



HAL
open science

Service recommendation for individual and process use

Ngoc Chan Nguyen

► **To cite this version:**

Ngoc Chan Nguyen. Service recommendation for individual and process use. Other [cs.OH]. Institut National des Télécommunications, 2012. English. NNT : 2012TELE0044 . tel-00789726

HAL Id: tel-00789726

<https://theses.hal.science/tel-00789726>

Submitted on 18 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



**DOCTORAT EN CO-ACCREDITATION
TELECOM SUDPARIS ET L'UNIVERSITE EVRY VAL D'ESSONNE**

Sp cialit : Informatique

Ecole doctorale: Sciences et Ing nierie

Pr sent e par

Nguyen Ngoc Chan

**Pour obtenir le grade de
DOCTEUR DE TELECOM SUDPARIS**

**SERVICE RECOMMENDATION
FOR INDIVIDUAL AND PROCESS USE**

Soutenue le 13/12/2012

devant le jury compos  de :

Directeur de th se :

M. Samir Tata Professeur, TELECOM SudParis, France

Rapporteurs :

M. Marlon Dumas Professeur, University of Tartu, Estonia

M. Schahram Dustdar Professeur, Vienna University of Technology, Austria

Examineurs :

M. Bruno Defude Professeur, TELECOM SudParis, France

M. Fran ois Charoy Professeur, Universit  de Lorraine, France

M. Sami Bhiri Ma tre de conf rences, National University of Ireland, Ireland

M. Walid Gaaloul Ma tre de conf rences, TELECOM SudParis, France (Encadrant)

to my parents

Acknowledgements



First and foremost, I would like to express my appreciation and gratitude to my advisor, Dr. Walid Gaaloul, who offered abundantly helpful assistance, support and guidance. His vision, creativeness and enthusiasm inspired me greatly to work. This dissertation would not have been possible without his help.

I am sincerely and heartily grateful to my supervisor, Professor Samir Tata, who gave me a lot of valuable advice. His expertise and experiences influenced and helped me be more mature in doing research.

I would like to thank to Dr. Sami Bhiri for his helpful comments and English correction. I would like to thank to Professor Bruno Defude, Brigitte Houassine and other staffs of the Information Department for their kind help and assistance.

I acknowledge the TELECOM SudParis institute for offering me a scholarship and providing me good environment and facilities to complete my thesis.

I am obliged to my colleagues, Mohamed Sellami, Mohamed Amin Sakka, Rami Selami, Olfa Bouchaala, Mourad Amziani, who not only made many useful discussions in research but also brought out many interesting stories and unforgettable memories.

I express my deepest gratitude to my loving wife, Vy. Her love and encouragement fostered me to concentrate at work. Her understanding and supports helped me to get through many difficult times.

Finally, I am forever indebted to my beloved parents for their understanding, endless patience and encouragement. They have given me constant supports, needed inspiration and always wish the best things to me. I dedicate this thesis to them.

Abstract

Web services have been developed as an attractive paradigm for publishing, discovering and consuming services. They are loosely-coupled applications that can be run alone or be composed to create new value-added services. They can be consumed as individual services which provide a unique interface to receive inputs and return outputs; or they can be consumed as components to be integrated into business processes. We call the first consumption case individual use and the second case business process use.

The requirement of specific tools to assist consumers in the two service consumption cases involves many researches in both academics and industry. On the one hand, many service portals and service crawlers have been developed as specific tools to assist users to search and invoke Web services for individual use. However, current approaches take mainly into account explicit knowledge presented by service descriptions. They make recommendations without considering data that reflect user interest and may require additional information from users. On the other hand, some business process mechanisms to search for similar business process models or to use reference models have been developed. These mechanisms are used to assist process analysts to facilitate business process design. However, they are labor-intensive, error-prone, time-consuming, and may make business analyst confused.

In our work, we aim at facilitating the service consumption for individual use and business process use using recommendation techniques. We target to recommend users services that are close to their interest and to recommend business analysts services that are relevant to an ongoing designed business process. To recommend services for individual use, we take into account the user's usage data which reflect the user's interest. We apply well-known collaborative filtering techniques which are developed for making recommendations. We propose five algorithms and develop a web-based application that allows users to use services. To recommend services for business process use, we take into account the relations between services in business processes. We target to recommend relevant services to selected positions in a business process. We define the neighborhood context of a service. We make recommendations based on the neighborhood context matching. Besides, we develop a query language to allow business analysts to formally express constraints to filter services. We also propose an approach to extract the service's neighborhood context from business process logs. Finally, we develop three applications to validate our approach. We perform experiments on the data collected by our applications and on two large public datasets. Experimental results show that our approach is feasible, accurate and has good performance in real use-cases.

Keywords: Service recommendation - business process design - neighborhood context - query language - process mining

Table of contents

1	General Introduction	9
1.1	Context	9
1.2	Thesis objectives	10
1.3	Thesis outline	11
2	Context and Research Problem	13
2.1	Thesis context and research problem	13
2.1.1	Web service discovery	13
2.1.2	Web services in business processes	15
2.1.3	Thesis research problem: How to recommend services for individual and process use?	16
2.2	Motivating example	18
2.3	Thesis principles, approach and contributions	22
2.3.1	Principles	22
2.3.2	Approach	23
2.3.3	Contributions	24
3	State of the Art	27
3.1	Introduction	27
3.2	On facilitating Web service discovery	28
3.2.1	Text-based Web service discovery	28
3.2.2	QoS-based Web service discovery	30
3.2.3	Semantic-based Web service discovery	32
3.2.4	Usage-based Web service discovery	34
3.2.5	Synthesis	35
3.3	On facilitating business process design	36
3.3.1	Business process modeling	36
3.3.2	Business process similarity	38
3.3.3	Business process querying	41
3.3.4	Business process mining	43
3.3.5	Synthesis	44
3.4	Conclusion	45
4	Service Recommendation Based on Past Usage Data	47
4.1	Introduction	47
4.2	Collaborative filtering techniques	48
4.2.1	Memory-based CF	48
4.2.2	Model-based CF	49
4.2.3	Hybrid CF	50
4.3	Illustrating example	50

4.4	Service recommendation based on past usage data	51
4.4.1	Service-based algorithm	51
4.4.2	User-based algorithm	53
4.4.3	Service-user combination algorithm	55
4.4.4	LSI-based algorithm	57
4.4.5	Power assignment algorithm	60
4.5	Conclusion	61
5	Service Recommendation based on Neighborhood Context Matching	65
5.1	Introduction	66
5.2	Preliminaries	67
5.2.1	Business process graph	67
5.2.2	Neighborhood context	69
5.2.3	Loop cases	72
5.3	Querying services	73
5.3.1	Query's grammar	73
5.3.2	Query's execution	75
5.3.3	Advantages of the query	76
5.4	Neighborhood context matching	77
5.4.1	Connection flow matching	77
5.4.2	Context matching	78
5.4.3	Zone weight consideration	78
5.4.4	Computational complexity	80
5.5	Recommendation	81
5.6	Taking into account parallel flow relations	83
5.6.1	Neighborhood context	83
5.6.2	Updating neighborhood context matching	85
5.7	Similarity between connection elements	87
5.7.1	Primitive rules	87
5.7.2	Similarity computation	88
5.7.3	Integration into the neighborhood context matching	90
5.8	Conclusion	90
6	Capturing Neighborhood Contexts from Business Process Logs	93
6.1	Introduction	93
6.2	Running example	94
6.3	Exploiting neighborhood context from logs	95
6.3.1	Preliminaries	96
6.3.2	Log-based business process	96
6.3.3	Log-based neighborhood context	97
6.4	Log-based neighborhood context matching	98

6.4.1	First zone matching	99
6.4.2	Further zone matching	101
6.4.3	Matching with zone weight consideration	103
6.5	Service recommendation	103
6.6	Conclusion	104
7	Implementation and Experiments	105
7.1	Introduction	105
7.2	Implementation	107
7.2.1	Application for individual use	107
7.2.2	Applications for business process use	108
7.2.3	Synthesis	114
7.3	Experiments	114
7.3.1	Individual use Experiments	115
7.3.2	Process use Experiments	119
7.4	Conclusion	126
8	Conclusion and Future Work	129
8.1	Conclusion	129
8.2	Future work	131
8.2.1	Improving recommendation quality	131
8.2.2	Integrating into cloud computing	132
Appendix A		135
A.1	Levenshtein distance of two inverse strings	135
A.2	Service location in layers	138
A.3	Checking primitive rules	139
Appendix B		143
B.1	Service matching in different layers	143
Appendix C		147
C.1	List of publications	147
Bibliography		149

List of Tables

3.1	Synthesis on service discovery approaches that satisfy our principles . . .	35
3.2	Synthesis on process design approaches that satisfy our principles . . .	44
4.1	Original usage matrix	51
4.2	User weights computed by TF-IDF.	53
4.3	Service weights computed by TF-IDF.	55
4.4	User weights given by the user-service combination algorithm.	57
4.5	Decomposed matrices in a 3-D space.	60
5.1	Query grammar	74
5.2	Query examples	75
5.3	Probability that a_j appears in the possible cases	89
5.4	Similarities between typical connection elements	89
6.1	Example: event logs of the liability claim process	95
7.1	Experiments with the two relevant sets	116
7.2	Details of the dataset	120
7.3	Examined cases	120
A.1	Probability that an output flow is executed	139
A.2	Similarities between typical connection elements	140

List of Figures

2.1	Web service architecture	14
2.2	Consuming Web services for individual use problematic	16
2.3	Consuming Web services for business process design problematic	18
2.4	Flight & hotel reservation processes	19
2.5	An incomplete train reservation process	20
2.6	Service recommendations for the ‘unknown’ service	20
2.7	The complete train reservation process	21
2.8	Recommendations for the selected services	21
2.9	New traveling process improved from the train reservation process	22
3.1	Basic architecture of the WS search engine using VSM [8].	29
3.2	Scenario of WS recommendation by syntactical matching [9]	30
3.3	QoS aware search engine for Web services [62]	31
3.4	Combining Service Request Expansion and LSI [18]	34
3.5	The architecture of the ICoP framework [50]	41
3.6	Example of a BPMN-Q query [35]	42
4.1	Decomposition in k dimensions [145]	58
4.2	Select services based on their selection probabilities	62
5.1	Incomplete train reservation process	66
5.2	Flight reservation business process	67
5.3	Business process graph of the ‘train-reservation’ process (Figure. 5.1)	68
5.4	Neighborhood context graphs of a_x (in Figure 5.1) and a_6 (in Figure 5.2)	71
5.5	Connection flows in loop cases	72
5.6	Recommendations for the ‘unknown’ service	82
5.7	Recommendations for the selected services	82
5.8	New traveling process improved from the train reservation process	82
5.9	Label-based business process graph of the ‘flight-reservation’ process	85
5.10	Updated neighborhood context graphs of services a_x and a_6	86
6.1	The liability claim business process discovered from event logs [155]	95
6.2	Log-based business process graph	97
6.3	Example: neighborhood context graph	98
6.4	Log-based business process graph	99
6.5	Neighborhood context graph	99
7.1	Architecture of our service recommendation tool (IRec) based on usage data	108
7.2	A screen-shot from IRec	109

7.3	PRec, a service recommendation application for process use	110
7.4	Querying web services for process design using WebRec	111
7.5	Business process extracted from logs and the related recommendations	113
7.6	Synthesized results of particular users	117
7.7	RMSE distribution with $k = 100$	118
7.8	Experiment with different k -parameter values	119
7.9	Percentage of services whose matching value ≥ 0.5	121
7.10	Percentage of services with different k^{th} -zone values	122
7.11	Precision and Recall values computed by taking into account the first zone	123
7.12	Precision values in case of two recommended services	123
7.13	Recall values computed based on the number of relevant processes . .	124
7.14	Comparing the simplest case of our approach to the random case . . .	125
7.15	Average computation time with $k = 3$	125
B.1	Current matching case	143
B.2	Ignored cases	143
B.3	Example: case 1	144
B.4	Example: case 2	145
B.5	Example: case 3	145
B.6	Example: case 4	146
B.7	Example: case 5	146

General Introduction



1.1 Context

The evolution of communication networks and technologies involves the explosion of services over the Internet. Service providers always compete to rapidly provide the best services to users. This circumstance requires the development of service oriented applications. Web services appeared as an attractive paradigm for publishing and consuming services. The goal of Web service development is to assist service providers to flexibly create new services and dynamically exchanging data with their partners for collaborative business. Web services are developed as loosely-coupled applications that can be run alone to provide a simple function or composed to create new value-added services. For instance, simple Web services can be services for city codes, local temperatures, quotes, up-to-date news, and composite Web services can be flight booking processes that compose functionality offered by other services such as customer authentication, online check-in, car rental, and payment to accomplish a flight booking transaction.

To consume a service, a user sends her request and obtains a response from the using service. To develop a composite service, a (business) process analyst designs a business process and looks for appropriate services to integrate into the designed process. Basically, services can be consumed in two different ways. They can be consumed as simple services which provide an interface to receive inputs and return outputs; or they can be consumed as components to be integrated into business processes. We call the first consumption case **individual use** and the second case **(business) process use**.

To find a service for individual use, a user can use a well-known search engine such as Google, Yahoo or Baidu. However, in most cases, she prefers to use specific service search engines that can not only provide her ‘good’ services but also can assist her to discover other interesting services. Similarly, to find a service for process use, a process analyst also needs specific tools that can understand the business context in order to rapidly find the most relevant services to integrate into the ongoing designed process.

The requirement of specific tools to assist consumers in the two cases involves many researches in both academics and industry. On the one hand, many service portals (such as XMethods, BindingPoint, WebServiceX.NET, WebServiceList, StrikeIron, RemoteMethods, Woogle, and eSynaps) and service crawlers (such as Seekda and EmbracereRegistry) have been developed as specific tools to assist users to search and invoke Web services for individual use. On the other hand, some business process search mechanisms (such as label matching, structural matching, behavioral matching) and querying languages (such as BPQL, BP-QL, BP-Mon and BPMN-Q) have been developed to assist process analysts to facilitate business process design.

To assist users to consume services for individual use, current approaches take into account data from *provider side* such as Web service descriptions, QoS and semantic concepts of services. They exploit *explicit knowledge* presented by service documents or QoS. They make recommendations *without considering data that reflect user interest*, such as usage data. In addition, they can meet text-based synonym and polysemy problems. Some of them are time consuming and some others *require efforts from users* such as rating Web services.

To assist users to consume services for process use, current approaches propose mechanisms to search for similar business process models. Some of them propose to use reference models. However, the design with reference models is still *labor-intensive*, which is absolutely *error-prone* and *time-consuming*. Meanwhile, searching entire process models costs much computation time and it can make business analysts *confused*, especially when the number of activities is large. In some cases, a compromise between the computational complexity and the quality of results needs to be found.

1.2 Thesis objectives

In this thesis, we aim at **facilitating the service consumption** in two cases: *individual use* and *process use*. Our purpose is twofold: (i) **recommending to users services that are close to their interest** and (ii) **recommending to process analysts services that are relevant to selected positions in a designed process**.

To achieve the first objective, we use usage data and adapt well-known collaborative filtering (CF) techniques. We aim at *discovering the user's interest that is hidden in the usage data*. We also aim at *using CF techniques*, which have been developed for item recommendation and prediction. We *do not ask users any effort* to provide additional information such as profile, rating or comments. Whenever a user selects a service, our approach dynamically recommend services that are relevant to her interest. To do so, we firstly identify user interests based on past usage data. Then, we integrate these interests in CF algorithms to calculate similarities between users and services. Based on the computed similarities, we select appropriate services for recommendations. Since usage data reflect user interest, our recommendations are

expected to be close to this interest.

To achieve the second objective, we capture relations of selected services with others. We define the *neighborhood context* of a service. We match neighborhood contexts to infer similarity between services. We make recommendations based on the neighborhood context matching. In our approach, we aim at *taking into account existing data* to generate service recommendations. We aim at *discovering implicit knowledge hidden in business process models*. We also *do not ask users any effort* to provide additional information. Whenever a business analyst selects a service, our approach recommend her services whose neighborhood contexts are similar to the selected service's context. Apart from service relations captured from business process models, we also *discover these relations from business process event logs*. In addition, we *develop a query language* in order to allow process analysts to formally express business constraints to filter services.

1.3 Thesis outline

This thesis includes 8 chapters:

In chapter 2, we introduce in details the thesis context, research problem and our contributions. We also present a typical scenario to motivate and illustrate our approach.

In chapter 3, we present the work related to our thesis. We study different approaches on improving Web services discovery and facilitating business process design. We introduce their models and analyze their solutions. This analysis allows us to justify the need of service recommendation based on usage data and interactions between services in business processes.

Chapters 4, 5 and 6 are the core of our thesis, which elaborate our approach to recommend services for individual and business process uses.

In chapter 4, we present our solution to recommend services based on past usage data. We present four algorithms based on CF techniques and one algorithm based on a power assignment strategy. These algorithms take into account usage data to make recommendations.

In chapter 5, we present the neighborhood context of a service. We also present a query language that allows business analysts to filter relevant services based on the neighborhood context matching. Then, we elaborate the neighborhood context matching computation and show how to recommend services that are relevant to a chosen position in a business process. Finally, we examine the parallel flow relation between services and the similarity between connection elements to improve the neighborhood context matching values.

In chapter 6, we examine business process logs. We present how we build neighborhood context from the execution orders recorded in business process logs. Then, we

present an algorithm to compute the similarity between the extracted neighborhood contexts. Finally, we present our solution to recommend services.

In chapter 7, we present applications and experiments to validate our approach.

Finally, in chapter 8, we summary our work and give an outlook to the future work.

Context and Research Problem

Contents

2.1 Thesis context and research problem	13
2.1.1 Web service discovery	13
2.1.2 Web services in business processes	15
2.1.3 Thesis research problem: How to recommend services for individual and process use?	16
2.2 Motivating example	18
2.3 Thesis principles, approach and contributions	22
2.3.1 Principles	22
2.3.2 Approach	23
2.3.3 Contributions	24

2.1 Thesis context and research problem

In this section, we present the context and research problem of our thesis. First, we introduce existing mechanisms to discover Web services (section 2.1.1). Then, we present the integration of Web services in business processes and the importance of business process design (section 2.1.2). Finally, we present our research problem related to service recommendation for individual and process use (section 2.1.3).

2.1.1 Web service discovery

Web services have been developed as a standard technology to deliver services over the Internet. The Web service architecture (Figure. 2.1) consists of three actors: *service provider*, *service requester* and *service registry*. Service providers traditionally describe their developed services using Web service Description Language (WSDL) [1] and publish them on service registries, such as Universal Description Discovery and Integration (UDDI) [2] registries. Service requesters who want to consume Web services use the search function provided by service registries to find their services. Service

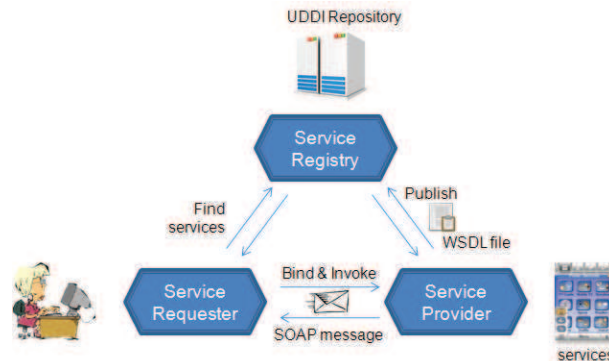


Figure 2.1: Web service architecture

registries return to requesters the locations of the requested services which are deployed at the provider site. The requesters then interact with the requested services by sending and receiving Simple Object Access Protocol (SOAP) [3] messages.

The Web service architecture creates fundamentals for the Web service development. Many companies developed and published their Web services. A survey of Gartner on 110 companies [4] showed that 54% are working on Web services. In another survey on 2847 executives worldwide in 2007 [5], 80% companies were using or going to use Web services and 78% identified that Web services are the most important technologies for their business. The growing rate of the number of Web services collected by search engines from Oct. 2006 to Oct. 2007 is 286% [6]. However, since 2006, most of large public UDDI registries such as SAP's, IBM's and Microsoft's were discontinued [7].

The growing up in the amount of Web services and the discontinuing of public UDDI registries make Web services more scattered. Many Web service portals, such as XMethods, BindingPoint, WebServiceX.NET, or Web service crawlers, such as Seekda and EmbraceService, were developed to allow service providers to continue publishing their service descriptions. They also provide interfaces for users to search and invoke Web services. Seekda¹ which has been launched since 2006 has collected 28606 Web services from 7739 providers. Meanwhile, EmbraceService² registry stores more than 8000 Web services. Apart from them, users can also discover Web services via search engines such as Google, Yahoo, AlltheWeb or Baidu.

The Web service technology has been proven as an efficient mean to delivering services to users. They are used worldwide in most of the companies. However, the Web service concept is still not familiar to end users, which still get difficulty in discovering Web services. They need advanced mechanisms to assist them to better discover services. This requirement leads to many researches on proposing solutions to

¹<http://webservices.seekda.com/browse>

²<http://www.embraceregistry.net/services>

enhance Web service discovery. Some approaches analyze Web service descriptions for better matching with the query strings [8, 9, 10], other approaches group Web services in clusters [11, 12], some approaches examine the quality of services [13, 14, 15], whereas [16, 17, 18] rely on the semantic descriptions of Web services.

2.1.2 Web services in business processes

Web services are developed as loosely-coupled applications that not only can be run alone to provide single-use functions but also can be composed with other services offered by other companies to create new value-added services. A service which is created by the composition of other services is called a *composite service*. It can be described from the view of a single company, called *service orchestration*, or from a global perspective, called *service choreography* [19, 20]. Interactions of service components in a composite service is generally specified by a process analyst/designer. There are many business process-based languages and standards to assist process analysts to specify interactions between services. In 2003, Business Process Execution Language for Web services (BPEL4WS or BPEL for short) [21] was issued by Microsoft, IBM, Siebel Systems, BEA and SAP as a standard for Web service orchestration. Intalio, Sun, BEA and SAP also released a joint specification named Web services Choreography Interface (WSCI) which is later superseded by the Web services Choreography Description Language (WS-CDL) [22] in 2005. BPEL provides an XML-based grammar for describing the control logic required to coordinate Web services participating in a process flow. WS-CDL is an XML-based language that is used to describe the interactions between multiple services. WS-CDL also specifies the ordering structures, such as sequence, parallel and choice, to define the interactions between Web services.

A business process can be implemented as a Web service composition that execute a set of services to achieve a business goal. It is managed by business process management (BPM) technology which includes methods, techniques, and tools to support the design, enactment, management, and analysis of business processes. The BPM life-cycle consists of 4 phases: *design*, *configuration*, *enactment* and *diagnosis* [23]. Among these phases, design is the initial and key phase of business process development as it helps to design the business process model, plan resources, identify new opportunities and foresee risks. In the design phase, process analysts sketch out business processes using a graphical modeling tool, such as ARIS and MID Innovator. However, designing a business process from scratch is always a labor-intensive and time-consuming task. Process analysts need tools and mechanisms to facilitate the business process design. Many approaches were proposed to address this need. They propose to use reference models [24, 25], match business processes for similarity search [26, 27, 28, 29, 30, 31, 32], create a query language [33, 34, 35, 36, 37] or discover process models from event logs [38, 39, 40, 41, 42].

2.1.3 Thesis research problem: How to recommend services for individual and process use?

As we mentioned in the thesis's introduction (section 1.1), basically, there are two service consumption cases: *individual use* and *process use*.

To find a service for individual use, users often spend much time to find, compare and decide the services that are best fitted to their needs. They may easily *get confused by the number of Web services* returned by search engines or service crawlers. Moreover, they may not be aware about the functionality and quality of the returned services (Figure. 2.2).

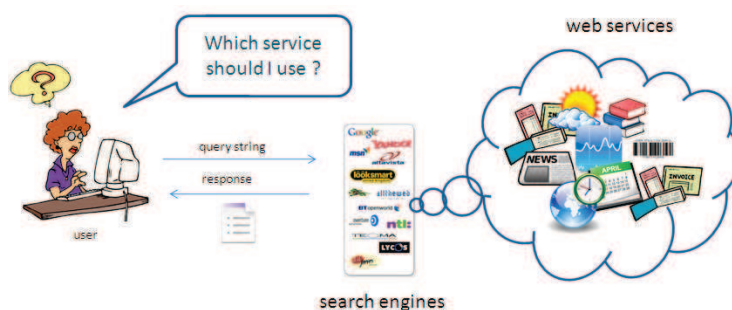


Figure 2.2: Consuming Web services for individual use problematic

Intuitively, users need support to understand their interests and suggest them appropriate services. In this case, recommender systems (RS) [43, 44] can be a good solution as they are developed to recommend users the most suitable items to their needs. Currently, many approaches apply RS techniques to assist users to discover services. Some of them [11, 8] apply content-based filtering technique to match WS descriptions and user's requests. Other approaches apply associated rules [45, 46] or collaborative filtering techniques on the service's QoS [13, 10]. These approaches, however, do not take into account user's behavior which is an important parameter for finding services that are close to user interest. User ratings are also considered for service recommendation [16]. Other solutions [47, 17, 48] use semantic annotation markup language, and rely on semantic description matching to recommend services.

However, the above approaches exploit *explicit knowledge* which is either represented by user ratings, semantic descriptions or service's QoS. They *do not* consider *user interest* which is implicitly reflected by usage data to make recommendations. Most of them take into account data from *provider side* (such as service descriptions, service's QoS, semantic annotations, etc.). Few of them consider data from customer side (profile, rating, comments, etc.).

In our work, we aim at recommending services that are close to **user interest**. We propose solution from the **customer side**. We target to exploit **implicit knowledge** hidden in usage data. We **do not ask** users **any effort** such as rating or comments.

To address this research problem, we need to answer the following questions:

1. How to identify user interest?
2. How to recommend services that are close to user interest?
3. How efficient our recommendations are?

From a process use perspective, process analysts need tools to facilitate the process design. It would be inefficient if every time a company engages in modeling or re-designing its process, it did so “from scratch” without consideration of previous design experiences, best practices or how other companies perform similar processes. In recent years, there have been many efforts on helping business analysts to create new business process models faster and more accurately by using available reference models [24, 25], or finding existing similar models to inspire the new process design [27, 28, 30, 26].

However, business analysts merely take reference models as a source of inspiration, but ultimately, they design their own models on the basis of the reference models. The design with reference models is still *labor-intensive*, which is absolutely *error-prone* and *time-consuming* [49]. Indeed, recommending *entire process models* costs much *computation time* and it can make business analysts *confused*, especially when the number of activities is large, e.g. hundreds of activities and transition flows. In some cases [27, 26, 50], a compromise between the *computational complexity* and the *quality of results* needs to be found.

On the other hand, in some circumstances, process analysts *need recommendations* for some *selected positions* instead of entire processes. For example, a process analyst is designing a process as shown in Figure 2.3. s_1 , s_2 and s_3 are services that execute some given tasks. The process analyst is looking for services that are suitable to the missing position (with the ‘?’ mark) in the ongoing designed process. In this case, recommending an entire business process is not helpful. Instead, service recommendation is more suitable and straightforward.

Service recommendation for process use not only helps to find suitable services for a missing positions but also helps to find other *alternatives* for a selected service. These alternatives can be useful in either *designing process variants* or *replacing a service* in case of failure. For example, assume that service s_1 (Figure 2.3) is vulnerable and the process analyst wants to find suitable services that can replace s_1 to keep the service available. In this case, she can use service recommendations.

In this work, our objective is to facilitate the business process design. We aim at **recommending services** that are relevant to selected positions of an ongoing design process. We target to exploit **implicit knowledge** hidden in business process models. We use **existing data** (previous process models, process logs) to make recommendations **instead of asking** process analysts **additional information**. We also want to **avoid** the **computational complexity** problem. To address this research problem, we need to answer the following questions:

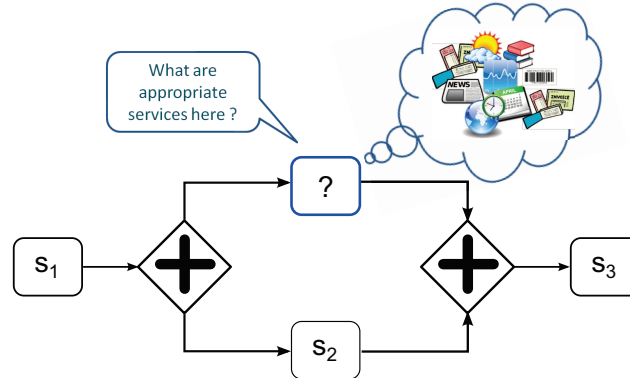


Figure 2.3: Consuming Web services for business process design problematic

1. How to recommend services for a particular position?
2. How to formally express constraints to filter services?
3. Can business process logs be useful? and How?
4. How efficient our recommendations are?

2.2 Motivating example

We present in the following a scenario to illustrate and motivate our approach. It is also used to explain our approach in the next chapters.

A traveler is planing a trip for her holidays. She intends to travel by train. So, she searches for train booking related services. She decides to use ‘Search trains’ Web service offered by our travel agency. Assume that this is the first time she uses our system. The system will record the ‘Search trains’ Web service as her first historical usage data. By analyzing usage data, our approach can infer the user interest and recommend her suitable services. For example, it detects that users who use the ‘Search trains’ service often use the ‘Search hotels’, ‘Rent cars’ and ‘Weather information’ services. So, it recommends her these services. Now, assume that the traveler is interested in the recommended ‘Search hotels’ service and she uses it to search for hotels. Our approach updates her usage data with the ‘Search hotels’ service and reanalyzes it. Assume that it detects that users who use ‘Search trains’ and ‘Search hotels’ services often use ‘Rent cars’, ‘Book tours’ and ‘Activities to do’ Web services. Then, it update the recommendation list with the new services.

By considering usage data, which reflect user interest, our approach can recommend services that are close to her interest. In addition, it recommends services without asking users any additional effort. Recommendations are generated dynamically whenever users select a service for individual use.

Back to the motivating example, our travel agency notices that their customers often consume a group of services in a time interval to achieve a specific business goal. They usually do not use just one service, rather they combine many of them. However, they do it in an ad-hoc manner. For instance, they start by searching for hotels using ‘Search hotels’ service, then collect manually the results delivered by this service, make their choice and thereafter call the ‘Process payment’ service. In order to satisfy the evolutive needs of their customers and offer a more complex and value-added service, the travel agency decides to take benefits from the multiple functionalities that business process management systems can offer by composing services as business processes. For example, it composes the ‘Search flights’ and ‘Search hotels’ services with other services such as, ‘Present alternatives’, ‘Request customer detailed Info.’, ‘Request customer basic Info.’, ‘Request credit card Info.’, ‘Process payment’ and ‘Send confirmation’, to provide a flight and hotel reservation processes (Figure 2.4).

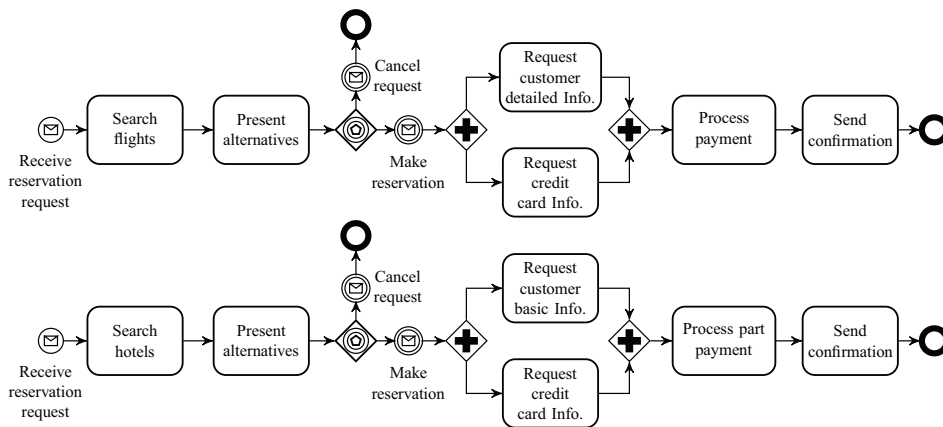


Figure 2.4: Flight & hotel reservation processes

Suppose that the traveler now wants to book a train and a hotel using a business process. Unfortunately, only hotel reservation process is supported (Figure 2.4). The train reservation process has not yet been developed. So, the user request is forwarded to our process analyst. The process analyst analyzes the user request and find that she has to develop either a train reservation process to support a separate payment or a combined trains-hotels reservation process to support a combined payment. These processes must be rapidly designed to answer these new business needs. Evolving user requirements impulse the business analyst to develop more and more value-added services by composing existing services.

The business analyst decides to design the “train-reservation” process. Based on her experience, she rapidly sketches-out the “train-reservation” process with some basic Web services as given in Figure 2.5. The ‘Search trains’ service receives a user’s request and searches for requested trains. The ‘Present alternatives’ service renders the results returned by the ‘Search trains’ service and presents it to the user. Then,

the user can either cancel her request or process the payment. The ‘Process payment’ service executes the payment and finally, the ‘Send confirmation’ service returns the user an invoice. The process analyst is looking for suitable services that can be executed before the ‘Process payment’ service. She marks it ‘unknown’ (notated by a round-corner rectangle with a ‘?’ symbol) and asks the system for recommendations.

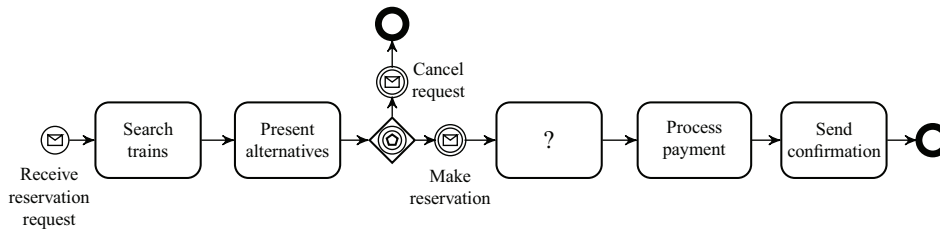


Figure 2.5: An incomplete train reservation process

Existing approaches (such as reference models, process matching, etc.) that propose entire process recommendations are not suitable in this circumstance because the process analyst look for services instead of business processes. Besides, existing service discovery approaches propose usually functionality-based search tools, which were mainly built for individual use service consumption, and do not take into account the process context (i.e. interaction with the other services in the process), which is important in the process use service consumption.

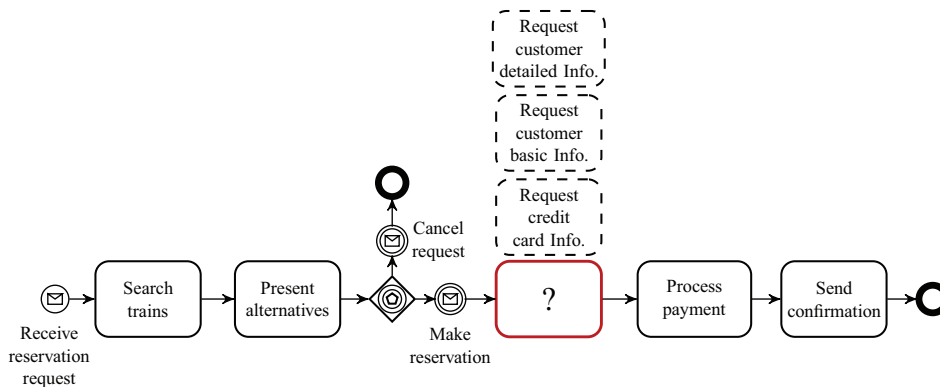


Figure 2.6: Service recommendations for the ‘unknown’ service

Indeed, The process analyst wants to know appropriate services that can be perfectly plugged in the missing position. By capturing the interactions between services in processes, our approach can recommend her services that are appropriate to the missing position. Concretely, it detects that the ‘unknown’ service has similar neighborhood context with the ‘Request customer detailed Info.’, ‘Request customer basic Info.’ and ‘Request credit card Info.’ in the previously designed Flight & hotel reservation processes (Figure 2.4) because they have similar connections with the same services, which are ‘Present alternatives’ and ‘Process payment’. So, we recommend

these services for the missing position (Figure 2.6).

The business analyst considers these recommended services. She thinks that the ‘train-reservation’ process should not bother users by requesting their personal information. In addition, to process the payment, this process needs the “Request credit card Info.” service. So, she replaces the missing service by the “Request credit card Info.” service. The complete “train-reservation” process is shown in Figure 2.7.

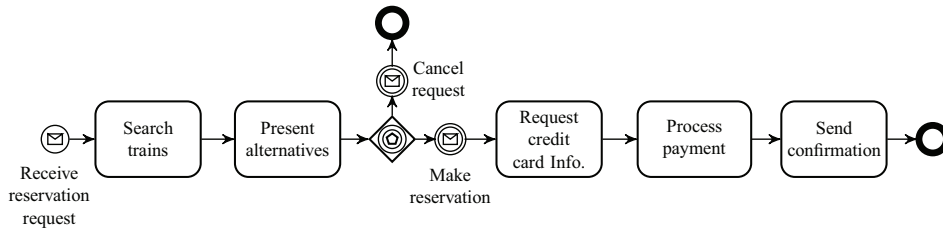


Figure 2.7: The complete train reservation process

After designing the ‘train-reservation’ process, she thinks about other process variants in order to create more value-added services. She wants to know services that are also appropriate to the positions of ‘Search trains’ and ‘Process payment’. So, she requests recommendations for these positions. She still keeps selecting the ‘Request credit card Info.’ service. By using the same principle, which is recommending services that have similar neighborhood context from previously designed business processes, recommendations for these selected positions are generated by our approach as shown in Figure 2.8.

