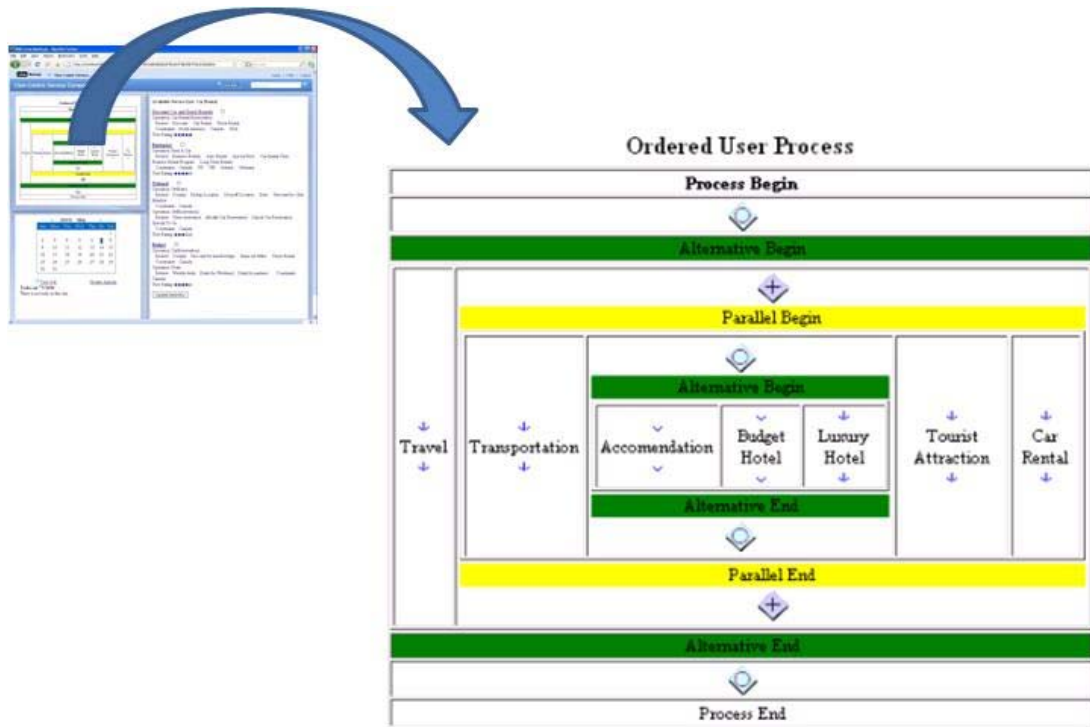


Figure 6-10 Ordered ad-hoc process in the Mashup page

imported into our prototype to ease the analysis. An ad-hoc process is automatically generated to capture a set of tasks that meet the specified goal shown in the *Process Editor*. A task in a generated ad-hoc process can be associated with one or more services. As shown in Figure 6-9, once an end-user selects the “Car Rental” task in the Process Editor, the associated services are displayed in the Service Selection Panel on the right side of the markup page. We allow an end-user to select the most desirable services. An end-user can refine and customize the ad-hoc process in the process editor. An end-user can remove a task if it is not needed by selecting the “Remove” check box. An end-user can also add a new task by specifying keywords for searching for services. We record the modifications as the end-user’s preferences. When an end-user specifies the same goal, our prototype provides the previously refined ad-hoc process. To guide

end-users to navigate through the task list to fulfill the goal, we identify the control flow among tasks based on the relations defined in the ontology. Figure 6-10 shows a screenshot of the suggested execution order of tasks in the ad-hoc process. Since the order ad-hoc process is only a suggested execution order, end-users still have the option to execute the tasks in other order.

6.7 Summary

In this chapter, we provide an approach that hides the complexity of SOA standards and tools from end-users and composes services to help end-users fulfill their daily activities. We propose a tag-based service description to allow end-users to understand the functionality of a service and add their own descriptive tags. Using our approach, an end-user only needs to specify the goal of their activities using keywords. Our approach automatically composes services that help an end-user achieve their desired goals without requiring the end-user to specify the detailed tasks.

Chapter 7

Case Studies

To evaluate the effectiveness of our proposed approaches, we conducted three case studies to examine: (1) whether our proposed approach can effectively extract desired process knowledge from websites and represent it using ontologies; (2) whether our proposed approach can correctly identify end-user's needs from an end-user's context and recommend the desired services to the end-user; and (3) whether our approach can effectively generate ad-hoc processes using ontologies.

This chapter is organized as follows. Section 7.1 evaluates our approach to extract process knowledge from the Web. Section 7.2 presents a case study to evaluate our approach to discover and recommend services using contexts. Section 7.3 examines our approach for ontology driven service composition. Finally, Section 7.4 gives a summary of the chapter.

7.1 Evaluation of Extracting Process Knowledge from the Web

The objectives of this case study are to: (1) evaluate the quality of the process knowledge extracted from each website; and (2) evaluate the quality of the integrated process knowledge.

7.1.1 Experiment Setup

To extract the desired process knowledge, we need to provide a set of goals. A goal describes the major objectives of the process knowledge. If we specify the goals for our case study by ourselves, it may impact the impartiality of the result since we design the approach. To avoid bias of goal selection for our case study, we gather 10 goals (*e.g.*, plan a trip, buy a car, and find a job)

from two websites eHow [46] and WikiHow [141]. eHow and WikiHow use articles and videos to provide online how-to instructions for fulfilling various goals.

The criteria for us to select goals are that (1) the goals should come from different domains, so that the case study can represent the general cases instead of just working one domain; (2) the goals require more than one online service work together to fulfill. The goals in eHow and WikiHow may only require one single service to fulfill. This criterion can help us filter the goals that do not need ad-hoc processes. Table 7-1 lists the selected goals used in our case study. The goals are distributed across 10 domains.

Table 7-1 List of goals

No.	Goal Description	Domain
1	Apply university	Education
2	travel	Travel
3	choose a gift	Shopping
4	Buy a cell phone	Shopping and Electronics
5	Buy a car	Shopping and Automobile
6	Buy a house	Real estate
7	Find a job	Career
8	Tax report	Tax
9	Canada health insurance	Insurance
10	Apply credit card	Finance

7.1.2 Evaluation Methods

To measure the extracted process knowledge, we use recall and precision defined as follows.

$$\mathbf{Precision} = \frac{|{\mathit{relevant\ items}} \cap {\mathit{retrieved\ items}}|}{|{\mathit{retrieved\ item}}|} \quad (7-1)$$

$$\mathbf{Recall} = \frac{|{\mathit{relevant\ items}} \cap {\mathit{retrieved\ items}}|}{|{\mathit{relevant\ items}}|} \quad (7-2)$$

As shown in Equation 7-1 and 7-2, precision is the ratio of the number of returned relevant items to the total number of returned items. Recall is the ratio of the number of returned relevant items to the total number of relevant existing items. In our case study, precision is the ratio of the total number of entities correctly extracted by a tool to the total number of entities extracted by the tool. And recall is the ratio of the total number of entities correctly extracted by a tool to the total number of entities existing in the websites.

To evaluate the quality of ontologies (*i.e.*, process knowledge) extracted from websites, it would be ideal if we could compare our approach with other tools which are specially designed for extracting process knowledge from commercial websites. However, according to the best of our knowledge, there are no public available tools similar to ours which can extract process knowledge from public Web pages without the prior knowledge of the server side source code. Text2Onto [35] is a public available tool to extract ontologies from unstructured or semi-structured textual resources. The extracted ontologies from Text2Onto represent all the information conveyed in the textual resources instead of process knowledge. Text2Onto is the available tool that is the most similar to ours. In this case study, we compare the ontologies extracted using our approach with the ontologies extracted using Text2Onto.

7.1.3 Experimental Procedure

For each goal in Table 7-1, we use Google [59] as the Web search engine to collect the websites related to goals by submitting the goal description as keywords to Google. For each query used to search for relevant websites, we collect the first 8 top ranked websites and apply our approach to each website. In our case study, we integrated Google AJAX Search API [60] into our system to automatically submit the keywords and get the searching results. 8 is the

maximum number of searching results supported by Google AJAX Search API. In total, we analyze 80 related websites for the 10 goals and generate 80 ontologies. For each goal, we produce an integrated ontology to combine the process knowledge extracted from the 8 websites returned by Google.

We apply our approach and Text2Onto tool to extract an ontology from the same website. We evaluate the effectiveness of both approaches by measuring the precision and recall. To assess the process knowledge extracted from each website, a subject spent 3 days manually examining the 80 websites to extract process knowledge. The subject is a graduate student with 5 years experience working on business process related projects. His knowledge and experience on business processes ensure that he can properly identify the process knowledge from these websites. To avoid any interference, the results from text2Onto and our prototype were not disclosed to the subject before he finished the analysis. We calculate the precision and recall by comparing the result from text2Onto and our approach with the result from the subject.

After we extract the process knowledge from each website, our approach combines the process knowledge from the websites relevant to the same goal to generate an integrated ontology. We import the same set of websites to Text2Onto and run Text2Onto to extract an ontology. To provide the standard integrated ontologies to evaluate the recall and precision of the integrated ontologies extracted by our approach and text2Onto, the subject examined all the returned websites for each goal and manually extracted the process knowledge from those websites. The manually identified process knowledge for the given goal is described as an ontology and treated as the standard result for comparisons.

7.1.4 Result Analysis

Table 7-2 lists the recall and precision of our approach and the Text2Onto tool for the ontologies extracted from each website. The results show that our approach can effectively extract most of the processes information from each website. It can achieve a recall of 0.80 and a precision of 0.82 comparing with Text2Onto which has a recall of 0.35 and a precision of 0.06. There are two major reasons that cause Text2Onto to have the low recall and precision. Firstly, Text2Onto simply extracts concepts from textual sources and does not have any mechanism to filter the concepts irrelevant to the goal. For most of Websites, Text2Onto identifies a large number of irrelevant concepts, such as copyright information and advertisements. Secondly, Text2Onto splits many phrases into separated words which make important phrases lose their meaning. For example, “Used Car Price Quotes” is a useful activity when an end-user wants to buy a used car. But Text2Onto may remove the adjectives and only keep the nouns “car” and “Quotes”.

Table 7-2 Recall and precision for ontologies extracted from each website

	Average Recall	Average Precision
Text2Onto	0.35	0.06
Our approach	0.80	0.82

We examined the false positives of our approach and found that one of the major problems is due to the JavaScript code. Our current approach does not parse and make use of any information from JavaScript code. It makes our approach miss some process knowledge in the Web page since JavaScript can be used to display any text on Web pages. If our approach takes JavaScript into consideration, it might increase the recall and precision.

Table 7-3 shows the results for evaluating the integrated ontologies. The results show that our approach can achieve a higher average recall and precision comparing with Text2Onto, for the purpose of extracting process knowledge. The major reason is that Text2Onto is designed as a general tool to extract ontologies from textual resources instead of focusing on extracting process knowledge.

Table 7-3 Recall and precision for integrated ontologies

	Average Recall	Average Precision
Text2Onto	0.52	0.05
Our approach	0.87	0.78

In this case study, we use the opinion from a subject to reach the correct answer (i.e., the correct ontologies extracted from websites). However, the opinion of subjects may not reflect the correct answers since different subjects may have different standards on the information needed to be extracted. In the future, we plan to conduct a Turing test [133] to evaluate our approach. Specifically, we want to check whether subjects can distinguish the results produced by our approach and the results provided by other person. If a system can fool a human being (i.e., make human being hard to distinguish the differences between the information provided by the system and the information provided by another human being), it indicates that the system has intelligence (i.e., has good performance).

7.2 Evaluation of Context-aware Service Discovery and Recommendation

The objective of our case study is to evaluate the effectiveness of our approach. In particular, we want to examine: 1) whether our approach can effectively recommend useful tasks

represented as classes, individuals and properties in the set *Potential Task Set*; and 2) whether the generated searching criteria can find the desired services.

7.2.1 Experimental Setup

Table 7-4 lists the context types used in our case study. By providing different context values for each context type, we can create different end-user scenarios. Each scenario is composed of the context types listed in Table 7-4 with assigned context values. For each scenario, our approach automatically detects various potential tasks for the end-user and recommends different services. In our case studies, we provide 5 different context values for each context type. Using different combinations of these context values, we generate 600 different context scenarios for our case studies.

Table 7-4 Context types used in our case study

Context Types	
Previous environment	Location (city and county)
Current environment	Location (city and country)
	Activity (described by keywords)
Future environment	Location (City and country)
	Activity (provided by calendar, described using keywords)
User's preferences and background	Favorite sports
	Favorite food
	Favorite celebrities
	Major
	Other preferences
	Income

Due to the limitation of time and resources, we cannot evaluate all the 600 scenarios. In our case studies, we randomly select 2% (i.e., 12) context scenarios from the 600 context scenarios to

evaluate our approach. To evaluate the identified potential tasks and the service searching criteria generated by our approach in different scenarios, we recruited 6 subjects, who are graduate students, to participate in our case study. These subjects have many years of experiences using online services and possess the basic knowledge on the context values appearing in the context scenarios.

7.2.2 Evaluation Methods

We use recall and precision [13] to evaluate the effectiveness of our approach. In our case, precision is the ratio of the number of correctly recommended tasks (or services) to the total number of recommended tasks (or services). Recall is the ratio of the number of correctly recommended tasks (or services) to the total number of tasks (services) needed to recommend in the given context scenario.

7.2.3 Experimental Procedure

To evaluate the potential tasks identified by our approach, we assign 2 context scenarios to each subject described in the previous section. For each scenario, a subject manually examines the context values and uses the subject's knowledge to identify the potential tasks that the subject would like to perform. Independent from the manual evaluation, we also use our prototype to automatically identify the potential tasks by analyzing the context values and the relations among context values. We compare the task sets produced by the subjects and the ones generated by our prototype tool to calculate the precision and recall of each scenario.

To evaluate the service searching criteria generated by our approach, we use the approach described in chapter 5 to generate the service searching criteria, then we submit the searching criteria to search engines Google [59] and Seekda [125] to search for online services. Seekda is a

search engine to search for Web services described using WSDL. A subject manually examines the available services in Seekda for each scenario. If there are available Web services in Seekda for a given topic, we use Seekda. Otherwise, we use Google to search for services.

We use the keywords in the generated searching criteria to search for services in Seekda. Then we use the generated WSDL query to check the top 20 returned services from Seekda to identify the matching services. For each query, our prototype chooses the top two returned services to recommend to the subject. The 6 subjects manually provide the description of desired services based on the given context scenarios. A subject manually compares the services recommended by our prototype with the desired services described by the subjects to evaluate if our prototype can correctly recommend services to a subject for the given context scenario.

7.2.4 Result Analysis

Table 7-5 Recall and precision for detecting potential tasks

Scenarios	# of Retrieved tasks	# of Retrieved relevant tasks	# of relevant tasks	Recall	Precision
1	2	2	2	100%	100%
2	1	1	1	100%	100%
3	3	2	3	67%	67%
4	3	3	4	75%	100%
5	2	2	2	100%	100%
6	3	1	1	100%	33%
7	3	3	3	100%	100%
8	3	2	2	100%	67%
9	4	4	4	100%	100%
10	1	1	1	100%	100%
Average				94%	87%

In the 12 context scenarios, 2 scenarios do not have any tasks recommended according to the results from the subjects as well as the results of our prototype. We manually examined both

scenarios. We found that the context values in both scenarios do not have any relations. Table 7-5 shows the results for detecting potential tasks from the remaining 10 scenarios. We notice that some tasks for a certain scenario are not included in the result from the subjects due to the limitation of the end-user's knowledge. However, such tasks are identified by our prototype. For example, in a travel scenario, "Michael Jordan" is a favorite celebrity of a subject, and one of the context values is the city "New York". Our prototype identifies that "New York" is the birth place of "Michael Jordan". As a fan of Michael Jordan, the subject would be interested to know this information and purchase the related souvenirs using an on line shopping service. However, such information is overlooked by the subject. When calculating recall and precision, we add the missed tasks into the relevant items set and treat the missed tasks as desired potential tasks. The 94% of recall reveals that our approach can identify most of the potential tasks based on the semantics of context values. Moreover, our prototype can identify the tasks that are overlooked by the subjects.

In our current approach, we adopt a very simply way to sort the potential tasks, i.e., the potential tasks which have relations with more context values have higher weight, and we sort the potential tasks according to the weight from high to low. In our case studies, we observed that most of the identified potential tasks are inferred from the common entities which are only shared by two context values. Therefore, in the most scenarios, the current sort algorithm does not impact these scenarios and the potential tasks are sorted randomly since the weights of most potential tasks are the same. It is the major reason of the low precision (i.e., 33%) in scenario 6.

Table 7-6 lists the evaluation results of service recommendation. The results show that our approach can recommend most of the needed services desired by subjects. However, as listed in

Table 7-6, the recall and precision are not very high in some context scenarios. Here are some reasons:

- 1) Some ontologies do not describe all the aspects of a context value. The incomplete ontologies cause incomplete service recommendations. Meanwhile, we only define one user-defined relation which is Figure 5-4 to capture the domain knowledge of “Income: low”. If we add more domain knowledge using user-defined relations, it could increase the recall and precision. For example, one subject in our case study lists “Tickets for Museums at Miami” as a potential task for a context scenario which specifies that the subject majors in “Art” and will attend a conference in “Miami”. Due to the lack of domain knowledge of “Art”, it is difficult for our prototype to automatically establish the relations between “Art” and “Museums”.

Table 7-6 Evaluation results of service recommendation

Scenarios	Total # of retrieved services	Total # of retrieved relevant services	Total # of relevant services	Recall	Precision
1	4	4	4	100%	100%
2	2	2	2	100%	100%
3	6	4	6	67%	67%
4	6	6	8	75%	100%
5	4	3	4	75%	75%
6	6	2	2	100%	33%
7	6	5	6	83%	83%
8	6	4	4	100%	67%
9	8	8	8	100%	100%
10	2	2	2	100%	100%
Avarage				90%	83%

- 2) Although WordNet can provide stems and synonyms for a single word, it cannot give the synonyms of phrases (i.e., two or more words in sequences to represent a specific meaning)

which are the most common expressions of entities in ontologies. The lack of phrases in our semantic analysis database (i.e., WordNet) makes it challenging for our prototype to identify the similarity of phrases defined in ontologies.