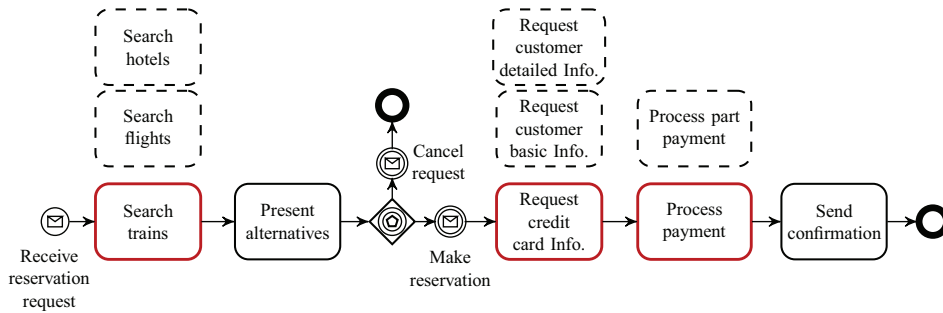


Figure 2.8: Recommendations for the selected services

With these recommendations, the business analyst may find that she can integrate the ‘Search hotels’ service to the designed process to create a combined ‘trains-hotels’ reservation process. In this case, apart from the credit card information, she needs also the user information for the hotel reservation. Finally, she creates the combined reservation service as shown in Figure 2.9.

By presenting this example, we show that *service recommendation* is essential to facilitate business process design. By taking into account the neighborhood context of services, we can make service recommendations *without asking* users any *additional*

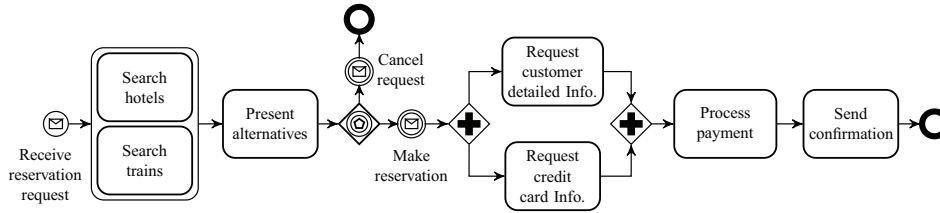


Figure 2.9: New traveling process improved from the train reservation process

information (such as service descriptions, locations, input, output etc.). Instead, we *exploit existing knowledge* which are the relations between services in business processes. We also show that service recommendation not only helps to *complete the design* of a business process but also helps to *create more value-added services*.

In summary, our approach can help to facilitate service consumption in two cases: *individual use* and *process use*. In the first case, we compare user usage data to detect the correlations between users and services in terms of usage profiles. We recommend to a user services that are similar to the currently used service, or were used by similar users. As the usage data reflect user interest, our approach can recommend services that are close to the user interest. In the second case, we assist business analysts to rapidly design new business processes. To do so, we compute the similarity between services based on relations with other services in business processes. Then, for a selected service, we recommend services that have similar relations.

2.3 Thesis principles, approach and contributions

2.3.1 Principles

In our approach, we consider the following principles:

- **Focused and fine-grained results.** In order not to confuse users, the approach should not recommend complicated results such as reference models or entire business processes, but rather focused and fine-grained results. Concretely, for a specific position in the ongoing designed process it should recommend users a ranked list of services.
- **No additional information.** The approach should not bother service consumers by asking them additional or complementary information.
- **Exploiting implicit knowledge.** The approach should extract and utilize implicit knowledge hidden in usage data such as user-service interactions, designed process models, or business process logs.
- **Balanced computational complexity.** The approach should make a compromise between the computational complexity and the quality of results.

Besides, we also consider the following general principles:

- *Simplicity.* The approach should not ask users complex interactions to achieve recommendations.
- *Flexibility.* Users should be able to adjust the number of recommended services. They should be able to select algorithms or set parameter values to get different recommendation strategies. They should also be able to specify constraints to filter recommended results.

2.3.2 Approach

To recommend services for individual use, we take into account *usage data*. We realize that usage data of each user reflect her *interest*. Therefore, by taking into account these data, we can make recommendations that are close to user interest.

collaborative filtering (CF) is one of the most successful technologies for developing recommender systems [51]. It has been developed and improved over the past decade with a wide variety of techniques [52]. CF techniques have been successful in research (with many projects such as GroupLens [53], Ringo [54], Video Recommender [55], and MovieLens [56]) and in commerce (with well-known websites such as Amazon, CDNow, MovieFinder, and Launch [57]). CF techniques are efficient and highly performant in generating recommendations [58].

As CF techniques bring out many advantages for recommendation, we apply them in the context of service use to generate service recommendations. In our approach, we firstly identify user interests based on past usage data. Then, we integrate these interests in CF algorithms to calculate similarities between users and services. Based on the computed similarities, we select appropriate services for recommendations. We implemented five algorithms, including three memory-based CF algorithms, one model-based CF algorithm and one algorithm to solve the *new item* (or cold-start) problem. Our memory-based CF algorithms take into account the correlations between services and users. The model-based CF algorithm applies the singular value decomposition and latent semantic indexing. The last algorithm deals with the usage frequency of services.

To recommend services for process use, we take into account relations between services in business processes. Inspired by the maxim “*Judge a man not by the words of his mother, but from the comments of his neighbors*”, we propose to recommend services that have similar neighborhood context with the selected one. This context is defined as a business process fragment around the service. For a selected service, we match its neighborhood context with the neighborhood contexts of other services. The matching between two neighborhood contexts is scored by a similarity value. Then, based on the similarity values, we recommend to the process analyst services that have the highest similarity values.

We have also identified parameters that have an impact on the neighborhood context matching. We integrated these parameters in our computation to improve the matching values. In addition, we proposed a query language to help process analysts

formally express neighborhood constraints to filter the recommendation results (see the research problem in section 2.1.3).

In most cases, neighborhood contexts are captured from business process models. However, in some cases, such as in a hospital information system [59], business processes are not explicitly presented. We realize that in these cases, business process event logs can be exploited to capture neighborhood contexts. So, we propose a solution to extract service neighborhood contexts from process event logs. We also take into account the importance of service execution, which is reflected by their occurrence in logs, for service recommendation.

We developed four applications, named IRec, PRect, WebRec and LogRec, to validate our approach. IRec generates service recommendations for individual use. PRect and WebRec makes recommendations for business process use. They run different algorithms of our approach. Finally, LogRec was developed as a proof of concept to validate our log-based approach.

We performed experiments on the data collected by our IRec application and on two large public datasets. We evaluated the efficiency of our approach based on precision/recall and root mean square error. We made statistics on the recommended services to estimate the recommendation quality. We also measured the performance of our algorithms based on the computation time.

2.3.3 Contributions

In summary, our contribution in this thesis are as following:

- *Four algorithms based on collaborative filtering to recommend to users services that are close to their interests.* By studying the advantages of collaborative filtering, we propose to apply this technique on service usage data to capture the correlation between users and services and generate recommendations.
- *One algorithm to improve the probability of selecting less used services.* This algorithm is developed to overcome the *new item* (or cold-start) problem: an item that has no (or limited) usage data is hard to be recommended.
- *The new concept “service neighborhood context”* which represent the business process fragment around a service. By taking into account the neighborhood context, our objective is twofold: (i) *focusing on specific parts of the business process* which can avoid the NP-complexity problem and (ii) benefiting from existing business processes by extracting *implicit knowledge* induced from business process fragments.
- *A solution to recommend to process analysts services that are relevant to selected positions in a designed process.* This solution helps to facilitate business process design, improve the designed process and create new business process variants.
- *A query language that uses the neighborhood context matching* to search and filter services that have similar neighborhood context. This query language allows

adding constrains, such as including/excluding services, to filter the returned results.

- *A solution to capture service neighborhood contexts from business process event logs and an algorithm to compute the matching of the extracted neighborhood contexts.* We extract execution orders of services from business process execution logs. Then, we create business processes based on the extracted orders. We construct service neighborhood contexts based on these processes. Finally, we match these contexts to generate recommendations.
- *Applications for service recommendation and experiments.* We deployed four applications to validate our approach. These applications are published in a public web site³. Besides, we also provide rich experiments on the collected usage data and large public datasets to demonstrate the feasibility and the efficiency of our proposed techniques.

³<http://www-inf.it-sudparis.eu/SIMBAD/tools/>

State of the Art

Contents

3.1	Introduction	27
3.2	On facilitating Web service discovery	28
3.2.1	Text-based Web service discovery	28
3.2.2	QoS-based Web service discovery	30
3.2.3	Semantic-based Web service discovery	32
3.2.4	Usage-based Web service discovery	34
3.2.5	Synthesis	35
3.3	On facilitating business process design	36
3.3.1	Business process modeling	36
3.3.2	Business process similarity	38
3.3.3	Business process querying	41
3.3.4	Business process mining	43
3.3.5	Synthesis	44
3.4	Conclusion	45

3.1 Introduction

Facilitating Web service discovery and business process design are two hot research topics. Many solutions have been proposed to assist users to retrieve their interested services and process analysts to design a process. Many problems have been identified and tackled such as service selection, service execution, service recommendation, business process modeling, business process searching and so on. Existing approaches analyzed different aspects of a Web service, such as service descriptions, execution conditions, service behaviors, etc.. Functionality properties (such as Web service operations, inputs, outputs) and non-functionality properties (such as quality of services, execution order) have been examined. Web service and process presentations, such as XML-based, semantic web, graph-based, have been also explored.

In this chapter, we study existing solutions on facilitating Web service discovery and business process design. In section 3.2, we present solutions that assist consumers

to discover Web service. We classify them in 4 categories: text-based, QoS-based, semantic-based and past-usage based. In section 3.3, we present solutions that assist process analysts to design business processes. We classify them in 4 categories: business process modeling, business process similarity, business process querying and business process mining. We present shortcomings of the related approaches, identify the difference and bring out the advantages of our approach.

3.2 On facilitating Web service discovery

Industrial and academic researchers have proposed many approaches to assist providers deliver the most relevant Web services to particular user's needs. Some approaches analyzed Web service descriptions [11, 8, 9, 12], some studied the QoS of services [14, 15, 13], while others used semantic concepts to discover services [16, 17, 10, 18]. Different methods in information retrieval, data mining and artificial intelligence domains were experimented such as collaborative filtering [8, 14, 15, 10], associated rules [46, 45, 60], clustering [11, 12, 61], divide and conquer [12] and so on. Different types of applications such as similarity search engines [11, 8, 12, 10] and Web service recommender systems [9, 46, 45, 60, 16] were also developed.

In order to clarify the difference between the existing approaches and identify the need of a Web service recommender system based on user's behavior, we classify them into 4 groups based on the used methodologies, including text-based approaches, QoS-based approaches, semantic-based approaches and usage-based approaches. In the following, we study in details the solutions in each category.

3.2.1 Text-based Web service discovery

An early work that analyzes Web service descriptions to build a similarity search engine for Web services was proposed by Dong et. al. [11]. The authors developed a Web service search engine that aims at computing the similarity between inputs, outputs and operations in a corpus. They proposed to pre-process the Web service descriptions in order to split the names of inputs, outputs and operations then cluster them in different concepts. Each input/output is formed as a vector with three elements, for example: the input i of a Web service operation op has a vector $i = (p_i; c_i; op)$, where p_i is the set of input parameter names, and c_i is the set of concepts associated with the parameter names. Whereas, each operation op is identified by a vector $op = (w; f; i; o)$, where w is the text description of the Web service to which op belongs, f is the textual description of op ; i and o denote the input and output parameters. Similarity of Web service operations is computed based on the similarities of vector elements. The similarity between inputs, outputs or Web service descriptions is computed by the similarity between the corresponding concepts using Term Frequency-Inverse Document Frequency (TF-IDF) measure.

Platzer et. al. [8] also proposed to match the user's query string with Web service

descriptions. However, they match the query vector directly with document vectors without structuring the input, output and operation vectors or clustering them in different concepts like Dong et. al. [11]. They firstly collect Web service descriptions, i.e. WSDL files, from different resources, such as user's uploading, links from websites, or references from UDDI repositories. Then, each Web service description file is parsed to generate a corresponding vector. Each element in the vector corresponds to a word in the description and its value is the number of time that the word appears in the description. The user's query string is also represented as a vector. Each term in the vectors is weighted by TF-IDF and the similarity between a query-vector and a document-vector is computed by Vector Space Model (VSM), i.e. the cosine of the angle between the two vectors. Finally, for each query string, the authors recommend the Web services whose descriptions have the highest similarity values with the query string. Figure. 3.1 shows the basic architecture of their system.

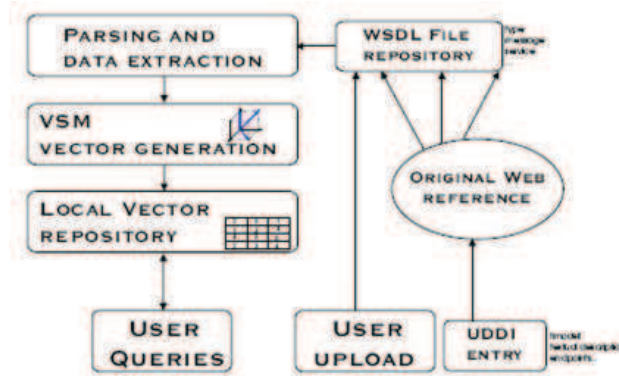


Figure 3.1: Basic architecture of the WS search engine using VSM [8].

In [9], Blake et. al. attempted to recommend Web services that are relevant to user's daily routine. Different from [11] and [8], which manipulated the user's query strings and Web service descriptions, they examined the similarity between the text data that a user was viewing/processing and the operations of a Web service. They firstly capture the text data that a user is working on such as HTML files, Word documents, File systems, messages (ICQ, SOAP, etc) (Figure. 3.2, step 1). Then, they extract text strings from the captured data (step 2). These extracted strings are compared to the operations of available Web services (step 3) to infer the similarity between them. In their approach, they captured four naming tendencies that software designers/developers used to name a Web service and proposed to apply Levenshtein Distance(LD) and Letter Pairing(LP) to compute the similarity between two strings. Finally, Web services that have the highest similarity values are recommended to the user (step 4).

The typical shortcomings of the text-based approaches are the **polysemy** and

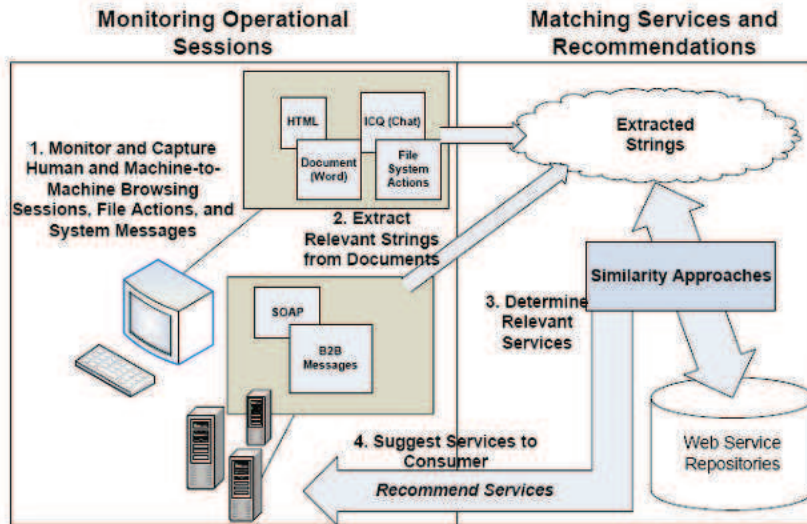


Figure 3.2: Scenario of WS recommendation by syntactical matching [9]

synonym problems, i.e. one word can have many meanings (polysemy) and one meaning can be expressed by different words (synonym). In addition, these approaches suppose that the service description text is always correct and meaningful. Consequently, they do not address the need to flexibly deal with the cases that a user either types incorrect words, abbreviated words or multi-meanings words in the query. Available **supported languages** could be also considered as another shortcoming of the text-based approaches. Currently, Web service documents are mostly described in English. Hence, the query should be in English and users must type correctly the queried words in English. This issue may limit the Web service discovery.

3.2.2 QoS-based Web service discovery

Many QoS driven approaches have been proposed for Web service search [62], Web service recommender system [14], Web service reliability prediction [63, 15], Web service selection [13, 64, 65, 66], optimal service composition [67, 68, 65, 69] and so on.

In [62], Zhang et. al. proposed a search engine that took into account both functionality and QoS for service ranking. The functionality is described by the Web service description, such as operation, input and output. The QoS is non-functionality features, such as price and response time. A user's query includes both functionality and QoS requests (Figure. 3.3). In the QoS query, 'Constraint' is a vector of queried values corresponding to QoS attributes. The 'Weight' vector allows user to specify the weight of each QoS attribute in her query. The authors computed both the

functionality matching and the QoS matching. A $\lambda \in [0,1]$ parameter is used to balance the bias of these matching.

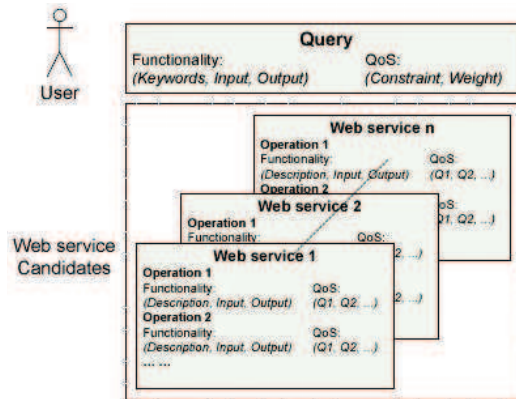


Figure 3.3: QoS aware search engine for Web services [62]

Different from [62], the authors in [14] took into account only QoS values of Web services to build a Web service recommender system. They aim at predicting the probability that a user a uses a Web service i in order to recommend her the Web services that have the highest probabilities. To do so, they built a user-item matrix, where the rows present the user IDs, the columns present the Web service IDs and each entry of this matrix is a vector of QoS values observed by the corresponding user on the corresponding Web service. Based on this matrix, they compute the similarity between Web services and users using Pearson Correlation Coefficient. The probability that a user a will use a Web service i is formulated based on the computed similarities. Two QoS metrics that are taken into account for their experiments are the round-trip-time and the failure-rate of the requests to each Web service.

The authors in [15] extended the prediction function proposed in [14] to a hybrid function that consists of item-based and user-based prediction functions. A parameter μ is added to balance the weight of item-based and user-based prediction functions. The authors in [63] also used the user-Web service QoS matrix and a prediction function. However, instead of predicting the usage probability, they predicted the failure of a composite Web services. This failure is aggregated from the failure probability of Web service components.

Other approaches [70, 13, 71, 69] also dealt with the QoS of composite Web services. However, instead of predicting the failure probability [63], their objective is to optimize Web service compositions by selecting the best Web services that satisfy given QoS constraints. The authors in [70, 13] proposed to apply two models: *the combinatorial model* which defines the problem as a multi-dimension multi-choice knapsack problem (MMKP) and *the graph model* that defines the problem as a multi-

constrained optimal path (MCOP) problem. In both models, a user-defined utility function of some system parameters may be specified to optimize application-specific objectives. Meanwhile, [71, 69] presented service execution paths in Directed Acyclic Graph (DAG) and formulated the service selection as an optimization problem which can be solved by linear programming methods.

Cardellini et. al. [65] proposed a service selection scheme which also optimizes the end-to-end aggregated QoS by means of a simple linear programming problem. However, instead of independently handling service requests [70, 71], they dealt with flows of requests, where each flow is a sequence of homogeneous requests originating by the same user/organization over time, all requiring the same QoS level.

The authors in [66] proposed an extensible, preference-oriented, open and fair QoS model for Web service selection. Their model allows providers to flexibly add new specific domain criteria for evaluating the QoS of Web service without changing the underlying computation model. It provides means for users to accurately express their preferences without resorting to complex coding of user profiles. It manipulates the QoS information collected from provider's site, such as service privacy, and user's site, such as feedbacks. Meanwhile, Haddad et. al. [64] focused on the transactional QoS criteria. They aimed at choosing Web services to create composite services that satisfy some transactional properties. For example, the composite services have to support atomic transactions, compensable transactions or they have to terminate after a finite number of execution steps.

Generally, QoS approaches facilitate the Web service discovery based on QoS attributes (such as throughput, bandwidth, execution time, failure rate, privacy or feedback) which are *explicit knowledge*. They can be classified in two categories based on service's consumption purposes: **individual use**, such as [62, 14, 15, 63], and **process use**, such as [13, 64, 65, 66, 67, 68, 69]. Our approach is also developed to facilitate these consumption cases. However, we do not take into account QoS information. Instead we exploit *implicit knowledge* hidden in user usage data and service composition structure. These parameters are important for improving Web service discovery as they expose the user's preferences and service's composition context. As taking into account different parameters for making recommendations, our approach can complement QoS approaches and vice-versa.

3.2.3 Semantic-based Web service discovery

A Web service recommender system based on semantic matching and rating prediction was proposed by Manikrao et. al. [16]. They proposed to describe Web services as ontologies using DARPA Agent Markup Language¹ (latterly Web Ontology Language [72]). Then, they matched all semantic attributes of Web services and considered that two Web services are similar if their individual semantic attributes matching is greater than given thresholds. The proposed system can predict the rating of a user

¹<http://www.daml.org>

to a Web service based on her previous ratings on other similar Web services. In their proposition, two services are more similar if their average ratings are less different.

The authors in [17] also proposed to use DARPA Agent Markup Language to semantically describe Web services. They aimed at utilizing a semantic language to present the Web service capabilities, which cannot be described by the conventional WSDL documents. They proposed to match the input/output concepts of a service request to the input/output concepts of service advertisements respectively in order to find relevant services to the request. The matching degrees are: exact, plug in, subsume and fail. The output concepts match has higher priority than the input concepts match. In other words, services whose advertisements are better matched on output concepts are preferred for recommendation rather than on input concepts.

Instead of describing Web services as ontologies like [16, 17], Ma et. al. [12] explored the semantic concepts hidden in the Web service descriptions. They proposed a two-phase approach that applies the Divide and Conquer methodology and Singular Value Decomposition technique. In the first phase, the collection of Web service descriptions is divided into a set of smaller clusters by using the Divide and Conquer approach (Bisecting k-means algorithm). In this phase, VSM and TF-IDF are applied to syntactically match the query and Web service descriptions in clusters to identify the most relevant cluster to the query. In the second phase, the SVD technique is applied on the selected cluster to decompose the document vectors and present them in a reduced space. Similarities between the query vector and the decomposed document vectors are inferred by the cosine of the angle between these vectors. Based on these similarities, the relevant services to the query are identified.

The authors in [10] also applied SVD and LSI and provided experiments on precision and recall metrics. However, instead of filtering Web services in two phase like Ma et. al. [12], they applied VSM and LSI directly on the corpus of Web service documents. Concretely, they presented Web service documents in a corpus as a matrix of documents and terms. Then, they decomposed the document-term matrix to an approximate matrix using SVD. Finally, they matched the query terms and the document rows in the decomposed matrix in order to find the closest documents to a given query.

Paliwal et. al. [18] proposed to link the Web service request and Web services descriptions to high-level ontology concepts before decomposing and matching them (Figure. 3.4). To do so, they firstly pre-process service request and determine the overall search category of Web services (step 1). Then, they retrieve relevant indexed service descriptions from the UDDI based on the high-level ontology concept of the Web service request (step 2). Next, they retrieve associated low-level ontology concepts related to the initial service request from an ontology framework based on the terms in the request (step 3). Then they expand the request with the retrieved concepts and transform the Web service descriptions into term-document matrices (step 4). These matrices are decomposed using SVD (step 5) and matched to the request vector to infer the similarity between them (step 6).

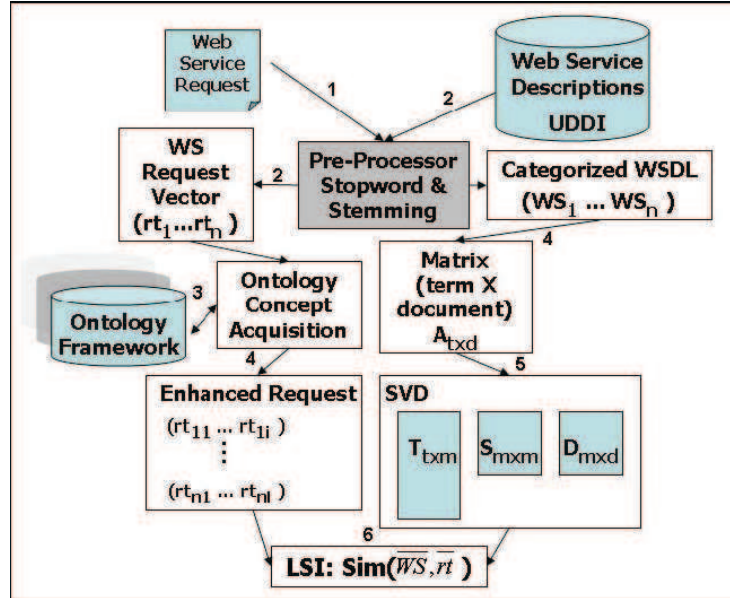


Figure 3.4: Combining Service Request Expansion and LSI [18]

Semantic-based approaches that described Web services as ontologies [16, 17] can present better functional and non-functional properties of Web services. However, creating and publishing ontology annotated content is **time-consuming** and **error prone** task as it needs to be done by domain experts using questionable editing tools. Meanwhile, rating-based solutions such as [16] may meet the *new item problem*, which means new items that have not yet rated are not selected for recommendation while items that have higher ratings are always recommended. Semantic-based approaches that exploit the latent semantics hidden in Web service descriptions [12, 10, 18] can generate recommendations without creating ontologies or asking any effort from users. In our work, we examine the latent semantics hidden in user usage data.

3.2.4 Usage-based Web service discovery

Although the past usage data are important resource exposing users' interests, there are still few approaches that take these data into account for Web service discovery. Birukou et. al. [45, 60, 46] proposed to utilize the past usage data for a Web service recommender system. They defined an 'Implicit Culture' concept as a relation between a set and a group of agents such that the set's elements behave according to the group's culture. Based on this concept, they consider that if a user has a similar request with other members of the community, he will be suggested operations that were used by those members. Therefore, they record the usage of users together

with their requests. And if they receive a new request, they will recommend the requester Web service operations that are used by other users who have similar requests. The similarity between service requests is computed based on VSM, TF-IDF and WordNet-based semantic similarity.

However, Birukou et. al. [45, 60, 46] did not process the past usage data. They used them in their proposed rule-based theory for generating recommendations. In addition, the similarity between requests is computed using text-based approaches. Hence, they ignore the correlation between users and Web services in the past usage data, which can infer the users' interests. Moreover, they can meet the shortcomings of text-based approaches while matching users' requests. In our approach, we manipulate the past usage data to exploit the hidden users' interests and we did not encounter the text-based approaches' shortcomings.

3.2.5 Synthesis

Existing approaches tried to examine different properties of Web services in order to find the best solutions for Web service discovery. They applied different techniques in information retrieval and artificial intelligence domains with different mathematical models. Those techniques were applied in different types of data, such as Web service requests and descriptions, quality of services, semantic descriptions and historical usage data. Most of existing approaches provided experiments on their solutions. Some of them developed applications such as search engines and recommender systems to realize their approaches.

Table 3.1 shows a synthesis on the presented approaches in term of our principles (section 2.3.1). '+' indicates that the corresponding principle is considered by the corresponding approach, '-' indicates that the corresponding principle is not considered and '+/-' indicates that the corresponding principle is optional.

Approaches	Principles			
	Focused result	No additional Information	Implicit knowledge	Balanced computation
[11], [17], [12], [10]	+	+	-	+
[9], [16], [18]	+	-	-	+
[8], [62]	+	+/-	-	+
[14], [15], [63], [45], [60], [46]	+	-	+	+
[70], [13], [71], [69], [65], [64], [66]	+	-	+	-

Table 3.1: Synthesis on service discovery approaches that satisfy our principles

In general, existing approaches attempted to find solutions from the **provider side**. They exploited the **explicit knowledge** that is presented in Web service

queries and Web service descriptions [11, 8] or in the QoS of Web services [62, 14, 63, 15]. Some of them applied associated rules [45, 60, 46] or semantic latent indexing technique [12, 10, 18] to exploit the **implicit knowledge** that is hidden from the Web service description and the service usage of users. Some of them **required efforts from users** to provide additional information such as user's daily routine files [9], preference [66] or rating [16]. Briefly, text-based approaches can meet the synonym and polysemy problems, QoS-based approaches miss user's interest properties, semantic-based approaches are time consuming in preparing semantic data and existing usage-based approaches do not exploit the hidden user's interest.

Our approach aims at improving the Web service discovery from the **consumer side**. We take the past usage data and the business process structure into account. We **do not encounter** the **synonym** and **polysemy** problems. We **do not face** the **misspelling words** or **supported languages** problem. We **do not ask** users any further **efforts** to provide additional information. We exploit the user's interest, which is **implicit knowledge** hidden in the past usage data. In addition, we propose an approach that can **overcome** the **new item** problem.

3.3 On facilitating business process design

Service providers integrate Web services in abstract business processes to develop composite services. An abstract business process consists of tasks and logic operators to specify the execution orders of the tasks, such as in sequence or in parallel. Each task can be executed by invoking a Web service. Business process design is an important step that helps to analyze a business process in reality, plan resources, identify risks and foresee opportunities.

In this section, we present different approaches that aim at facilitating the business process design. These approaches are classified into four categories: business process modeling (section 3.3.1), business process similarity (section 3.3.2), business process querying (section 3.3.3), and business process mining (section 3.3.4).

3.3.1 Business process modeling

Business process modeling refers to tools and techniques that assist to design, simulate and monitor business process models. The tools include graphical interface applications, such as Visio, Workflow designer, Process marker, Tibco, Holosofx, Questetra, Bizagi, Bonita and so on. The techniques are standards and languages used to describe business processes, such as Unified Modeling Language (UML), Event-driven process chain (EPC), XML Process Definition Language (XPDL), Extended Business Modeling Language (xBML), Business Process Model and Notation (BPMN), and so on. In spite of the great support of tools and techniques, the business process modeling is still time-consuming and labor-intensive, especially when such models are required to be detailed to support the development of software systems [49].

It would be inefficient if every time a company engages in modeling and re-designing its process, it did so “from scratch” without consideration of how other companies perform similar processes. Indeed, some business processes recur in similar forms from one company to another [73]. Thereafter, to avoid the effort of creating process models from scratch, several consortia and vendors have defined so-called reference process models. These models capture proven practices and recurrent business operations in a given domain. They are designed in a generic manner and are intended to be individualized to fit the requirements of specific organizations or IT projects. Commercial process modeling tools which come with standardized libraries of reference process models such as the Supply Chain Operations Reference (SCOR) models [24] or the SAP reference models [25], aim at enabling systematic reuse of proven practices across process (re-)design projects. They do so by capturing knowledge about common activities, information artifacts and flows encountered in specific application domains.

However, analysts take the reference models merely as a source of inspiration, but ultimately, they design their own model on the basis of the reference model, with little guidance as to which model elements need to be removed, added or modified to address a given requirement. Briefly, the support by reference models still has shortcomings as (1) the reference models are human based and provided manually (this work is absolutely error-prone and time-consuming) and (2) they are always studied as a whole while sometimes only some parts of the model need to be considered.

The Workflow Patterns initiative² has been developed since 1999 to delineate the fundamental requirements that arise during business process modeling. It examines various perspectives (control flow, data, resource, and exception handling) that need to be supported by a workflow language or a business process modeling language [74, 75]. It presents patterns that are used to describe the control-flow perspective of workflow systems, such as branching and synchronize patterns, multiple instance patterns, state-based patterns, etc.. It also provides detailed evaluations of various process languages for Web service compositions, and workflow systems in terms of control-flow patterns. The results of this initiative can be used for examining the suitability of a particular process language or workflow system for a particular project, assessing relative strengths and weaknesses of various approaches to process specification, implementing certain business requirements in a particular process-aware information system, and as a basis for language and tool development.

In [76], Gschwind et. al. proposed a tool to support users to correctly apply simple workflow patterns (exclusive choice, parallel split, simple merge, synchronization and sequence) [74] during the business process modeling. They proposed three scenarios in which patterns can be applied. In the first scenario, a user selects a *single edge* in a model. This edge is suggested to be replaced by a pattern compound such as sequence, parallel patterns (parallel split, synchronization) or alternative patterns (exclusive choice, simple merge), or cyclic compound. In the second scenario, a user selects

²<http://workflowpatterns.com>

a *pair of edges*. She is recommended to apply a parallel pattern or an alternative pattern to connect these edges. In the third scenario, a user selects a *set of edges*. The system considers the direction of the selected edges and recommends to apply parallel patterns or alternative patterns according to different situations. Some of other pattern-based approaches are [77, 78, 79]. In our approach, we recommend process analysts relevant Web services (or activities) instead of workflow patterns. We organize the relations between Web services in layers and zones and we apply the approach on a general case instead of three typical scenarios.

A business process auto-completion assistant approach was proposed by Lincoln et. al. [80]. For a new business process, they suggest candidates for the first activity based on the process's description. When the designer selects an activity to start designing the process, they suggest the next possible activities. The suggestion is continued until the process is completely designed. To do so, they proposed to describe business process using Process Descriptor Catalog notations. Each activity is described by four elements: object, object qualifier, action and action qualifier. For example, the activity "Arrive at appropriate terminal with luggages" is described by for elements: 'arrive', 'with luggages', 'terminal', 'appropriate'. They presented possible actions and objects on hierarchy trees. They computed the closeness between objects or actions based on the distance between them in these trees. Closeness between activities is inferred from the closeness of respective objects and actions. They recommend the first activity for a new business process models based on the business process descriptor. For the next activity, they recommend the activities that are close to the last selected activity or its siblings. The business designer selects an activity and creates links between it and previous activities. In our approach, we do not specify the relations between objects or actions of an activity by hierarchy trees. We do not suppose having more information apart from service interactions. We compute the closeness between Web services based on their neighborhood context instead of the distance between objects or actions. Moreover, our approach can adapt to all business process descriptions.

3.3.2 Business process similarity

Evaluating the similarity between business process models can help to detect the duplication or overlap between a new model and the existing process models. It also helps to identify the redundancy in the repository of process models. It is an essential tool for business process analysts.

Many approaches proposed different solutions to measure the similarity between business process models such as [26, 27, 28, 29, 30, 31, 32]. In [26], Dijkman et. al. proposed to rank all business process models in a repository according their similarity with respect to a given process model. They presented a business process model as a directed attributed graph and adopted the graph-edit distance [81] to compute the similarity between models. The similarity is computed based on the number of

deletions/insertions/substitutions of nodes and edges in order to transfer from one model to the other.

Yan et. al. [27] also aim at estimating the similarity between two business process models for similarity search purpose. They focused on the characteristics of elements in a business process model instead of the whole business process structure [26]. They defined label and structural features of a business process model. Label feature is the text string used to describe an activity, i.e. activity's name. Structural features are connection elements of a business process, including start, end, sequence, split and join. They computed the similarity between two label features using the text edit distance [82]. The similarity between two structural features is inferred by the average number of input and output paths of the corresponding connection elements. And the similarity between two business process models is synthesized from the similarity of label feature and structural features.

An approach that measures the difference between two processes based on the high-level change operations was proposed by Li et. al. [30]. They computed the difference based on the number of operations that need to be performed to transform one process to the other. However, different from [26], which measures the difference based on the number of deletions/insertions/substitutions of nodes, they took into account the execution orders between activities and measured the difference based on the deletions/insertions/movements of activities. They targeted to keep the execution orders when transforming one process model to the other to guarantee the soundness of the business process. They drove the business process matching problem to the optimization problem and aim at finding the minimum number of operations that need to be performed to achieve the transformation.

In [28], the authors proposed to compare two business process models based on the behaviors obtained from the process executions. They presented business process models using Petri-Net. The executions of these models are recorded as sequences of activities, called log traces, and their frequencies. An activity is considered in the enable state if it is going to be executed in the next execution step. The authors defined the notion 'fitness' and computed the 'fitness' between two traces based on the enable state of activities in the traces. To compute the similarity between two business process models, they consider one model as an "original model" and the other as some "new model". The similarity between the "original model" and the "new model" is evaluated based on precision and recall metrics. These metrics are computed based on the fitness between the log traces of the two processes and their frequencies.

Different from [26, 27, 30, 28], Dongen et. al. [31] proposed to measure the similarity between process models using vector space model (VSM). They modeled business processes using Event-driven Process Chains (EPCs) language [83]. They also defined the 'causal footprint' which is a graph presenting the execution orders of activities. Each causal footprint is consistent to an EPC, which is a presentation of a process model. Similarity between two business process models is inferred from the similar-

ity between two corresponding causal footprints. The author presented each causal footprint as a vector of activities that belongs to both business process models. Each element in the causal footprint vector has a value corresponding to its execution orders. The similarity between two causal footprints is computed using VSM, i.e. the cosine value of the angle created by the two corresponding vectors.

The authors in [84] merged the work in [26] and [31] to propose a new approach for business process models similarity computation. They proposed to evaluate the similarity in three metrics: label matching, structural matching and behavioral matching. The label matching is synthesized from the matching of the labels attached to process model elements. The matching between two element's labels is computed based on the string-edit distance [82]. The structural matching is inferred from element labels matching as well as the topology matching of process models. The topology matching is computed by the graph-edit distance [26]. For the behavioral matching, they defined causal footprint vectors [31] which are consistent to the business process models on causal relations and compute the similarity between footprint vectors using vector space model.