- 3) When the number of keywords increases, the results returned by Google or Seekda are likely to diminish. Especially, we may extract general terms from ontologies, such as “people”, “person”, and “location”. Such terms in the searching keywords often result in drastically reduction of the quality of searching results.

There are different types of threats which may affect the validity of the result of our case study. The major threats to validity are as follows.

External validity refers to the generalization of the results. In our case studies, we automatically generated 600 different context scenarios and randomly selected 12 scenarios out of the 600 context scenarios. We believe that the automated generation and random selection of context scenarios can reflect the situation of the practices. However, there are various context types and many variations of context values in a context-aware system. Our case studies only evaluate a limited number of context types and values. In the future, we plan to expand our context scenarios with more context types and values. When the number of context types and values increases in our case studies, we expect that the *precision* and *recall* is likely to be lower than the result of our current experiment.

Construct validity is the degree to which the independent and dependent variables accurately measure the concepts which they are intended to measure. We have carefully chosen the criteria to avoid the threats of construct validity. To evaluate the effectiveness of identified context relations and recommended services, we use *recall* and *precision* which are the well adopted evaluation criteria in literature. However, the potential tasks and recommended services for

context scenarios contain subjective issues. For example, one subject may satisfy the recommended task but another subject may not like the recommended task at all. In our case studies, we ask the 6 subjects to provide the potential tasks and evaluate the returned services according to the relations of context values and their understanding of the context scenario. The identified potential tasks and relevant services recommended by the 6 subjects may not reflect the potential tasks of all the end-users in practice. Especially, in our case study, all the 6 subjects are graduate students. In the future, we plan to hire more subjects with different backgrounds to participate in our case studies.

Internal validity is a concern with the cause-effect relationship between independent and dependent variables. In our case studies, the retrieved tasks are automatically identified by our prototype, and the relevant potential tasks are identified by the subjects who did not observe the result of our prototype. Therefore, we can rule out the learning effect that the subjects may be impacted by the results of our prototype.

7.3 Evaluation for Ontology Driven Service Composition

We conduct a case study to evaluate the effectiveness of our approach that can support end-users without the knowledge of SOA technologies to compose services. Our approach can generate ad-hoc processes for end-users and discover the relevant services to fulfill the tasks in the ad-hoc processes. The objectives of our case study are to examine 1) if the ad-hoc process generated by our approach can reflect an end-user's goals; 2) if the tag-based description and the service searching criteria extracted from ontologies can help locate the relevant services with high precision and recall; and 3) if end-users is satisfied with the experience of the service composition.

7.3.1 Experiment Setup

In our case study, we create an ontology database and use two approaches to obtain ontologies. One approach is to retrieve ontologies over the Web. We collect the online ontologies from Freebase [54], DBpedia [40], and Swoogle [132]. Freebase and DBpedia are two ontology databases which extract structured information from Wikipedia [142]. Swoogle is an online ontology search engine. However, for some goals, we are not able to find the matching ontologies from the Internet. In this case, we extract ontologies from the Web to overcome the availability of ontologies.

In the case study, an subject specifies 20 goals and each goal requires a certain process to achieve. Then we use the 20 goals to find 20 different ontologies from our ontology database. The 20 goals describe daily activities in different domains, such as shopping, travel, banking, and entertainment. For example, in the banking domain, we specified the goal as “Credit Card Application”, “Stock Analysis”, and “investment”. The goal of “Credit Card Application” includes tasks such as “Credit Card Options”, “Application Requirements”, and “Online Applications”. Table 7-7 summarizes the distribution of the goals and the number of tasks identified in each domain after we use our approach to generate the ad-hoc processes.

Table 7-7 Characteristics of generated ad-hoc processes

Domain	# of goals	# of tasks
Shopping	6	79
Travel	3	23
Banking	3	17
Entertainment	3	15
Others	5	59

To collect Web services and create a service repository, we manually searched for Web services from the Internet using Seekda and Google. We registered 1,000 Web services into our service repository. Each service in the repository is described using our proposed tag-based service description. The services of different domains are selected to ensure that we can find services that match with the end-users' goals specified in the case study.

To evaluate the generated ad-hoc processes, we recruited 8 end-users to participate in our experiment. Nielsen [103] suggests that the best end-user study for gathering qualitative measures should involve three to five end-users. We carefully select the 8 subjects to make sure that they do not have any background on SOA since they are intended to represent the group of non-professional end-users. Four of the 8 subjects do not have background on SOA but are very familiar with the Internet. The remaining 4 subjects have not knowledge SOA at all and they have the basic knowledge about Internet.

To evaluate the performance of our proposed tag-based service description in the service discovery, we use a baseline approach which requires a developer to manually search for relevant services using keywords as a comparison. We recruit a novice developer, who knows the general concepts of SOA and has limited experience in developing SOA systems, to conduct the manual searching.

7.3.2 Evaluation Methods

In our case study, the number of returned services can be too large for an end-user to review. Instead, an end-user would only go through the first k returned items. Therefore, in addition to recall and precision, we also use the top- k precision and r -precision to measure the effectiveness of our approach for generating ad-hoc processes and discovering appropriate services. The definitions of top- k precision and r -precision are defined as follows.

$$\mathbf{Precision}_k = \frac{|\{\mathit{relevant\ items}\} \cap \{\mathit{top\ }k\ \mathit{retrieved\ items}\}|}{k} \quad (7-3)$$

$$\mathbf{Precision}_r = \mathbf{Precision}_{k=|\{\mathit{relevant\ iterm}s\}|} = \frac{|\{\mathit{relevant\ items}\} \cap \{\mathit{top\ }k\ \mathit{retrieved\ items}\}|}{|\{\mathit{relevant\ items}\}|} \quad (7-4)$$

The top-k precision evaluates the precision for the top-k returned items. For example, consider the case of getting 9 relevant services when 50 services are returned as a result of a query. Those 9 relevant services are listed in the top 10 returns, and there are 20 relevant services in total in the service repository. The top-10 precision is 9/10=90% whereas the precision would be 9/50=18%. The R-precision calculates the precision based on the number of relevant items at the top r returned items, and r is the total number of existing relevant items. In the prior example, the r-precision evaluates the precision of the top 20 returned services since there are 20 relevant services in the entire repository. The r-precision in this example would be 9/20=45%.

To evaluate the end-user satisfaction of using our approach to compose services, we conducted a user study. We ask each of the 8 subjects as described in Section 7.3.1 to compose 3 ad-hoc processes by providing 3 goals using our prototype. After the subjects completed the service composition, we asked them to complete a short survey to assess their experience with the service composition.

7.3.3 Experimental Procedure

7.3.3.1 Evaluating the Generated Ad-hoc Processes

We conduct a user study to evaluate if the generated ad-hoc process can match well with an end-user's expectation to achieve a goal. We asked the 8 aforementioned subjects to manually specify 6 ad-hoc processes for 6 given goals based on their experience and domain knowledge.

Due to the limitation of time and recourses, we are not able to ask subjects to describe the ad-hoc processes for all the 20 goals described in Section 7.3.1. We randomly select 6 goals out of the 20 goals and provide them to the 8 subjects. Given a goal, the tasks in an as-designed ad-hoc process which is provided by the subjects are treated as the relevant tasks (*i.e.*, *relevant items* in Equation 7-1 and 7-2), and the tasks generated by our approach are treated as returned tasks (*i.e.*, *retrieved items* in Equation 7-1 in 7-2). We calculate the recall and precision of each ad-hoc process, and use the average recall and precision to evaluate the overall performance of our generated ad-hoc processes.

7.3.3.2 Evaluating the Performance of Service Discovery

We compare the performance of our proposed tag-based service in discovering Web services with a baseline approach which requires manually searching for relevant services using keywords. Due to a large number of tasks generated in 20 ad-hoc processes, we are not able to evaluate all associated services. To compare the recall and precision of our approach with the baseline approach, we use 6 ad-hoc processes from the 20 generated ad-hoc processes. To make the *top-k precision* meaningful, each task in the 6 ad-hoc processes has at least 6 relevant services in our service repository. The novice developer manually searches for services to match with the tasks generated from the 6 ad-hoc processes.

To compare our approaches with the baseline approach using the same set of tasks, we provide the 30 discovered tasks in 6 ad-hoc processes as described in Table 7-8 to the developer, who manually specifies keywords from their knowledge of the tasks as search criteria to query the service repository. The services are described using WSDL, without the tag-based service description as proposed in Chapter 6.

To calculate the *recall* and *r-precision*, one graduate student spent around 3 weeks manually analyzing the 1,000 Web services registered in the service repository and identifying the relevant services for each task.

Table 7-8 Characteristics of the six ad-hoc processes

Process ID	Name of ad-hoc processes	# of Entities	# of Tasks
1	Planning a Trip	47	6
2	Watching Movie	28	5
3	Online Shopping	29	5
4	Credit Card Application	24	4
5	Stock Analysis	27	3
6	job-hunting	18	7

7.3.3.3 Evaluating User Experience of the Service Composition

The survey for evaluating user experience of the service composition contains the following questions. We indicate the rationale of each question in parentheses.

- 1) Whether the relations among tasks help you to navigate among different tasks? (Easiness of task navigation)
- 2) Does the tag description help you understand the functionality of services and select services? (Understandability of tag-based service description)
- 3) How helpful is the prototype for you to find online services compared with the other service mediator websites (*e.g.*, Expedia)? (Helpfulness to end-users)
- 4) How much SOA knowledge is required for composing services using this prototype? (Requirement of SOA background)

The survey provides five choices for each question to measure the degree of the answer, such as “almost never”, “a little”, “sometimes”, “often”, and “almost always”. Those answers are mapped to the score of 0 to 5. 5 represents the most positive answer and 0 represents the most negative answer.

7.3.4 Results Analysis

7.3.4.1 Results of Evaluating the Generated Ad-hoc Processes

The average recall and precision for the tasks in the generated ad-hoc processes are 0.80 and 0.84 respectively. We find that the as-designed ad-hoc processes vary a lot due to the differentiation of the subjects’ preferences. Therefore, the recall and precision of tasks in the generated ad-hoc processes are not very high. The result of the case study shows that the generated ad-hoc process can provide the major tasks to fulfill an end-user's goal. As a complement, in our approach, end-users can edit the generated ad-hoc process to make them satisfy their own specific requirements for future reuse.

7.3.4.2 Results of Evaluating Service Discovery

Table 7-9 Recall and R-precision comparison

	Recall	R-precision
Our approach	0.93	0.61
Baseline	0.67	0.33

Table 7-9 lists the average recall of our approach and the baseline approach. In the searches for 30 tasks, our approaches can find all the relevant services with a recall of 93%. In the baseline approach, a few relevant services are not returned using the provided keywords since the developer does not use the same words as the WSDL description to search for Web services. In

summary, our approach has a higher recall. Table 7-9 lists the average r-precision of each task for both approaches. Our approach has higher r-precision than the baseline approach.

We calculate the averages of the *top-k* precisions (ranging from *top-1* precision to *top-6* precision) for all the tasks. Figure 7-1 shows the results for average *top-k* precision for all 30 tasks, when *k* ranges from 1 to 6. As shown in Figure 7-1, our approach outperforms the baseline approach. The ontology definition used in our approach captures the expert knowledge and provides more relevant search keywords for each task; and therefore increases the success of the service discovery.

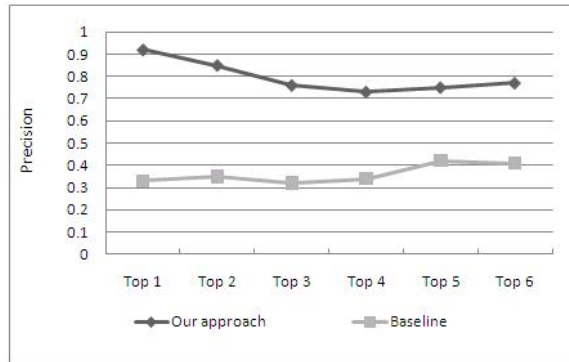


Figure 7-1 Top-k precision

We use the precision vs. recall graph to display the performance of both approaches as shown in Figure 7-2. The ideal approach should achieve high precision and high recall. A good performance is indicated by the trend line of an approach appearing in the upper right portion of the graph shown in Figure 7-2. As shown in Figure 7-2, the precision rate of our approach decreases slower than the baseline approach as the recall increases. As a result, our approach demonstrates higher precision and recall.

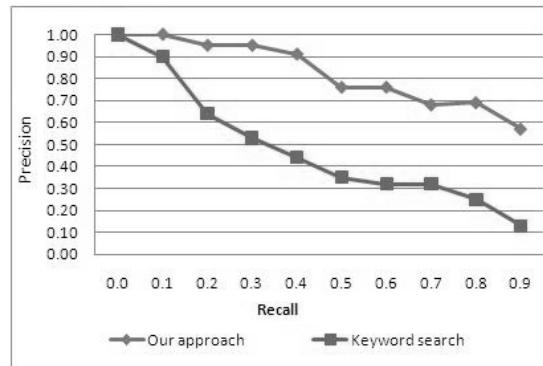


Figure 7-2 Recall vs. precision curves

When searching for services, we observe that the baseline approach is highly dependent on the keywords provided by the novice developer and his domain knowledge. When a novice developer is not familiar with the application domain, the search with the provided keywords often returns no services although several relevant services exist in the service repository. The service retrieval in our approach uses the expert knowledge captured in an ontology and the tag-based services description. Our approach is independent from an end-user’s familiarity with the domain and their knowledge of Web services. Therefore, our approach achieves high precision and recall in the service discovery.

7.3.4.3 Results of Evaluating User Experience

Table 7-10 lists the average value of each question in our survey. In Table 7-10, the first column shows the question number corresponding to the questions in the survey. The last column is the average value of the scores provided by the 8 subjects. The results indicate that our tag-based service description can help subjects to understand the service descriptions and the generate ad-hoc process can reduce the workload of composing services. The subjects can get relevant services automatically without having to search for the services over the Web. The services are

organized in an abstract ad-hoc process which makes it easy for them to navigate through the services. Moreover, the subjects found that the tag-based service information intuitive for understanding the functionality and usage of the services.

Table 7-10 Results of satisfaction evaluation

Question No.	Metrics	Average Value
1	Easiness of task navigation	4.6
2	Understandability of tag-based service description	4.4
3	Helpfulness to end-users	4.2
4	Requirement of SOA background	0.3

There are different types of threats which may affect the validity of the result of our case study. The major threats to validity are as follows.

External validity refers to the generalization of the results. Instead of simulating Web services and using self-designed ontologies, our case studies use publically available Web services and use the ontologies collected from the Internet or generate the ontologies from on-line Web pages. We believe that such Web services and ontologies can better reflect the situation in practice. However, there are a large number of Web services available on the Internet in various domains. Our case study only considered a limited number of Web services from six domains. In the future, we plan to expand our services repository and collect more services from different domains. When the number and the domains of Web service increase in our service repository, we expect that the *precision* and *recall* is likely to be lower than the results of our experiment.

In the experiment, we note that the generated process depends on the quality of the ontology. If an ontology does not define the main concepts of the goal or does not represent concepts in a good structure, the generated process may include useless tasks or the generated process might miss some important tasks for achieving the goal.

In addition, our approach cannot adequately control the granularity of tasks in the ad-hoc processes. We find that some generated ad-hoc processes in our experiment contain too many detailed tasks. Those tasks are relevant to the goal but useless to help end-users fulfill their needs. In the future, we plan to collect and analyze the context information of end-users (*e.g.*, end-user's preferences and historical data). The context information might help us to select high quality ontologies and refine the ad-hoc processes.

Construct Validity is the degree to which the independent and dependent variables accurately measure the concepts which they are intended to measure. In our case studies, we have carefully chosen the criteria to avoid the threats of construct validity. To evaluate the effectiveness of generated ad-hoc processes and the tag-based service description, we use *recall* and *precision* which are the well adopted evaluation criteria in literature. In the survey of end-user's satisfactions, we use multiple-choice to help participants provide their feedback accurately. However, the as-designed ad-hoc processes and end-user's satisfactions contain subjective issues. The results of our evaluations may not exactly represent the feedback of all the end-users in the real world.

Internal validity is a concern with the cause-effect relationship between independent and dependent variables. While comparing the baseline approach and our tag-based approach for service discovery, the tag-based approach is executed automatically using our prototype and the baseline approach is conducted by a developer who did not observe the result of the tag-based approach. Therefore, we can rule out the learning effect that the developer may learn from the generated ad-hoc processes. To avoid interference, the experiment was run with subjects who had never done a similar experiment.

Communication between the subjects influences the response of subjects. We ruled out this threat by executing the experiment in an observed room where the subjects were not allowed to communicate. In addition, subject's knowledge on the relevant domains can also impact the result. In our experiment, all the ad-hoc processes in our case studies are relevant to daily activities. Therefore, every subject can use their knowledge to provide rational evaluations on the ad-hoc processes.

7.4 Summary

This chapter presents the evaluation of our proposed framework. Firstly, we conduct a case study to compare our process knowledge extraction approach with a tool that extracts ontologies from textual sources. The result of the case study shows that our approach can extract process knowledge from online applications with higher precision and recall comparing to the ontology learning tool. Secondly, a case study is conducted to evaluate the effectiveness of our approach for context-aware service recommendation. The results show that our approach can use contexts to find end-users' requirements and recommend their desired services with high precision and recall. Thirdly, we compare the performance of our approach in automatic service composition with a baseline approach which consists of the manual process of searching for services using keywords. The results show that our approach can achieve higher precision and recall than the baseline approach.

Chapter 8

Conclusions and Future Work

To support non-IT professional end-users to compose services, we present a framework that composes services on-the-fly and provides personalized service recommendation for end-users. The framework hides the complexity of SOA standards from end-users and helps end-users fulfill their daily activities. Instead of requiring end-users to specify detailed steps during service composition, our framework only requires the end-users to specify the goals of their desired activities using a few keywords to generate an ad-hoc process. The process knowledge from the existing ontologies and websites are used to guide the generation of ad-hoc processes. By discovering the semantic relations among context values using ontologies, our approach can identify an end-user's needs hidden in the context values and recommend the desired services.