Weidlich et. al [50] focused on activities. They proposed a framework, named ICoP, to identify the correspondences between activities in two business processes in order to compare them. This framework is tailored to deal with complex 1:n matches, i.e. each activity can be matched to an arbitrary number of other activities. It includes 4 main components: searcher, booster, selector and evaluator (Figure. 3.5). *Searchers* identify potential 1:1 and 1:n matches between two process models based on different similarity metrics and heuristics along with a score that indicates the quality of the match. The result of the search stage is a multiset of matches. The scored potential matches are conveyed to *boosters*. The boosters transform the *multiset* of potential matches received from the searchers into a *set* of potential matches by aggregating them. The *selector* selects the best matches from the set of potential matches, satisfying that all the matches are not overlapping. Then it passes these matches to the *evaluator* to evaluate them with a matching score. The evaluator may use knowledge about the original process models to compute this score. Then, the evaluator returns to the selector the computed matching score. The selector re-select the best matches based on the score received from the evaluator and re-send them to the evaluator. This process is continued until the matching score does not increase. Then, the selector returns the final mapping between elements of the process models.

Ehrig et. al [32] proposed to model business process models using Petri-net. Each element in the Petri-net, i.e. place, transition, attribute and value, is described by an ontology with a set of properties. They specified the weight of each property and computed the similarity between elements based on the syntactic and semantic matching of the respective properties. For the syntactic matching, they apply string-edit distance and for the semantic matching, they use WordNet library. They also modeled a business process as a concept consisting of multiple instances. Each instance corresponds to a Petri-net element and the similarity between instances is the

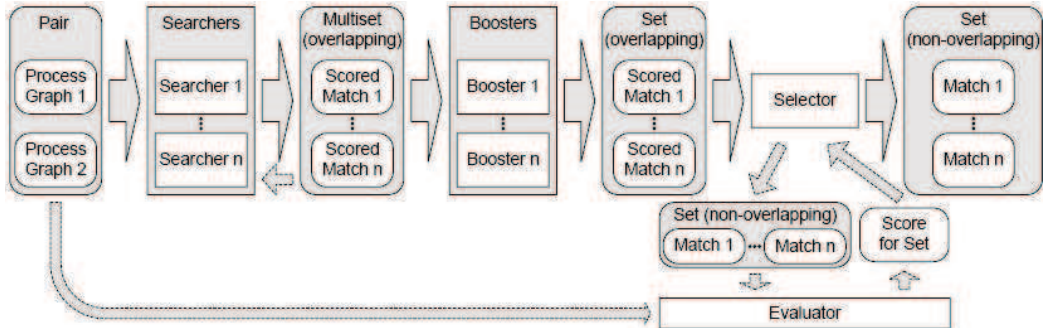


Figure 3.5: The architecture of the ICoP framework [50]

similarity between corresponding Petri-net elements. Similarity between two business process models is synthesized from the syntactic similarity, semantic similarity and instances' similarity of the two business processes' concepts.

The business process similarity can help to deploy a similarity search engine. It assists process analysts to rapidly search for business processes related to a specific requirement, then design a new business process inspired by a selected process from the search result. However, this may be efficient with *small-size business processes*, i.e. with few activities and operations. In contrast, the large-size business processes, e.g. consist of hundreds of activities and operations, may consume much computation time. Moreover, they may make process analysts confused and hard to detect how the business processes are similar and which parts should be inherited from the recommended processes to use for the current design. In addition, the matching of the whole business processes often leads to the graph-matching problem, which is *NP-complete* [85], and they, e.g. [27, 26, 50], have to deal with the trade-off among the complexity, accuracy (efficiency) and system performance. In our approach, we focus *partially* on the business process and take into account only the activity neighborhood context for recommendations. Consequently, we retrieve related activities without facing the complexity problem.

3.3.3 Business process querying

One of solutions to assist process analysts to speed up the design is providing him a tool that can help to query appropriate activities or business processes [33, 35, 36, 37, 86, 87, 88, 89, 90].

Awad et. al. [33, 34, 35, 36, 37] developed a visual business process query language, named BPMN-Q. This query language extends the BPMN notations by adding some new elements such as variable node, generic gateway and path [36]. These elements are used to define a query graph. They matched the query graph to existing business process models to retrieve partial business processes that satisfy the query pattern.

For example, a query with the path element is shown in Figure 3.6. It is for finding all the execution paths (and the involving activities) between activities *B* and *D*. The variable node element is used for searching activities that are executed before or after a given activity. The generic gateway element is used for searching activities that are connected by a gateway.

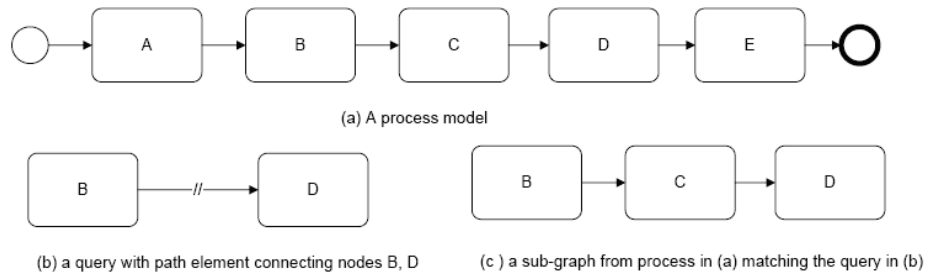


Figure 3.6: Example of a BPMN-Q query [35]

BPMN-Q finds business process patterns that are compliant to a query and limited by two activities. It processes the perfect match between the given activities and the activities in existing process models then retrieves exactly the patterns connecting the matched activities. In our approach, we also develop a query language for querying Web services. However, we match the neighborhood contexts instead of the Web services. Our search context is defined by layers and zones around a service instead of the connections between two services. We retrieve Web services that have similar neighborhood contexts instead of the same context. And we build a query language in Extended Backus–Naur Form (EBNF) instead of visual notations.

In [86], Hornung et. al. proposed an approach that allows process analysts to query business processes or to get recommendations from the system. To do so, they transformed business process models from Petri-net to text based documents by extracting the labels of the component elements. Then, to query business processes, they query the corresponding documents and return the original process models. The query is executed by Java Lucene, an open source search engine that was developed by Apache for the documents querying. The selected part of the editing business process is considered as a document and it is matched to the documents in the repository based on the term frequency - inverse document frequency (TF-IDF) and vector space model (VSM). In our approach, we develop a query language to query services. However, we do not transform business processes to text-based documents. We deal with the neighborhood contexts of Web services. And we retrieve relevant services (with the business processes that contain them) instead of relevant business processes.

Lincoln et. al. [87] continued their previous work [80] (section 3.3.1) to propose a search framework for retrieving business process segments. They defined the object grouping model (OGM) which includes the relationship between a primary object and

others in a process segment. They weighted the edges that connecting objects in an OGM by their repetitions. Similarity between segments is computed based on the term frequency-inverse document frequency (TF-IDF). In our approach, we organize Web services in a neighborhood context in layers based on their shortest path lengths to a primary service instead of putting them in the same layer. We take into account the sequence of connection flow elements instead of the repetition of edges. And we match connection flows in zones to infer the similarity instead of using TF-IDF.

Some approaches proposed query languages to support business process monitoring during the execution, such as BPQL [88, 90], BP-QL [89] and BP-Mon [91]. Beheshti et. al. [92] inspired from SPARQL language to propose FPSPARQL which is a query language for analyzing business process logs. These approaches assist process analysts to analyze business process instances during the execution. For example, the analyst can monitor an auction process to guarantee the fair-play. She can detect users that register as sellers but repeatedly cancel bids or auctions and warn them. She can also identify illegal accesses, such as users who attempt to submit bids without first registering to the system, and block these process instances. In our approach, we also propose a query language. However, we focus on the design phase, not execution phase. Our query helps to facilitate the business process design rather than monitor business process instances. We take into account design-based business process properties, such as the shortest path between two Web services or the similar between connection elements, instead of execution-based properties, such as user's role or login time.

3.3.4 Business process mining

Today's information systems, such as workflow management systems (e.g. Staffware), ERP systems (e.g. SAP), case handling systems (e.g. FLOWer), PDM systems (e.g. Windchill), CRM systems (e.g. Microsoft Dynamics CRM), middle ware (e.g., IBM's WebSphere), hospital information systems (e.g., Chipsoft), etc., record their business transactions as event logs [59]. These logs back up not only the business execution but also the knowledge related to the a-priori business process models.

The goal of process mining is to extract information from these logs in order to exploit the hidden knowledge that may be helpful for the business analysis. For instance, it can discover the process models [38, 39, 40, 41, 42] from an enormous set of log traces. It can mine the business constraints to check the conformance of a-priori models [28, 93, 94, 95, 96]. It can also detect execution errors [97, 98], observe social behaviors between groups of users [99, 100], and so on.

Business process mining is a discipline that sits between machine learning and data mining on the one hand and process modeling and analysis on the other hand [42]. It was firstly introduced in 1998 [101] and was proven as a powerful technique to discover behaviors observed from event logs. It has involved the development of many tools and techniques that support mining event logs [102, 103, 104, 105, 106, 107, 108].

In our approach, we also exploit the business execution logs to extract the *relations between activities*. However, our target is not to discover the a-priori business process or to check the process conformance. Instead, we discover the neighborhood contexts of activities based on their execution orders. We prove that the business event logs also contain knowledge that is essential for recommending activities during the business process design.

3.3.5 Synthesis

Many approaches have been proposed to assist process analysts to rapidly design business processes. Most of them dealt with the **whole business process model**. They measured the *similarity between business process models* by comparing their structures [26, 27], behaviors [28, 84], the execution orders of activities [30, 31], etc. to help to detect the duplication or the redundancy of process models. They built different *query languages* [33, 34, 35, 36, 37, 86] to help to retrieve business process models that are similar to a query process model. They mined the business process *event logs* in order to discover the business process models [38, 39, 40, 41, 42] to help to check the conformance of a-priori models [28, 93, 94, 95, 96].

Approaches	Principles			
	Focused result	No additional Information	Implicit knowledge	Balanced computation
[24], [25], [30], [31], [33], [34], [35], [36], [37], [86]	-	+	-	+
[76]	+	+	-	-
[80], [87]	+	+	-	+
[26], [27], [84], [50]	-	+	-	-
[28], [38], [39], [40], [41], [42], [28], [93], [94], [95], [96], [97], [98], [99], [100]	-	+	+	+
[32]	-	-	-	+

Table 3.2: Synthesis on process design approaches that satisfy our principles

Some of approaches dealt with **parts of business process model**. They attempted to recommend process analysts either appropriate *workflow patterns* for a business process structure [76, 77, 78, 79] or appropriate *activities* for a business process [80, 50] during the design. They also proposed a framework to retrieve *process segments* that are similar to a given segment [87]. Some of them dealt with all possible matching between business processes [26, 27] or activities [50]. They **faced** the **NP-complexity problem** of the graph matching and they needed to find a trade-off between the computational complexity and the quality of recommendations or imple-

ment other strategies. One of them **require additional information** (WordNet library) for semantic matching computation [32].

Table 3.2 shows a synthesis on the presented approaches in terms of our principles (section 2.3.1).

In our approach, we focus on *part of business process model*. We aim at **recommending services** that are relevant to a selected position in a business process. We take into account the **neighborhood context** of a service instead of the service's name or features. Therefore, we can retrieve relevant services for not only **a selected service** but also **an empty position** in a business process (recommendations in our motivating example in section 2.2). We make recommendation based on the existing data. We **do not ask for additional information**. As our approach can recommend services at different positions in a business process, it can be applied for **business process auto-completion** during the design. We also exploit business process event logs to extract the **hidden knowledge**, which is the execution orders between services, for recommendations. In addition, as we do not deal with the whole business process topology or possible correspondences between services in two processes, our approach **do not face the complexity** problem.

3.4 Conclusion

In this chapter we present different approaches that assist users in two service consumption cases: *individual use* and *process use*. For the first case, we classify the existing approaches into four categories: *text-based*, *QoS-based*, *semantic-based* and *usage-based*. For the second case, we also classify them into four categories: *business process modeling*, *business process similarity*, *business process querying* and *business process mining*. We briefly introduce these approaches and identify their principles. We show that existing approaches still miss some features that can help to facilitate the service consumption. We also also present the difference between current approaches and our approach.

We start presenting in detail our approach in the next chapters. In chapter 4, we elaborate how we recommend services that are close to user interest based on past usage data. In chapter 5, we present our approach to facilitate the business process design based on service recommendation. In chapter 6, we show how we can extract service relations from event logs to make recommendations.

Service Recommendation Based on Past Usage Data

Contents

4.1	Introduction	47
4.2	Collaborative filtering techniques	48
4.2.1	Memory-based CF	48
4.2.2	Model-based CF	49
4.2.3	Hybrid CF	50
4.3	Illustrating example	50
4.4	Service recommendation based on past usage data	51
4.4.1	Service-based algorithm	51
4.4.2	User-based algorithm	53
4.4.3	Service-user combination algorithm	55
4.4.4	LSI-based algorithm	57
4.4.5	Power assignment algorithm	60
4.5	Conclusion	61

4.1 Introduction

This chapter presents our contribution to improve service consumption for *individual use*. It presents algorithms and strategies to process *past usage data* for *service recommendation*. *Only user past usage data* is used as input to our approach. We do not ask users any further data such as their profiles, comments or ratings. As usage data present user interest on certain categories of services, our approach can recommend services that are close to user interest. We apply collaborative filtering techniques on past usage data to generate recommendations. We also propose an algorithm to address the ‘new item problem’ that can happen in collaborative filtering techniques.

In this chapter, we firstly give some basic backgrounds on collaborative filtering techniques (section 4.2). Then, we present an illustrating example with a usage matrix

(section 4.3). We use this example to illustrate our algorithms (section 4.4). Finally, we conclude the chapter in section 4.5.

The work of this chapter is published in [109, 110, 111, 112].

4.2 Collaborative filtering techniques

In this section, we present an overview on collaborative filtering techniques. We introduce memory-based CF (section 4.2.1), model-based CF (section 4.2.2) and hybrid CF (section 4.2.3). In our approach, we apply principles of these CF techniques to make service recommendations.

The term “collaborative filtering” (CF) was firstly coined by the developers of the Tapestry recommender system [113]. The fundamental assumption of CF is that if users X and Y rate n items similarly, or have similar behaviors (e.g., buying, watching, listening), then they will rate or act on other items similarly [114]. For example, GroupLens [115, 53] uses user rating data to compute the similarity between either users or movies then make recommendations according to the similarity values.

CF includes a set of techniques (mathematical, statistical, etc.) applied on user rating data to find the correlative relations between users or items in order to make predictions or recommendations. CF techniques can be classified into three basic categories: *memory-based CF*, *model-based CF* and *hybrid CF* [58].

4.2.1 Memory-based CF

Memory-based CF algorithms compute the similarity between users or items based on a user-item matrix. This matrix presents the usage data of users in a system. Each row of the matrix presents the items that a user used and each column presents a set of users who used a corresponding item. The value of each element in the matrix can be the rating of a user to an item or the number of times that a user used (or viewed, purchased, listened, etc.) an item. Some systems, such as Amazon [116, 117] or GroupLens, apply memory-based CF algorithms as they are easy to implement and retrieve high correlated similar items.

The similarity or the correlation between two users, is computed based on their co-purchased or co-rated items. In contrast, the similarity or the correlation between two items, is computed based on the common users who purchased or rated the considered items [118]. The CF techniques that compute the similarity between users is called *user-based CF*. Meanwhile, the CF techniques that compute the similarity between items is called *item-based CF*. The *Vector Space Model* [119] measure is one of the most used and popular measures to compute the similarity between users or items in memory-based CF.

A user who is interacting with the system is called an *active user*. The top- N recommendation [117] method is one of the most used and popular methods to make predictions or recommendations for an active user using memory-based CF.

The *top-N recommendation* method [117] aims at recommending a user the top-N items that are most relevant to her interest. There are two types of algorithms to implement this method: *user-based top-N* and *item-based top-N* [58]. User-based top-N recommendation algorithms firstly identify the k most similar users to the active user. Let C be the set of items that are purchased by the k most similar users. The user-based algorithms recommend the top-N most frequent items in C that the active user has not purchased. Whereas, the item-based top-N recommendation algorithms compute the k most similar items to the item that the active user is purchasing. Then, they remove the items that the active user has purchased, sort the rest items in descending order of the similarity, and pick up n item in the top list for the recommendation.

As CF techniques have been developed as efficient tools to make predictions and recommendations, we apply these techniques in our approach. We present usage data as a matrix. Each row of this matrix presents the usage of a service, each column presents the usage of a user and the value of each element in the matrix presents the number of times that the corresponding user used the corresponding service. We propose two algorithms to make service recommendations based on the *user-based top-N* and *item-based top-N* CF methods. We choose Vector Space Model (VSM) as it is one of the most popular memory-based CF technique and especially widely used in Information Retrieval [119]. Detail of our algorithms is presented in section 4.4.1 and section 4.4.2.

4.2.2 Model-based CF

The memory-based CF techniques are easy to implemented and highly effective. However, as they rely on the commonly rated items, their performance decreases when data are sparse or common items are few. Consequently, model-based CF techniques were investigated to overcome the memory-based CF problem. They alleviate the sparsity problem by discovering hidden correlations between users or items. Some mathematical techniques are applied in model-based CF to reduce the sparsity such as Singular Value Decomposition [120] or Eigentaste [114]. They transform the original user-matrix to a new approximate matrix by removing unrepresentative or insignificant users or items. Later, mathematical or statistical models are applied to make prediction.

Popular model-based CF techniques include Bayesian belief nets (BNs) CF [121, 122], clustering CF [123, 124, 125], regression-based CF [126], Markov decision process CF [127] and latent semantic CF [128, 129, 130, 131]. Some other model-based CF techniques are association rule based CF [132], maximum entropy CF [133], decision tree CF [134], graph-based CF [135] and probabilistic CF [136, 137, 129].

In our approach, we propose an algorithm based on model-based CF technique to overcome the sparsity problem of our memory-based algorithm. We choose Latent Semantic Indexing (LSI) [128, 130] as it is one of the common used model-based

CF technique and it implements the Singular Value Decomposition (SVD) which is a mathematical model that greatly reduce the sparsity of the usage data. Our algorithm is presented in section 4.4.4.

4.2.3 Hybrid CF

Hybrid CF systems combine different CF techniques [138, 139] or CF techniques with other recommendation techniques, typically content-based systems [140, 141, 137], to make predictions or recommendations. They utilize the advantages of different techniques to overcome limitations of memory-based and model-based models. They improve the prediction performance. However, they also increase the implementation complexity and cost. In addition, they generally need external information that is usually not available [58].

In our approach, we propose an algorithm based on hybrid CF technique. Our algorithm combines the two memory-based CF methods: *user-based top-N* and *item-based top-N*. However, our proposed hybrid CF technique do not ask for additional information. Detail of this algorithm is presented in section 4.4.3.

4.3 Illustrating example

It is worthwhile to notice that a service provides one or more operations. To avoid the confusion of services and operations, we consider that a service offers only one operation throughout our approach¹.

To illustrate our approach, we use an example with four users who used a set of services as following: $U_1 = \{\text{getWeather (2), getLocation (2), getNews (1), getHotels (3)}\}$; $U_2 = \{\text{getLocation (4), getBook (1), getNews (2), getHotelID (4), getPlaces (3), getWeatherByZipCode (2)}\}$; $U_3 = \{\text{getPlaces (2), getWeatherByDate (5), getWeatherByZipCode (3), getHotelID (5), getCityDescByName (2)}\}$ and $U_4 = \{\text{getHotel (7), getCityDescByName (8)}\}$.

The set U_i , ($i = 1..4$) presents the past usage data of the user u_i . The record “getWeather(2)” in the set U_1 means that the user u_1 has used the service getWeather two times. The service-user matrix corresponding to these usage data, so named $A_{[m \times n]}$, is represented in Table.4.1, where $m = 10$ and $n = 4$. The value of an element $a_{i,j}$ in this matrix is the number of times that the user u_j has used the service s_i . This usage matrix is used to illustrate the three memory-based CF algorithms and the model-based algorithm we propose in the following.

¹In the case of multi-operation services, we consider service operation instead of service. Usage data is presented by an operation-user matrix instead of the service-user matrix currently presented in our approach. Then, recommendations would have the same level of accuracy.

[service×user]: $A_{[m\times n]}$	U_1	U_2	U_3	U_4
getWeather	2	0	0	0
getLocation	2	4	0	0
getNews	1	2	0	0
getHotels	3	0	0	7
getBook	0	1	0	0
getHotelID	0	4	5	0
getPlaces	0	3	2	0
getWeatherByZipCode	0	2	3	0
getWeatherByDate	0	0	5	0
getCityDescByName	0	0	2	8

Table 4.1: Original usage matrix

4.4 Service recommendation based on past usage data

In this section, we detail our algorithms to make recommendations based on past usage data. We firstly present a service-based algorithm (section 4.4.1), which makes recommendations based on the similarity between services. Then, we present a user-based algorithm (section 4.4.2), which makes recommendations based on the similarity between users. Next, we present a combination of the service-based and user-based algorithms (section 4.4.3). This later algorithm makes recommendations based on the similarity between services that have been used by relevant users. We also present an algorithm based on Latent Semantic Indexing (LSI) and Singular Value Decomposition (SVD) (section 4.4.4) techniques to overcome the sparsity problem of the service-based and user-based algorithms. Finally, we present an algorithm which aims at improving the probability of selecting less used (or never used) services (section 4.4.5). This algorithm is developed to overcome the *new item problem* (or cold-start problem).

4.4.1 Service-based algorithm

In this algorithm, we aim at finding services relevant to the service that a user is currently using. We apply the item-based top-N CF algorithm on the service-user matrix. We compute the similarity between the currently used service with other services using VSM. Then, we sort the services in descending order of similarity. We pick up the l -top services to recommend them to the user. The pseudo code of service recommendation based on item-based top-N CF is described in Algorithm 1.

The key step of the algorithm is finding the similarity between a service s_i and another service s_x (line 3 of Algorithm 1). To compute this similarity, we apply the vector space model (VSM).

VSM is firstly introduced by Gerard Salton et al. [142]. It is developed to compute the similarity between two individual documents. It presents documents in a k

Algorithm 1: Service-based recommendation

input : s_x :currently used service
output: a recommended list of l services

- 1 S = set of services;
- 2 **foreach** service s_i in S **do**
- 3 | Compute the similarity between s_i and s_x ;
- 4 **end**
- 5 Sort $s_i \in S$ in descending order of similarity ;
- 6 Select top- l services for recommendation;

dimensional space, where k is the number of different terms. Each document is presented as a vector with k elements. Each element of a document vector corresponds to a term appearing in the document. The value of a vector element is the weight of the corresponding term. This weight is computed by term frequency (TF) and inverse document frequency (IDF). Similarity between two documents is computed by the cosine value of the angle created by the two corresponding vectors.

In our approach, we consider analogically each row (service) in the usage matrix as a document and each column (user) as a term. The value of each element in the usage matrix is considered as the number of times that the corresponding term appears in the corresponding documents. Similarity between two services is inferred from the similarity between two row vectors. We also apply the term-frequency (TF) and inverse document frequency (IDF) on the usage matrix to compute the weight of each user (term).

The weight of a user u_j w.r.t a service s_i , denoted by $w_{i,j}$, $i = 1..m, j = 1..n$, computed by TF-IDF is given by Equation 4.1.

$$\begin{aligned}
 w_{i,j} &= tf_{i,j} \times idf_{j,S} \\
 &= \frac{a_{i,j}}{\sum_{k=1}^n a_{i,k}} \times \log \frac{m}{|s_t \in S : a_{t,j} > 0|}
 \end{aligned} \tag{4.1}$$

where $a_{i,j}$ is the number of times that the service s_i was used by the user u_j ; $\sum_{k=1}^n a_{i,k}$ is the number of times that s_i was used by users; S is set of all services; m is the number of services; and $|s_t \in S : a_{t,j} > 0|$ is the number of different services that were used by the user u_j .

By applying the TF-IDF computation (Equation 4.1) on the original usage matrix, we have a new matrix $W_{[m \times n]}^u$ which contains the weights of all users. For example, by applying the TF-IDF on our motivating example (Table 4.1), we have $W_{[m \times n]}^u$ as given in Table 4.2.

[service×user]: $W_{[m×n]}^u$	U_1	U_2	U_3	U_4
getWeather	0.92	0	0	0
getLocation	0.31	0.34	0	0
getNews	0.31	0.34	0	0
getHotels	0.27	0	0	1.13
getBook	0	0.51	0	0
getHotelID	0	0.23	0.39	0
getPlaces	0	0.31	0.28	0
getWeatherByZipCode	0	0.02	0.42	0
getWeatherByDate	0	0	0.69	0
getCityDescByName	0	0	0.14	1.29

Table 4.2: User weights computed by TF-IDF.

Each row in the weight matrix presents a service vector. Similarity between two services is computed by the cosine value of the angle created by the two corresponding vectors (Equation 4.2).

$$sim(s_a, s_b) = \frac{\vec{w}_a^u \times \vec{w}_b^u}{|\vec{w}_a^u| \times |\vec{w}_b^u|} \quad (4.2)$$

where \vec{w}_a^u, \vec{w}_b^u are the weight vectors of services s_a and s_b respectively, $\vec{w}_a^u = \{w_{a1}, w_{a2}, \dots, w_{an}\}$, $\vec{w}_b^u = \{w_{b1}, w_{b2}, \dots, w_{bn}\}$, $w_{ak}, w_{bk} \in W_{[m \times n]}^u, k = 1..n$, and n is the total number of users.

In our example, suppose that a user uses the service *getPlaces* and we are going to recommend her the top-4 services relevant to *getPlaces*. By applying the similarity computation (Equation 4.2) on the user weight matrix (Table 4.2), we get the similarity values of other services to *getPlaces* as following: {getWeather (0.00), getLocation (0.55), getNews (0.55), getHotels (0.00), getBook (0.74), getHotelID (0.95), getWeatherByZipCode (0.92), getWeatherByDate (0.67), getCityDescByName (0.07)}. Based on the similarity values, we select 4 services to recommend them to the user. The selected services are {getHotelID, getWeatherByZipCode, getBook, getWeatherByDate} as they have the greatest similarity values.

The complexity of the item-based CF algorithm is $O(mn)$ where m is the number of terms and n is the number of documents. In our approach, to shorten the response time, we process data offline and store them on temporary tables. We also update the similarities between services periodically offline.

4.4.2 User-based algorithm

Inspired by the fact that users who have similar interest will tend to select similar items, we aim in this algorithm at finding users who have similar interest, i.e. they used similar services. We select then the most frequently used services that were used

by the most relevant users and were not used by the active user to make recommendations.

Contrary to the service-based algorithm, we consider in this algorithm each user as a document and each service as a term. We apply the VSM to compute the similarity between users. We also use TF-IDF to weight vector elements. Concretely, the weight of a service s_i which was used by a user u_j is computed by Equation 4.3.

$$\begin{aligned} w_{i,j} &= tf_{i,j} \times idf_{i,U} \\ &= \frac{a_{i,j}}{m} \times \log \frac{n}{|u_t \in U : a_{i,t} > 0|} \end{aligned} \quad (4.3)$$

where $a_{i,j}$ is the number of times that the service s_i was used by the user u_j ; $\sum_{k=1}^m a_{k,j}$ is the number of times that u_j used services; U is the set of all users; n is the number of users; and $|u_t \in U : a_{i,t} > 0|$ is the number of users who used s_i .

By applying Equation 4.3 on the usage matrix, we get a weight matrix $W_{[m \times n]}^s$ that contains the weight of all services. Based on this matrix, we compute the similarity between users using VSM. Concretely, the similarity between two users u_x and u_y is given by Equation 4.4.

$$sim(u_x, u_y) = \frac{\overrightarrow{w}_x^s \times \overrightarrow{w}_y^s}{|\overrightarrow{w}_x^s| \times |\overrightarrow{w}_y^s|} \quad (4.4)$$

where $\overrightarrow{w}_x^s, \overrightarrow{w}_y^s$ are weight vectors of users u_x and u_y respectively, $\overrightarrow{w}_x^s = \{w_{1x}, w_{2x}, \dots, w_{mx}\}$, $\overrightarrow{w}_y^s = \{w_{1y}, w_{2y}, \dots, w_{my}\}$, $w_{kx}, w_{ky} \in W_{[m \times n]}^s, k = 1..m$, and m is the total number of services.

We generate recommendations in three steps algorithm (see the pseudo code presented in Algorithm 2). Firstly, we compute the similarity between the active user and others based on their usage data (line 3 of Algorithm 2). Secondly, we sort other users in descending order of similarity (line 5 of Algorithm 2) and select the top- k users in the list (line 6 of Algorithm 2). Finally, for each selected user, we select the t -most-frequently-used services that were not used by the active user to make recommendations (line 7 of Algorithm 2).

Back to our motivating example, after applying Equation 4.3 on the past usage matrix (Table 4.1), we get the service weight matrix $W_{(m \times n)}^s$ illustrated in Table.4.3.

Suppose that we are going to provide a recommended list of 4 ($l = 4$) services to u_1 based on the past usage data of the two closest users ($k = 2$). By using VSM, we compute the similarity between user u_1 and others as follows: $\{u_2(0.25), u_3(0.00), u_4(0.36)\}$. As $k = 2$, u_4 and u_2 are selected as they have the highest similarity values. Then, for each selected user, we select the two most used services for recommendation. Therefore, we have the recommended services to u_1 are: $\{\text{getLocation}, \text{getHotelID}, \text{getHotel}, \text{getCityDescByName}\}$.

Algorithm 2: User-based recommendation

input : u_x :active user
output: a recommended list of l services

- 1 U = set of users;
- 2 **foreach** user u_j in U **do**
- 3 | Compute the similarity between u_j and u_x ;
- 4 **end**
- 5 Sort $u_j \in U$ in descending order of similarity values;
- 6 Select top k users from the sorted list of $u_j \in U$;
- 7 For each of k selected users, select the t -most-frequently-used services to make a recommended list of $l = k \times t$ services;

[service×user]: $W_{[m \times n]}^s$	U_1	U_2	U_3	U_4
getWeather	0.35	0	0	0
getLocation	0.17	0.17	0	0
getNews	0.09	0.09	0	0
getHotels	0.26	0	0	0.32
getBook	0	0.09	0	0
getHotelID	0	0.17	0.20	0
getPlaces	0	0.13	0.08	0
getWeatherByZipCode	0	0.09	0.12	0
getWeatherByDate	0	0	0.41	0
getCityDescByName	0	0	0.08	0.37

Table 4.3: Service weights computed by TF-IDF.

In the last step of Algorithm 2, we select the t -most used services from the k -top similar users to provide the recommendation list. Suppose that u_x , u_y and u_z are the most similar users. Our algorithm always automatically suggest the top- t services of u_x , u_y and u_z , even if the $(t+i)^{th}$ ($i > 0$) service of u_x is much more used than t^{th} service of u_y or u_z . On the other hand, if a similar user used less than t services, the $(t+i)^{th}$ service of the other users would not be selected to fulfill the recommendation list. For example, if we select 3 services from u_4 and u_2 to be recommended to u_1 , the recommendation list will have 5 services as u_4 has just used 2 services. We call this *potential missing problem*. This problem may decrease the effectiveness of our approach. In the next section, we present an algorithm that can overcome this problem.

4.4.3 Service-user combination algorithm

In this section, we present a combination of the service-based and user-based algorithms. We also make recommendations based on the usage data of relevant users.

However, instead of selecting the mostly used services of relevant users, we compute the similarity between services used by these users. By combining these algorithms, we aim at improving the recommendation performance and avoiding the *potential missing problem* of the user-based algorithm.

Algorithm 3: User-service combination recommendation

input : current user u_x , current used service s_y
output: a recommended list of l services

- 1 U = set of users;
- 2 **foreach** user u_j in U **do**
- 3 | Compute the similarity between u_j and u_x ;
- 4 **end**
- 5 Sort $u_j \in U$ in descending order of similarity values;
- 6 Select top k users from the sorted list of $u_j \in U$;
- 7 $A'_{[m \times k]}$ = usage data of the selected users;
- 8 S = set of services;
- 9 **foreach** service s_i in S **do**
- 10 | Compute the similarity between s_i and s_y based on the new usage matrix
 | $A'_{[m \times k]}$;
- 11 **end**
- 12 Sort $s_i \in S$ in descending order of similarity values;
- 13 Select top- l services for recommendation;

Concretely, the *service-user combination algorithm* (see Algorithm 3) generates recommendations in three steps. Suppose that a user u_x currently uses a service s_y . First, we find the k -most similar users to u_x using the user-based algorithm (lines 3-6 of Algorithm 3). Second, we eliminate the unselected users' data from the original usage matrix to get a smaller matrix $A'_{[m \times k]}$, m is the number of services and k is the number of selected users (line 7 of Algorithm 3). Third, we recompute the weight of each user in the new matrix $A'_{[m \times k]}$ and use the service-based algorithm to find the l most relevant services to s_y for recommendation (lines 10-13 of Algorithm 3).

In our example, suppose that we are going to make recommendations for u_1 , who is using *getPlaces*. Also suppose that we will recommend 4 services ($l=4$) based on the two most similar users ($k=2$). We firstly select two users that are the two most similar to u_1 . In this step, we select U_4 and U_2 as they have the highest similarity values to u_1 which are 0.36 and 0.25. Second, we eliminate the unselected users' data, i.e. U_3 , from the original matrix to have a smaller matrix with three columns. The user weights computed based on the new matrix are given by Table 4.4. Third, we apply the service-based algorithm on this weight matrix to find the most relevant services ($l=4$) to *getPlaces*. Finally, we select 4 services {getBook, getHotelID, getWeatherByZipCode, getLocation} for recommendation according to

	U_1	U_2	U_4
getWeather	0.92	0	0
getLocation	0.31	0.34	0
getNews	0.31	0.34	0
getHotels	0.27	0	1.13
getBook	0	0.51	0
getHotelID	0	0.51	0
getPlaces	0	0.51	0
getWeatherByZipCode	0	0.51	0
getWeatherByDate	0	0	0
getCityDescByName	0	0	1.61

Table 4.4: User weights given by the user-service combination algorithm.

their similarity values to *getPlaces*, which are {1.00, 1.00, 1.00, 0.74} respectively.

4.4.4 LSI-based algorithm

The memory-based CF techniques, e.g. the service-based and user-based algorithms in our approach, compute the similarity based on the explicit relations between users and items, i.e. the usage matrix. They match directly user vectors or item vectors to infer their similarity. They do not take in to account the correlation between two vectors and a third-party vector. For example, in our illustrating example, all users used the same category of services, which provide traveling information. Their interests are not totally different. However, when computing the similarity between users, the user-based CF technique does not detect the similarity between u_1 and u_3 , u_2 and u_4 as $\text{sim}(u_1, u_3) = \text{sim}(u_2, u_4) = 0$ (by Equation 4.4).

To detect the similarity between users or services via a third-party item, we present in this section the application of a model-based CF technique, which is Latent Semantic Indexing (LSI).

LSI is a mathematical and statistical technique for extracting hidden correlations between documents and terms [143, 130]. It applies the Singular Value Decomposition (SVD), which is a factorization algorithm to decompose a rectangle matrix into three matrices. The original matrix is equal to the multiplication of these matrices. The mathematical fundamental of SVD and its computation are explained in [130, 144, 145, 146]. Some examples about LSI and SVD are presented in [147, 148].

Basically, a matrix $A_{[m \times n]}$ can be decomposed into three matrices $U_{[m \times n]}$, $\Sigma_{[n \times n]}$ and $V_{[n \times n]}^T$ using SVD. This decomposition is given by Equation 4.5.

$$A_{[m \times n]} = U_{[m \times n]} \Sigma_{[n \times n]} V_{[n \times n]}^T \quad (4.5)$$

where $U_{[m \times n]}$ and $V_{[n \times n]}$ are orthogonal matrices, which present the left and right singular vectors of A . $\Sigma_{[n \times n]}$ is an n-by-n diagonal matrix holding the singular values.

In $\Sigma_{[n \times n]}$, only the elements on the diagonal have values greater than or equal to 0 and they are sorted in descending order. Other elements are equal to 0. So, if we present the values of the elements on the diagonal of $\Sigma_{[n \times n]}$ as a vector $\vec{\sigma}$, we will have $\vec{\sigma} = (\sigma_1, \sigma_2, \dots, \sigma_n)$, $\sigma_i > 0$ for $1 \leq i \leq r \leq n$ and $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_n = 0$. $r > 0$ is called *rank* of A .

As $\sigma_{r+1} = \dots = \sigma_n = 0$, the rows and columns $(r+1)^{th}, \dots, n^{th}$ in $\Sigma_{[n \times n]}$ are zero vectors, i.e. vectors whose all element values are equal to 0. So, the multiplication by these vectors has value 0. Therefore, if we reduce the $\Sigma_{[n \times n]}$ to $\Sigma_{[r \times r]}$ by removing the zero vectors, and remove the corresponding columns in $U_{[m \times n]}$ and rows in $V_{[n \times n]}^T$, the multiplication of these matrices also yields to the original matrix (Equation 4.6).

$$A_{[m \times n]} = U_{[m \times r]} \Sigma_{[r \times r]} V_{[r \times n]}^T \quad (4.6)$$

On the other hand, as elements on the diagonal of $\Sigma_{[n \times n]}$ are sorted in descending order of their values ($\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$), the last $r - k$ elements have the smallest positive values. So, if we consider these $r - k$ smallest values equal to 0, and thereafter remove zero vectors and corresponding columns in $U_{[m \times r]}$ and rows in $V_{[r \times n]}^T$, the multiplication of $U_{[m \times k]}$, $\Sigma_{[k \times k]}$ and $V_{[k \times n]}^T$ will yield a matrix $A_{[m \times n]}^k$ that is approximated to $A_{[m \times n]}$ (Figure 4.1, Equation 4.7).

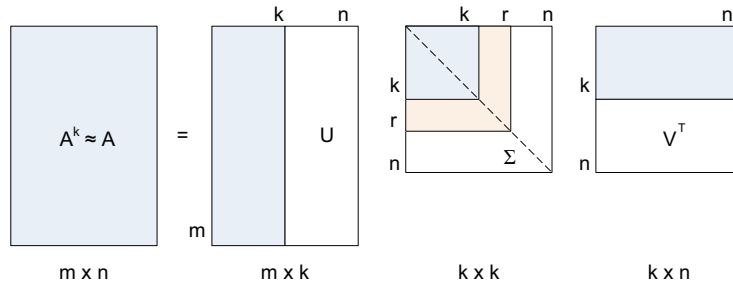


Figure 4.1: Decomposition in k dimensions [145]

$$A_{[m \times n]} \approx A_{[m \times n]}^k = U_{[m \times k]} \Sigma_{[k \times k]} V_{[k \times n]}^T \quad (4.7)$$

Assume that $A_{[m \times n]}$ is a service-user usage matrix. The derived $A_{[m \times n]}^k$ matrix does not reconstruct the original matrix $A_{[m \times n]}$ exactly. However, the truncated SVD not only captures most of the important underlying structure in the association of services and users but also removes the noise or variability in service usage. Services that are used by similar users, for example, will be near each other in the k -dimensional space even if they never be co-consumed by the same user.

A query is a set of services. It can be considered as a user. So, to retrieve the relevant services (or users) to a query, this query must be represented in the same k -dimensional space. The values of a query vector $\vec{q}_{[m]}$ in the k -dimensional space

is represented by Equation 4.8 [145].

$$\vec{q}_{[k]} = \vec{q}_{[m]}^T \times U_{[m \times k]} \times \Sigma_{[k \times k]}^{-1} \quad (4.8)$$

As the query is presented in the same k -dimensional space with services (or users), it can be compared to the services (or users) based on the similarity between two vectors.

In our approach, we apply LSI on the service usage data to make recommendations (pseudo codes is presented in Algorithm 4). Following the principles of LSI, we firstly decompose the service-user matrix $A_{[m \times n]}$ into three matrices U , Σ and V (by Equation 4.5) (line 2 of Algorithm 4). As decomposed by SVD technique, these matrices hold the values that reflect the correlations between services. Second, we reduce the service space to a k -dimensional space (Figure 4.1) (line 3 of Algorithm 4). The original matrix $A_{[m \times n]}$ is approximated to a matrix $A_{[m \times n]}^k$ (by Equation 4.7). In other words, we represent the existing services as vectors in a k -dimensional space. Third, we compute the vector of the service that a user is using in order to present it in the same k -dimensional space with other services (by Equation 4.8) (line 4 of Algorithm 4). Finally, we compute the similarity between the current used services with others using VSM, sort the services in descending order of similarity, and select top- l services for recommendation (line 6-9 of Algorithm 4).

Algorithm 4: LSI-based recommendation

input : u_x, s_y
output: a recommended list of l services

- 1 $A_{[m \times n]}$ = usage matrix;
- 2 Decompose $A_{[m \times n]}$ into 3 matrices U, Σ, V using SVD;
- 3 Compute U^k, Σ^k, V^k whose $A_{[m \times n]}^k$ is an approximated matrix of $A_{[m \times n]}$;
- 4 Present s_y as a query vector \vec{q} in the same k -dimensional space;
- 5 **foreach** service vector s_i in U_k **do**
- 6 | Compute the similarity between \vec{q} and s_i ;
- 7 **end**
- 8 Sort $s_i \in U_k$ in descending order of similarity with \vec{q} ;
- 9 Select top- l services from the sorted list for recommendation;

It is noticed that the mathematical computation of SVD is elaborated in [144] and SVD has been implemented in different languages such as C, C++ or Java². Therefore, we do not present the SVD computation. Instead, we present how and when SVD is applied in LSI technique as aforementioned.

Suppose that a user u_j is using a service s_i . To make recommendations for u_j , we consider s_i as a query vector $\vec{q}_{[m]}$, in which only the element q_i has value $a_{i,j}$, other

²<http://web.eecs.utk.edu/research/lsi/soft.html>

elements are equal to 0. m is the number of services (Equation 4.9).

$$q_t = \begin{cases} a_{i,j} & \text{if } t = i, t \in [1..m] \\ 0 & \text{if } t \neq i, t \in [1..m] \end{cases} \quad (4.9)$$

In our motivating example, services are presented in a 4-D space (corresponding to 4 users). After applying the SVD and truncating the weakest dimension, i.e. the dimension corresponding to the smallest singular value of S_k , we have the decomposed matrices (U^k , Σ^k and V^k) in a 3-D space ($k = 3$) as given in Table 4.5.

$$U^k = \begin{bmatrix} 0.04 & 0.01 & 0.18 \\ 0.10 & -0.26 & 0.66 \\ 0.05 & -0.13 & 0.33 \\ 0.62 & 0.30 & 0.25 \\ 0.02 & -0.07 & 0.12 \\ 0.22 & -0.64 & -0.03 \\ 0.11 & -0.35 & 0.15 \\ 0.13 & -0.35 & -0.07 \\ 0.16 & -0.37 & -0.51 \\ 0.71 & 0.19 & -0.23 \end{bmatrix} \quad \Sigma^k = \begin{bmatrix} 11.13 & 0.00 & 0.00 \\ 0.00 & 9.28 & 0.00 \\ 0.00 & 0.00 & 5.50 \end{bmatrix}$$

$$V^k = \begin{bmatrix} 0.19 & 0.03 & 0.50 \\ 0.18 & -0.61 & 0.66 \\ 0.35 & -0.69 & -0.56 \\ 0.90 & 0.39 & -0.02 \end{bmatrix}$$

Table 4.5: Decomposed matrices in a 3-D space.

Now, assume that user u_1 uses service *getPlaces*, the query vector of this usage is presented by $\vec{q}_{[m]} = (0,0,0,0,0,0,1,0,0,0)$. The coordinates of $\vec{q}_{[m]}$ in the 3-D space (computed by Equation 4.8) are given in (4.10).

$$\vec{q}_{[k]} = \vec{q}_{[m]}^T \times U_{[m \times k]} \times \Sigma_{[k \times k]}^{-1} = (0.01, -0.04, 0.03) \quad (4.10)$$

Next, we compute the similarity between the query with other services whose coordinates are presented by the matrix U_k . Suppose that we need to recommend 3 services for the selected service. We apply the VSM for the similarity computation. Then, we select the top-3 services that have the highest similarity values for recommendation. The selected services are {getBook, getNews, getLocation} according to their similarity values to the query, which are 0.92, 0.85 and 0.85.

4.4.5 Power assignment algorithm

A common problem of CF techniques occurs when a new item has just entered the system. A new item, which has no usage data, cannot be recommended until users enough use it. An item, which has been rarely used, can be hardly selected for recommendation. This problem is called *cold start* problem [58] or *new item problem* [149, 150].

In this section, we present an algorithm that can help to improve the probability of selecting new items. We do not apply CF techniques or compute the similarity between items. Concretely, we propose to assign each service a power value that is inversely proportional to its used time. The less used a service is, the greater its

power value is. Let M be the maximum used time of any service, p_i is the power value of a service s_i , we have:

$$p_i = (M + 1) - |s_i| \quad (4.11)$$

where $|s_i|$ is the number of used times of s_i by all users. If s_i has never been used, it will have the highest power value, which is $M + 1$. In contrast, if it is the most used service, it will have the lowest power value, which is 1.

Selection probability of s_i computed based on the power value is given by Equation 4.12.

$$Prob(s_i) = \frac{p_i}{\sum_j p_j} \quad (4.12)$$

where n is the number of services. The less used a service is, the greater its selection probability is. If s_i has never been used, it will have the highest selection probability.

Algorithm 5: Recommendation based on power assignment

input : past usage data
output: a recommended list of l services
1 **foreach** service s_i in S **do**
2 Compute the power value p_i of s_i .;
3 Compute the selection probability of s_i .;
4 **end**
5 Arrange all services on a line according to their selection probability;
6 Select l services to make recommendations;

Pseudo codes of this technique is presented in Algorithm 5. We firstly compute the power value and selection probability of each service (line 1-3 of Algorithm 5). Then, we arrange all services on a line (line 5 of Algorithm 5). On this line, each service is presented by a segment whose length is equal to its selection probability (Figure 4.2). The less used a service is, the longer its segment is. Finally, we select l services from the line for recommendation (line 6 of Algorithm 5). As less used services corresponds to longer segments, they have higher probability to be selected.

Our recommendation algorithm based on the power assignment can improve the probability of selecting less used services.

4.5 Conclusion

In this chapter, we answer the following research question raised in section 2.1.3: *How to identify user interest?*. user interest in our approach is identified by services that the user has used. Usage data of each user is presented by a vector. Each element in

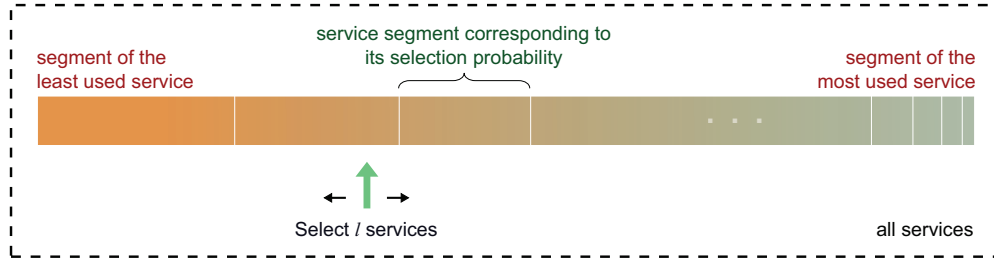


Figure 4.2: Select services based on their selection probabilities

this vector is weighted using TF-IDF metrics to calculate its importance w.r.t. other services.

We also answer another research question: *How to recommend services that are close to user interest?*. We make recommendations based on *usage data*. We recommend a user services that are used by other users who have the same interest with her. We also recommend her services that are similar to her used services. As the recommended services are generated based on user usage data, they are close to user interest.

In our approach, we apply collaborative filtering (CF) techniques. We developed five algorithms to make recommendations, three of them are based on the memory-based CF technique, one algorithm is based on the model-based CF technique and a non CF algorithm to resolve the *cold start* problem.

As mentioned in section 3.2.4, there are few approaches [45, 60, 46] that take into account past usage data for service recommendation. Previous work uses these data as references for a rule-based and text-based solution. They do not take into account usage data in their computation. In our approach, we use past usage data to exploit hidden users' interests.

The principles presented in section section 2.3.1 are respected by the techniques proposed in this chapter:

- *Focused and fine-grained results.* We recommend suitable services instead business processes.
- *No additional information.* We do not ask users any effort to provide additional information such as profiles, ratings and comments.
- *Exploiting implicit knowledge.* Our recommendations are made based on the correlation between users and services. This knowledge is implicitly presented in usage data.
- *Balanced computational complexity.* The computation time in our approach is polynomial.

In addition, we *do not* take into account the *text-based* query or service descriptions. Therefore, we do not face the problems of text-based approaches (section 3.2.1) as discussed in chapter 3.

To validate our work, we implemented an application that allows user to select services and obtain recommendations. We also performed experiments using not only the usage data collected by our application and but also a large public dataset. Details of implementation and experiments are presented in chapter 7.

Service Recommendation based on Neighborhood Context Matching

Contents

5.1	Introduction	66
5.2	Preliminaries	67
5.2.1	Business process graph	67
5.2.2	Neighborhood context	69
5.2.3	Loop cases	72
5.3	Querying services	73
5.3.1	Query's grammar	73
5.3.2	Query's execution	75
5.3.3	Advantages of the query	76
5.4	Neighborhood context matching	77
5.4.1	Connection flow matching	77
5.4.2	Context matching	78
5.4.3	Zone weight consideration	78
5.4.4	Computational complexity	80
5.5	Recommendation	81
5.6	Taking into account parallel flow relations	83
5.6.1	Neighborhood context	83
5.6.2	Updating neighborhood context matching	85
5.7	Similarity between connection elements	87
5.7.1	Primitive rules	87
5.7.2	Similarity computation	88
5.7.3	Integration into the neighborhood context matching	90
5.8	Conclusion	90

5.1 Introduction

This chapter presents how we can recommend services for *process use*. It introduces the concept of *service neighborhood context* and details our recommendation approach in order to facilitate *business process design*.

We start the chapter by presenting some preliminaries that help to formally define business processes and service neighborhood contexts (section 5.2). Basically, the context of a service within a process includes its *characteristics* and *relations* to other component services. Service characteristics include the service name, operation, input, output, etc.. Service relations refer to the dependencies that may have with other services within the same process. We define the concept of *service neighborhood context* which corresponds to a process fragment around the given service. We recommend services based on similarity of their neighborhood contexts.

Next, we present a query language that is used to find relevant services to selected positions in a business process (section 5.3). This query language is based on the neighborhood context. Then, we present our approach to compare the similarity between services based on their neighborhood context matching (section 5.4). We show how to make recommendations for a selected position using its neighborhood context matching (section 5.5).

Finally, we study the impact of parallel flow relations between services (section 5.6) and the similarity between connection elements on neighborhood context matching (section 5.7).

We reuse our motivating example to illustrate our approach (section 2.2). We assume that a process analyst is designing a ‘train-reservation’ process. She sketches out the business process as given in Figure 5.1. She needs recommendations for a selected position, which is represented by the service labeled by ‘?’, to complete the process.

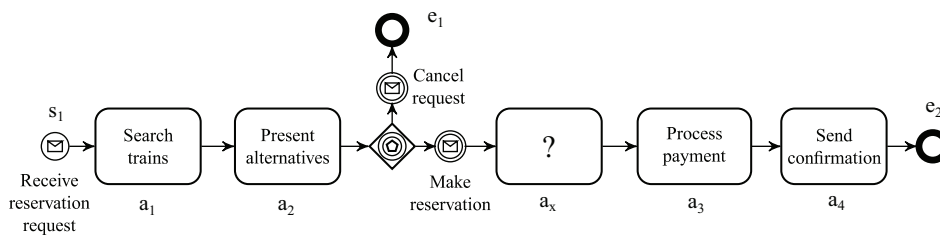


Figure 5.1: Incomplete train reservation process

We also assume that there exists a ‘flight-reservation’ process in the system (Figure 5.2, taken from our motivating example in section 2.2). We demonstrate our approach to recommend the process analyst relevant services for the selected position.

The work in this chapter was published in conference proceedings [151, 152, 153].

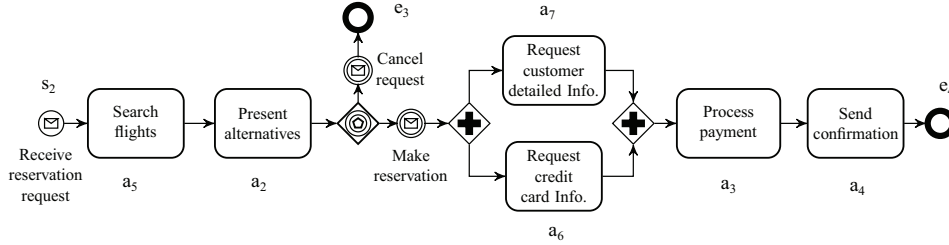


Figure 5.2: Flight reservation business process

5.2 Preliminaries

In this section, we present notations and definitions that we use to formally define a business process (section 5.2.1) and a neighborhood context of a service (section 5.2.2). We also describe the representation of neighborhood contexts in loop cases (section 5.2.3).

5.2.1 Business process graph

As the structure of a business process can be mapped to a graph, we choose graph theory to present a business process. Indeed, there are a number of graph-based business process modeling languages, e.g. BPMN, EPC, YAWL, and UML activity diagram. Despite their variances in expressiveness and modeling notations, they all share the common concepts of tasks, events, gateways, artifacts and resources, as well as relations between them, such as transition flows [37]. Without loss of generality, we select and use BPMN in our approach as it is one of the most popular business process modeling language. In our work, we focus on services and the connections to their neighbors.

A *service* is presented as a round rectangle like a task or a sub-process in BPMN. Termination activities such as start or end events are considered as *termination services*. We define a *connection element* as either a *connecting object*, e.g. sequence flow and message flow, or a *gateway*, e.g. AND-split, OR-split, etc., or an *intermediate event*, e.g. error message, message-catching, etc.. For example, in Fig. 5.1, s_1 , a_1 , a_2 , e_1 are services; and ‘flow-transition’, ‘event-based-gateway’ and ‘message-catching’ are connection elements.

Services and connection elements in our approach are defined using BPMN notations. However, they are easily mapped to equivalent notations in other business process modeling and workflow languages, e.g. tasks and workflow control patterns [23]. In the following, we present some definitions that are used in our approach to present a business process and the concept of neighborhood context.

Let A_P be the set of services and C_P be the set of connection elements in a business process P .

Definition 5.2.1 (next relation). Let $e_i, e_j \in A_P \cup C_P$. A next relation e_i to e_j , denoted by $e_i \rightarrow_P e_j$, indicates that e_j is situated right after e_i in P .

Definition 5.2.2 (connection flow). A connection flow from a_i to a_j , $a_i, a_j \in A_P$, denoted by ${}^{a_j}f_P$, is a sequence of connection elements $c_1, c_2, \dots, c_n \in C_P$ that satisfies: $a_i \rightarrow_P c_1, c_1 \rightarrow_P c_2, \dots, c_{n-1} \rightarrow_P c_n, c_n \rightarrow_P a_j$. ${}^{a_j}f_P \in C_P^*$, C_P^* is the set of sequences of connection elements in P .

For example, in Figure. 5.2, the connection flow from s_2 to a_5 or from a_5 to a_2 is ‘flow-transition’; the connection flow from a_2 to e_3 is ‘event-based-gateway’‘message-catching’; the connection flow from a_2 to a_7 is ‘event-based-gateway’‘message-catching’ ‘parallel-split’ and so on.

Definition 5.2.3 (Business process graph). Let A_P be a set of services and C_P^* be the set of sequences of connection elements in a business process P . The business process graph of P is a labeled directed graph $G_P = (V_P, E_P, L_P)$, where:

- $V_P = A_P$.
- $E_P \subseteq A_P \times A_P$.
- $L_P = \{((a_i, a_j), {}^{a_j}f_P) : (a_i, a_j) \in E_P, {}^{a_j}f_P \in C_P^*\}$.

Elements of V_P are *vertexes*, elements of E_P are *edges* and elements of L_P are ordered pairs presenting the mapping from edges to labels. In a business process graph, each vertex is a service; each edge is an ordered pair of services; and each edge is labeled by the connection flow between the two associated services. *For simplicity, we label a connection element by its type.*

For example, the ‘train-reservation’ process in Figure. 5.1 can be presented by a business process graph $G_{P_1} = (V_{P_1}, L_{P_1}, E_{P_1})$ (Figure 5.3), where: $V_{P_1} = \{a_1, a_2, a_x, a_3, a_4, s_1, e_1, e_2\}$, a_1 =“Search trains”, a_2 =“Present alternatives”, a_x =“?”, a_3 =“Process payment”, a_4 =“Send confirmation”, s_1 =“start message”, e_1 =“end event”, e_2 =“end event”; $E_{P_1} = \{(s_1, a_1), (a_1, a_2), (a_2, e_1), (a_2, a_x), (a_x, a_3), (a_3, a_4), (a_4, e_2)\}$; $L_{P_1} = \{((s_1, a_1), \text{‘flow-transition’}), ((a_1, a_2), \text{‘flow-transition’}), ((a_2, e_1), \text{‘event-based-gateway’ ‘message-catching’}), ((a_2, a_x), \text{‘event-based-gateway’ ‘message-catching’}), ((a_x, a_3), \text{‘flow-transition’}), ((a_3, a_4), \text{‘flow-transition’}), ((a_4, e_2), \text{‘flow-transition’})\}$.

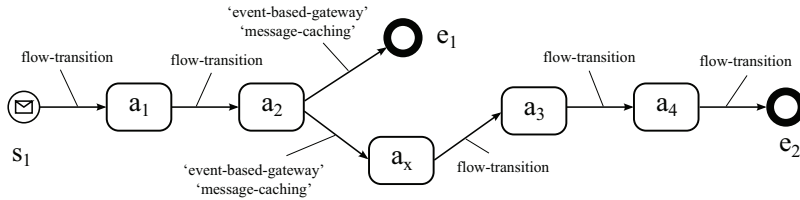


Figure 5.3: Business process graph of the ‘train-reservation’ process (Figure. 5.1)

5.2.2 Neighborhood context

We define the neighborhood context of a service as a business process fragment that includes the associated service and the closest relations to its neighbors. A neighborhood context is presented as a graph of which the associated service is located at the center. Its neighbors are located in layers according to their shortest path lengths to the associated service. The neighbor context of a service can be considered as a business fragment that presents the behavior of the associated service.

We present in the following definitions that are used to define the neighborhood context.

Definition 5.2.4 (connection path). *A connection path from a_i to a_j in a business process graph G_P , denoted by ${}_{a_i}^{a_j}\mathcal{P}_P$, is a sequence of services a_1, a_2, \dots, a_k where $a_1 = a_i, a_k = a_j$ and $\exists ({}_{a_t}^{a_{t+1}}f_P \in C_P^* \vee {}_{a_{t+1}}^{a_t}f_P \in C_P^*) \forall 1 \leq t \leq k-1$.*

According to Definition 5.2.4, a connection path in a business process graph is *undirected*. It means that the edges in a connection path can be oriented in different directions. For example, in Figure. 5.2, a connection path from ‘Search flights’ (a_5) to ‘Request customer detailed Info.’ (a_7) can be either (‘Search flights’, ‘Present alternatives’, ‘Request customer detailed Info.’) or (‘Search flights’, ‘Present alternatives’, ‘Request credit card Info.’, ‘Process payment’, ‘Request customer detailed Info.’).

Definition 5.2.5 (connection path length). *The length of a connection path ${}_{a_i}^{a_j}\mathcal{P}_P$, denoted by $\mathcal{L}({}_{a_i}^{a_j}\mathcal{P}_P)$ is the number of connection flows in the path.*

Definition 5.2.6 (shortest connection path). *The shortest connection path between a_i and a_j , denoted by ${}_{a_i}^{a_j}\mathcal{S}_P$, is the connection path between them that has the minimum connection path length.*

For example, in Figure. 5.2, the shortest path from ‘Search flights’ to ‘Request customer detailed Info.’ is (‘Search flights’, ‘Present alternatives’, ‘Request customer detailed Info.’) and its length is 2.

Definition 5.2.7 (k^{th} -layer neighbor). *a_j is a k^{th} -layer neighbor of a_i in a business process P iff $\exists {}_{a_i}^{a_j}\mathcal{P}_P : \mathcal{L}({}_{a_i}^{a_j}\mathcal{P}_P) = k$. The set of k^{th} -layer neighbors of a service a_i is denoted by $N_P^k(a_i)$. $N_P^0(a_i) = \{a_i\}$.*

For example in Figure. 5.2, ‘Receive reservation request’ and ‘Present alternatives’ are the 1st-layer neighbors of ‘Search flights’; ‘Search flight’, ‘end-event’, ‘Request customer detail Info.’ and ‘Request credit card Info.’ are the 1st-layer neighbors of ‘Present alternatives’; ‘Request credit card Info.’ is one of the 2nd-layer neighbors of ‘Search flights’ and so on.

As the distance from a service a_i to its k^{th} -layer neighbors is k , we can imagine that the k^{th} -layer neighbors of a service a_i are located on a circle whose center is a_i and k is the radius. The circle is latent since it exists but is not explicitly represented

in the business process graph. We call this latent circle a *connection layer* and the area limited by two adjacent latent circles a *connection zone*. Connection layers and connection zones of a service are numbered. A connection flow connecting two $(k-1)^{th}$ -layer neighbors, or a $(k-1)^{th}$ -layer neighbor to a k^{th} -layer neighbor is called a k^{th} -zone flow (Definition 5.2.8).

Definition 5.2.8 (k^{th} -zone flow). ${}_{a_u}^{a_v} f_P$ is a k^{th} -zone flow of a_i iff $\exists {}_{a_u}^{a_v} f_P : (a_u, a_v \in N_P^{k-1}(a_i)) \vee (a_u \in N_P^{k-1}(a_i) \wedge a_v \in N_P^k(a_i)) \vee (a_v \in N_P^{k-1}(a_i) \wedge a_u \in N_P^k(a_i))$. The set of all k^{th} -zone flows of a service $a_i \in P$ is denoted by $Z_P^k(a_i)$. $Z_P^0(a_i) = \emptyset$ and $|Z_P^k(a_i)|$ is the number of connection flows in the k^{th} connection zone of a_i .

For example in Figure. 5.2, the connection from ‘Present alternatives’ to ‘Request customer detailed Info.’ is the 2^{nd} -zone flow of ‘Search flights’ while the connection from ‘Request customer detailed Info.’ to ‘Process payment’ is its 3^{rd} -zone flow. $|Z_{P_2}^2(\text{‘Search flights’})| = 3$ as in the 2^{nd} -zone of ‘Search flight’, there are three connection flows, which are from ‘Present alternatives’ to ‘Request customer detailed Info.’, ‘Request credit card Info.’ and an end event.

Intuitively, the connection paths between two services present their relation in term of closeness. The longer the connection path is, the weaker their relation is and the shortest connection path between two services presents their best relation. To illustrate the best relations of a service to others services in a business process, we define the *neighborhood context graph* (formally defined in Definition 5.2.9) which presents all the shortest paths from a service to others. Each service in a business process has a neighborhood context graph. Each vertex in the neighborhood context graph is associated to a number which indicates *the shortest path length of the connection path to the associated service*. The vertexes that have the same shortest path length value are considered to have the same distance to the associated service and are located on the same layer around the associated service. We name the number associated to each service in a neighborhood context graph the *layer number*. The area limited between two adjacent layers is called zone. The edge connecting two vertexes in a neighborhood context graph belongs to a zone. We assign to each edge in the neighborhood context graph a number, so-called *zone number*, which determines the zone that the edge belongs to.

The edge connecting two services a_i, a_j in the neighborhood context graph of a service a_x is associated to a zone number such that: if a_i and a_j are located on two adjacent layers, the edge (a_i, a_j) will belongs to the zone limited by the two adjacent layers; and if a_i and a_j are located on the same layer, the edge connecting them belongs to the outer zone of the layer they are located on.

Concretely, assume that e_{ij} is the edge connecting two vertexes a_i and a_j in the neighborhood context graph of a service a_x . The lengths of the shortest connection paths connecting a_i and a_j to a_x are $l_1 = \mathcal{L}_{(a_i)}^{(a_x)} \mathcal{S}_P$ and $l_2 = \mathcal{L}_{(a_j)}^{(a_x)} \mathcal{S}_P$ respectively. Let $d = |l_1 - l_2|$, d has only two possible values, which are 0 and 1 (see Appendix A.2 for the proof). In the case that $d = 0$ ($l_1 = l_2$), i.e. a_i and a_j are both l_1^{th} -layer neighbors

of a_x , we assign to e_{ij} $l_1 + 1$ as zone value. In the case that $d = 1$, i.e. a_i and a_j belong to two adjacent layers, e_{ij} is the k^{th} -zone flow connecting a_i and a_j and we assign to e_{ij} the zone value k , i.e. $\min(l_1, l_2) + 1$. Consequently, we assign to the connection flow connecting a_i and a_j in the neighborhood context graph of a_x the value $\text{Min}(\mathcal{L}_{a_i}^{a_x} \mathcal{S}_P, \mathcal{L}_{a_j}^{a_x} \mathcal{S}_P) + 1$. The maximum zone value of all connection flows in the context graph of a_x will be $\text{Max}(\mathcal{L}_{a_t}^{a_x} \mathcal{S}_P) + 1 \forall a_t \in P$.

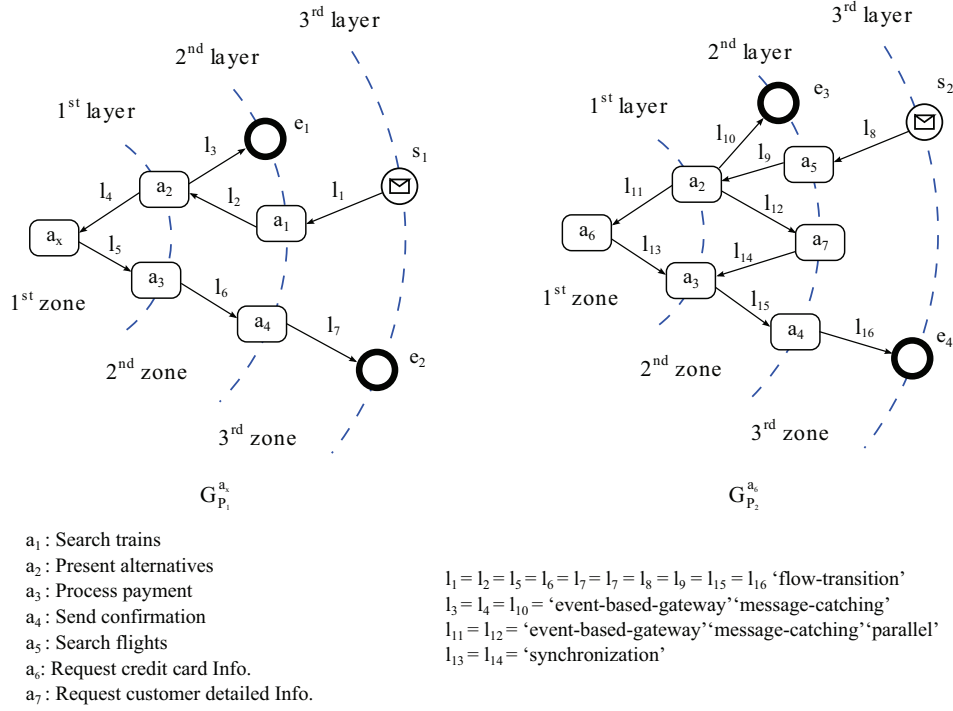


Figure 5.4: Neighborhood context graphs of a_x (in Figure 5.1) and a_6 (in Figure 5.2)

Definition 5.2.9 (Neighborhood context graph). *The neighborhood context graph of a service $a_x \in P$, denoted by $G_P^{a_x} = (V_P^{a_x}, E_P^{a_x}, L_P^{a_x})$, is a labeled directed graph created from $G_P = (V_P, E_P, L_P)$. $V_P^{a_x}$ is a set of vertexes associated to their layer numbers; $E_P^{a_x}$ is a set of edges associated to their zone numbers; and $L_P^{a_x}$ is a set of edge labels in $G_P^{a_x}$. $V_P^{a_x}$, $E_P^{a_x}$, and $L_P^{a_x}$ are defined as:*

- $V_P^{a_x} = \{(a_i, \mathcal{L}_{a_i}^{a_x} \mathcal{S}_P) : a_i \in V_P\}$
- $E_P^{a_x} = \{((a_i, a_j)_{a_i}^{a_j} z_P^{a_x}) : (a_i, a_j) \in E_P, a_i z_P^{a_x} = \text{Min}(\mathcal{L}_{a_i}^{a_x} \mathcal{S}_P, \mathcal{L}_{a_j}^{a_x} \mathcal{S}_P) + 1\}$
- $L_P^{a_x} = L_P$

For example, the neighborhood context graph of 'unknown' service (a_x) in the 'train-reservation' process (Figure. 5.1) is $G_{P_1}^{a_x} = (V_{P_1}^{a_x}, E_{P_1}^{a_x}, L_{P_1}^{a_x})$ where $V_{P_1}^{a_x} = \{(a_x, 0), (a_2, 1), (a_3, 1), (a_1, 2), (e_1, 2), (a_4, 2), (s_1, 3), (e_2, 3)\}$; $E_{P_1}^{a_x} = \{((a_2, a_x), 1), ((a_x, a_3), 1), ((a_2, e_1), 2), ((a_1, a_2), 2), ((a_3, a_4), 2), ((s_1, a_1), 3), ((a_4, e_2), 3)\}$; $L_{P_1}^{a_x} = L_{P_1} = \{((s_1, a_1),$

‘flow-transition’), $((a_1, a_2)$, ‘flow-transition’), $((a_2, e_1)$, ‘event-based-gateway’‘message-catching’), $((a_2, a_x)$, ‘event-based-gateway’‘message-catching’), $((a_x, a_3)$, ‘flow-transition’), $((a_3, a_4)$, ‘flow-transition’), $((a_4, e_2)$, ‘flow-transition’)}. This neighborhood context graph and the neighborhood context graph of the service ‘Request credit card Info.’ created from ‘flight-reservation’ process (Figure. 5.2) are depicted in Figure. 5.4.

In any business process graph, including graphs that contain loops, we can always calculate the shortest path length between two services. Therefore, in the neighborhood context graph of a service, we can always identify the layers on which other services are located. Consequently, we can always assign layer number to a service and thus, zone number to a connection flow in a neighborhood context graph.

5.2.3 Loop cases

In this section, we show how we handle the loop cases in neighborhood context graphs. There are three typical loop cases in a business process: *self-loop*, *loop via another service* and *loop via other services*. By applying Definition 5.2.9, the possible layers to which the loop services can belong are depicted in Figure 5.5.

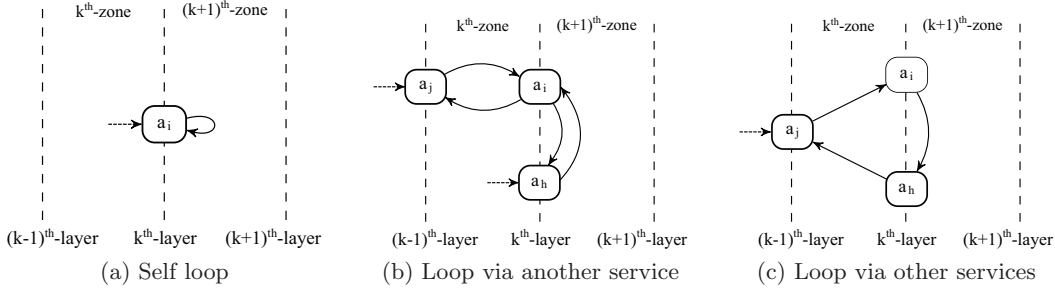


Figure 5.5: Connection flows in loop cases

In the self-loop case (Figure. 5.5a), if a service a_i is located on the k^{th} -layer, then its self-loop edge belongs to the zone $(k + 1)^{th}$ because according to Definition 5.2.9, the zone number of this edge is $\frac{a_i}{a_i} z_P^{a_x} = \text{Min}(\mathcal{L}_{a_x}^{a_i} \mathcal{S}_P, \mathcal{L}_{a_x}^{a_i} \mathcal{S}_P) + 1 = \text{Min}(k, k) + 1 = k + 1$.

In the loop-via-another-service case (Figure. 5.5b), there are two possibilities: (i) the two services are located on adjacent layers (e.g. a_j and a_i) and (ii) the two services are located on the same layer (e.g. a_i and a_h). In the first possibility, assume that a_i and a_j are respectively located on the $(k - 1)^{th}$ -layer and k^{th} -layer, the edges connecting them are assigned the zone number $\frac{a_j}{a_i} z_P^{a_x} = \frac{a_i}{a_j} z_P^{a_x} = \text{Min}(\mathcal{L}_{a_x}^{a_i} \mathcal{S}_P, \mathcal{L}_{a_x}^{a_j} \mathcal{S}_P) + 1 = \text{Min}(k - 1, k) + 1 = k$, i.e. they are on the zone limited by these adjacent layers. In the second possibility, assume that a_i and a_h are located on the same k^{th} -layer, the edges connecting them are assigned the zone number $\frac{a_h}{a_i} z_P^{a_x} = \frac{a_i}{a_h} z_P^{a_x} = \text{Min}(\mathcal{L}_{a_x}^{a_i} \mathcal{S}_P, \mathcal{L}_{a_x}^{a_h} \mathcal{S}_P) + 1 = \text{Min}(k, k) + 1 = k + 1$, i.e. they are presented on the outer zone of the layer where the services are located.

In the loop-via-other-services case (for example, Figure. 5.5c shows the loop created by 3 services), the edges between services are assigned their zone numbers following Definition 5.2.9. For the two services located on adjacent layers (e.g. a_j and a_i , or a_j and a_h in Figure. 5.5c), the edge connecting them belongs to the zone limited by these layers. For the two services located on the same layer (e.g. a_i and a_h in Figure. 5.5c), the edge connecting them belongs to the outer zone of their layer.

5.3 Querying services

Each service in a business process has a neighborhood context which presents the relations between the service and its neighbors. By matching neighborhood contexts, we can find services that have similar relations to common neighbors. In this section, we present a query language used to retrieve relevant services to a selected position in a business process based on its neighborhood context matching. We focus on presenting the query language (section 5.3.1), query's execution (section 5.3.2) and its advantages (section 5.3.3). The neighborhood context matching is elaborated in the next section (section 5.4).

5.3.1 Query's grammar

The query in our approach not only helps to search for relevant services based on neighborhood contexts but also allows process analysts to add constraints to the requested context to filter the searching results. The query language in our approach consists of three parameters, which are: *associated service*, *connection constrain*, and *radius*. The associated service is the service whose neighborhood context is taken into account to match with other contexts (the neighborhood context matching is presented in section 5.4). Connection constrains are services or connection flows to be included/excluded to filter the query's results. The radius is the number of connection layers taken into account for the neighborhood context matching. It specifies the largeness of the considered neighborhood contexts.