8.1 Thesis Contributions

The major contributions of this thesis are summarized as follows:

- **Derive automatic techniques to extract ontologies with process knowledge from the Web.** Ontologies are used as a form of knowledge representation and sharing to compose services. Most existing ontologies provided by ontology search engines and online knowledge bases are designed for classifying and representing general knowledge without capturing the process knowledge on how an end-user activity can be accomplished. We observe that many websites provide specific services to fulfill different goals. For example, expedia.com helps end-users plan a trip. The process knowledge is embedded in such websites. To obtain the process knowledge for service composition, we provide an approach which extracts ontologies with the process knowledge from various websites. Our approach uses existing Web search engines

to find websites with embedded process knowledge. By analyzing the content and the structure of the relevant websites, we identify the tasks needed for completing an embedded process. To provide comprehensive process knowledge for achieving a goal, our approach merges the process knowledge extracted from multiple websites that serve for the same goal (e.g., travel planning) to generate an integrated ontology with rich process knowledge.

- **Propose an approach to personalize service recommendation using contexts.** Effective service recommendation requires the ability to detect an end-user's context at the run-time environment to dynamically recommend services. We design and develop techniques that automatically analyze an end-user's context which denotes changes in an end-user's environment (i.e., operational and physical environments), and use the context to recommend services. Our approach reduces the amount of information required from end-users in order to specify well-defined criteria for searching Web services. More specifically, we use ontologies to enhance the meaning of an end-user's context values and automatically identify the relations among different context values. By discovering the semantic relations among context values, we can identify an end-user's needs hidden in the context values and generate searching criteria for service recommendation.
- **Develop strategies to dynamically compose services for end-users on-the-fly.** In the current state of practice, SOA practitioners manually describe the details of each task and the interactions among tasks using BPEL. In an end-user environment, most of the activities or goals (e.g., plan a trip) are spontaneously prompted from end-users. End-users may not have a clear plan on achieving a goal. To guide end-users to achieve their goal, we devise techniques that dynamically search and compose services on-the-fly to assist end-users in fulfilling their desired goals (e.g., planning a trip). Our approach uses ontologies to extend the meaning of

end-user's goals and generate ad-hoc processes. Our approach shelters end-users from complex programming issues.

8.2 Future Research Directions

This section describes our future research directions based on some of the limitations that we observed in our current approaches.

- **Automatic Techniques to generate input data for services:** Automatic generation of input data for services can reduce the workload of end-users and further increase the automation of service composition. Currently, automatic generation of input data for executing services is still a challenging work. In our framework, end-users need to manually provide the input data for services. In the future, we plan to develop techniques to automatically generate service input data by analyzing end-user's context, historical execution data and the output of other services.
- **Effective criteria to choose an appropriate ontology with desired process knowledge:** we found that the quality of generated ad-hoc processes highly depends on the quality of the ontology relevant to the goal. There may have several matching ontologies for the same goal. However, there are no effective criteria to help us select the appropriate ontologies for the generation of an ad-hoc process. A further study can be conducted to evaluate the effectiveness of different criteria for ontology selection and identify the effective criteria for our framework.
- **Service composition techniques to use the process knowledge from multiple resources:** In our framework, the process knowledge can be derived from online knowledge bases. It can also be extracted from existing online websites. However, many other resources contain the process knowledge, such as online documents, publicly available BPEL processes, business

process databases (e.g., The MIT Process Handbook Project [92]). To improve the quality of the generated ad-hoc processes, we can develop new service composition approaches to use the process knowledge from multiple resources.

- **Approaches to enhancing the tag-based service description schema:** Our framework provides a tag-based schema to enable end-users to describe WSDL services using tags. We believe that enabling end-users to refine the service descriptions can improve the quality of semantic service description. However, tags (i.e., keywords) have limited capability to describe semantic meanings comparing with natural languages (e.g., English) or formal languages (e.g., ontology language). The future work could enhance the expressiveness of the tag-based service description schema. The enhancement should ensure that the improved schema can be easily processed by computers and also simple enough to make non-IT professional end-users to understand and edit.

Bibliography

- [1] W. M. P. van der Aalst, A. J. M. M. Weijters and L. Maruster, “Workflow Mining: Which Processes Can Be Rediscovered?” BETA Working Paper Series, WP 74, Eindhoven University of Technology, Eindhoven, 2002
- [2] S. Abbar, M. Bouzeghoub, S. Lopez, “Context-aware recommendation systems: a service-oriented approach,” In Proceedings of the International Conference on Very Large Data Bases (VLDB) Profile Management and Context Awareness (PersDB) Workshop, Lyon, France, 2009
- [3] ActiveVOS Designer, available at: <http://www.activevos.com/products/activevos/overview>, last time accessed on September 21, 2011
- [4] D. Aguilar-Lopez, I. Lopez-Arevalo, V. Sosa-Sosa, “Toward the Semantic Search by Using Ontologies,” In Proceedings of the International Conference on Electrical Engineering, Computing Science and Automatic Control, Mexico City, Mexico, November 2008, pages: 328 – 333.
- [5] R. Agrawal, D. Gunopulos, and F. Leymann, “Mining Process Models from Workflow Logs”, In Proceedings of the Sixth International Conference on Extending Database Technology, 1998, pages: 469-483.
- [6] R. Akkiraju, J. Arrell, J. Miller, etc., “Web Service Semantics-WSDL-S,” W3C member submission, 7 November, 2005
- [7] E. Al-Masri, Q. Mahmoud, “Investigating Web Services on the World Wide Web: an Empirical Study,” In Proceedings of the 17th ACM International World Wide Web Conference, Beijing, China, 2008, pages: 795-804
- [8] Angele J. and Lausen G. “Ontologies in F-logic,” In: S. Staab and R. Studer (editors), Handbook on Ontologies in Information Systems, Springer Verlag, Berlin, Germany, 2004, pages: 29-50
- [9] A. Ankolekar, M. Burstein, J. R. Hobbs, R. Lassila, et al. “DAML-S: Web Service Description for the Semantic Web,” In Proceedings of the First International Semantic Web Conference (ISWC), Sardinia, Italy, June 10-12, 2002, pages. 348-363

- [10] K. Arabshian, C. Dickmann and H. Schulzrinne, "Ontology-Based Service Discovery Front-End Interface for GloServ," LNCS in the Semantic Web: Research and Application, 2009, pages: 684 - 696
- [11] I. B. Arpinar, B. Aleman-Meza, R. Zhang, and A. Maduko, "Ontology-Driven Web Services Composition Platform," In Proceedings of the IEEE International Conference on E-Commerce Technology, 2004, pages: 146-152
- [12] D. Bachlechner, K. Siorpaes, "Web Service Discovery - A Reality Check," In Proceedings of the Third European Semantic Web Conference (ESWC 2006), Demos and Posters, Budva, Montenegro, June, 11-14, 2006,
- [13] R. Baeza-Yates, B. Ribeiro-Neto, "Modern Information Retrieval," New York: ACM Press, Addison-Wesley, 1999
- [14] M. Baldauf, S. Dustdar and F. Rosenberg, "A Survey on Context-aware Systems," International Journal of Ad Hoc and Ubiquitous Computing, Volume 2, Issue 4, June 2007, pages: 263-277
- [15] W. T. Balke, M. Wagner, "Towards Personalized Selection of Web Services," In Proceedings of the International World Wide Web Conference (WWW 03) 2003, Budapest, Hungary, 2003, pages: 725-733
- [16] D. Beckett, B. McBride, "RDF/XML Syntax Specification (Revised)", W3C Recommendation (2004), available at <http://www.w3.org/TR/rdf-syntax-grammar/>, last accessed on May 23, 2011
- [17] B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, F. Toumani, "On Automating Web Services Discovery," VLDB(Very Large Data Bases) Journal (2005) 15, pages:84-96.
- [18] Bing, <http://www.bing.com>, last time accessed on May 25, 2011
- [19] M. B. Blake, D. R. Kahan, M. F. Nowlan, "Context-aware Agents for end-user-oriented Web Services Discovery and Execution," Distribute Parallel Databases (2007) 21, pages: 39-58
- [20] P. Brézillon, "Focusing on context in human-centered computing," IEEE Intelligent Systems 18, 3, May/June 2003, pages: 62-66
- [21] S. Brin and L. Page, "The Anatomy of a Large-Scale Hyper-textual Web Search Engine," In Proceedings of the 7th international conference on World Wide Web, Brisbane, Australia, April 14-18, 1998, pages:107-117