We present in the following our proposed query's grammar using the Extended Backus-Naur Form (EBNF)¹ [154]. We use ';' to separate the input parameters; '(' and ')' to separate query's constrains; '<' and '>' to group services or connection flows; '[' and ']' to separate a connection flow and its ending services. We use '+' and '-' signs to include/exclude constraints; and '|' sign for multiple choice operator. Details of the grammar are presented in Table 5.1.

Some notations in our query grammar are similar to the EBNF notations. In EBNF, comma (,) is used for concatenation; semicolon (;) is used for termination; vertical bar (|) is used for alternation; [...] is used for option; (...) is used for grouping; {...} is used for repetition and '...' is used for terminal string.

¹the EBNF standard is adopted by ISO, no. ISO/IEC 14977

1.	Query	::=	ServiceID,':',[Constraint],':',Radius;
2.	ServiceID	::=	Character,{Character Digit};
3.	Constraint	::=	('+' '-')Term Constraint, ' ',Term;
4.	Term	::=	Item Term, '+',Item Term, '-',Item;
5.	Item	::=	ServiceID ConFlow '(,Constraint,')';
6.	ConFlow	::=	'<',[ServiceID], '[,FlowString,']',[ServiceID]'>;
7.	FlowString	::=	ConElement,{ConElement};
8.	ConElement	::=	'sequence' 'AND-split' 'AND-join' 'OR-split' 'OR-join' 'XOR-split' 'XOR-join';
9.	Radius	::=	DigitNotZero,{Digit};
10.	Character	::=	'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z';
11.	DigitNotZero	::=	'1' '2' '3' '4' '5' '6' '7' '8' '9';
12.	Digit	::=	'0' DigitNotZero;

Table 5.1: Query grammar

In the following, we detail the grammar of our query language:

- The query is defined in line 1 with three parameters separated by ':'. The constraint is optional. It can be defined to filter the query result. It can also be eliminated if we want to execute only the neighborhood context matching without filtering.
- In line 2, the service identifier is defined as a string of characters or digits. However, it has to start by a character.
- In line 3, we define the constraint. A constraint can be an included/excluded service or connection flow. It can also include different items and operators. Operators in a constraint can be OR, INCLUDE, EXCLUDE. We use '|', '+', and '-' to specify these operators. Consequently, we define a constraint as a 'Term' or another constraint with '|' operator.
- In line 4 we define a 'Term' as an 'Item' or another constraint with the '+' and '-' operators.
- The 'Item' is defined in line 5. It can be a service or a connection flow that is included/excluded in the query. It can also be another constraint which is grouped by '(' and ')'. Definitions in line 3, 4 and 5 allow specifying a constraint with multiple services, connection flows, operations and grouped conditions.
- In line 6, we define a connection flow which is presented within '<' and '>' signs. It includes a string of connection elements that connects two services.
- The string of connection elements is defined in line 7. It includes at least a connection element which is defined in line 8.
- In line 9, we define the radius as a natural number greater than 0.

- Finally, lines 10, 11, 12 define literal characters and digits.

Examples of our query are given in Table 5.2. All of the queries are finding services that have neighborhood contexts similar to the neighborhood context of s_x . These neighborhood contexts are limited in 3 layers. In Table 5.2, we explain the concerned constraints used to filter the query result.

Query	Explanation
$s_x::3$	neighborhood context matching without filtering
$s_x:+s_1-s_2:3$ $s_x:-s_2+s_1:3$	services in the results have s_1 but do not have s_2 in their neighborhood contexts
$s_x:+(s_1 s_2)-s_3:3$ $s_x:-s_3+(s_1 s_2):3$	services in the results have s_1 or s_2 but do not have s_3 in their neighborhood contexts
$s_x:+<s_1['sequence']s_2>:3$	services in the results have a connection flow s_1 'sequence' s_2 in their neighborhood contexts
$s_x:-s_1+<['AND-split']\text{'OR-split'}>:3$	services in the results do not have s_1 but have a connection flow 'AND-split''OR-split' in their neighborhood contexts
$s_x:-(s_1 <['OR-split']s_2>):3$	services in the results do not have s_1 or a connection flow 'OR-split' s_2 in their neighborhood contexts
$s_x:+<s_1['AND-split']s_2>+<s_2['AND-join']s_3>:3$	services in the results have both connection flows s_1 'AND-split' s_2 and s_2 'AND-join' s_3 in their neighborhood contexts

Table 5.2: Query examples

In our motivating example (Figure 5.1), assume that a process analyst wants to find services that have similar context to 'Search trains' (a_1) within the first zone, he can make a query as: $a_1::1$. In the case that he wants to know possible payment methods, he may select 'Process payment' service (a_3) and add a constraint which does not include a 'sequence' that connects to 'Send confirmation' service (a_4). He may also widen the considered neighborhood contexts in two layers to get more results. Consequently, his query can be: $a_3:-(<['sequence']a_4>):2$. In another context, if he wants to know existing services that are similar to 'Search trains', include a new service 'Request payment Info.' (a_8) and 'Process payment' but not the AND-join connection between them, he will select the 'Search trains' service and declare the constraints to be excluded. His query can be: $a_1:+a_8+a_3-(<a_8['AND-join']a_3>):5$.

5.3.2 Query's execution

Our query is developed to filter the services returned by the neighborhood context matching. In general, the procedure of the query execution is the following:

1. We capture the neighborhood context of the associated service. This neighborhood context is identified by the associated service and connection flows to its neighbors. The largeness of the neighborhood context is specified by the radius parameter.
2. We match the neighborhood context of the associated service to others in other business processes.
3. We refine the matching result by selecting only services whose neighborhood contexts satisfy the query's constraint.
4. We sort the selected services based on the matching values and pick up top-N services. N can be flexibly tuned by the process analyst for the query's response.

The neighborhood context matching is executed beforehand. Then the query constraint is applied on the matching result to filter the unrelated services.

5.3.3 Advantages of the query

Our query can help process analysts to retrieve relevant services to facilitate the business process design. Some advantages of our query are presented in the following.

- Query's presentation: The query in our approach is described as a text string. It contains predefined information, e.g. service's ID, included/excluded connection flows. Hence, it can be extended in other standard formats such as XML, JSON. In addition, it can be expressed by an SQL-like language.
- Flexible result: The query can be extended to query fragments of business processes. The simplest way is to add an additional parameter to specify the radius of the requested fragments. Then, instead of returning a list of relevant services, we return a list of process fragments that include the relevant services and their neighborhood contexts which are specified by the additional radius.
- Querying an 'unknown' service: The associated service's ID in our query is used to specify the neighborhood context for querying. So, if we know the neighborhood context of an associated service regardless its ID, we can execute the query. Consequently, we can find relevant services to an 'unknown' service in a business process if we know the connection flows between it and its neighbors.
- Business process auto-completion: As our query can be used to find services that are relevant to an 'unknown' service, it can be applied for business process auto-completion, which dynamically suggests a process analyst next relevant services to his design. For example, consider a process analyst is designing a process. She selects a service s_x and drags from it a 'sequence' connection element. She needs recommendations for the 'next' service of s_x . We imagine that there is an 'unknown' service connected from s_x by a 'sequence'. The neighborhood context of this 'unknown' service is specified by a 'sequence' connection element from s_x . Based on this neighborhood context, using our query, we can retrieve services

that are relevant to the ‘unknown’ service. Consequently, we can recommend the process analyst possible ‘next’ services of s_x .

5.4 Neighborhood context matching

The k^{th} -zone neighbors of a service and their connection flows create a process fragment around the associated service. This fragment contains the business context that reflects the behavior of the associated service. In this section, we present our methodology to compute the matching between two neighborhood contexts. This matching is used for service recommendation. To compute the neighborhood context matching, we propose to *match all connection flows that belong to the same connection zone and have the same ending services*. Particularly, for the first zone, we *match the connection flows connecting the associated services to the same services in the first layer*.

To illustrate the computation process, we demonstrate the matching between the neighborhood context of the ‘unknown’ service in the ‘train-reservation’ process (a_x in Figure. 5.1) and the ‘Request credit card Info.’ service in the ‘flight-reservation’ process (a_6 in Figure. 5.2). The neighborhood context graphs of these services are shown in Figure 5.4.

5.4.1 Connection flow matching

To compute the similarity between two services, we propose to match all the connection flows that connect them to their neighbors. Since each connection flow is a sequence of connection elements which can easily mapped to a sequence of characters, we propose to use the Levenshtein distance (LD for short) [82] to compute the matching between two connection flows. In information theory, the LD is a metric for measuring the difference between two sequences of characters. The LD is defined as the minimum number of edits needed to transform one sequence of characters into the other, with the allowable edit operations being *insertion*, *deletion*, or *substitution* of a single element. For example, the LD between “gumbo” and “gambol” is 2, between “kitten” and “sitting” is 3, etc. Inspired from this, we consider each connection element as a character, then *a connection flow is presented by a sequence of characters* and the similarity between two connection flows can be easily computed by LD.

Concretely, given two connection flows ${}^{a_j}f_{P_1} = c_1c_2 \dots c_n$ and ${}^{a_y}f_{P_2} = c'_1c'_2 \dots c'_m$, the pattern matching between them is computed by Equation. (5.1).

$$M({}^{a_j}f_{P_1}, {}^{a_y}f_{P_2}) = 1 - \frac{\text{LevenshteinDistance}({}^{a_j}f_{P_1}, {}^{a_y}f_{P_2})}{\text{Max}(n, m)} \quad (5.1)$$

In our example, we have $M({}^{a_2}f_{P_1}, {}^{a_2}f_{P_2}) = M(\text{‘flow-transition’}, \text{‘flow-transition’}) = 1$; $M({}^{a_x}f_{P_1}, {}^{a_6}f_{P_2}) = M(\text{‘event-based-gateway’}, \text{‘message-caching’}, \text{‘event-based-gateway’}, \text{‘message-caching’}, \text{‘parallel’}) = 0.67$ and so on.

5.4.2 Context matching

To compute the neighborhood context matching between two services, we propose to synthesize the matching of the connection flows in the two contexts. There are two cases to consider: matching in the *first zone* and matching in *other zones*. In the first zone, we match the connection flows that connect the two associated services and same services in the first layer. In other zones, we match the connection flows that connect the same services. We sum all matching values then divide them by the number of connection flows in the considered zones of the first service.

Concretely, the neighborhood context matching between $a_i \in P_1$ and $a_j \in P_2$ within k zones, denoted by $\text{MC}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j})$, is computed by Equation 5.2.

$$\text{MC}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j}) = \frac{\sum_{t=1}^k \sum_{\substack{a_u f_{P_1} \\ a_m f_{P_2} \in Z_{P_1}^t}} \text{MF}^t(a_u f_{P_1}, a_m f_{P_2})}{\sum_{t=1}^k |Z_{P_1}^t(a_i)|} \quad (5.2)$$

where k is the number of considered zones, $|Z_{P_1}^t(a_i)|$ is the number of connection flows in zone t^{th} of $G_{P_1}^{a_i}$, and $\text{MF}^t(a_u f_{P_1}, a_m f_{P_2})$ is the matching between two connection flows $a_u f_{P_1}$ and $a_m f_{P_2}$ in zone t :

$$\text{MF}^t(a_u f_{P_1}, a_m f_{P_2}) = \begin{cases} M(a_u f_{P_1}, a_m f_{P_2}) & \text{if } \begin{cases} t = 1, & (a_u = a_m) \vee (a_u = a_n) \\ & \vee (a_u = a_v \wedge a_m = a_n) \\ t \neq 1, & (a_u = a_m \wedge a_v = a_n) \end{cases} \\ 0 & \text{other cases} \end{cases}$$

For example, the neighborhood context matching between a_x and a_6 (Figure.5.4) computed by Equation 5.2 is: $\text{MC}^3(G_{P_1}^{a_x}, G_{P_2}^{a_6}) = (M(a_3 f_{P_1}, a_6 f_{P_2}) + M(a_2 f_{P_1}, a_6 f_{P_2}) + M(a_2^e f_{P_1}, a_6^e f_{P_2}) + M(a_3^a f_{P_1}, a_6^a f_{P_2}) + M(a_4^e f_{P_1}, a_6^e f_{P_2})) / (2 + 3 + 2) = (0 + 0.66 + 1 + 1 + 1) / 7 = 0.52$.

5.4.3 Zone weight consideration

The behavior of a service is strongly reflected by the connection flows to its closet neighbors while the interactions with other neighbors in the further layers do not heavily reflect its behavior. Therefore, we propose to assign a weight (w_t) for each t^{th} connection zone, so called *zone-weight* and integrate this weight into the similarity computation. Since the zone-weight has to have greater values in smaller t^{th} connection zone, we propose the zone-weight a value computed by a polynomial function which is $w_t = \frac{k+1-t}{k}$, where t is the zone number ($1 \leq t \leq k$) and k is the number of considered zones around the service. All connection flows connecting either to or from the associated service have the greatest weight ($w_1 = 1$) and the connection flows

connecting to/from services in the furthest zone have the smallest weight ($w_k = \frac{1}{k}$). The aggregated weight values is given by Equation 5.3.

$$W = \sum_{t=1}^k \frac{k+1-t}{k} = \frac{k+1}{2} \quad (5.3)$$

Then, the neighborhood contexts matching between $G_{P_1}^{a_i}$ and $G_{P_2}^{a_j}$ within k zones and with zone weight consideration, denoted by $\text{MW}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j})$, is given by Equation 5.4.

$$\text{MW}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j}) = \frac{2}{k+1} \times \sum_{t=1}^k \frac{k+1-t}{k} \times \text{MZ}^t(G_{P_1}^{a_i}, G_{P_2}^{a_j}) \quad (5.4)$$

where $\text{MZ}^t(G_{P_1}^{a_i}, G_{P_2}^{a_j})$ is the matching value of all connection flows in the t^{th} zone of $G_{P_1}^{a_i}$ and $G_{P_2}^{a_j}$.

If the neighborhood contexts of a_i and a_j are completely different, i.e. there is no similar service on the same layer, or the matching of all connection flows of their neighborhood contexts is 0, $\text{MZ}^t(G_{P_1}^{a_i}, G_{P_2}^{a_j}) = 0, \forall t = 1..k$. This yields $\text{MW}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j}) = 0$.

If the neighborhood contexts of a_i and a_j are completely matched, i.e. they have the same neighbors on the same layers and the matching of all connection flows is equal to 1, we have: $\text{MZ}^t(G_{P_1}^{a_i}, G_{P_2}^{a_j}) = 1, \forall t = 1..k$, and:

$$\begin{aligned} \text{MW}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j}) &= \frac{2}{k+1} \times \sum_{t=1}^k \frac{k+1-t}{k} \\ &= \frac{2}{k+1} \times \frac{k+1}{2} \quad (\text{by Equation 5.3}) \\ &= 1 \end{aligned}$$

Consequently, the neighborhood context matching of two services a_i and a_j , i.e. $\text{MW}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j})$, has value in the range of $[0, 1]$.

To compute $\text{MZ}^t(G_{P_1}^{a_i}, G_{P_2}^{a_j})$, for the first zone ($t = 1$), we match the connection flows connecting the two associated services to the same neighbors in the first layer. Then we divided the result by the number of connection flows in the first zone of the first service. For further zone ($t > 1$), we match all connection flows that are in the same zone and connect the same ending services. Then we divided the result by the number of connection flows in the considered zone of the first service. The connection flow matching in the t^{th} zone is given by Equation 5.5.

$$\text{MZ}^t(G_{P_1}^{a_i}, G_{P_2}^{a_j}) = \frac{\sum_{\substack{a_u f_{P_1} \in Z_{P_1}^t, \\ a_n f_{P_2} \in Z_{P_2}^t}} \text{MF}^t(a_u f_{P_1}, a_n f_{P_2})}{|Z_{P_1}^t(a_i)|} \quad (5.5)$$

where:

$$\text{MF}^t(a_u f_{P_1}, a_n f_{P_2}) = \begin{cases} M(a_u f_{P_1}, a_n f_{P_2}) & \text{if } \begin{cases} t = 1, & (a_u = a_m) \vee (a_v = a_n) \\ & \vee (a_u = a_v \wedge a_m = a_n) \\ t \neq 1, & (a_u = a_m \wedge a_v = a_n) \end{cases} \\ 0 & \text{other cases} \end{cases}$$

From Equation 5.4 and Equation 5.5, we have:

$$\text{MWW}^k(G_{P_1}^{a_i}, G_{P_2}^{a_j}) = \frac{2}{k+1} \times \sum_{t=1}^k \frac{k+1-t}{k} \times \frac{\sum_{\substack{a_u f_{P_1} \in Z_{P_1}^t \\ a_n f_{P_2} \in Z_{P_2}^t}} \text{MF}^t(a_u f_{P_1}, a_n f_{P_2})}{|Z_{P_1}^t(a_i)|} \quad (5.6)$$

where:

$$\text{MF}^t(a_u f_{P_1}, a_n f_{P_2}) = \begin{cases} M(a_u f_{P_1}, a_n f_{P_2}) & \text{if } \begin{cases} t = 1, & (a_u = a_m) \vee (a_v = a_n) \\ & \vee (a_u = a_v \wedge a_m = a_n) \\ t \neq 1, & (a_u = a_m \wedge a_v = a_n) \end{cases} \\ 0 & \text{other cases} \end{cases}$$

For example, the service context matching between a_x and a_6 (Figure. 5.4) computed by Equation 5.6 is: $\text{MWW}^3(G_{P_1}^{a_x}, G_{P_2}^{a_6}) = \frac{2}{3+1} \times (\frac{3}{3} \times \frac{M(a_2 f_{P_1}, a_6 f_{P_2}) + M(a_3 f_{P_1}, a_6 f_{P_2})}{2} + \frac{2}{3} \times \frac{M(a_2^e f_{P_1}, a_2^e f_{P_2}) + M(a_3^e f_{P_1}, a_3^e f_{P_2})}{3} + \frac{1}{3} \times \frac{M_p(a_4^e f_{P_1}, a_4^e f_{P_2})}{2}) = \frac{2}{4} \times (\frac{0.66+0}{2} + \frac{2}{3} \times \frac{1+1}{3} + \frac{1}{3} \times \frac{1}{2}) = 0.47$.

5.4.4 Computational complexity

In our approach, only the connection flows connecting common neighbors in two adjacent layers are taken into account for the matching computation. So, by using queues (data structure) to store the common neighbors and track them from the nearest layers to the furthest layers, *we avoid the redundant checking of unrelated neighbors*. On the other hand, the number of services as well as the number of common neighbors in a business process are not great², our algorithm can run fast in computing the neighborhood context matching of two services. The worst case of this algorithm's computation time is $\mathcal{O}(n_S \times n_P \times n \times k)$, where n_S is the number of services, n_P is the number of business processes, n is the maximum number of common services located on a layer and k is the number of considered layers. The worst case only happens when all the business processes in the system are completely

²We made statistics on the public dataset used in our experiment. This dataset consists of real business processes designed for financial services, telecommunications, and other domains. We have that in average, there are 11.36 services in a business process (section 7.3.2)

matched. In our experiments (section 7.3.2), we measured the computation time and the result shows that our algorithm runs fast in making recommendations. In addition, the performance of the algorithm can be improved by processing the neighborhood context matching periodically off-line.

5.5 Recommendation

We make recommendations based on the neighborhood context matching. For a selected service, we compute its neighborhood context graph matching with other services in other business processes. Then, we sort the computed matching values in descending order and pick up top- N services that have the highest matching values for recommendation.

As the neighborhood context graph presents the interactions between the associated service and its neighbors, it infers the associated service's behavior. Therefore, the matching between service context graphs exposes the similarity between associated services in terms of their behaviors. In our approach, the higher the service context matching value is, the more similar the services are.

There are two typical cases that a process analyst needs service recommendation: *discovering services* or *improving the ongoing designed business process*.

In the first case, when the process analyst wants to discover services that are suitable to a position in a business process, she marks this position as an 'unknown' service (a round rectangle with a '?' symbol). Our approach will capture the neighborhood context of the 'unknown' service. Then, it matches the captured context with others and retrieves relevant services to the selected position. For example, the process analyst wants to discover services that are suitable to the position of a_x in the 'train-reservation' process (Figure 5.1). She selects this position. Our approach captures the neighborhood context of a_x and matches it with neighborhood contexts of other services in other processes. The neighborhood context matching between a_x and other services in the 'flight-reservation' process (Figure 5.2) are $\text{MW}^3(G_{P_1}^{a_x}, G_{P_2}^{a_6}) = 0.47$; $\text{MW}^3(G_{P_1}^{a_x}, G_{P_2}^{a_7}) = 0.47$; $\text{MW}^3(G_{P_1}^{a_x}, G_{P_2}^{a_5}) = 0$; etc. In our motivating example (section 2.2), we compute the matching between a_x and services in 'flight-reservation' and 'hotel-reservation' process. According to the matching values, we recommend for a_x services 'Request customer detailed Info.', 'Request customer basic Info.' and 'Request credit card Info.' (Figure 5.6).

In the second case, when the process analyst wants to extend (or improve) the ongoing designed process, she may need recommendations provided by our approach. For example, if she wants to find other possibilities at some positions in the process, she will select services at these positions. Our approach will recommend her relevant services. With these recommendations, the process analyst can create different process variants from the current designed process. In our motivating example (Figure 5.7, taken from section 2.2), when the process analyst selects services 'Search trains' and 'Process payment' to know other possibilities at these positions, our ap-

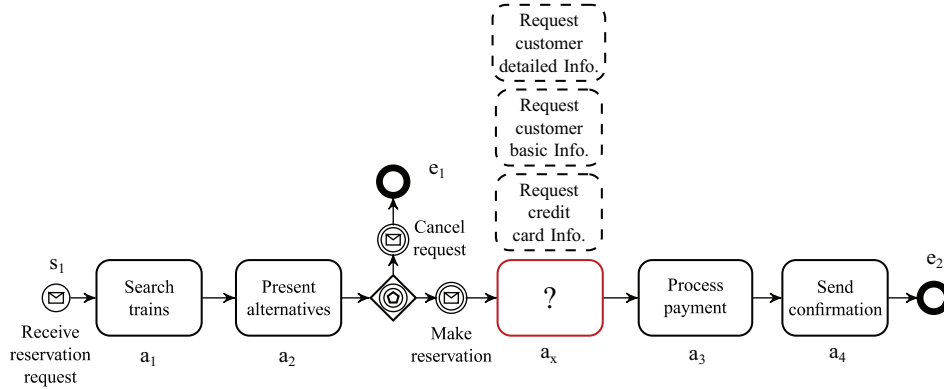


Figure 5.6: Recommendations for the 'unknown' service

proach recommends services as shown in Figure 5.7. Then, she may improve the 'train-reservation' process to a combined service as shown in Figure 5.8 (also taken from our motivating example in section 2.2)

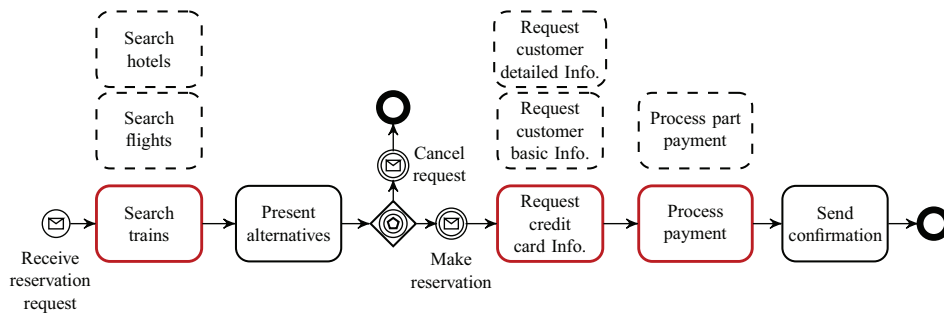


Figure 5.7: Recommendations for the selected services

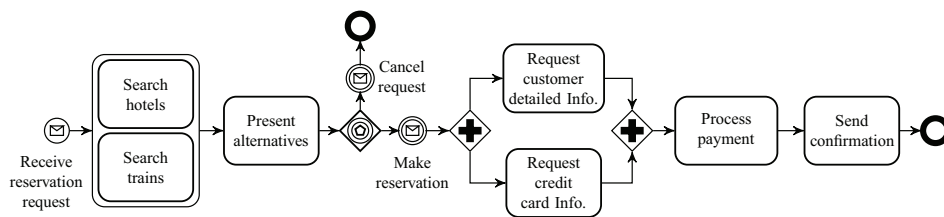


Figure 5.8: New traveling process improved from the train reservation process

In addition, our approach can retrieve services that have the same neighborhood context with the selected service (neighborhood context matching value is equal to 1). It can be associated to a functionality-filtering approach, which can filter services that have the same function, to find services that have the same function and behavior with a given service. This can help to find services that can replace a given service in case of unavailability.

5.6 Taking into account parallel flow relations

In our previous model, we take into account the causal relation between services. We consider only the case where a service is situated next to another service. We did not consider the parallel flow relations between services, i.e. the relations between services that belong to parallel flows. These relations express concurrent and choice (or condition) relations. For example, in previous model, we do not take into account the relation between a_6 and a_7 in Figure 5.2 as they are presented in parallel flows.

In this section, we examine the parallel flow relations (created by AND, OR and XOR operators) between services. We re-define the connection flow in order to capture both causal and parallel flow relations. The new connection flow definition yields the enrichment of the neighborhood context and the updated matching computation.

5.6.1 Neighborhood context

To capture the causal and parallel flow relations between services, we define the *connected relation* based on the *next relation* (which was defined in Definition 5.2.1) and update the connection flow definition as following.

Definition 5.6.1 (connected relation, connection flow). *Let A_P be the set of services and C_P be the set of connection elements in a business process P .*

Let $e_i, e_j \in A_P \cup C_P$. e_i is connected to e_j in P , denoted by $e_i \leftrightarrow_P e_j$, iff $e_i \rightarrow_P e_j$ or $e_j \rightarrow_P e_i$.

A connection flow from a_i to a_j , $a_i, a_j \in A_P$, denoted by ${}_{a_i}^{a_j}f_P$, is a sequence of connection elements $c_1, c_2, \dots, c_n \in C_P$ satisfying: $a_i \leftrightarrow_P c_1, c_1 \leftrightarrow_P c_2, \dots, c_{n-1} \leftrightarrow_P c_n, c_n \leftrightarrow_P a_j$ ³. ${}_{a_i}^{a_j}f_P \in C_P^$, C_P^* is set of sequences of connection elements in P .*

Definition 5.6.2 (connected relation label, connection flow label). *The label of a connected relation $e_i \leftrightarrow_P e_j$, $e_i, e_j \in A_P \cup C_P$, denoted by $l(e_i \leftrightarrow_P e_j)$, is defined as following:*

$$l(e_i \leftrightarrow_P e_j) = \begin{cases} e_i e_j, & \text{if } e_i \rightarrow_P e_j \\ e_j e_i, & \text{if } e_j \rightarrow_P e_i \end{cases}$$

The label of a connection flow ${}_{a_i}^{a_j}f_P$, denoted by $l({}_{a_i}^{a_j}f_P)$, is defined as following:

$$l({}_{a_i}^{a_j}f_P) = l(a_i \leftrightarrow_P c_1).l(c_1 \leftrightarrow_P c_2) \dots l(c_{n-1} \leftrightarrow_P c_n).l(c_n \leftrightarrow_P a_j)$$

where $c_1, c_2, \dots, c_n \in C_P$: ${}_{a_i}^{a_j}f_P = c_1 c_2 \dots c_n$.

For example, the label of the connection flow from “Search flights” to “Present alternatives” in Figure 5.2 is: a_5 ‘sequence’.‘sequence’ a_2 ; from “Present alternatives” to “Request customer detailed Info.” is: a_2 ‘event-based-gateway’.‘event-based-gateway’‘message-catching’.‘message-catching’‘parallel-split’.‘parallel-split’ a_7 .

We notice that:

³The connection flow from a_j to a_i is the inverse of the connection flow from a_i to a_j

- An edge connecting two services $a_i, a_j \in A_P$ can be labeled by either $l_{(a_i^j f_P)}$ or $l_{(a_j^i f_P)}$. For example, the edge connecting a_5 to a_2 in Figure. 5.2 can be label by $l_{(a_5^2 f_P)}=a_5$ ‘sequence’.‘sequence’ a_2 or $l_{(a_2^5 f_P)}=$ ‘sequence’ $a_2.a_5$ ‘sequence’.
- There can be more than one connection flow between two services. In this case, we *number* these connection flows to distinguish them. For example, there are two connection flows from a_7 to a_6 in Figure. 5.2 and we number them as follows: $l_{(a_7^6 f_P^1)}=$ ‘parallel-split’ a_7 .‘parallel-split’ a_6 and $l_{(a_7^6 f_P^2)}=a_7$ ‘synchronization’. a_6 ‘synchronization’.

We re-define the business process graph with regard to the new definition of connection flow. In this graph, we define the set of edges is a *multiset* in order to present the number of times that an edge appears. We call this graph ‘label-based business process graph’ (Definition 5.6.3).

Definition 5.6.3 (Label-based business process graph). *Let A_P be the set of services and C_P^* be the set of sequences of connection elements in a business process P . A label-based business process graph of P is an undirected labeled multigraph $G_P = (V_P, E_P, L_P, l)$ in which V_P is a set of nodes, E_P is a multiset of edges, L_P is a set of edge labels, and l is a mapping function that maps edges to labels, where:*

- $V_P = A_P$,
- $E_P \subseteq \langle A_P \times A_P, g \rangle$, $g : A_P \times A_P \rightarrow N$ is a mapping function. $g((a_i, a_j))$ is the multiplicity of (a_i, a_j) . If $g((a_i, a_j)) > 1$, the edges connecting a_i to a_j are numbered as $(a_i, a_j)^t$, $t = 1..k$, $k > 1$.
- $L_P = l(E_P)$, where:

$$l : E_P \rightarrow L_P$$

$$(a_i, a_j) \mapsto l_{(a_i^j f_P)}, \text{ if } g((a_i, a_j)) = 1$$

$$(a_i, a_j)^t \mapsto l_{(a_i^j f_P^t)}, \text{ if } g((a_i, a_j)) = k > 1, t = 1..k$$

For example, the label-based business process graph of the ‘flight-reservation’ process (Figure. 5.2) is presented in Figure. 5.9.

The modification of the connection flow definition yields the adjustment of the business process graph with additional edges connecting parallel services. These edges are taken into account to update the shortest paths between services. Consequently, they impact on the distribution of services on layers and connection flows in zones.

For example, the neighborhood context graphs of a_x (Figure. 5.1) and a_6 (Figure. 5.2) are presented in Figure. 5.10. In these graphs, all causal and parallel flow relations are presented. The parallel flow relation between a_6 and a_7 is presented by two connection flow labels l_{16} and l_{17} . Whereas, this relation can not be presented in our previous model (Figure 5.4).

By using the updated connection flow definition, we can present any connection flow connecting two parallel services. This connection flow is presented by a sequence of connected relations between them. Consequently, we can present any parallel flow relation created by an AND, an OR, an XOR or a combination of these operators.

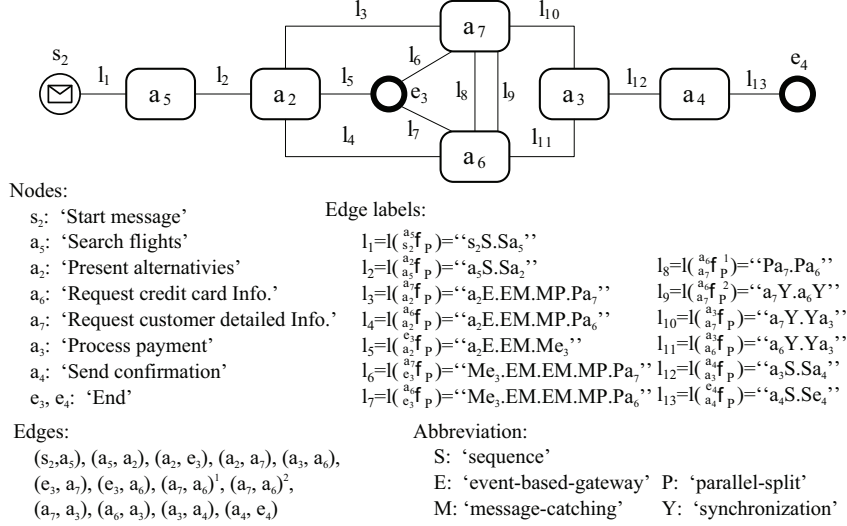


Figure 5.9: Label-based business process graph of the 'flight-reservation' process

5.6.2 Updating neighborhood context matching

As a label of a connection flow is sequence of characters, we reuse the Levenshtein distance (LD) to compute the matching between two connection flows. However, an edge connecting a_u and a_v in P_1 can be labeled by either $^{a_v}f_{P_1}$ or $^{a_u}f_{P_1}$ because G_{P_1} and G_{P_2} are undirected graphs. In addition, there can be multiple connection flows between two services. We update the matching between $^{a_u}f_{P_1}$ and $^{a_n}f_{P_2}$ as following.

Let $st_1 = l(^{a_u}f_{P_1})$, $st_2 = l(^{a_n}f_{P_2})$. Let $Diff$ be a function that computes the difference between two connection flows. We have:

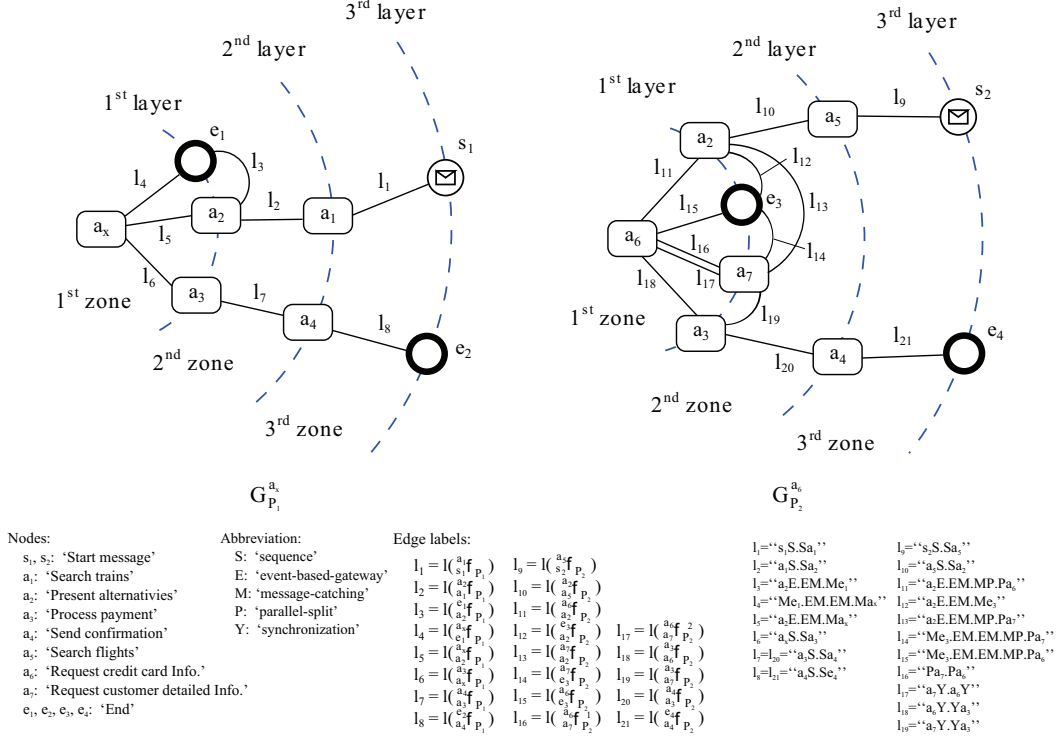
$$M(^{a_v}f_{P_1}, ^{a_n}f_{P_2}) = 1 - \frac{Diff(st_1, st_2)}{Max(length(st_1), length(st_2))} \quad (5.7)$$

where:

- $Diff(st_1, st_2) = LD(l(^{a_v}f_{P_1}), l(^{a_n}f_{P_2}))$, if $(a_u = a_m) \wedge (a_v = a_n)$
- $Diff(st_1, st_2) = LD(l(^{a_v}f_{P_1}), l(^{a_m}f_{P_2}))$, if $(a_u = a_n) \wedge (a_v = a_m)$
- $Diff(st_1, st_2) = Max(length(st_1), length(st_2))$, i.e. $M(^{a_v}f_{P_1}, ^{a_n}f_{P_2}) = 0$ in other cases.

We prove that LD of two strings is equal to LD of their inverse strings (see Appendix A.1). So, whatever the edges (a_u, a_v) and (a_m, a_n) are labeled by $l(^{a_u}f_{P_1})$ or $l(^{a_v}f_{P_1})$ and $l(^{a_m}f_{P_2})$ or $l(^{a_n}f_{P_2})$, Equation 5.7 give a unique value.

Equation 5.7 to used compute the matching between two connection flows that connecting the same pair of services in two business processes. In case of neighborhood context matching, we apply Equation 5.7 to compute the matching between

Figure 5.10: Updated neighborhood context graphs for services a_x and a_6

connection flows that are located on the k^{th} -zone, $k > 1$. For the 1^{st} -zone, we compute the difference between connection flows, which connect to two associated services, as following.

Let $u_i \in \{a_u, a_v\}$, $u_x \in \{a_m, a_n\}$: u_i, u_x are two associated services. Let $u_j = \{a_u, a_v\} \setminus u_i$, $u_y = \{a_m, a_n\} \setminus u_x$. Let $st_1 = l_{(a_u, a_v)}^{a_u f_{P_1}}$, $st_2 = l_{(a_m, a_n)}^{a_m f_{P_2}}$, then:

$$Diff(st_1, st_2) = LD(l_{(a_u, a_v)}^{a_u f_{P_1}}, l_{(a_m, a_n)}^{a_m f_{P_2}}) \quad (5.8)$$

In the case that there is more than one connection flow between two services, we compute all possible matching between them and we select the best matching value.

To compute the neighborhood context matching between two services, we reuse the principles presented in section 5.4.1, 5.4.2 and 5.4.3. We consider two matching cases: matching in the first zone and in other zones. In the first zone, we match all connection flows that connect the two associated services to the same services in the 1^{st} -layer. In other zones, we match connection flows that are in the same zone and connect to the same services.

We also take into account the zone weight in the neighborhood context matching. We present in chapter 7 (section 7.3.2) our experiments in both cases: with and without zone weight consideration.

5.7 Similarity between connection elements

Connection elements serve as fundamental factors to specify execution constraints and dependencies between services. For example, a sequence specifies a causal relation, an AND-split specifies a parallel execution, an OR-split specifies a choice, etc. The connection elements can show common execution behavior such as sequential, concurrent, choice, etc. So, their behaviors are not totally different. For example, consider two services a_i and a_j that can be connected by: (1) a ‘sequence’ or (2) an ‘AND-split’. The two connection elements are different, however, their behavior is quite similar: a_j has to be executed and it is always executed after a_i .

In our previous model, we consider that the matching between two connection elements has only two values: 0 (in the case that they do not have the same type) and 1 (in the case that they have the same type). In this section, we propose a metric to compute the similarity between connection elements in terms of execution properties. This metric allows evaluating their similarity by a value between 0 and 1.

We firstly indicate the primitive rules that the similarity has to satisfy (section 5.7.1). Secondly, we present our proposition to compute the similarity between typical connection elements (section 5.7.2). Finally, we describe how to integrate the computed similarity into the neighborhood context matching (section 5.7.3).

5.7.1 Primitive rules

In our work, we deal with basic connection elements, which are *sequence*, *AND*, *OR* and *XOR* as they are commonly used in most of business processes. Other elements and complex gateways are not considered. However, the same principles expressed below can be applied.

We present our analysis on the ‘-split’ elements, including *AND-split*, *OR-split* and *XOR-split*. A ‘-split’ connection element consists of *one input and multiple output* flows. The similarities between ‘-join’ elements can be inferred with the same reasoning.

To compute the similarity between connection elements, we firstly specify three primitive rules that our approach satisfies, which are:

- ① If two connection elements are *identical*, their similarity is 1.
For example, similarity of two ‘sequence’ is 1; similarity of two ‘AND-split’ elements is 1 and so on.
- ② Similarity between two connection elements that *have the same types but different number of output flows* is 1. Similarity between two *different* connection elements is less than 1.
For example, similarity between two ‘AND-split’ elements that have different number of output flows is 1 whereas similarity between an ‘AND-split’ and an ‘OR-split’ is less than 1.

- ③ In the case that two connection elements are *different*, the less different the numbers of output flows are, the greater similarity value is.
- For example, consider the matching between an ‘AND-split’ that has 2 output flows and two ‘OR-split’ that have respectively (1) 2 output flows and (2) 3 output flows. The similarity value of the ‘AND-split’ and the first ‘OR-split’ must be higher than the similarity value of the ‘AND-split’ and the second ‘OR-split’.

5.7.2 Similarity computation

Connection elements indicate also the number of possible execution cases. The number of possible execution cases is impacted by the type of connection elements and the number of output flows derived by the connection elements. In our approach, we analyze the number of possible execution cases and compute the similarity between connection elements based on the probability that an output flow is executed.

Consider the case where two services a_i and a_j are connected by a connection element c . The connection flow from a_i to a_j is notated by $\frac{a_j}{a_i} f = c$. c can be a sequence, an ‘AND-split’, an ‘OR-split’ or an ‘XOR-split’.

We call a case that satisfies the constraints of a connection element a ‘*possible execution case*’ or a ‘*possible case*’ in short. For example, if c is a ‘sequence’, it has only one possible execution case, in which the service following c is executed; if c is an ‘AND-split’, it also has one possible execution case in which all services located on the output flows of c are executed; but if c is an ‘OR-split’, it has multiple execution cases: ‘at least one of the services located on output flows needs to be executed’.

Let x , y and z be the number of output flows that an ‘AND-split’, an ‘OR-split’ and an ‘XOR-split’ respectively have. The numbers of possible cases of these connection elements are given in the 4th column of Table 5.3. The last column of Table 5.3 presents the *probabilities that an output flow is executed*. We explain in the following how we compute these values.

For the ‘sequence’ and ‘AND-split’ cases, there is only one possible execution case, in which all services are executed. Hence, the probability that an output flow is executed is 1.

For the ‘OR-split’ case, the execution step is completed if at least one of y services in the y output flows is executed. The number of cases where at least one of y services is executed is $2^y - 1$. On the other hand, consider an output flow. The number of possible cases where this flow is ‘executed’ among 2^y possible execution cases is $\frac{2^y}{2} = 2^{y-1}$. So, the probability that an output flow is executed is $\frac{2^{y-1}}{2^y - 1}$.

For the ‘XOR-split’ case, for z output flows, there are z possible execution cases. Therefore, the probability that an output flow is executed is $\frac{1}{z}$.

To compute the similarity between these connection elements, we propose to combine the *weight* of an output flow and the probability that it is executed. The weight of an output flow is specified by the inverse number of output flows. For example,

Element	Presentation	No. paths	No. possible cases	Probability that an output flow is executed
Sequence		1	1	1
AND-split		x	1	1
OR-split		y	$2^y - 1$	$\frac{2^{y-1}}{2^y - 1}$
XOR-split		z	z	$\frac{1}{z}$

 Table 5.3: Probability that a_j appears in the possible cases

consider an ‘AND-split’ that derives 3 output flows. The weight of each output flow is $\frac{1}{3}$.

Consequently, the similarity between two connection elements c_u and c_v , denoted by $S(c_u, c_v)$, is given as following:

$$S(c_u, c_v) = w_{c_u}^* \times w_{c_v}^* \times P_{c_u}^+ \times P_{c_v}^+ \quad (5.9)$$

where $w^*(c_u)$, $w^*(c_v)$ are respectively weights of an output flow of c_u and c_v ; $P_{c_u}^+$ and $P_{c_v}^+$ are probabilities that an output flow of c_u and c_v is executed.

The similarities between aforementioned connection elements are given in Table 5.4. In Appendix A.3, we prove that the similarities computed by our approach satisfy the primitive rules.

Similarity	Sequence	AND-split(x)	OR-split(y)	XOR-split(z)
Sequence	1	$1 \times \frac{1}{x} \times 1 \times 1$	$1 \times \frac{1}{y} \times 1 \times \frac{2^{y-1}}{2^y - 1}$	$1 \times \frac{1}{z} \times 1 \times \frac{1}{z}$
AND-split(x)		1	$\frac{1}{x} \times \frac{1}{y} \times 1 \times \frac{2^{y-1}}{2^y - 1}$	$\frac{1}{x} \times \frac{1}{z} \times 1 \times \frac{1}{z}$
OR-split(y)			1	$\frac{1}{y} \times \frac{1}{z} \times \frac{2^{y-1}}{2^y - 1} \times \frac{1}{z}$
XOR-split(z)				1

Table 5.4: Similarities between typical connection elements

For example, with $x = 2$, $y = 3$, $z = 4$, $S(AND - split, sequence) = 0.5$, $S(OR - split, sequence) = 0.19$, $S(XOR - split, sequence) = 0.06$, $S(AND - split, OR -$

$split) = 0.10$, $S(AND-split, XOR-split) = 0.03$ and $S(OR-split, XOR-split) = 0.01$.

5.7.3 Integration into the neighborhood context matching

The neighborhood context matching is synthesized from the connection flow matching (section 5.4). A connection flow is a sequence of connection elements that connect two services (Definition 5.2.2). It can contain one or more connection elements.

We integrate the similarity between connection elements into the neighborhood context matching in the case where connection flows contain only one connection element. For example, consider two connection flows ${}_{a_x}^{a_y}f_{P_1}$ and ${}_{a_u}^{a_v}f_{P_2}$ of two neighborhood context graph $G_{P_1}^{a_i}$ and $G_{P_2}^{a_j}$. Assume that they are located on the same connection zone and connect the same ending services. Each of them contains only one connection element, which is c and c' respectively. Similarity between them is inferred by the similarity between the connection elements, which is computed by Equation 5.9.

In the case that connection flows contain more than one connection element, we transform them to strings of characters and reuse Levenshtein distance to compute their similarity (section 5.4.1).

As the similarity between connection elements is applied to compute the similarity of connection flows, it impacts on the neighborhood context matching. In our experiments, we run our different neighborhood context matching computations with and without the similarity between connection elements to identify its impact on the final results.

5.8 Conclusion

In this chapter, we answer the two questions raised in the thesis problematic (section 2.1.3), which are *How to recommend services for particular positions?* and *How to formally express constraints to filter services?*

To recommend services for a particular position, we take into account connection flows between services. We define the *neighborhood context* of a service as a process fragment that contains the associated service and relations to its neighbors. We capture the neighborhood context of a selected position and match it to existing neighborhood contexts to find relevant services. Our recommendations can be used in different cases: (1) a process analyst wants to *discover* suitable services for an ‘empty’ position or (2) to have recommendations to *extend* (or evolve) a designed process or (3) to *replace* an existing service.

To help process analysts formally express constraints to filter services, we propose a *query language*. Using this language, a process analyst can formally specify filtering constraints. These constraints are applied on the neighborhood context of the

returned services. For example, the returned services have to contain (or not contain) a given service (or a given connection flow) in their contexts, etc..

To capture more service relations, we examine *parallel flow relations* and the *similarity between connection elements*. We also present how to integrate these factors in the neighborhood context matching.

Our principles presented in section 2.3.1 are respected:

- *Focused and fine-grained results.* We recommend services instead of business processes. So, our recommendations do not make users confused.
- *No additional information.* We do not ask for any additional information. We make recommendations based on existing data, which are relations between services in business process models.
- *Exploiting implicit knowledge.* We exploit service neighborhood context which is implicit knowledge hidden in process models. This context is used to induce the service behavior within a process.
- *Balanced computational complexity.* We neither focus on a service nor consider entire business processes. Instead, we take into account neighborhood contexts which are business fragments around services. The computational complexity of our approach is polynomial. In addition, we do not face the NP-complete problem.

Capturing Neighborhood Contexts from Process Logs for Service Recommendation

Contents

6.1	Introduction	93
6.2	Running example	94
6.3	Exploiting neighborhood context from logs	95
6.3.1	Preliminaries	96
6.3.2	Log-based business process	96
6.3.3	Log-based neighborhood context	97
6.4	Log-based neighborhood context matching	98
6.4.1	First zone matching	99
6.4.2	Further zone matching	101
6.4.3	Matching with zone weight consideration	103
6.5	Service recommendation	103
6.6	Conclusion	104

6.1 Introduction

In this chapter, we present another approach for service recommendation. Instead of capturing the neighborhood contexts from existing business processes, we propose to discover the neighborhood contexts from business process logs. We study business process logs because of four reasons:

1. *They exist in all transactional information systems* such as ERP, CRM, or workflow management systems [39]. So, they are large resources that are always available to be discover.

2. *Business process models do not always exist.* For example, in case of the flow of patients in a hospital, all activities are logged but information about the underlying process is typically missing [39]. In this case, our previous techniques presented in the previous chapter cannot be applied. Therefore, we propose to use logs as an alternative input.
3. *They present the reality of the business process execution.* They include the service execution frequency, which is useful information to identify the importance of a service and is not presented by the a-priori process model.
4. *They contain useful information that can be discovered to assist the business process design and diagnosis.* For example, they can be mined to check the conformance of a-priori business process models [28, 93, 94, 95, 96], to detect execution errors [97, 98] or to observe social behaviors between users or services [99, 100].

By extracting business process logs, we capture the execution orders between web services. Then, we construct a log-based business process model from the captured execution orders. We define the service neighborhood context as a fragment of the log-based business process that contains the considered service and relations to its neighbors mined from the business process logs. Each relation between two web services is associated to a weight value which is the number of time that it occurs in the business process logs. We match neighborhood contexts to infer the similarity between services. This similarity is used for service recommendation to facilitate the business process design.

This chapter is organized as following: we firstly present a running example that is used for illustrating our approach (section 6.2). The log-based business process model and the neighborhood context graph extracted from logs are presented in section 6.3. In section 6.4, we elaborate our technique for service matching based on business process logs. Section 6.5 presents the recommendation strategy. Finally, we conclude the chapter in section 6.6.

6.2 Running example

We use an example about liability claim within an insurance company, which is presented in [155]. The liability claim process is described as following: first, some data related to the claim is registered (service A), and then either a full check or a policy-only check is performed (B or C). Afterwards, the claim will be evaluated (D), and then it is either rejected (F) or approved (E and G). Finally, the case is archived and closed (H).

The log traces collected from the execution of the liability claim process are given in Table 6.1. We assume that there is no error occurs during the business process execution. A process instance contains multiple services and a service can be executed by different process instances. Each process instance is recorded as a *trace*. Each

trace is a sequence of services, which presents the execution orders of the services involved by a process instance. In a trace, the following service (service on the right) is executed after the followed service (service on the left). For example, consider a trace $\sigma = ABCD$, service B is executed after service A and before service C.

Traces	Log traces
trace 1	ACDGEH
trace 2	ABDFH
trace 3	ABDEGH
trace 4	ABDFH
trace 5	ACDGEH
trace 6	ABDGEH
trace 7	ACDFH

Table 6.1: Example: event logs of the liability claim process

Figure 6.1 shows the liability claim process model which is discovered from the above event logs [155].

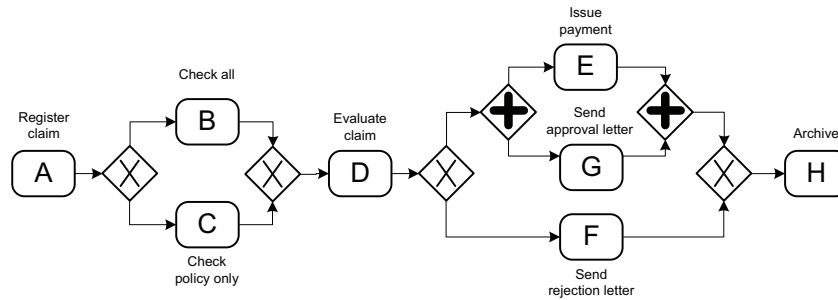


Figure 6.1: The liability claim business process discovered from event logs [155]

In the following sections, we present our approach to build a log-based business process model. Based on this model, we also build service neighborhood contexts and present a matching technique for service recommendation. We notice that the only input in our approach is business process log.

6.3 Exploiting neighborhood context from logs

In this section, we present the log-based business process and the neighborhood context that are captured from business process logs. We firstly present some definitions related to business process logs (section 6.3.1). Then, we present log-based business process (section 6.3.2) and service neighborhood context (section 6.3.3) definitions.

6.3.1 Preliminaries

We reuse definitions in [39] and in [156] for the business process log and log traces definitions. In our approach, as we take into account the service execution frequency, we add a definition about full business process log (see Definition 6.3.2).

Definition 6.3.1 (Log trace, business process log). *Let A be a set of services. A^* denotes the set of finite sequences over A and $\sigma = a_1a_2 \dots a_n \in A^*$ is a log trace. $L \in \mathcal{P}(A^*)$ is a business process log¹.*

A business log defined by Definition 6.3.1, which is proposed by [39, 156], does not include the number of times that a trace is executed. To include this number in the log, we define a full business process log (see Definition 6.3.2) as an original business process log without the elimination of similar traces.

Definition 6.3.2 (Full business process log). *A full business process log is the original business process log with the consideration of the trace frequency, i.e., number of times that traces are repeated in the log. The full business log is denoted by L^* , $L^* \in \mathcal{P}^*(A^*)$. $L \subseteq L^*$.*

For example, the *full* business process log of the liability claim process in the running example includes all traces given in Table 6.1 while the business process log (by Definition 6.3.1) includes only case 1, 2, 3, 5 and 7.

Definition 6.3.3 (Log-based ordering relation). *Let L be a business process log over A , i.e., $L \in \mathcal{P}(A^*)$. Let $a, b \in A$. $a >_L b$ iff $\exists \sigma = a_1a_2 \dots a_n$, $i \in \{1, 2, \dots, n-1\}$: $\sigma \in L \wedge a_i = a \wedge a_{i+1} = b$.*

6.3.2 Log-based business process

The sequence of services in a log trace $\sigma = a_1a_2 \dots a_n \in A^*$ present their ordering relations. A relation between a service a_i and its followed service a_{i+1} in the trace σ , $1 \leq i \leq n-1$ can be presented as a *directed edge* from a_i to a_{i+1} . The service relations in a business process log L can be presented in a weighted directed graph where the edge's weight presents the number of times that the edge was repeated in the log L . This graph is called *log-based business process graph* (Definition 6.3.4).

Definition 6.3.4 (Log-based business process graph). *A log business process graph is a weighted directed graph $G_L = (V_L, E_L, w)$ built from a business process log $L^* \in \mathcal{P}^*(A^*)$ where:*

- $V_L = A = \{a_1, a_2, \dots, a_n\}$,
- $E_L = \{(a_i, a_j) \in A \times A : a_i >_L a_j\} \subseteq A \times A$,

¹ $\mathcal{P}(A^*)$ is the power set of A^* , i.e., $L \subseteq A^*$

- w is a weight function from E_L to N :

$$\begin{aligned} w : E_L &\longrightarrow N \\ (a_i, a_j) &\mapsto |a_i >_L a_j| \end{aligned}$$

$|a_i >_L a_j|$ is number of times that $a_i >_L a_j$ comes about in the log L^* .

$w(a_i, a_j) = 0$ if $\nexists \sigma = a_1 a_2 \dots a_n$ and $k \in \{1, 2, \dots, n-1\} : a_k = a_i \wedge a_{k+1} = a_j$.

The log-based business process graph of the given running example is depicted in Figure 6.2. The flow's weight is the number of times that the flow is executed. It is emphasized by the path's thickness. The log-based graph indicates the frequency of each flow in real business execution. This information should be taken into account for recommendation because it can suggest the likelihood of this relation to happen again.

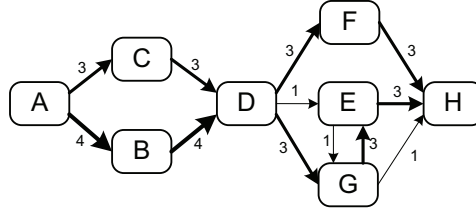


Figure 6.2: Log-based business process graph

6.3.3 Log-based neighborhood context

Inspired by the neighborhood context definition in the previous chapter (Definition 5.2.9), we define the log-based neighborhood context as a directed labeled graph that presents the shortest path from a service to its neighbors. Intuitively, the closeness between services is presented by the paths connecting them. The shortest path between services presents their closest relation. The log-based neighborhood context of a service presents the best relations between the service and its neighbors.

In a log-based neighborhood context graph, each vertex is associated to a number that indicates the shortest path length from it to the associated service. Vertexes that have the same shortest path length are considered to be located on the same layer around the associated service. Thus, we name the number associated to each service in a neighborhood context graph *layer number*. The layer number of a service a is denoted by $l(a)$. The area limited between two adjacent layers is called zone. The edge connecting two vertexes in a neighborhood context graph belongs to a zone as the vertexes are on the same or adjacent layers. We assign to each edge in the neighborhood context graph a number, so-called *zone number*, which determines the zone that the edge belongs to.

The edge connecting a_j, a_k in the neighborhood context graph of a service a_i is assigned a zone number $z(a_j, a_k) = \min(l(a_j), l(a_k)) + 1$. This means, if a_j and a_k

are located on two adjacent layers, the edge (a_j, a_k) will belong to the zone limited by $l(a_j)$ and $l(a_k)$. In the case that a_j and a_k are located on the same layer, the edge connecting them belongs to the outer zone of their layer, which is limited by layers $l(a_j)$ and $l(a_j) + 1$.

Definition 6.3.5 (Log-based neighborhood context graph). *A log-based neighborhood context graph of a service a_i , denoted by $G_C(a_i)$, is an extension of the log-based graph $G_L = (V_L, E_L, w)$ with vertex layer numbers and edge zone numbers. The layer number of an vertex a_j , denoted by $l(a_j)$, is the shortest path length from a_j to a_i and the zone number of an edge (a_j, a_k) , denoted by $z(a_j, a_k)$, has value $\min(l(a_j), l(a_k)) + 1$:*

1. $l(a_j) = \text{ShortestPathLength}(a_j, a_i)$,
2. $z(a_j, a_k) = \min(l(a_j), l(a_k)) + 1, a_j >_L a_k \vee a_k >_L a_j$.

The vertex layer numbers and edge zone numbers are different in different neighborhood context graphs. We denote $l(a_j)_{G_C(a_i)}$ the layer number of vertex a_j and $z(a_j, a_k)_{G_C(a_i)}$ the zone number of the edge (a_j, a_k) in the neighborhood context graph $G_C(a_i)$.

For example, the neighborhood context graph of activity D in Figure 6.2 is depicted in Figure 6.3.

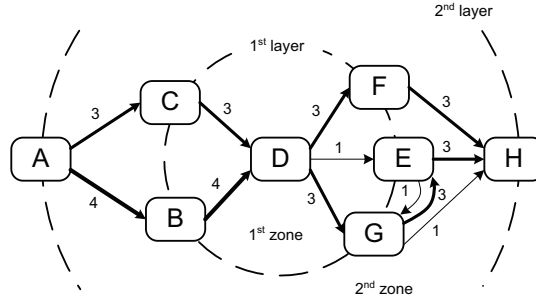


Figure 6.3: Example: neighborhood context graph

Definition 6.3.6 (k^{th} -neighbor). *a is the k^{th} -neighbor of b , iff $l(a)_{G_C(b)} = k$. Set of k^{th} -neighbors ($k \geq 1$) of a service a_i is denoted by $N^k(a_i)$. $N^k(a_i) = \{a_j : l(a_j)_{G_C(a_i)} = k\}$.*

6.4 Log-based neighborhood context matching

The layer number and the zone number in a neighborhood context graph present the closeness between services, while the weights of edges in the log-based graph present the strength of their relations. In this section, we detail our approach that is based on

log-based neighborhood contexts matching (including layer number and edge weight) for service recommendation.

Assume that P_p and P_q are two log-based business processes constructed from the event logs L_p and L_q . Let A_p, A_q be sets of services of P_p and P_q respectively. Similarity between services $a \in A_p$ and $b \in A_q$ is computed using vector space model (VSM). The computation is divided in two parts: first zone matching and further zone matching.

To illustrate our computation, we assume that there is another log-based process built from event logs of another business process as given in Figure 6.4.

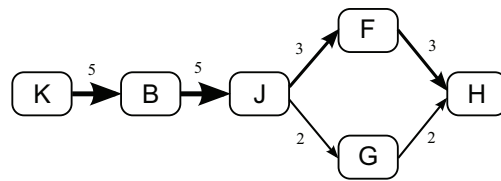


Figure 6.4: Log-based business process graph

We are going to compute the similarity between service D in the liability claim process (Figure 6.2) and service J in Figure 6.4. The neighborhood context graph of J is illustrated in Figure 6.5.

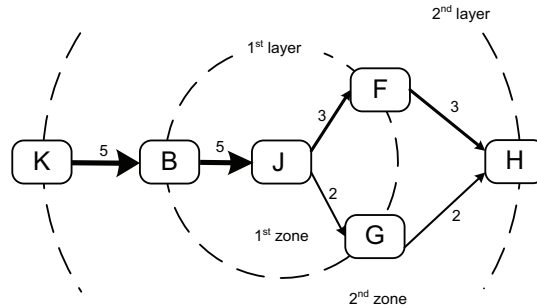


Figure 6.5: Neighborhood context graph

In the following we present the matching computation in the first zone and in the further zones using VSM.

6.4.1 First zone matching

The first zone matching computes the similarity between services' neighborhood contexts for the first zone.

Let $E_{P_p}^1(a), E_{P_q}^1(b)$ be the sets of edges connecting a in P_p and b in P_q in the first

zone respectively. Let $\overrightarrow{e(a)}$, $\overrightarrow{e(b)}$ be vectors of weights of these edges.

$$\begin{aligned}
E_{P_p}^1(a) &= \{(a, x) \cup (y, a) : x, y \in N^1(a) | a >_{L_p} x, y >_{L_p} a\} \\
&= \{(a, x_1), (a, x_2), \dots, (a, x_m), (y_1, a), (y_2, a), \dots, (y_n, a)\} \\
\overrightarrow{e(a)} &= (w(a, x_1), w(a, x_2), \dots, w(a, x_m), w(y_1, a), w(y_2, a), \dots, w(y_n, a)) \\
&= (u_1, u_2, \dots, u_{m+n}) \\
E_{P_q}^1(b) &= \{(b, e) \cup (f, b) : e, f \in N^1(b) | b >_{L_q} e, f >_{L_q} b\} \\
&= \{(b, e_1), (b, e_2), \dots, (b, e_d), (f_1, b), (f_2, b), \dots, (f_h, b)\} \\
\overrightarrow{e(b)} &= (w(b, e_1), w(b, e_2), \dots, w(b, e_d), w(f_1, b), w(f_2, b), \dots, w(f_h, b)) \\
&= (v_1, v_2, \dots, v_{d+h})
\end{aligned}$$

Let $N_c^1(a, b)$ be the set of common neighbors of a and b in the first layer.

$$N_c^1(a, b) = N_{P_p}^1(a) \cap N_{P_q}^1(b)$$

Let $E_c^1(a)$, $E_c^1(b)$ be the sets of edges that connect a and b to their common neighbors in the first layer respectively. Let $\overrightarrow{e_c(a)}$, $\overrightarrow{e_c(b)}$ be vectors of weights of these edges.

$$\begin{aligned}
E_c^1(a) &= \{(a, x') \cup (y', a) : x', y' \in N_c^1(a, b) | a >_{L_p} x', y' >_{L_p} a\} \\
&= \{(a, x'_1), (a, x'_2), \dots, (a, x'_t), (y'_1, a), (y'_2, a), \dots, (y'_z, a)\} \\
\overrightarrow{e_c(a)} &= (w(a, x'_1), w(a, x'_2), \dots, w(a, x'_t), w(y'_1, a), w(y'_2, a), \dots, w(y'_z, a)) \\
&= (a_1, a_2, \dots, a_{t+z}) \\
E_c^1(b) &= \{(b, e') \cup (f', b) : e', f' \in N_c^1(a, b) | b >_{L_q} e', f' >_{L_q} b\} \\
&= \{(b, e'_1), (b, e'_2), \dots, (b, e'_r), (f'_1, b), (f'_2, b), \dots, (f'_z, b)\} \\
\overrightarrow{e_c(b)} &= (w(b, e'_1), w(b, e'_2), \dots, w(b, e'_r), w(f'_1, b), w(f'_2, b), \dots, w(f'_z, b)) \\
&= (b_1, b_2, \dots, b_{t+z})
\end{aligned}$$

The similarity between a and b for the 1st - zone is computed by VSM (Equation 6.1).

$$M^1(a, b) = \frac{\sum_{i=1}^{t+z} a_i \times b_i}{\sqrt{\sum_{i=1}^{m+n} u_i^2} \times \sqrt{\sum_{i=1}^{d+h} v_i^2}} \quad (6.1)$$

For example, from Figure 6.2 and Figure 6.4, we have:

$$\begin{aligned}
E^1(D) &= \{(D, F), (D, E), (D, G), (C, D), (B, D)\} \\
\overrightarrow{e(D)} &= (3, 1, 3, 3, 4) \\
E^1(J) &= \{(J, F), (J, G), (B, J)\} \\
\overrightarrow{e(J)} &= (3, 2, 5)
\end{aligned}$$

Common neighbors of D and J in the 1st-layer are:

$$N_c^1(D, J) = \{B, F, G\}$$

Sets of edges connecting to the same neighbors are:

$$\begin{aligned}
E_c^1(a) &= \{(D, F), (D, G), (B, D)\} \\
E_c^1(a) &= \{(J, F), (J, G), (B, J)\}
\end{aligned}$$

Vectors of these edges are:

$$\begin{aligned}
\overrightarrow{e_c(D)} &= (3, 3, 4) \\
\overrightarrow{e_c(J)} &= (3, 2, 5)
\end{aligned}$$

So, the matching between D and J for the 1st-zone is:

$$M^1(D, J) = \frac{3 \times 3 + 3 \times 2 + 4 \times 5}{\sqrt{3^2 + 1^2 + 3^2 + 3^2 + 4^2} \times \sqrt{3^2 + 2^2 + 5^2}} = 0.86$$

6.4.2 Further zone matching

The further zone matching computes the similarity between services' neighborhood contexts for the k^{th} -zone ($k > 1$). To do so, we present edges that connect the same vertexes in the same layers of the two neighborhood context graphs in vectors. The item values in each vector are the edge weights in the log-based graphs.

Let $E_{P_p}^k(a)$ and $E_{P_q}^k(b)$ be the sets of edges in k^{th} -zone of a in P_p and b in P_q respectively. Let $\overrightarrow{e(a)}$, $\overrightarrow{e(b)}$ be vectors of weights of these edges.

$$\begin{aligned}
E_{P_p}^k(a) &= \{(x, y) : z(x, y) = k, x, y \in A_p\} \\
&= \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\} \\
\overrightarrow{e(a)} &= (w(x_1, y_1), w(x_2, y_2), \dots, w(x_m, y_m)) \\
&= (u_1, u_2, \dots, u_m) \\
E_{P_q}^k(b) &= \{(e, f) : z(e, f) = k, e, f \in A_q\} \\
&= \{(e_1, f_1), (e_2, f_2), \dots, (e_n, f_n)\} \\
\overrightarrow{e(b)} &= (w(e_1, f_1), w(e_2, f_2), \dots, w(e_n, f_n)) \\
&= (v_1, v_2, \dots, v_n)
\end{aligned}$$

Let $N_c^{k-1}(a, b)$ and $N_c^k(a, b)$ be the sets of common neighbors of a and b on layers $k-1$ and k , $k > 1$.

$$\begin{aligned} N_c^{k-1}(a, b) &= N_{P_p}^{k-1}(a) \cap N_{P_q}^{k-1}(b) \\ N_c^k(a, b) &= N_{P_p}^k(a) \cap N_{P_q}^k(b) \end{aligned}$$

Let E_c^k be the set of common edges of a and b in k^{th} -zone.

$$\begin{aligned} E_c^k &= \{(r, t) : (r \in N_c^{k-1}(a, b), t \in N_c^k(a, b) | r >_{L_p} t \wedge r >_{L_q} t) \\ &\quad \cup (r \in N_c^k(a, b), t \in N_c^{k-1}(a, b) | r >_{L_p} t \wedge r >_{L_q} t)\} \\ &= \{(r_1, t_1), (r_2, t_2), \dots, (r_m, t_m)\} \end{aligned}$$

Let $\overrightarrow{e_c(a)}$, $\overrightarrow{e_c(b)}$ be vectors of weights of these common edges.

$$\begin{aligned} \overrightarrow{e_c(a)} &= (w(r_1, t_1), w(r_2, t_2), \dots, w(r_z, t_z)), (r_i, t_i) \in E_L(A_p), 1 \leq i \leq z) \\ &= (a_1, a_2, \dots, a_z) \\ \overrightarrow{e_c(b)} &= (w(r_1, t_1), w(r_2, t_2), \dots, w(r_z, t_z)), (r_i, t_i) \in E_L(A_q), 1 \leq i \leq z) \\ &= (b_1, b_2, \dots, a_z) \end{aligned}$$

The similarity between a and b for the k^{th} zone is given by VSM (Equation. 6.2).

$$M^k(a, b) = \frac{\sum_{i=1}^z a_i \times b_i}{\sqrt{\sum_{i=1}^m u_i^2} \times \sqrt{\sum_{i=1}^n v_i^2}} \quad (6.2)$$

For example, the matching between D and J in the second zone is computed as following:

$$\begin{aligned} E^2(D) &= \{(A, C), (A, B), (E, H), (F, H), (G, H), (E, G), (G, E)\} \\ \overrightarrow{e(D)} &= (3, 4, 3, 3, 1, 1, 3) \\ E^2(J) &= \{(K, B), (F, H), (G, H)\} \\ \overrightarrow{e(J)} &= (5, 3, 2) \end{aligned}$$

$$\begin{aligned} N_c^1(D, J) &= \{B, F, G\} \\ N_c^2(D, J) &= \{H\} \end{aligned}$$

$$E_c^2 = \{(F, H), (G, H)\}$$

$$\begin{aligned}\overrightarrow{e_c(D)} &= (3, 1) \\ \overrightarrow{e_c(J)} &= (3, 2)\end{aligned}$$

Then, similarity between D and J for the second zone is:

$$M^2(D, J) = \frac{3 \times 3 + 1 \times 2}{\sqrt{3^2 + 4^2 + 3^2 + 3^2 + 1^2 + 1^2 + 3^2} \times \sqrt{5^2 + 3^2 + 2^2}} = 0.24$$

6.4.3 Matching with zone weight consideration

The behavior of a service is strongly reflected by the connections to its closet neighbors while the interactions among other neighbors in the further layers do not heavily reflect its behavior. Therefore, to improve the final matching precision, we propose to consider a zone weight in our matching. Concretely, as the zone-weight has to have greater values in smaller k^{th} connection zone, we propose to assign the zone-weight a value computed by a polynomial function which is $w_j^z = \frac{k+1-j}{k}$, where j is the zone number ($1 \leq j \leq k$), k is the number of considered zones around an associated service. The final matching formula improved with the zone weight consideration is given in Equation 6.3.

$$M^*(a, b) = \frac{2}{k+1} \times \sum_{i=1}^k \frac{k+1-i}{k} \times M^i(a, b) \quad (6.3)$$

For example, the matching between the neighborhood contexts of D and J with zone weight consideration is:

$$M^*(D, J) = \frac{2}{3} \times (M^1(D, J) + \frac{1}{2} \times M^2(D, J)) = \frac{2}{3} \times (0.86 + \frac{1}{2} \times 0.24) = 0.65$$

6.5 Service recommendation

The neighborhood context graph presents the interactions between the associated service and its neighbors in layers. It can infer the behavior of the associated service. Therefore, the matching between neighborhood context graphs exposes the similarity between the associated services. In our approach, the higher the matching value is, the more similar the services are. Basically, the steps to make recommendations based on log-based neighborhood context matching are:

1. We represent the business execution logs in a log-based graph. This graph contains relations between services and their weight values.
2. For each service in the log-based graph, we build a neighborhood context graph which contains the closet relations between the associated service and its neighbors. In a neighborhood context graph, services are presented in layers and relations between them are presented in zones.

3. We compute the neighborhood context graph matching using vector space model. The matching between two neighborhood context graphs presents the similar between two corresponding services in terms of relations to their neighbors.
4. For a selected service, we sort other services in descending order of similarity and pick up top- n services for recommendation.

6.6 Conclusion

In this chapter, we answer the second last question raised in the thesis problematic (section 2.1.3), which is *Can business process logs be useful? and How?*. We show that business process logs can be useful to make service recommendations. To do so, we capture execution orders between services and the ‘importance’ of the connection flows connecting them. This ‘importance’ is reflected by the number of time that the connection flow is repeated in the business process logs. We build business process models and neighborhood context graphs of services based on business process graphs. We match neighborhood context graphs to infer the similarity between corresponding services. This similarity is used for service recommendation during the design phase. Concretely, for each selected service in an on-going designed process, we recommend a list of services that are similar in terms of neighborhood context using only business process logs.

Our principles (section 2.3.1) are also respected in this chapter:

- *Focused and fine-grained results.* We recommend services instead of business processes.
- *No additional information.* We do not ask users any further information or resources to make recommendations.
- *Exploiting implicit knowledge.* We exploit log-based neighborhood contexts which is implicit knowledge hidden in business process logs.
- *Balanced computational complexity.* The computational complexity of our approach is polynomial and we do not face the NP-complete problem.

Our approach is self-contained and it can be associated to other recommendation approaches such as functionality based, user’s interest based, service-rating based, etc. to improve recommendation quality.

Implementation and Experiments

Contents

7.1	Introduction	105
7.2	Implementation	107
7.2.1	Application for individual use	107
7.2.2	Applications for business process use	108
7.2.2.1	PRec	109
7.2.2.2	WebRec	111
7.2.2.3	LogRec	112
7.2.3	Synthesis	114
7.3	Experiments	114
7.3.1	Individual use Experiments	115
7.3.1.1	Experiments on the dataset collected by IRec	115
7.3.1.2	Experiments on a large public dataset	117
7.3.1.3	Synthesis	119
7.3.2	Process use Experiments	119
7.3.2.1	Approach feasibility and parameter impact	120
7.3.2.2	Algorithms accuracy	122
7.3.2.3	Algorithms performance	125
7.3.2.4	Synthesis	126
7.4	Conclusion	126

7.1 Introduction

In this chapter, we present the *implementation* we have done to realize our recommendation techniques, and the *experiments* we have made to evaluate the effectiveness of our solutions. Our goal is to prove that our approach is *feasible* and *accurate* in *real*

use-cases. Besides, we analyze the *parameters* that impact on the *recommendation quality*.

We implemented four applications that recommend relevant services in the two consumption cases: *individual use* and *process use*.