- [22] T. Broens, S. Pokraev, M. V. Sinderen, J. Koolwaaij, P. D. Costa, "Context-Aware, Ontology-Based Service Discovery," In Proceedings of the European Symposium on Ambient Intelligence, Eindhoven, The Netherlands, 2004, LNCS 3295, 2004, pages: 72-83
- [23] Business Process Model and Notation (BPMN), FTF beta for version 2.0, available at: <http://www.omg.org/cgi-bin/doc?dtc/09-08-14.pdf>, last accessed on May 25, 2011
- [24] M. P. Carlson, A. H. H. Ngu, R. M. Podorozhny, L. Zeng, "Automatic Mash Up of Composite Applications," In Proceedings of the International Conference on Service Oriented Computing (ICSOC) 2008, Sydney, Australia, December 1-5, 2008, pages: 317-330
- [25] K. S. M. Chan, J. Bishop and L. Baresi, "Survey and Comparison of Planning Techniques for Web Services Composition," Technical Report, Polelo Research Group Department of Computer Science, University of Pretoria Pretoria, South Africa, 2007
- [26] S. Chandrasekaran, J. A. Miller, G. S. Silver, B. Arpinar and A. P. Sheth, "Performance Analysis and Simulation of Composite Web Services," Electronic Markets 2003, Volume 13(3), pages:120-132
- [27] C. H. Chang, M. Kayed, M. R. Girgis, K. F. Shaalan, "A Survey of Web Information Extraction Systems," IEEE transactions on Knowledge and Data Engineering, Vol. 18, No. 10, October 2006, pages: 1411-1428
- [28] G. Chen and Kotz D., "A Survey of Context-Aware Mobile Computing Research," Dartmouth Computer Science Technical Report TR2000-231, 2000
- [29] I. Y. L. Chen, S. J. H. Yang, J. Jiang, "Ubiquitous provision of context aware Web services," In Proceedings of the IEEE International Conference on Services Computing (SCC) 2006, Chicago, USA, September 18-22, 2006, pages: 60-68
- [30] P. P. Chen, "The Entity-Relationship Model - Toward a Unified View of Data," ACM Transactions on Database Systems (TODS), volume 1, issue 1, 1976, pages: 9-36
- [31] X. Chen, X. Liu, Z. Huang, H. Sun, "RegionKNN: a Scalable Hybrid Collaborative Filtering Algorithm for Personalized Web Service Recommendation," In Proceedings of the International Conference on Web Services, Miami, FL, USA, 2010, pages: 9-16

- [32] R. Chinnici, J. J. Mreau, A. Ryman, S. Weerawarana (editors), "Web Services Description Language (WSDL) Version 2.0," W3C Recommendation, June 26, 2007, available at: <http://www.w3.org/TR/wsdl20/>, last accessed on September 30, 2011.
- [33] E. Christensen, F. Curbera, G. Meradith, S. Weerawarana (editors), Web Services Description Language (WSDL) 1.1, W3C Note, March 15, 2001, available at <http://www.w3.org/TR/wsdl>, last accessed on May 25, 2011
- [34] U. Chukmol, "A Framework for Web Service Discovery: Service's reuse, quality, evolution and end-user's data handling," In Proceedings of the 2nd SIGMOD PhD Workshop on Innovative Database Research (IDAR 2008), Vancouver, Canada, June 13, 2008, pages: 13-18
- [35] P. Cimiano, J. Völker, "Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery," In Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB), vol. 3513 of Lecture Notes in Computer Science, Springer, Alicante, Spain, June 2005, pages: 227-238
- [36] L. Clement, A. Hately, C. von Riegen, T. Rogers, etc., "UDDI Version 3.0.2," UDDI Spec Technical Committee Draft, October 19, 2004, available at: http://www.uddi.org/pubs/uddi_v3.htm, last accessed on May 25, 2011
- [37] D. Connolly, F. V. Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "DAML+OIL Reference Description," W3C Note, December 2001, available at: <http://www.w3.org/TR/daml+oil-reference>, last time accessed on May 17, 2010.
- [38] A. D'Ambrogio, "A WSDL Extension for Performance-enabled Description of Web Services," LNCS vol. 3733/2005, In Proceedings of the 20th International Symposium on Computer and Information Sciences (ISCIS'05), Istanbul, Turkey, October 26-28, 2005, pages: 371-381
- [39] AnDrea D' Ambrogio, Paolo Bocciarelli, "A Model-driven Approach to Describe and Predict the performance of Composite Services," In Proceedings of the Sixth International Workshop on Software and Performance (WOSP'07), Buenos Aires, Argentina, February 5-8, 2007, pages: 78-89
- [40] DBpedia, <http://dbpedia.org/About>, last accessed on August 16, 2011

- [41] A. K. Dey, et al. "Toward a Better Understanding of Context and Context-Awareness," In Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing, Karlsruhe, Germany, September 27-29, 1999, pages: 304-307
- [42] A. K. Dey, D. Salber, G. D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications," Human-Computer Interaction (HCI) Journal, Vol. 16(2-4), 2001, pages: 97-166
- [43] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity Search for Web Services," In Proceedings of the 30th International Conference on Very Large Data Bases, Toronto, Canada, August 29-September 3, 2004, pages: 372-383
- [44] Dublin Core, available at <http://dublincore.org/>, last accessed on May 25, 2011
- [45] S. Dustdar and W. Schreiner, "A survey on Web Service Composition," International Journal of Web and Grid Services, Vol. 1, No. 1/2005, 2005, pages: 1-30
- [46] EHow, <http://www.ehow.com/>, last accessed on May 25, 2011
- [47] Expedia, <http://www.expedia.com/>, last accessed on May 25, 2011.
- [48] Facebook, <http://www.facebook.com/>, last accessed on May 25, 2011
- [49] W. Fang, L. Zhang, Y. Wang, S. Dong, "Toward a Semantic Search Engine Based on Ontologies," In Proceedings of the International Conference on Machine Learning and Cybernetics, Guangzhou, China, April, 2005, pages: 1913 - 1918
- [50] J. Farrell, H. Lausen (editors), "Semantic annotation for WSDL and XML schema," W3C Recommendation, 28 August, 2007
- [51] J. Floch, E. Stav, and E. Blakstad, "Compose Your Own City Guide," VERDIKT Conference, Oslo, Norway, November 3-4, 2009, available at: <http://www.sintef.no/home/Publications/Publication/?page=125679>, last accessed on May 25, 2011
- [52] M. Flügge, D. Tourtchaninova, "Ontology-derived Activity Components for Composing Travel Web Services," In Proceedings of the International Workshop on Semantic Web Technologies in Electronic Business (SWEB2004), 2004
- [53] C. D. Francescomarino, A. Marchetto and P. Tonella, "Reverse Engineering Of Business Process Exposed As Web Applications", In Proceedings of the European Conference On

- Software Maintenance And Reengineering, Kaiserslautern, Mar. 24-27, 2009, pages: 139-148.
- [54] Freebase, <http://www.freebase.com/>, last accessed on August 23, 2011.
- [55] A. Gao, D. Yang, S. Tang, M. Zhang, "Web Service Composition Using Markov Decision Processes," In Proceedings of the 6th International Conference on Web-Age Information Management (WAIM) 2005, Hangzhou, China, October, 11-13, 2005, pages: 308-319
- [56] J. Garofalakis, Y. Panagis, E. Sakkopoulos, "Web Service Discovery Mechanisms: Looking for a Needle in a Haystack," In Proceedings of the International Workshop on Web Engineering 2004, Santa Crus, CA, USA, August 10, 2004
- [57] J. Gekas, "Web Service Ranking in Service Networks," Demos and Posters of the 3rd European Semantic Web Conference (ESWC 2006), Budva, Montenegro, June 11-14, 2006
- [58] J. Gekas and M. Fasli, "Automatic Web Service Composition Using Web Connectivity Analysis Techniques," W3C Workshop on Frameworks for Semantics in Web Services 2005 Position Paper, Innsbruck, Austria, June 9-10, 2005
- [59] Google, <http://www.google.com/>, last accessed on August 23, 2011
- [60] Google AJAX Search API: Class Reference, <http://code.google.com/apis/ajaxsearch/documentation/reference.html>, last accessed on May 11, 2011
- [61] T. Gruder, "Ontology," Encyclopedia of Database Systems, Springer-Verlag, 2009
- [62] S. Gupta, G. Kaiser, D. Neistadt, P. Grimm, "DOM-based Content Extraction of HTML Documents," In Proceedings of the Twelfth International World Wide Web Conference, Budapest, Hungary, May 20-24, 2003, pages: 207-214
- [63] M. J. Hadley, "Web Application Description Language (WADL)," Sun Microsystems Inc., February 2, 2009
- [64] Y. Hao, Y. Zhang, and J. Cao, "WSXplorer: Searching for Desired Web Services," In Proceedings of the 19th International Conference on Advanced Information System Engineering (CAiSE), LNCS 3395, 2007, pages: 173-187
- [65] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. Miller, M. Wang, "A framework for semantic link discovery over relational data," In Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM), Hong Kong, China, 2009, pages: 1027-1036

- [66] J. He, T. Gao, W. Hao, I.-L. Yen, and F. Bastani, "A flexible content adaptation system using a rule-based approach," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, No. 1, January, 2007, pages: 127-140
- [67] J. He, I. L. Yen, "Adaptive end-user Interface Generation for Web Services," In *Proceedings of the 2007 IEEE International Conference on e-Business Engineering*, Hong Kong, October 24-26, 2007, pages: 536-539
- [68] C. Hesselman, A. Tokmakoff, P. Pawar, S. Iacob, "Discovery and Composition of Services for Context-Aware Systems," In *Proceedings of the 1st European Conference on Smart Sensing and Context 2006 (EUROSSC 2006)*, Enscheda, The Netherlands, 25-27 October, 2006, pages: 67-81
- [69] J. Hoxha, and S. Agarwal, "Semi-automatic Acquisition of Semantic Descriptions of Processes in the Web," In *Proceedings of the 2010 International Conference on Web Intelligence and Intelligent Agent Technology*, Toronto, Canada, Aug. 31- Sept. 3, 2010, pages: 256-263
- [70] M. N. Huhns, M. P. Singh, "Service-Oriented Computing: Concepts, Characteristics and Directions," *IEEE Internet Computing*, January-February, 2005, pages: 3-12
- [71] S. Hu, V. Muthusamy, G. Li, H. Jacobsen, "Distributed Automatic Service Composition in Large-Scale Systems," In *Proceedings of the Distributed Event-Based Systems Conference (DEBS) Rome, Italy, July 1-4, 2008*, pages: 233-244
- [72] IBM Mashup Center, <http://www-01.ibm.com/software/info/mashup-center/>, last accessed on May 25, 2011.
- [73] IBM WebSphere Business Modeler, available at: <http://www-01.ibm.com/software/integration/wbimodeler/>, last accessed on May 15, 2011
- [74] IBM WebSphere Integration Developer (WID), <http://www-01.ibm.com/software/integration/wid>, last access on March 23, 2011
- [75] IBM WebSphere Service Registry and Repository, <http://www-01.ibm.com/software/integration/wsrr/>, last accessed on May 25, 2011
- [76] M. Keidl, A. Kemper, "Towards Context-Aware Adaptable Web Services," In *Proceedings of the International World Wide Web Conference (WWW) 2004*, New York, NY, USA, 2004, pages: 55-65

- [77] D. Khshraj, O. Lassila, "Ontological Approach to Generating Personalized end-user Interfaces for Web Services," *Lecture Notes in Computer Science*, Volume 3729/2005, page 916-927, 2005, pages: 916-927
- [78] G. Klyne and J. J. Carroll (editors), "Resource description framework(RDF): Concepts and abstract syntax," *W3C Recommendation*, 2004
- [79] M. Krause, C. Linnhoff-Popien, M. Strassberger, "Concurrent Inference of High Level Context Using Alternative Context Construction Trees," In *Proceedings of the 3rd International Conference on Automatic and Autonomous Systems (ICAS)*, Athens, Greece, 2007, page 7-7
- [80] U. Küster, M. Stern, B. König-Ries, "A Classification of Issues and Approaches in Service Composition," *International Workshop on Engineering Service Compositions*, 2005.
- [81] H. Lausen and T. Haselwanter, "Finding Web Services," In *Proceedings of the 1st European Semantic Technology Conference (ESTC)*, Vienna, Austria, April 18, 2007
- [82] Q. Liang, J. Chung, S. Miller, Y. Ouyang, "Service Pattern Discovery of Web Service Mining in Web Service Registry-Repository," In *Proceedings of the IEEE International Conference on E-Business Engineering*, , Shanghai, China, October 24-26, 2006, pages: 286-293
- [83] G. Li, V. Muthusamy, and H. Jacobsen, "A Distributed Service Oriented Architecture for Business Process Execution," *ACM Transaction on the Web*, Vol. 4, No. 1, January 2010, Article No. 2
- [84] Y. Li, Y. Liu, L. Zhang, G. Li, B. Xie, and J. Sun, "An Exploratory Study of Web Services on the Internet," In *Proceedings of the 2007 IEEE International Conference on Web Services*, Salt Lake City, Utah, USA, 2007, pages:380-387
- [85] X. Liu, G. Huang, H. Mei, "Towards End end-user Service Composition," In *Proceedings of the 31st Annual International Computer Software and Applications Conference*, Beijing, China, 2007, pages: 667-678
- [86] X. Liu, G. Huang, H. Mei, "A User-Oriented Approach to Automated Service Composition," In *Proceedings of the 2008 IEEE International Conference on Web Services (ICWS)*, Short paper, Beijing, China, September 23-26, 2008, pages: 773-776

- [87] X. Liu, Y. Hui, W. Sun and H. Liang, "Towards Service Composition Based on Mashup," In Proceedings of the 2007 IEEE Congress on Services (SERVICES), Salt Lake City, Utah, USA, July 9-13, 2007, pages: 332-339
- [88] Y. Liu and A. Agah, "A Prototype Process-Based Search Engine," In proceeding of the 2009 IEEE International Conference on Semantic Computing, 2009, pages: 481- 486
- [89] Y. Liu, and A. Agah, "Crawling and Extracting Process Data from the Web," In Proceedings of the International Conference on Advanced Data Mining and Applications (ADMA), Beijing, China, August 17-19, 2009, pages: 545-552
- [90] I. W. Ma, "Ontology base Web Services Composition," Master Thesis, National Central University, Taiwan, 2007
- [91] Z. Maamar, D. Benslimane and N. G. Narendra, "What can Context do for Web Services," Communications of ACM, Vol. 49. No. 12, 2006, Pages: 98-103
- [92] T. W. Malone, K. Crowston and G. A. Herman (editors), "Organizing Business Knowledge: The MIT Process Handbook," Cambridge, MA:MIT press, 2003
- [93] F. Manola, E. Miller, B. McBride (editors), "RDF primer," W3C recommendation, available at <http://www.w3.org/TR/rdf-primer/>, 2004, last accessed on May 25, 2011
- [94] D. Martin, M. Burstein, J. Hobbs, et al (editors)., "OWL-S: Semantic Markup for Web Services," Technical Report, W3C Member Submission, November 22, 2004
- [95] P. Massimo, S. Katia, K. Takahiro, "Delivering Semantic Web Services," In Proceedings of the International World Wide Web Conference (WWW) 2003, Alternate Paper Tracks, Budapest, Hungary, May 20-24, 2003
- [96] S. McIlraith, T. C. Son, "Adapting Golog for Composition of Semantic Web Services," In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning, Toulouse, France, April 2002, pages: 482-493
- [97] S. McIlraith, T. C. Son, "Semantic Web Services," IEEE Intelligent Systems 16(2), March/April 2001, pages: 46-53
- [98] N. Milanovic, M. Malek, "Current Solutions for Web Service Composition," IEEE Internet Computing Magazine, Vol. 8, Issue 6, 2004, pages: 51-59
- [99] J. Montgomery, "Microsoft Popfly: Building Games without a CS Degree," available at: <http://expression.microsoft.com/en-us/cc963994.aspx>, last accessed on August 23, 2011.

- [100] S. K. Mostefaoui, B. Hirsbrunner, "Context Aware Service Provisioning," In Proceedings of the International Conference on Pervasive Services (ICPS) 2004, Beirut, Lebanon, July 19-23, 2004, pages: 71-80
- [101] M. Y. Muriankara, "SOA Governance Framework and Solution Architecture," IBM DeveloperWorks, May 15, 2008, available at: http://www.ibm.com/developerworks/webservices/library/ws-soa-govframe/index.html?S_TACT=105AGX04&S_CMP=EDU, last accessed on May 25, 2011
- [102] S. Narayanan, S. A. McIlraith, "Simulation, Verification and Automated Composition of Web Service," In Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA, May 2002, pages: 77-88
- [103] J Nielsen, "Success Rate: The Simplest Usability Metrics," Jakob Nielsen's Alertbox, Feb. 2001
- [104] Z. Obrenovic and D. Gasevic, "End-User Service Composition: Spreadsheets as a Service Composition Tool," IEEE Transactions on Service Computing, vol. 1, No. 4, October-December, 2008, pages: 229-242
- [105] Oracle BPEL Process Manager, available at: <http://www.oracle.com/technetwork/middleware/bpel/overview/index.html>, last accessed on September 22, 2011
- [106] Organization for the Advancement of Structured Information Standards (OASIS), "Introduction to UDDI: Important Features and Functional Concepts," Technical White Paper, October 2004
- [107] B. Orriüens, J. Yang and M. P. Papazoglou, "A Framework for Business Rule Driven Web Service Composition, ER 2003 Workshops, LNCS 2814, Springer-Verlag Berlin Heidelberg, 52-64, 2003, pages: 52-64
- [108] B. Orriüens, J. Yang, and M. P. Papazoglou, "Model Driven Service Composition," In Proceedings of the International Conference on Service-Oriented Computing (ICSOC) 2003, Trento, Italy, December 15-18, 2003, pages: 75-90
- [109] OWL API, <http://owlapi.sourceforge.net/>, last accessed on May 25, 2011
- [110] M. Paolucci, T. Kawamura, "Semantic Matching of Web Services Capabilities," In proceedings of the International Semantic Web Conference (ISWC) 2002, Sardinia, Italy, June 10-12, 2002, pages: 333-347