- *The first application*, named IRec, is a web-based application. It allows users to search for services, select a service for *individual use*, and get recommendations. In this application, we collected the user's usage data and implemented the algorithms presented in chapter 4 to make recommendations.
- *The second application*, named PRec, demonstrates our recommendation techniques for *process use*. It allows users to add/remove services, create business process and get recommendations for a selected service. In this application, we implemented the algorithm proposed in chapter 5 (without considering the parallel flow relation and the similarity between connection elements). We made recommendations based on the existing designed business processes.
- *The third application*, named WebRec, is another web-based application. It makes recommendations for business process use. It is developed based on Signavio, which is a public platform for BPMN-based business process design. It implements the algorithm proposed in chapter 5. This application considers the parallel flow relation and the similarity between connection elements.
- *The forth application*, named LogRec, demonstrates the service recommendations from business process logs. It implements the algorithm proposed in chapter 6. It is developed on ProM, an open-source framework for implementing process mining tools.

We perform experiments on the usage data collected by our applications. We also tested our algorithms on public datasets.

- For *individual use*, we evaluated the effectiveness of our algorithms based on the *search data* and the user's *last usage data*. We used as metrics for the evaluation *Precision* and *Recall*. We also tested our algorithms on AudioScrobbler, which is a large public dataset that contains similar behaviors to the service usage data. We used as metric for this experiment the *Root Means Square Error* (RMSE).
- For *process use*, we provided *statistics* on the number of recommended services and the context matching values. We tested our algorithms on a large public dataset of real business processes. This dataset is shared by the Business Integration Technologies (BIT) research group at the IBM Zurich Research Laboratory. We computed Precision and Recall values. We also measured the computation time and usage memory of our algorithms to evaluate their performance.

The results show that our *approach are feasible and accurate*. They can be deployed as standalone applications or integrated in the existing business process management applications to facilitate service consumption.

We present in section 7.2 our four applications and basic use cases. The experiments and related discussion are presented in section 7.3. Finally, we conclude the chapter in section 7.4.

7.2 Implementation

In this section, we present the applications that we have developed to realize our approach. We firstly present IRec, an application that recommends web services for *individual use* (section 7.2.1). Second, we present three applications that recommend services for *process use* (section 7.2.2). These applications include PRec, a standalone java-based application, WebRec, a web-based business process design application and LogRec, an application that uses business process logs to make recommendations.

7.2.1 Application for individual use

The architecture of IRec is shown in Figure 7.1. It consists of 7 components: a search engine, an invocation engine, a recommender component, an evaluation component, a database management component, a data manipulation component and a web service collector and checker component. *The search engine* allows users to search for services. *The invocation engine* provides interface for service invocation. It forwards the usage data to the database management component. *The recommender component* implemented the proposed algorithms to recommend users suitable services to their interests. *The data management component* provides functions to interact with the database system. *The service collector and checker component* collects web service descriptions from different sources such as providers, crawlers, search engines and other systems. It checks web services availability and the correctness of their descriptions before sending them to the data management component. *The data preparation component* pre-processes the usage data to facilitate the recommendation step. It also periodically updates the similarity between web services. Finally, *the evaluation component* evaluates the recommendation algorithms. It helps to identify the user's interest to improve the recommendation quality.

Based on this architecture, we developed IRec as a web-based server-client application. It allows users to register, create their profiles and use web services. It also provides an interface to upload web service descriptions. It implements the algorithms presented in chapter 4. It includes two parts: front-end and back-end. The front-end processes data at the user-side. It provides interfaces to interact with users. It is written in Flex¹, which is a programming language developed by Adobe. The back-end processes data at the server-side. It is written in Java. The mechanisms used to exchange data between front-end and back-end are RemoteObject and BlazeDS framework². Tomcat server 5.5 was set up for hosting the application.

¹<http://www.adobe.com/devnet/flex/>

²<http://sourceforge.net/adobe/blazeds/>

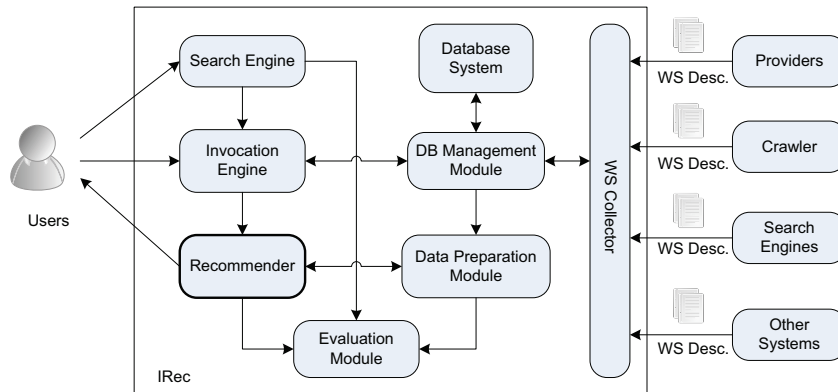


Figure 7.1: Architecture of our service recommendation tool (IRec) based on usage data

The screen-shot of IRec is shown in Figure 7.2. Its layout consists of 5 areas, in which the top area provides a menu and a search interface; the center area provides the search results or the service invocation interface; the left, right and bottom areas provide lists of recommended services according to the proposed algorithms. Our tool can be found at <http://www-inf.it-sudparis.eu/SIMBAD/tools/IRec/> and the user guide is available at <http://www-inf.it-sudparis.eu/SIMBAD/tools/IRec/tutorial.html>.

A basic scenario to get recommendations is as follows:

1. A user firstly searches for services in which she is interested. She types keywords and click on *Get services* button.
2. Our application returns a list of services that match to the input keywords.
3. The user now selects a service. We assume that the selected service is the one in which the user is interested.
4. Our application generates recommendations based on the user's ID and the selected service's ID using the the five algorithms presented in chapter 4.

7.2.2 Applications for business process use

We present in the following our three developed applications to make recommendations for process use. The first application, named PRec (section 7.2.2.1), allows users to design business process and get recommendation from our approach. The second application, named WebRec (section 7.2.2.2), was developed as a plugin of a business process design platform. The third application, named LogRec (section 7.2.2.3), makes recommendations based on process event logs. These applications implemented the different algorithms presented in chapters 5 and 6.



Figure 7.2: A screen-shot from IRec

7.2.2.1 PRec

PRec is developed to show how our approach helps to facilitate the business process design. It allows process analysts to design new business processes, select a service and get recommendations from our approach. It also allows managing services and processes. It is written in Java and published at <http://www.inf.it-sudparis.eu/SIMBAD/tools/PRec/>³.

A screenshot of the application is shown in Figure 7.3. It includes three tabs, named *Composition design*, *Composition* and *Service*. The ‘Composition’ tab provides functions to manage business processes. It allows business analysts to create/delete/modify a business process. The ‘Composition design’ tab provides functions to design a process. It allows creating/deleting connection flows between services in a selected process. The ‘Service’ tab (Figure 7.3) provides functions to manipulate services. It allows selecting services for the process design (area 2). It also allows adding/deleting/modifying a service (area 3). It provides recommendations for a selected service (area 4) and allows configuring parameters for the recommendation (area 5).

The basic scenario to get recommendations is as follows:

1. A process analyst firstly selects the *Composition* tab and creates a new business process.
2. A new process contains no services. Therefore, the process analyst clicks on the *Service* tab and select services for the new business process design from the list of available services.

³A test version, where users are free to use its functions and select algorithms to run, is published at <http://www-inf.it-sudparis.eu/SIMBAD/tools/PRec/test/>



Figure 7.3: PRec, a service recommendation application for process use

3. If the needed services are not available in the list, the process analyst can add them as new services.
4. After adding necessary services to the new business process, the process analyst clicks on the *Composition design* tab to start designing the process. She can select services and define the connection flows⁴ between them.
5. Finally, the process analyst can back to the *Service* tab and click on one of selected services to see the recommendations. The number beside the recommended service's name is the neighborhood context matching between it and the selected service. The process analyst can adjust the k parameter, which corresponds to the number of zones that are taken into account for the neighborhood context matching.

This tool does not take into account the parallel flow relations between services and the similarity between connection elements. These parameters are considered in the next application (section 7.2.2.2).

⁴There are 9 connection elements, including sequence, parallel split, synchronization, exclusive choice, simple merge, multi choice, structured synchronizing merge, deferred choice and cancel task.

7.2.2.2 WebRec

Our first application for process use, PRec, is not a perfect user-friendly to use because the design and recommendation functions are text-based. So, to improve the usability of our implementation, we develop WebRec, an application that provides a graphical user interface for business process design. WebRec is developed based on Signavio⁵.

Signavio is a platform developed for business process design. It provides a web-based graphical interface to design business processes. It uses BPMN notations. It has two versions: commercial and open source. The open source version with limited features is published⁶ for free downloading and testing.

By developing our approach based on Signavio, we achieve two targets: (1) we make our approach more *user-friendly* through the graphical suite and (2) we *widen* the *user community* and make our approach more *visible* as Signavio is widely known in the community. Our tool is published at <http://www-inf.it-sudparis.eu/SIMBAD/tools/WebRec/>.

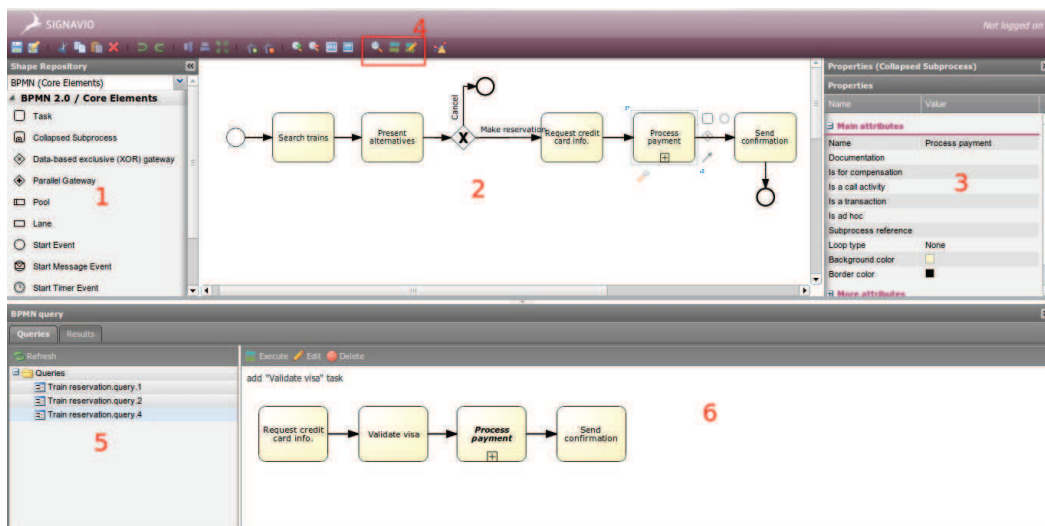


Figure 7.4: Querying web services for process design using WebRec

In this application, a process analyst can design and store business processes. During the design, she can select a service and activate the recommendation feature. She needs to specify the number of layers/zones needed to be taken into account and an algorithm to be executed to make recommendations. We consider each selected service and relations to other services within the selected zone as a query. The queries and their results can be saved and reloaded for a future use.

A screen-shot of the application is shown in Figure. 7.4. The areas 1, 2 and 3 show

⁵<http://www.signavio.com/>

⁶<http://code.google.com/p/signavio-core-components/>

the BPMN elements for design, canvas and property of the selected element. They are provided by the Signavio platform. We developed areas 4, 5, and 6. Area 4 contains the buttons for activating the context matching and query, area 5 shows the list of previous queries and area 6 can show both the query design and the recommendation results.

The recommendation feature of this application runs the neighborhood context matching algorithm (chapter 5) that takes into account the parallel flow relations and the similarity between connection elements. Whenever a user selects a service, the application provides a list of recommended services together with the corresponding similarity values computed by the neighborhood context matching.

A basic scenario⁷ is as follows:

1. A business process analyst opens the application and designs a new process. She can also open an existing process for editing⁸.
2. During the design, the analyst can select a service and activate the recommendation feature. She is free to select one of the four algorithms to make recommendations. The four algorithms take into account the parallel flow relations between services (chapter 5). They correspond to the four cases: with/without zone weight consideration and with/without similarity between connection elements.
3. She also selects the number of layers (or zones) needed to be taken into account for the neighborhood context matching. A query is generated. The designer can save, modify or execute it (a saved query can be loaded and modified in the next usage).
4. After executing the query, she gets a list of relevant services recommended by the application.
5. If she selects a service from the recommendation list, the application will return the corresponding process in which the selected service is highlighted.
6. She can copy services from the recommended process and paste them to the ongoing designed process.

7.2.2.3 LogRec

Our last application, LogRec, was developed to make recommendations based on process event logs. It implements the algorithm presented in chapter 6. It is developed based on ProM⁹, which is a well-known open-source framework for implementing

⁷Tutorial at: <http://www-inf.it-sudparis.eu/SIMBAD/tools/WebRec/tutorial.html> and video demo at: <http://www-inf.it-sudparis.eu/SIMBAD/tools/WebRec/demo.html>

⁸As Signavio uses BPMN notations, we consider each task as a service and the task's name as its identifier.

⁹<http://www.promtools.org/prom6/>

process mining tools¹⁰. Source codes and tutorial of our LogRec application are published at <http://www-inf.it-sudparis.eu/SIMBAD/tools/LogRec/>. Our objective is twofold: (1) we validate our approach using a *proof of concept* to show the feasibility of our approach and (2) we implement a tool, which is a plug-in within ProM, to make our approach more *visible* and *widely used* by the community.

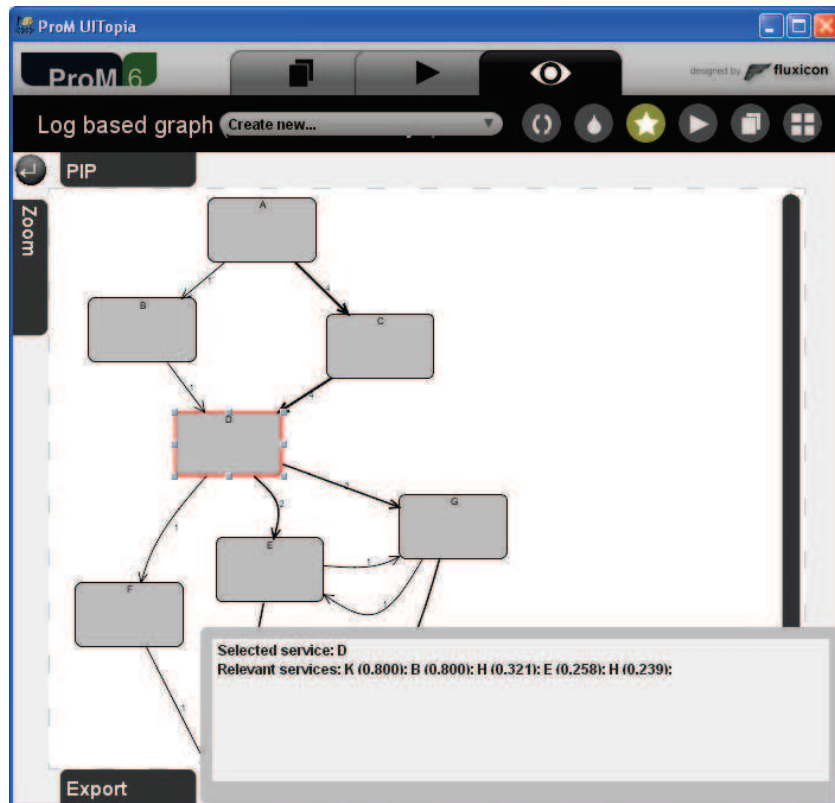


Figure 7.5: Business process extracted from logs and the related recommendations

A screen-shot of the application is shown in Figure 7.5. The displayed graph is built based on the relations between services extracted from event logs. Each edge connecting two services is associated to a number which presents the number of times that the edge is repeated in the logs. This number (corresponds to the thickness of the edge) reflects the importance of the relation between two services as it shows the execution frequency of that relation. During the process design, if a business analyst selects a service and clicks on the *Recommendation* button, a list of recommended services to that selected position will be shown (Figure 7.5).

A basic scenario to get recommendations is as follows:

¹⁰ProM has been developed by the Process Mining group (<http://www.processmining.org>)

1. A business analyst firstly runs the ProM application, open the *Workspace* tab and import the log file to be used.
2. After importing the log file in ProM, the business analyst opens the *Action* tab, selects the *Mine log-based graph* action and clicks *Start* to execute it.
3. The application will process log and present it as a graph with edge weights.
4. The business analyst now is able to select a service (presented in rectangle shape) and press the *Recommendation* button to get the recommendations based on the log-based neighborhood context matching presented in chapter 6.

7.2.3 Synthesis

In our work, we implemented 4 applications: IRec, PRec, WebRec and LogRec.

- IRec (section 7.2.1) was developed as a web-based application to collect service usage data for individual use. Based on these data, we will perform experiments in the following section to evaluate the accuracy of our approach. IRec implements our algorithms presented in chapter 4.
- PRec (section 7.2.2.1) serves for process use service consumption. It allows managing services, designing processes and viewing recommendations. However it is not user-frequently. It implements the neighborhood context matching algorithm without considering the parallel flow relations and the similarity between connection elements.
- WebRec (section 7.2.2.2) was developed as a user-frequently application with graphical user interface to design business processes. It was developed based on Signavio platform to *widen* the *user community* and make our approach more *visible* as Signavio is widely known in the community. It implements our algorithms by considering the parallel flow relations and the similarity between connection elements.
- LogRec (section 7.2.2.3) was developed to validate our approach using a *proof of concept*. It runs our log-based algorithm presented in chapter 6.

The four applications were implemented to prove that our approach is *feasible*. It can be flexibly developed as a standalone application or integrated into other applications to recommend services for *individual use* and *process use*.

7.3 Experiments

In this section, we present the experiments that we performed to evaluate our approach. We firstly present experiments about *individual use* service consumption (section 7.3.1). We run the five algorithms presented in chapter 4 on the data collected by IRec, our first application. We evaluate their performance using *Precision* and *Recall* metrics. We also perform experiments with LSI-based and service-user combination-based algorithms on a large dataset.

Second, we present experiments about process use service consumption (section 7.3.2). We run the algorithms presented in chapter 5 on a public dataset. This dataset consists of real business processes designed for financial services, telecommunications and other domains. We evaluate the accuracy of our algorithms using Precision and Recall. We also present the computation time of our algorithms.

7.3.1 Individual use Experiments

We performed our experiments on two different datasets: the dataset collected by IRec, our first application (section 7.3.1.1) and AudioScrobbler, a large public dataset (section 7.3.1.2). Our objective is to show that our algorithms can be used to *widen the view* of users and they can produce high quality recommendations in the case that users have *stable behavior*. Moreover, we target to show that our algorithms are of *good performance* on a large-scale system.

7.3.1.1 Experiments on the dataset collected by IRec

We used *Precision* and *Recall* metrics to measure the accuracy of our algorithms. Precision and Recall (and often associated F-measure) are two popular metrics to evaluate the accuracy of an information retrieval system [157]. They are computed based on the matching between data retrieved by the system and relevant (or ground-truth) data. Precision is equal to 1 if all retrieved data belong to the relevant set. Recall is equal to 1 if all relevant data are retrieved by the system. Precision and Recall are computed by Equation 7.1.

$$\begin{aligned} \text{Precision} &= \frac{\text{Retrieved data} \cap \text{Relevant data}}{\text{Retrieved data}} \\ \text{Recall} &= \frac{\text{Retrieved data} \cap \text{Relevant data}}{\text{Relevant data}} \end{aligned} \quad (7.1)$$

In our approach, we identify two relevant sets, which are used as ground-truth data, to compute Precision and Recall:

- The first set is *the most-used services returned by our search engine*, which is a traditional query-string search engine. Whenever a user searches for a service, we capture the most-used services in the search result. Whenever she selects a service, she gets recommendations from our application. We match the recommended services with the services that we captured from her last search to compute the Precision and Recall. By using this set, we target to compare the services recommended by our algorithms with the services returned by our search engine. We do not target to replace a search engine by our tool. Instead, we aim at evaluating how far (or how close) our recommendations and search results are.

- The second set is *the user's last used services*. Whenever a user selects a service, we match the services recommended by our algorithms with the user's last used services. By using this set, we target to detect the relation between user's behavior and recommendation quality. We measure the recommendation quality in two cases: (1) a user whose behavior changes frequently and (2) a user who has stable behavior.

During two weeks, our IRec application collected 271 iterations. Most of them are performed by invited PhD students and researchers. The relevant data were set as the 10 most-used services returned by the search engine and the 10 last-used services of each user. We compute the average Precision and Recall values of each algorithm. Table 7.1 shows the results based on the two relevant sets.

	Search based		Last-used based	
	Precision	Recall	Precision	Recall
Service-based	0.107	0.521	0.351	0.704
User-based	0.206	0.623	0.27	0.382
Service-user combination	0.093	0.346	0.296	0.577
LSI-based	0.118	0.54	0.333	0.689
Power assignment	0.184	0.109	0.075	0.069

Table 7.1: Experiments with the two relevant sets

Experiments on first relevant data set show that recommendations made by our algorithms are not “too close” (low Precision values) and not “too far” (high Recall values) from the results returned by a query string search engine. On the one hand, it means that our algorithms and the query-string based solution are not identical and our approach could be a good solution along with the query-string based search approach to *widen the user's view* and give him interesting web services that the classical query-based approach could not give. On the other hand, the services returned by our recommendation algorithms should not be too far from the query-based results to avoid incoherent services and to be quite close to the search context to replace the query-based approach in the case that it fails.

They also show that the user-based algorithm has the highest Precision and Recall values. Indeed, the usage data is collected from the usage of PhD students who have somehow similar behavior. Hence, the user-based algorithm can return good results and becomes the most suitable algorithm for this context. It is obvious that the power assignment algorithm has very low Precision and Recall because its objective is to recommend the least used services instead of services that are close to user interest.

Experiments with the last usage-based relevant data show that the algorithms which take into account the relations of all services (service-base and LSI-based) achieved the best results. The user-based algorithm and service-user combination al-

gorithm make recommendations based on the usage data of selected users, hence, they can easily miss the potential services which can make the evaluation more accurate.

Figure 7.6 shows the synthesized Precision and Recall values computed by the second evaluation method for particular users. If a user (for instance User ID=21 in Figure 7.6a) changes his behavior frequently, the recommendations generated by our algorithms may not fit to his interest. This causes the low and unstable Precision and Recall values when we run the second evaluation method, which is based on the user’s usage data. In contrary, if a user (for instance User ID=24 in Figure 7.6b) keeps or slightly changes his interest, our recommendations are of higher quality.

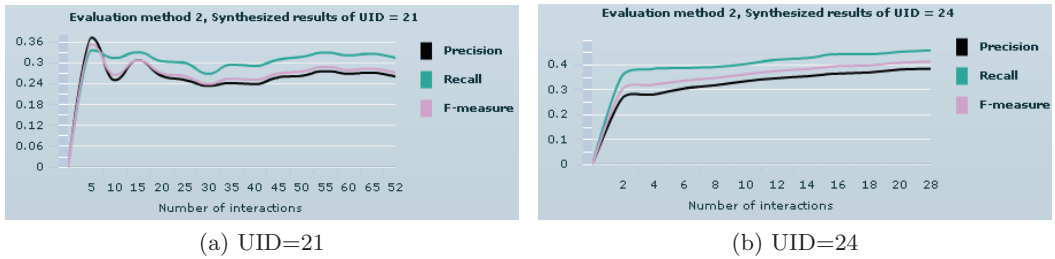


Figure 7.6: Synthesized results of particular users

7.3.1.2 Experiments on a large public dataset

We used AudioScrobbler¹¹, which is a public dataset, for this experiment. This dataset records the musical usage. We analyzed that the behavior of users in this dataset is similar to the behavior of users in using a web service because in both cases, users interact with items in which they are interested. In addition, this dataset contains records that correspond to user’s IDs, service’s IDs and the number of times that a user uses a web service. Therefore, it is suitable to evaluate our algorithms.

As the provided dataset does not include the search results or user’s last-user items, we can not process the evaluation with Precision and Recall metrics like we did with the data collected by our application. We decided to use Root Mean Square Error (RMSE) [52] metric, which is a metric to evaluate the performance of prediction system.

RMSE is used to compare the predicted data to existing ground-truth data. It is used in the Netflix Prize¹² Contest as the significant metric to evaluate the performance of participated recommendation algorithms [158, 159]. Small RMSE values indicate good prediction. Basically, assume that $P_{[m \times n]}$ is the prediction matrix and $A_{[m \times n]}$ is the ground-truth matrix. Let $I_{[m \times n]} \in \{0, 1\}_{[m \times n]}$ be the indicator of A .

¹¹<http://www.audioscrobbler.net/development/>

¹²<http://www.netflixprize.com/>

The RMSE between the prediction P and the answer A is computed by Equation 7.2.

$$RMSE(P, A) = \sqrt{\frac{\sum_{i=1}^m \sum_{j=1}^n I_{ij} (A_{ij} - P_{ij})^2}{\sum_{i=1}^m \sum_{j=1}^n I_{ij}}} \quad (7.2)$$

To evaluate our algorithms using this measure, we divided the AudioScrobbler dataset, which consists of 1033 user's IDs, 3435 artist's IDs, 644.281 records and 12.332.214 hits, into two parts: the training set, which consists of 4/5 the number of records, and the test set, which contains the rest 1/5 number of records. We run our algorithms on the training set and captured recommendations generated by our algorithms. We compared these recommendations (corresponding to the P matrix) to the test set (corresponding to the A matrix) to compute the RMSE. We performed experiments using two algorithms of two CF different types: our *service-user combination algorithm* for the memory-based CF, and our *LSI-based algorithm* for model-based CF.

We run our algorithms with different k parameter values. In the service-user combination algorithm, k is the number of the selected users and in the LSI-based algorithm, k is the number of dimensions reduced by the singular value decomposition (see more details in section 4.4.3 and 4.4.4).

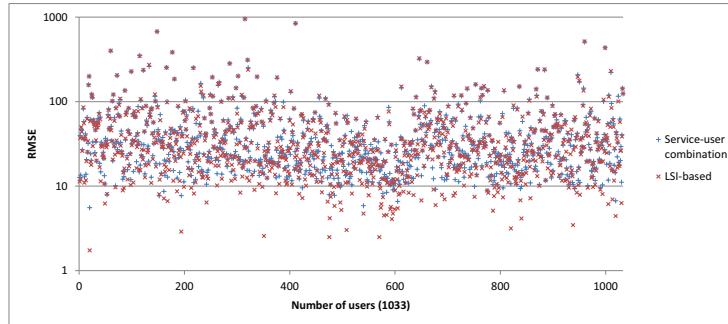


Figure 7.7: RMSE distribution with $k = 100$

In our experiments, we set $k = \{1, 2, 5, 10, 20, 50, 100\}$. Figure 7.7 shows the distribution of RMSE values computed by the service-user combination and LSI-based algorithms with $k = 100$ for 1033 users. The average and minimum RMSE values with different k are shown in Figure 7.8. These results show that in most cases, the LSI-based algorithm performs better than the service-user combination algorithm in a large scale system. They also show that RMSE values decrease when k increases and the service-user combination algorithm recommends services that are more accurate to user need. It is because when k increases that the number of selected users (in the service-user combination algorithm) and the number of dimensions (in the LSI-based algorithm) increases, useful data for recommendations are larger and our algorithms make better recommendations.

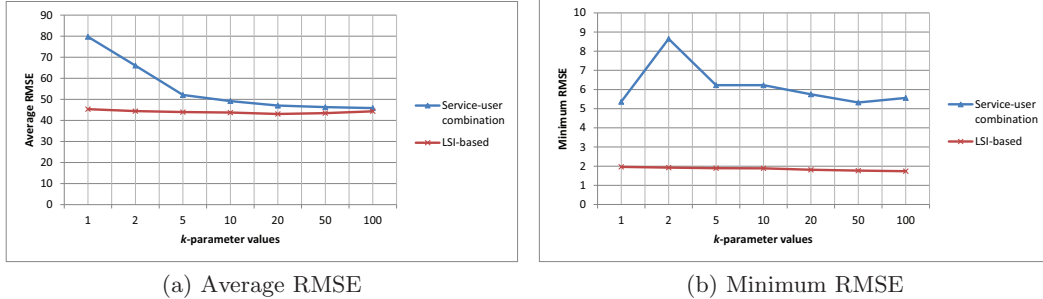


Figure 7.8: Experiment with different k -parameter values

7.3.1.3 Synthesis

Experiments on the first search data showed that the recommendations made by our algorithms can help to *widen the user's view* and give him interesting web services that the classical query-based approach could not give. They also showed that the user-based algorithm achieved the best results in the case that users have similar behaviors.

Experiments on the last-used data showed that the algorithms that take into account the relations of all services (service-based and LSI-based) achieved the best results. They also showed that our recommendations were *accurate* in the case that users had *stable behavior*.

Experiments on a large dataset showed that our algorithms were of *good-performance* in a *large scale system* with quite small RMSE values. They also showed that the *LSI-based* algorithm achieved *better* results than the *service-user combination* algorithm because the LSI-based algorithm overcomes the sparsity problem of the service-user combination algorithm (see section 4.4.3 for details), especially when the number of users and services are very large.

7.3.2 Process use Experiments

We present in the following our experiments for the neighborhood context matching algorithms (presented in chapter 5) using a large public dataset. This dataset is shared by the Business Integration Technologies (BIT) research group¹³ at the IBM Zurich Research Laboratory. It was presented in [160]. It contains business process models designed for financial services, telecommunications, and other domains. It is presented in XML format following BPMN 2.0 standard. Each XML file stores the data of a business process, including elements' IDs, activity names, and the sequence flows between elements. The dataset consists of 560 business processes with 6363

¹³<http://www.zurich.ibm.com/csc/bit/>

activities. There are 3781 different activities in which 1196 activities appear in more than one process. In average, there are 11.36 activities, 2.59 ‘start’ elements, 3.42 ‘end’ elements, 18.96 gateways (including parallel, exclusive and inclusive gateways) and 46.81 sequence flows in a process (Table 7.2).

	Min.	Max.	Average
No. of activities in a process	1	195	11.36
No. of start events in a process	1	32	2.59
No. of end events in a process	1	32	3.42
No. of gateways in a process	1	139	18.96
No. of sequence flows in a process	2	326	46.81

Table 7.2: Details of the dataset

We consider each activity as a service¹⁴ and the activity’s name as the service’s identifier. We performed three experiments to show that our approach is *feasible*, *accurate* and of *good performance*. In the first experiment, we evaluate the feasibility of our approach by measuring the number of services whose matching values with others are greater than a given threshold. We also observe the impact of parameters, including the k^{th} -zone, zone weight and similarity between connection elements, on the number of recommended services (section 7.3.2.1). In the second experiment, we evaluate the accuracy of our algorithms based on Precision and Recall metrics (section 7.3.2.2). In the third experiment, we measure the performance of our algorithms based on the computation time (section 7.3.2.3).

7.3.2.1 Approach feasibility and parameter impact

In our experiments, we run the proposed algorithms with different k^{th} -layer values. We compute the similarity between each service and the other services in the other processes. We observed the number of recommended services and their similarity values. We also analyze the impact of the zone weight and the similarity between connection elements.

We performed experiments with the neighborhood context matching in 5 cases presented in Table 7.3.

	Case 1	Case 2	Case 3	Case 4	Case 5
parallel flow relations		x	x	x	x
Zone-weight	x		x		x
Connection element similarity				x	x

Table 7.3: Examined cases

¹⁴The mapping between activities and services in a business process is discussed in section 3.3

Figure 7.9 shows the percentages of services whose matching values with at least one other service are greater than or equal to 0.5. It shows that cases 2, 3, 4, 5, which take into account the parallel flow relations, retrieve greater number of services than case 1. The zone-weight parameter has no impact in the first zone (see section 5.4.3). Hence, with $k = 1$, case 2 and case 3 has the same number of services (similar to case 4 and 5). However, with $k = 2$, the number of services and similarity values decreased. Case 3 and case 5, which take into account zone weight in their computation, retrieve more services than case 2 and case 4 respectively. It means that *zone weight impacts on the number of retrieved services*. When we take into account zone weight, we retrieve more services. Similarly, *the similarity between connection elements impacts on the number of retrieved services*. When we take into account similarity between connection elements (case 4 and case 5 compared to case 2 and case 3 respectively), we retrieve more services. Case 5, which takes into account both zone-weight and connection element similarity, retrieves the highest number of services.

In this experiment, we obtain 77.7% services whose matching values are greater than 0 and 21.48% services whose matching values are greater than 0.5. In the worst cases, we obtain 61.3% services whose matching values are greater than 0 and 8.57% services whose matching values are greater than 0.5. These results show that our approach can provide recommendations for a majority services as we can retrieve similar services for more than 2/3 number of services in average. It means that our approach is *feasible* and can be applied in real use-cases.

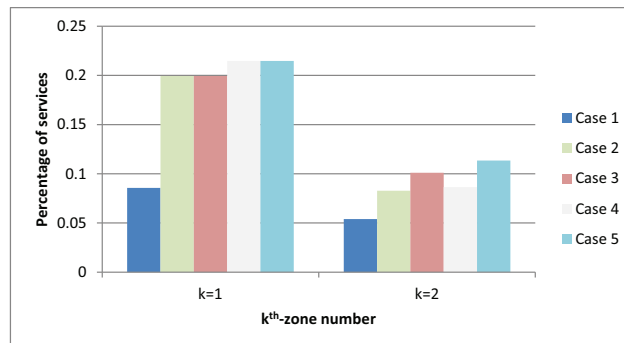


Figure 7.9: Percentage of services whose matching value ≥ 0.5

To examine the impact of k^{th} -zone values, we run our algorithms with k from 5 to 1. The experiment (Figure 7.10) shows that when k decreases, the number of recommended services increases. It is because when k decreases that the number of unmatched services in further layers decreases, the matching values between neighborhood contexts increase, the number of services increases.

This experiment also shows that zone-weight and similarity between services impact on the number of retrieved services. Cases 3 and case 5, which take into account zone-weight, has more services than case 2 and 4 respectively. Similarly, cases 4 and

5, which take into account similarity between services, has more services than case 2 and 3 respectively.

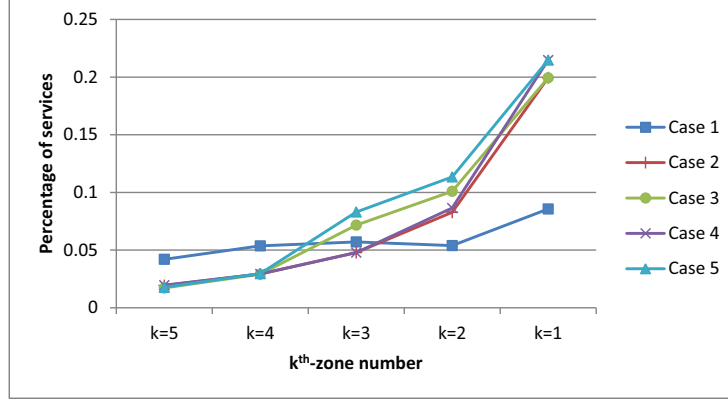


Figure 7.10: Percentage of services with different k^{th} -zone values

Figure 7.10 also shows the impact of *parallel flow relations*. When we take into account the parallel flow relations between services, many services located on the further layers of a neighborhood context graph are relocated on the nearer layers (as we discussed in section 5.6). Therefore, when we take into account parallel flow relations, the matching values in the first zone are high and these values decrease quickly when we consider further zones. Figure 7.10 show that in Cases 2, 3, 4 and 5, which take into account parallel flow relations, the number of services is very low with great k and very high with small k . Meanwhile, in Case 1, the number of services slightly change when k decreases.

7.3.2.2 Algorithms accuracy

Our approach makes recommendations based on the neighborhood context matching regardless the identifier of the considered service. The identifier of a service is used to determine the associated neighborhood context. In this experiment, we aim at using it as the ground-truth data for Precision and Recall computation. Our objective is to assess how accurate is our approach when it is used to recommend services for an empty position in a process. To do so, *for a selected service in a process, we consider this service as an unknown service*. We compute recommendations for this selected position. A relevant recommendation should contain the selected service.

Concretely, consider a selected service s in a process P . Assume that s appears in n processes. The recommendations for this selected position consist of l services, in which t ($t \leq l$) services are s . Precision and Recall of these recommendations are given by Equation 7.3.

$$Precision = \frac{t}{l}; \quad Recall = \frac{t}{n} \quad (7.3)$$

In our experiment, we tune the number of recommended services for each position from 5 to 1. We consider the matching in *the first zone*. To ignore the noise of the irrelevant processes, we compute Precision and Recall for only the services that appear in at least 10 business processes. Consequently, 29 services and 267 processes are considered for our experiment.

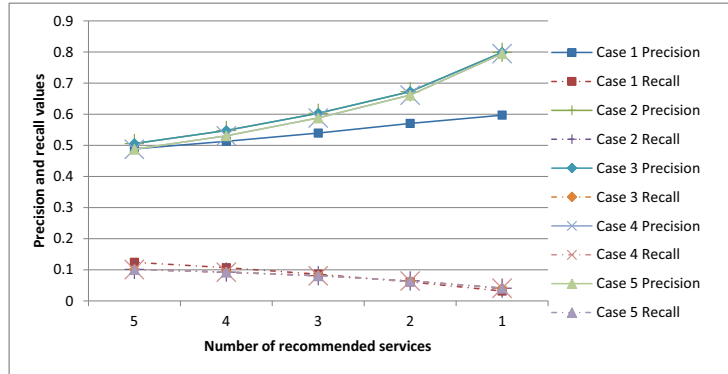


Figure 7.11: Precision and Recall values computed by taking into account the first zone

The average Precision and Recall values are shown in Figure 7.11. The Precision and Recall values of the examined cases are not so different, as the different parameters that distinguish these cases has a slight impact if we consider just the *the first zone* (as explained in the previous section). The Precision values increase when the number of recommended services decreases. This means that the relevant services mostly appear at the top of the recommendation list. In other words, when we shorten the recommendation list, the recommendations generated by our approach are more focused and precise.