- [111] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: A Research Roadmap," *International Journal Cooperative Information System*, 17(2), 2008, pages: 223-255
- [112] T. Pedersen, S. Patwardhan, J. Michelizzi, "WordNet::Similarity - Measuring the Relatedness of Concepts ," In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, Intelligent Systems Demonstration, San Jose, CA July 25-29, 2004, pages: 1024-1025.
- [113] R. Perrey, M. Lycett, "Service-Oriented Architecture," In *Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT)*, - Orlando, FL, USA, 27-31 January 2003, pages: 116-119
- [114] M. Pistore, A. Marconi, P. Bertoli and P. Traverso, "Automated Composition of Web Services by Planning at the Knowledge Level," In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) 2005*, Pasadena, California, USA, pages: 1252-1259
- [115] M. Pistore, P. Traverso, P. Bertoli, A. Marconi, "Automated Synthesis of Composite BPEL4WS Web Services," In *Proceedings of the International Conference on Web Services (ICWS) 2005*, Orlando Florida, USA, July 11-15, 2005, pages: 293-301
- [116] Princeton University, "About WordNet," 2010, Available at: <http://wordnet.princeton.edu>, last accessed on May 25, 2011
- [117] Protégé, <http://protege.stanford.edu/>, last time accessed on May 25, 2011.
- [118] Y. Qi, S. Qi, P. Zhu, L. Shen, "Context-Aware Semantic Web Service Discovery," In *Proceedings of the 3rd International Conference on Semantics, Knowledge and Grid*, Xi'an, China, Oct. 29-31, 2007, pages: 499-502
- [119] D. Raggett, A. Hors, L. Jacobs (editors), "HTML 4.01 specification," W3C Recommendation, December 24, 1999
- [120] P. Rajasekaran, J. Miller, K. Verma, A. Sheth, "Enhancing Web Services Description and Discovery to Facilitate Composition," In *Proceedings of the International Workshop on Semantic Web Services and Web Process Composition (SWSWPC) 2004*, San Diego, California, USA, July 6-9, 2004, pages: 55-68
- [121] S. Ran, "A Model for Web Service Discovery with QoS," *ACM SIGecom Exchanges*, V.4 n.1, Spring, 2003, page1-10

- [122] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, San Diego, CA, USA, July 2004, pages: 43-54
- [123] Sap News Desk, "Microsoft, IBM, SAP To Discontinue UDDI Web Services Registry Effort," SOA World Magazine, available at <http://soa.sys-con.com/node/164624>, last accessed on May 25, 2011
- [124] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker, "Modeling and Composing service-base and Reference Process-based Multi-enterprise Processes," In Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, June 2000, pages: 247-263
- [125] Seekda, <http://webservices.seekda.com/>, last accessed on May 25, 2011
- [126] M. Sheshagiri, M. desJardins, T. Finin, "A planner for composing services described in DAML-S," AAMAS Workshop on Web Services and Agent-Based Engineering, Melbourne, Australia, July 14, 2003.
- [127] K. Sivashanmugam, J. A. Miller, A. P. Sheth, and K. Verma, "Framework for Semantic Web Process Composition," International Journal of Electronic Commerce (IJEC), Special Issue on Semantic Web Services and Their Role in Enterprise Application Integration and E-Commerce, Winter 04/05 issue, 9 (2), pages: 71-106.
- [128] K. Sivashanmugam, K. Verma, A. P. Sheth, "Discovery of Web Services in a Federated Registry Environment," In Proceedings of the International Conference on Web Services 2004, San Diego, California, USA, , July 6-9, 2004, pages: 270-278
- [129] M. K. Smith, C. Welty, D. L. McGuinness (editors) , "OWL Web Ontology Language Guide," W3C Recommendation (2004), available at <http://www.w3.org/TR/owl-guide/>, last accessed on May 25, 2011
- [130] D. Sprott and L. Wilkes, "Understanding Service Oriented Architecture," The Architecture Journal, January 2004
- [131] T. Strang, and C. Linnhoff-Popien, "A context modeling survey," The First International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham, England, September, 2004
- [132] Swoogle, <http://swoogle.umbc.edu/>, last accessed on May 25, 2011

- [133] A. M. Turing, "Computing Machinery and Intelligence," *Mind* LIX(236) , 1950, pages: 443-460
- [134] UbiCompForAll - Ubiquitous Service Composition for All End-Users, <http://www.sintef.no/Projectweb/UbiCompForAll/Home/>, last accessed on August 23, 2011
- [135] UDDI data Model, available at: http://www.tutorialspoint.com/uddi/uddi_data_model.htm, last accessed on May 25, 2011
- [136] A. S Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, U. Yalcinalp (editors), "Web Service Policy 1.5 - Framework," W3C Recommendation, 04 September, 2007
- [137] M. Vukovic and P. Robinson, "Adaptive, Planning-based, Web Service Composition for Context Awareness," In Proceedings of the International Conference on Pervasive Computing, Vienna, April 18-23, 2004
- [138] Y. Wang and E. Stroulia, "Semantic Structure Matching for Assessing Web Service Similarity," In Proceedings of the First International Conference on Service Oriented Computing (ICSOC03), Springer, Berlin, December 15-18, 2003, pages: 194-207
- [139] Web Services Business Process Execution Language Version 2.0, available at: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>, last accessed on May 25, 2011.
- [140] Web Service Modeling Ontology (WSMO), W3C Member Submission June 3, 2005, <http://www.w3.org/Submission/WSMO/>, last accessed on June 19
- [141] WikiHow, <http://www.wikihow.com/>, last accessed on May 25, 2011
- [142] Wikipedia, <http://en.wikipedia.org/wiki/Wikipedia:About>, last accessed on May 25, 2011
- [143] Woogle, <http://db.cs.washington.edu/webService/>, last accessed on March 20, 2011
- [144] D. Wu, B. Parsia, E. Sirin, J. Hendler and D. Nau, "Automating DAML-S Web Services Composition Using SHOP2," In Proceedings of the 2nd International Semantic Web Conference (ISWC) 2003, Sardinia, Italy, 9-12 June, 2003, pages: 195-210
- [145] Z. Wu and M. Palmer, "Verb Semantics and Lexical Selection," In Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, 1994, pages: 133-138.
- [146] H. Xiao, Y. Zou, J. Ng, L. Nigul, "An Approach for Context-aware Service Discovery and Recommendation," In Proceedings of the 8th International Conference on Web Services (ICWS) 2010, Miami, Florida, USA, July 5-10, 2010, pages: 163-170

- [147] H. Xiao, R. Tang, Y. Zou, J. Ng and L. Nigul, "Context-aware Service Composition", Technology showcase, CASCON 09, November, Toronto, 2009
- [148] H. Xiao, Y. Zou, J. Ng and L. Nigul, "Intelligent Service Selection and Composition", Technology showcase, CASCON 08, October, Toronto, 2008
- [149] H. Xiao, Y. Zou, J. Ng, L. Nigul, "Personalized Service Discovery and Composition" , In Proceedings of the Smart Internet Technologies Working Conference (SITCON), Markham, Ontario, Canada, November 2, 2009
- [150] H. Xiao, Y. Zou, R. Tang, J. Ng, L. Nigul, "An Automatic Approach for Ontology-Driven Service Composition," In Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications 2009, Taipei, Taiwan, December, 2009, pages: 17-24
- [151] H. Xiao, Y. Zou, R. Tang, J. Ng, and L. Nigul, "Ontology-Driven Service Composition for End-Users", Service Oriented Computing and Applications, Springer-Verlag, Issn 1863-2386, Vol. 5, Num. 3, 2011, pages: 159-181
- [152] H. Xiao, Y. Zou, R. Tang, J. Ng, L. Nigul, "A framework for Automatically Supporting End-Users in Service Composition," In Book "The Smart Internet", Lecture Notes in Computer Science (LNCS), Springer-Verlag, Vol. 6400, 2010, pages: 115-136
- [153] H. Xiao, "A Survey on Service Composition," Ph.D. Depth paper, School of Computing, Queen's University, ON, Canada, December, 2009
- [154] Yahoo, <http://www.yahoo.com>, last accessed on March 20, 2011
- [155] Yahoo! Pipes, <http://pipes.yahoo.com/pipes/>, last accessed on May 25, 2011
- [156] W. Yan, S. Hu, V. Muthusamy, H. Jacobsen, L. Zha, "Efficient Event-based Resource Discovery," In Proceedings of the ACM Distributed Event-based Systems Conference (DEBS) 2009, Nashville, TN, USA, July 6-9, 2009
- [157] S. J. H. Yang, J. Zhang, I. Y. L. Chen, "A JESS-enabled context elicitation system for providing context-aware Web services," Export Systems with Applications, Volume 34, Issue 4 (May 2008), pages: 2254-2266
- [158] M. Yoshida, , K. Torisawa and J. Tsujii, "Extracting Ontologies from World Wide Web via HTML tables," In Proceedings of the Pacific Association for Computational Linguistics (PACLING), Kitakyushu, Japan, 2011, pages: 332-341

- [159] Y. Zou, J. Guo, K. C. Foo, M. Hung, “Recovering Business process from Business Applications,” *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 21(5), Sept. 2009, pages: 315-348
- [160] Y. Zou, H. Xiao, and B. Chan, “Weaving Business Requirements into Model Transformations,” In *Proceedings of the 11th International Workshop on Aspect-Oriented Modeling (AOM)*, co-located with the *IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Nashville, TN, USA, September 30, 2007