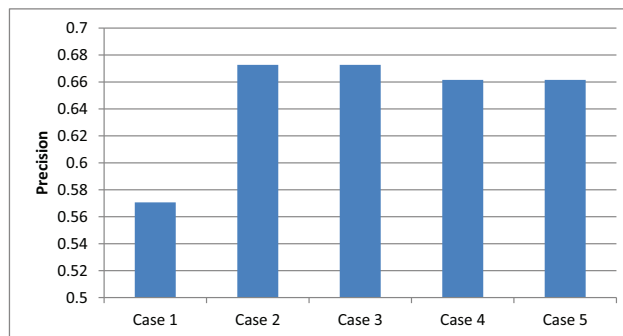


Figure 7.12: Precision values in case of two recommended services

Figure. 7.12 shows the Precision values in the case that we recommend only 2

services. These Precision values decrease when the k^{th} -layer value increases. The first algorithm which does not take into account the concurrent relations and the similarity between connection elements has the lowest Precision values.

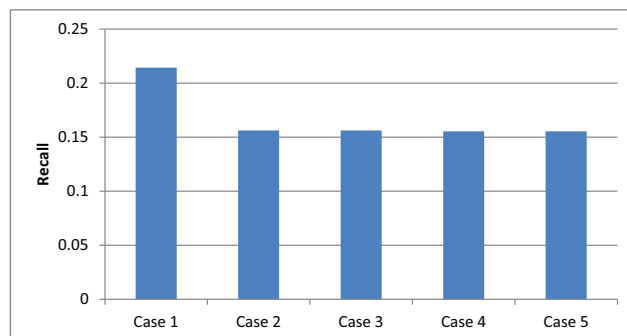


Figure 7.13: Recall values computed based on the number of relevant processes

The Recall value is not so high in the examined cases (Figure 7.11) because the number of recommended services is much more smaller than the relevant services. For example, suppose that a service s appears in 20 processes. We recommend 5 services for each selected position. The maximum value of Recall computed by Equation (7.3) is $\frac{5}{20} = 0.25$. In our approach, as we consider only services that appear in at least 10 processes (*i.e.*, $l \geq 10$) and we recommend maximum 5 services for each selected position (*i.e.*, $n \leq 5$), the value of $\frac{l}{n}$ is always less than $\frac{5}{10} = 0.5$. If we increase the number of recommended services to be equal to the number of relevant services, the Recall values increase by almost twice (Figure 7.13). Figure 7.13 also shows that the Recall values decrease when the k^{th} -layer value increases and the first algorithm has the highest Recall values.

Currently, there are few approaches [80, 87] that consider the matching between services in processes. They however use the matching results to search for relevant processes. In addition, they do not provide experiments with Precision and Recall values. So, we can not compare the accuracy of our approach to them. Instead, we consider the *random case*, where a system recommends randomly services for a selected position. We compare the simplest case of our approach, which make recommendations without considering the concurrent relation and similarity between connection elements, with the *random case*.

Figure 7.14 shows the result of our experiment. In this experiment, we make recommendations randomly for each service which appears in at least 10 business processes and compute the average Precision and Recall values. Figure 7.14 shows that the *worst* result of our approach is still much better than the *best* result of the random case (with 5 recommended services, our approach achieves 10.01 times higher Precision value and 20.62 times higher Recall value).

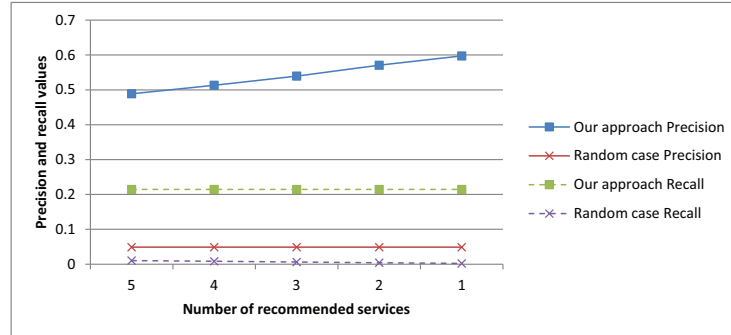


Figure 7.14: Comparing the simplest case of our approach to the random case

7.3.2.3 Algorithms performance

We performed all experiments on a computer running Ubuntu 11.10 with configuration: Pentium 4 CPU 2.8GHz, cache 512KB, RAM 512MB, HDD 80GB. We evaluate the performance of our algorithms by considering the computation time.

We measure the time that each algorithm consumes to perform the matching between a service and all other services in the dataset. Figure 7.15 shows the average computation time of all algorithms in the case that the k^{th} -zone value is 3. In average, our algorithms spend less than 2 seconds to compute the matching between a service and the other 6362 services. This means that these algorithms have *acceptable* computation time as they can make recommendations in a very short time by considering a large number of services. The experimental result also shows that Case 1, which does not take into account parallel flow relations, is the least time-consuming. Other cases, which consider more parameters, are more time-consuming. To shorten the response time for recommendations, the matching computation in our approach can be done offline.

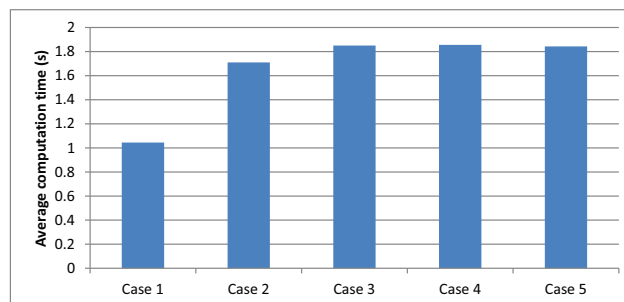


Figure 7.15: Average computation time with $k = 3$

7.3.2.4 Synthesis

We performed experiments to evaluate the *feasibility* of our approach. We also observe the *impact of parameters* (including the k^{th} -zone, zone weight and connection element similarity) on the number of recommended services. Experimental results showed that our approach is *feasible*. They also showed that when k increases, the number of similar services decreases. When taking into account zone weight or connection element similarity, our algorithms retrieve more services that have similarity values greater than a given threshold. The results also showed that the algorithms that take into account parallel flow relations between services retrieved more services than the one that does not take into account these relations.

To evaluate the accuracy of our algorithms, we compute Precision and Recall values. We examined the case in which we *recommend services for an empty position*. We have not yet examined the case in which we recommend services that are similar to a selected service as we did not have the relevant data for this case to compute Precision and Recall. We will consider this case in our future work. We also compare our algorithms to the case that makes recommendations randomly. Experimental results showed that our algorithms achieved *high Precision values*. In addition, precise services are mostly at the top of the recommendation list. So, when we shorten the recommendation list, the recommendations generated by our approach are more focused and precise. The results also showed that our algorithms were much more *accurate* than the random case.

To evaluate the performance of our algorithms, we measure the computation time. Experimental results showed that our algorithms have *acceptable* computation time as they computed the context matching of a large number of services (6362 services) within a short time (less than 2 seconds) which is not very long for a business process designer waiting for recommendation.

7.4 Conclusion

In this section, we answer the question *How accurate the recommendations are?*, which is raised in the thesis problematic (section 2.1.3). We present the implementations and experiments to validate our approach.

Four applications were implemented to prove that our approach is *feasible*. It can be flexibly developed as a standalone application or integrated into popular and specific applications to recommend services for *individual use* and *process use*.

We performed experiments with the data collected by our applications and large public datasets. Experimental results of individual-use service consumption case showed that our approach could help to *widen user's view* on the services that are related to their behaviors. The quality of recommendations is *higher* in the case that users have *stable behavior*.

Experimental results of business process-use service consumption case showed that

the number of considered layers, zone weight and connection element similarity impacted on the neighborhood context matching results. They also showed that our approach is *accurate* and of *good performance*. It performs much more better than a random approach.

Due to the general limitation of public business process datasets, which provide only elements' identifier and service' names, the validation of our approach so far is done with only the perfect match of services' names. However, our approach can be easily improved to take into account other parameters such as input, output, service description, etc. In addition, the validation can be extended for the imperfect matching. For instance, we can consider more paths on layers for the neighborhood context matching if similarity on another comparison metric between the related services is greater than a certain threshold.

Conclusion and Future Work

The research problem of this thesis is expressed by this interrogation: *How to recommend services for individual and process use?* Previous chapters presented in details our solutions to answer the raised question. In this chapter, we summary our work (section 8.1) and present the future work (section 8.2).

8.1 Conclusion

A service can be either consumed for *individual use* or *process use*. Facilitating service consumption for the individual and process use has been a hot topic since the beginning of the last decade. It became a big challenge that involves many researches in both academics and industry.

Contemporary approaches address this problem in different ways. They analyzed service descriptions, service quality, user's behavior, semantic relations, service execution order, business process topology, etc.. They applied different mathematical models and technologies such as singular value decomposition, probabilistic methods, clustering, mining, and so on.

In our work, we tackle this problem using *recommendation techniques*. We exploit *implicit knowledge* hidden in service usage data; we study *service relations* in a business process; and we capture *execution orders* from event logs for service recommendation.

To make recommendations for individual use, we proposed to utilize *past usage data* and retrieve services that are *close to user interest*. We proposed four algorithms based on collaborative filtering techniques, which have been developed for item recommendation and prediction. We also developed a new algorithm to resolve the cold-start problem. By making recommendations based on only usage data, our approach is *self-contained* and *independent* from any explicit or human centric or error prone knowledge. Whereas other information such as user rating or service reputation is hard to be captured, our approach is a good solution as it does not require such explicit knowledge.

To validate our approach, we implemented a web-based application, named IRec, that allows users to select services for their individual use and obtain recommendations. We performed experiments using the usage data collected by our application. We also run our algorithms on a large public dataset to measure their performance in a large scale system. Experiments showed that our approach could help to widen user's view on the services that are related to their interests. The quality of recommendations is high when users have stable behavior.

To make recommendations for process use, we proposed to recommend relevant services for selected positions on an ongoing designed process. These recommendations can be used in three typical cases:

- when a process analyst wants to find suitable services for an empty place in order to facilitate the process design;
- when she wants to extend (or improve) the ongoing designed process for new (or different) business goals;
- and when she wants to find services that have similar behavior to an existing one to replace it in case of failure.

We proposed to compute the similarity between services by matching their neighborhood contexts. The recommendations are generated based on the computed similarities. We also proposed an approach to extract the service neighborhood contexts from process event logs for service recommendation. Our approach not only helps to find suitable services that can be plugged-in into selected positions in a business process but also can recommend services for the auto-completion during business process design.

To validate our approach, we developed three applications: PRec, WebRec and LogRec. PRec was developed to demonstrate our recommendation technique for *process use*. It allows managing services, designing processes and viewing recommendations. WebRec was developed as a user-friendly application based on Signavio platform to widen the tool user community and make our approach more visible. Finally, LogRec was developed as a proof of concept to validate our log-based recommendation technique. We performed different experiments on a real public dataset. Experimental results showed that our approach was accurate (according to obtained precision values) and has good performance (according to computation time).

The principles presented in section 2.3.1 have been respected:

- *Focused and fine-grained results.* Our approach recommends a list of services instead of complicated results (such as entire business processes) in both service consumption cases. So, our recommendations are clear and easy to apprehend.
- *No additional information.* We do not ask users any effort to provide additional information such as user's ratings, comments or profiles. We make recommendations based on existing data, which are usage data, business process models and process logs.

- *Exploiting implicit knowledge.* We exploit implicit knowledge hidden in existing data. Concretely, we capture the correlations between users and services; and we extract neighborhood contexts of services from business processes and logs.
- *Balanced computational complexity.* The complexity of our algorithms is polynomial and we do not face the NP-complete problem. The computation time is acceptable for users.

Besides, we also consider the announced general principles:

- *Simplicity.* To obtain recommendations, users just perform a very simple action. In case of individual use, a user selects a service in which she is interested. In case of business use, a process designer selects a position for which she wants to have recommendations.
- *Flexibility.* In our approach, we allow adjusting the number of recommended services; selecting algorithms to run; setting parameters such as the number of related users and the number of considered zones; and adding constraints to filter results.

Last but not least, our approach is self-contained and independent. So, it can be associated to other approaches to better make recommendations. For example, in the business process design context, our approach can be associated to functionality-based service discovery approaches to find services that have similar function and behavior.

8.2 Future work

In the future work, we intend to improve the recommendation quality of our current work (section 8.2.1) and extend the applicability of our work to other contexts, such as the possibility to integrate our approaches into the cloud computing environment to support the business process management (section 8.2.2).

8.2.1 Improving recommendation quality

Currently, our work takes into account only non-functionality data, which are past usage data and service relations. In future work, we intend to take into account functionality data such as service descriptions, inputs, outputs, actors, resources or pre- and post-conditions. We intend to discover the correlations between these data to infer the similarity between services in term of functionality. Then, we will combine non-functionality based similarity computed by our current approach to this functionality-based similarity to make recommendations.

We also intend to associate our approach to existing functionality based recommendation methods, such as [11, 8, 9, 12, 62, 16, 17]. In this case, our approaches can play as a post- or pre-processing filter to limit the search space of relevant services. We expect that the combination of our approach with others will increase result quality.

Moreover, We aim to propose a new method to compute the matching between services using multiple criteria. So far we consider each service's name as the service's identifier and we match only services that have the same name. In the future, we will identify a service by a vector of characteristics, including name, description, location, input, output, etc. The matching between services becomes the matching between two characteristic vectors. To compute the matching between two vectors, we can, for instance, apply the vector space model or synthesize the respective matching of vector elements. Consequently, two services in the same layer will be considered equal if the matching between the corresponding characteristic vectors is greater than a given threshold.

Our approach can also be extended to deal with the matching in different layers. Currently, we match only same services on same layers to compute the neighborhood context matching. We do not take into account the similarity between services in different layers. This means that the matching algorithms may miss some meaningful information which can improve the recommendation quality. So, we are going to extend our algorithm by considering service matching from different layers. In Appendix B.1, we list out the possible matching cases and proposed some solutions. We will deal with these cases in our future work.

In the log-based approach, we currently use ProM, which is a popular mining application, to validate our approach. In the upcoming work, we will integrate the log-based recommendation feature into a process design application, such as Eclipse BPEL designer, BonitaSoft or Signavio, in order to find relevant services for an ongoing designed business process based on its logs. We will also perform more experiments with different metrics (such as ROC, AUC and top-k precision) to obtain richer results. We will identify relevant data to evaluate the accuracy of our approach in the case that a business analyst wants to find services that are relevant to a selected service in order to improve the current designed process create more business process variants.

8.2.2 Integrating into cloud computing

Cloud computing has emerged as an advanced paradigm for developing and providing services over the Internet. Its innovation brings out many benefits to small and medium enterprises, such as no up-front investment, lowering operating cost, highly scalable, easy access and reducing business risks and maintenance expenses [161]. The migration of business process management to the Cloud involves an extension of the cloud computing architecture. A new layer, so called Business Process as a Service (BPaaS) layer, was proposed to locate upon the application layer to provide process management services [162, 163].

All entities that are hosted, deployed and run on the same cloud provider somehow share their implicit and explicit data. A cloud provider can capture the exposed data to improve its services. For example, a cloud provider can analyze the logs captured from the interactions between entities on the lower layers to know the relations

between entities. This information is very useful for a cloud computing system to generate recommendations for business process design at the BPaaS layer.

Integrating our approaches into the BPaaS layer can help to (1) flexibly expand/adjust a current business process to provide new services, (2) create business process variants which are very necessary to the multi-tenant companies and (3) improve the quality of participated services in business processes which involves the improvement of the cloud environment.

In the future work, we will integrate our work into the Cloud environment. We will exploit the implicit and explicit data owned by a cloud service provider to discover the relevant services to a business process. We will infer the relations between services based on the exchanged messages in the Cloud. In addition, we will associate our recommendation techniques to other functionality-based service recommendation approaches and business model checking approaches, such as [160, 164, 165] to dynamically create business process variants.

A.1 Levenshtein distance of two inverse strings

Lemma A.1.1. *The Levenshtein distance of two strings is equal to the Levenshtein distance of their inverse strings.*

Which means, assume that:

$$\begin{aligned} s_a &= a_0 a_1 \dots a_n \\ s_b &= b_0 b_1 \dots b_n \end{aligned} \tag{A.1}$$

Let:

$$\begin{aligned} \overline{s_a} &= \text{inverse}(s_a) = a_n a_{n-1} \dots a_0 \\ \overline{s_b} &= \text{inverse}(s_b) = b_n b_{n-1} \dots b_0 \end{aligned} \tag{A.2}$$

We have:

$$LD(\overline{s_a}, \overline{s_b}) = LD(s_a, s_b) \tag{A.3}$$

Proof:

Let T be a function to transform a string to another string. $|T|$ is the number of substitution, insertion or deletion steps to process the transformation. To prove the Lemma, we need to prove that:

1. $\exists T : |T(\overline{s_a}, \overline{s_b})| = LD(s_a, s_b)$, i.e. $\overline{s_a}$ can be transformed to $\overline{s_b}$ by $LD(s_a, s_b) = l$ steps. (*)
2. $\nexists T : |T(\overline{s_a}, \overline{s_b})| < LD(s_a, s_b)$, i.e. there does not exist a solution that transforms $\overline{s_a}$ to $\overline{s_b}$ by $l' < l$ steps. (**)

Proof ():*

Assume that we need x substitution steps, y insertion steps and z deletion steps to transform from s_a to s_b . Let P_S, P_I, P_D be sets of positions in s_a that are substituted, inserted and deleted, we have

$$\begin{aligned} P_S &= \{r_1, r_2, \dots, r_x\} \\ P_I &= \{i_1, i_2, \dots, i_y\} \\ P_D &= \{d_1, d_2, \dots, d_z\} \end{aligned} \tag{A.4}$$

and

$$l = |P_S| + |P_I| + |P_D| \quad (\text{A.5})$$

Assume without loss of generality that the substitution operation (O_S) is firstly executed, then the insertion operation (O_I) and deletion (O_D), which means:

$$s_b = O_D(O_I(O_S(s_a))) \quad (\text{A.6})$$

Let :

$$\begin{aligned} s_{s_a}^r &= O_S(s_a) \\ s_{s_a}^{ir} &= O_I(s_{s_a}^r) = O_I(O_S(s_a)) \\ s_{s_a}^{dir} &= O_D(s_{s_a}^{ir}) = O_D(O_I(s_{s_a}^r)) = O_D(O_I(O_S(s_a))) = s_b \end{aligned} \quad (\text{A.7})$$

Generally, we have:

$$s' = \text{inverse}(s) \Leftrightarrow \begin{cases} s'[i] = s[n - i], \forall 0 \leq i < n = \text{length}(s) \\ \text{length}(s') = \text{length}(s) \end{cases} \quad (\text{A.8})$$

From (A.2), we have:

$$\begin{cases} \overline{s_a}[i] = s_a[n - i], \forall 0 \leq i < n = \text{length}(s_a) \\ \text{length}(s'_a) = \text{length}(s''_a) \end{cases} \quad (\text{A.9})$$

If we replace the character at the position r_t ($0 \leq r_t < \text{length}(s_a)$) in s_a to get s'_a and the character at the position $n - r_t$ in $\overline{s_a}$ by the same character c , to get s''_a , we have

$$s'_a[i] = s''_a[n - i] = c, i = r_t. \quad (\text{A.10})$$

Characters in other positions in s'_a and s''_a are kept from s_a and $\overline{s_a}$, therefore:

$$s'_a[i] = s_a[i] = \overline{s_a}[n - i] = s''_a[n - i], 0 \leq i < n = \text{length}(s_a), i \neq r_t. \quad (\text{A.11})$$

From (A.10) and (A.11), we have:

$$\begin{cases} s'_a[i] = s''_a[n - i], \forall 0 \leq i < n = \text{length}(s'_a) \\ \text{length}(s'_a) = \text{length}(s''_a) \end{cases} \quad (\text{A.12})$$

From (A.8) and (A.12), $s''_a = \text{inverse}(s'_a)$.

Inductively, if we replace characters at the positions $r_t \in P_S$ in s_a , to get $s_{s_a}^r$ and the characters at the position $n - r_t$ in $\overline{s_a}$ by the same characters, to get $s_{s_a}^{*r}$ we also have: $s_{s_a}^r[i] = s_{s_a}^{*r}[n - i], i = r_t \in P_S$.

With the same argument as in (A.11), we have:

$$\begin{cases} s_{s_a}^r[i] = s_{s_a}^{*r}[n - i], \forall 0 \leq i < n = \text{length}(s_{s_a}^r) \\ \text{length}(s_{s_a}^r) = \text{length}(s_{s_a}^{*r}) \end{cases} \quad (\text{A.13})$$

From (A.8) and (A.13), $s_{s_a}^{*r} = \text{inverse}(s_{s_a}^r) = \overline{s_{s_a}^r}$. Which means that with $|P_S|$ substitution steps, we can transform $\overline{s_a}$ to $\overline{s_{s_a}^r}$ which is the inverse string of $s_{s_a}^r$. This means:

$$\exists T : |T(\overline{s_a}, \overline{s_{s_a}^r})| = |P_S| \quad (\text{A.14})$$

Next, from (A.4) and (A.7), we have: $s_{s_a}^{ir}$ is created by inserting y characters from s_b into $s_{s_a}^r$. Since $\overline{s_{s_a}^r}$ is the inverse string of $s_{s_a}^r$, $s_{s_a}^r[i_j] = \overline{s_{s_a}^r}[n - i_j]$ and if we insert a character b_i at the position i in $s_{s_a}^r$, to get s_p and the same b_i at the position $n - i$ in $\overline{s_{s_a}^r}$, to get s_q , we will have: $s_q = \overline{s_p}$.

Inductively, if we insert all b_{uj} in I at the positions $i_j \in P_I (j = [1 \dots y])$ in $s_{s_a}^r$, to get $s_{s_a}^{ir}$ and the same b_{uj} at the position $n - i_j$ in $\overline{s_{s_a}^r}$, we will get $\overline{s_{s_a}^{ir}}$ which is the inverse string of $s_{s_a}^{ir}$.

Which means that with $|P_I|$ insertion steps, we can transform $\overline{s_{s_a}^r}$ to $\overline{s_{s_a}^{ir}}$ which is the inverse string of $s_{s_a}^{ir}$. This means:

$$\exists T : |T(\overline{s_{s_a}^r}, \overline{s_{s_a}^{ir}})| = |P_I| \quad (\text{A.15})$$

Also from (A.4) and (A.7), $s_{s_a}^{dir}$ is created by deleting z characters from $s_{s_a}^{ir}$ to get s_b . We have:

$$\begin{aligned} \overline{s_{s_a}^{ir}} &= \text{inverse}(s_{s_a}^{ir}) \\ \implies s_{s_a}^{ir}[i] &= \overline{s_{s_a}^{ir}}[n - i] \forall i \in [0 \dots n] \end{aligned}$$

Which means if we delete a character at position d_i in $s_{s_a}^{ir}$ to get a new string s_e and delete a character at position $n - d_i$ in $\overline{s_{s_a}^{ir}}$ to get a new string s_f , we have $s_f = \text{inverse}(s_e)$.

Inductively, if we delete characters at all positions $d_i \in P_D, i = [1 \dots z]$ in $s_{s_a}^{ir}$ to get $s_{s_a}^{dir}$ and characters at the positions $n - d_i$ in $\overline{s_{s_a}^{ir}}$, we will get:

$$\overline{s_{s_a}^{dir}} = O_D(\overline{s_{s_a}^{ir}}) = \text{inverse}(s_{s_a}^{dir}) = \text{inverse}(s_b) = \overline{s_b}$$

Which means that with $|P_D|$ deletion steps, we can transform $\overline{s_{s_a}^{ir}}$ to $\overline{s_{s_a}^{dir}} = \overline{s_b}$ which is the inverse string of s_b . Which means:

$$\exists T : |T(\overline{s_{s_a}^{ir}}, \overline{s_b})| = |P_D| \quad (\text{A.16})$$

From (A.14), (A.15), (A.16), we conclude that:

$$\exists T : T(\overline{s_a}, \overline{s_b}) = |P_S| + |P_I| + |P_D| = l = LD(s_a, s_b) \quad \square \quad (\text{A.17})$$

*Proof (**):*

We prove (**) by contradiction.

Assume that there exist a $l' < l$ steps that can transform $\overline{s_a}$ to $\overline{s_b}$, $l' = |P'_S| + |P'_I| + |P'_D|$.

Following (A.17), we have:

$$\exists T : |T(\overline{s_a}, \overline{s_b})| = |P'_S| + |P'_I| + |P'_D| = l' \quad (\text{A.18})$$

On the other hand, we have:

$$\begin{aligned} \text{inverse}(\overline{s_a}) &= s_a \\ \text{inverse}(\overline{s_b}) &= s_b \end{aligned} \quad (\text{A.19})$$

From (A.18) and (A.19), we have:

$$\exists T : |T(s_a, s_b)| = l' < l \quad (\text{A.20})$$

(A.20) is illogical because as the Levenshtein Distance definition, l is the minimum steps to transform from s_a to s_b .

Therefore, there does not exist a solution that transforms $\overline{s_a}$ to $\overline{s_b}$ with a $l' < l$ steps. In the other words:

$$\nexists T : |T(\overline{s_a}, \overline{s_b})| < LD(s_a, s_b) \quad \square \quad (\text{A.21})$$

From (A.17) and (A.21), we conclude that:

$$\boxed{LD(\overline{s_a}, \overline{s_b}) = LD(s_a, s_b)} \quad (\text{A.22})$$

A.2 Service location in layers

Lemma A.2.1. *Let e_{ij} be the edge connecting two vertexes a_i and a_j in the neighborhood context graph of a service a_x . $l_1 = \mathcal{L}_{a_i}^{(a_x)} \mathcal{S}_P$ and $l_2 = \mathcal{L}_{a_j}^{(a_x)} \mathcal{S}_P$ are the lengths of the shortest connection paths connecting a_i and a_j to a_x respectively. Let $d = |l_1 - l_2|$, then $d \leq 1$.*

Proof:

Assume without loss of generality that $l_1 < l_2$. We prove the lemma by contradiction.

Assume that $d = |l_1 - l_2| > 1$. As $l_1 < l_2$, we have:

$$l_2 > l_1 + 1 \quad (\text{A.23})$$

On the other hand, as e_{ij} is the edge connecting two vertexes a_i and a_j , there exists a path from a_j to a_x via a_i whose length is equal to $l_1 + 1$. This means:

$$\exists_{a_j}^{a_x} \mathcal{P} : \mathcal{L}_{a_j}^{(a_x)} \mathcal{P}_P = \mathcal{L}_{a_i}^{(a_x)} \mathcal{S}_P + 1 = l_1 + 1 \quad (\text{A.24})$$

From (A.23) and (A.24), we have:

$$\exists_{a_j}^{a_x} \mathcal{P} : \mathcal{L}(a_j \mathcal{P}_P) < l_2 = \mathcal{L}(a_j \mathcal{S}_P) \tag{A.25}$$

which means that there exists a path that has the length smaller than the shortest path. This is illogical according to the definition of the shortest path.

So, $d = |l_1 - l_2| \leq 1 \quad \square$

A.3 Checking primitive rules

Primitive rules are:

- ① If two connection elements are *identical*, their similarity is 1.
- ② Similarity between two connection elements that *have the same types but different number of output flows* is 1. Similarity between two *different* connection elements is less than 1.
- ③ In the case that two connection elements are *different*, the less different the numbers of output flows are, the greater similarity value is.

Probabilities that an output flow is executed with different connection flows are given in Table A.1.




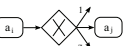
Element	Presentation	No. paths	No. possible cases	Probability that an output flow is executed
Sequence		1	1	1
AND-split		x	1	1
OR-split		y	$2^y - 1$	$\frac{2^y - 1}{2^y - 1}$
XOR-split		z	z	$\frac{1}{z}$

Table A.1: Probability that an output flow is executed

Proof that the similarities between connection elements given in Table A.2 satisfy the primitive rules.

Proof:

- 1. Proof of the first primitive rule:

Similarity	Sequence	AND-split(x)	OR-split(y)	XOR-split(z)
Sequence	1	$\frac{1}{x}$	$\frac{1}{y} \times \frac{2^{y-1}}{2^y - 1}$	$\frac{1}{z^2}$
AND-split(x)		1	$\frac{1}{x} \times \frac{1}{y} \times \frac{2^{y-1}}{2^y - 1}$	$\frac{1}{x} \times \frac{1}{z^2}$
OR-split(y)			1	$\frac{1}{y} \times \frac{2^{y-1}}{2^y - 1} \times \frac{1}{z^2}$
XOR-split(z)				1

Table A.2: Similarities between typical connection elements

It is clear from the Table A.2 that similarity between two connection flows that are the same types and same number of output flows is 1.

2. Proof of the second primitive rule:

Consider two connection elements c_1 and c_2 that connect a_i and a_j in two different business processes P_1 and P_2 . If c_1 and c_2 have the same type and different number of output flows, the connection flows from a_i to a_j in these processes are labeled by the same string. Therefore, their similarity is 1 (computed by Levenshtein distance).

If $c_1 \neq c_2$, consider the similarities presented in Table A.2, we have: $\frac{1}{x} < 1$; $\frac{1}{y} < 1$; $\frac{1}{z} < 1$. We also have: $2^{y-1} < 2^y - 1, \forall y > 1$ which infer $\frac{2^{y-1}}{2^y - 1} < 1$. So, the multiplication of these elements is always less than 1. In other words, similarity between two different elements is always less than 1.

3. Proof of the third primitive rule:

Consider a connection element c_1 . We compare c_1 to others connection elements c_2 and c_3 that have the same type and are different from c_1 . Assume that c_2 has x_1 output flows and c_3 has x_2 output flows. We are going to prove that:

$$x_1 < x_2 \implies \text{Sim}(c_1, c_2) > \text{Sim}(c_1, c_3) \quad (\text{A.26})$$

As c_2 and c_3 have the same type, $\text{Sim}(c_1, c_2)$ and $\text{Sim}(c_1, c_3)$ is computed by the same function with different input parameters. So, if we prove that $\text{Sim}(c_1, c_2)$ is a contra-covariant function by the number of output flows, we can conclude that (A.26) is always true.

We have: $\frac{1}{x}$, $\frac{1}{y}$ and $\frac{1}{z}$ are contra-covariant functions. If we prove that $\frac{2^{y-1}}{2^y-1}$ is also a contra-covariant function, we can conclude that $\text{Sim}(c_1, c_2)$ is a contra-covariant function as it is computed by a multiplication of contra-covariant functions (according to Table A.2).

To prove that $\frac{2^{y-1}}{2^y-1}$ is a contra-covariant function, we consider the function:

$$f(y) = \frac{2^{y-1}}{2^y-1}, \quad y \geq 2 \quad (\text{A.27})$$

We have:

$$\begin{aligned} & \frac{2^{y_1-1}}{2^{y_1}-1} \geq \frac{2^{y_2-1}}{2^{y_2}-1} \\ \Leftrightarrow & \frac{(2^{y_1-1})}{(2^{y_2-1})} \times \frac{(2^{y_2-1})}{(2^{y_1-1})} \geq 1 \\ \Leftrightarrow & (2^{y_1-y_2}) \times (2^{y_2-1}) \geq (2^{y_1-1}) \\ \Leftrightarrow & 2^{y_1} - 2^{y_1-y_2} \geq 2^{y_1} - 1 \\ \Leftrightarrow & 1 \geq 2^{y_1-y_2} \end{aligned} \quad (\text{A.28})$$

Assume that $y_1 < y_2$, we have: $1 \geq 2^{y_1-y_2}$. Then, from (A.28), we have:

$$\frac{2^{y_1-1}}{2^{y_1}-1} \geq \frac{2^{y_2-1}}{2^{y_2}-1} \quad (\text{A.29})$$

So, function $f(y)$ in (A.27) is contra-covariant.

Which means, $\text{Sim}(c_1, c_2)$ is a contra-covariant function and (A.26) is always true. We conclude that the similarity given in Table A.2 satisfy the second condition. \square

APPENDIX B

B.1 Service matching in different layers

In our current approach, we match only services that have the same name and are located on the same layers. For example, in Figure.B.1, we compute the matching between two connection flows $s_1 \rightarrow d_1$ and $s_2 \rightarrow d_2$ only if: $s_1 = s_2$ (or s_1, s_2 are two associated services) and $d_1 = d_2$.

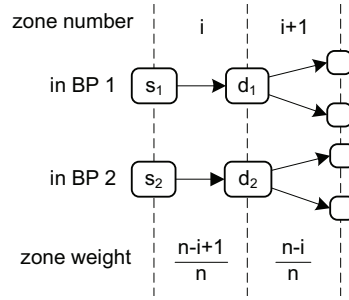


Figure B.1: Current matching case

In the case that $d_1 = d_2$ but d_1 is not the closest neighbor of s_1 or d_2 is not the closest neighbor of s_2 , even though neighbors of d_1 are completely matched with neighbors of d_2 (Figure.B.2), our approach ignores these common neighborhood contexts.

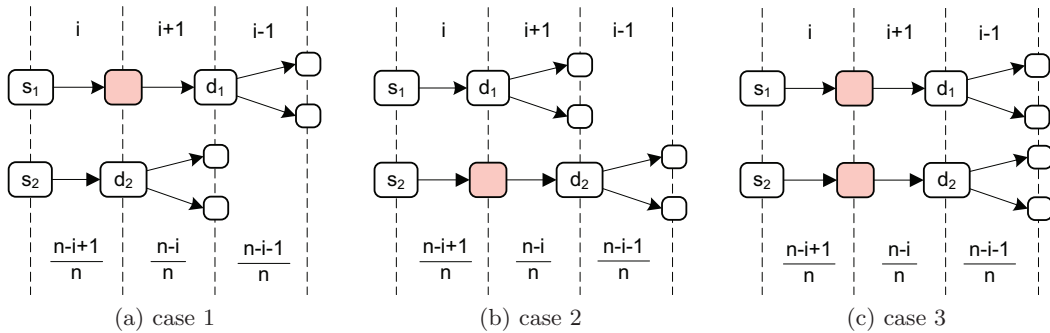


Figure B.2: Ignored cases

In this section, we present a solution that consider the matching between services not only on the same layer but also on the next layers. This solution proposes an adjustment of the connection flow matching computation.

Concretely, assume $d_1 \in N^1(s_1)$ and $d_2 \in N^1(s_2)$ and $d_1 \neq d_2$. We consider the matching of 2^{nd} -layer neighbors of s_1 and s_2 . If there exist $d'_1 \in N^2(s_1)$ and $d'_2 \in N^2(s_2)$ such that $d'_1 = d_2$ or $d'_2 = d_1$ or $d'_1 = d'_2$, we will (1) compute the pattern matching between $s_1 \rightarrow d'_1$ and $s_2 \rightarrow d'_2$, (2) assign d'_1 to d_1 , d'_2 to d_2 and (3) repeat the matching process.

To compute the matching between $s_1 \rightarrow d'_1$ and $s_2 \rightarrow d'_2$, we ignore the intermediated services on the connection flows and take into account a new zone weight which is computed by the *average weights of the zones that the flows cross*.

For example, in Fig. B.2a and B.2b, the average zone weight is:

$$\bar{w} = \frac{2 \times \frac{n-i+1}{n} + \frac{n-i}{n}}{3}$$

while in Fig. B.2c, the average zone weight is:

$$\bar{w} = \frac{2 \times \frac{n-i+1}{n} + 2 \times \frac{n-i}{n}}{4}$$

In the case that $d_1 = d_2$, the zone weight computed in this solution becomes the zone weight defined in our current approach.

In the following, we present some typical cases where we can apply our solution. In these cases, we compute the matching between connection flows $x \rightarrow f$ and $x' \rightarrow f'$

a. Simple cases: ordered connection flows.

First example (Figure. B.3):

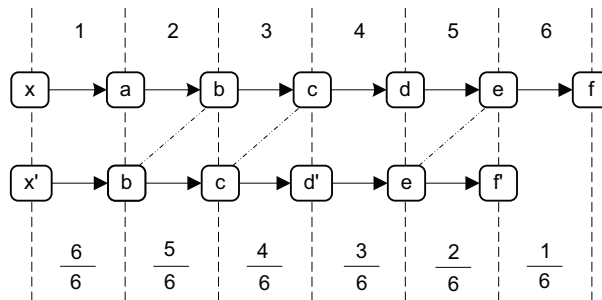


Figure B.3: Example: case 1

$$M^+ = \frac{9}{2} \times M({}_b^c f_{P_1}, {}_b^c f_{P_2}) + \frac{12}{4} \times M({}_c^e f_{P_1}, {}_c^e f_{P_2})$$

Second example (Figure. B.4):

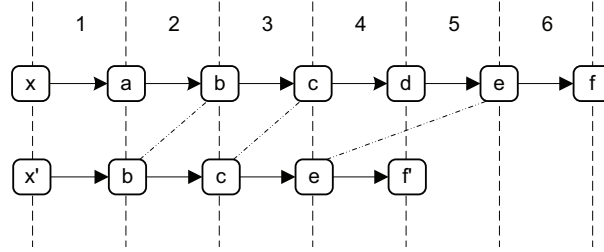


Figure B.4: Example: case 2

$$M^+ = \frac{9}{6} \times M({}_b^c f_{P_1}, {}_b^c f_{P_2}) + \frac{9}{3} \times M({}_c^e f_{P_1}, {}_c^e f_{P_2})$$

b. Complex cases: disordered connection flows

First example (Figure. B.5):

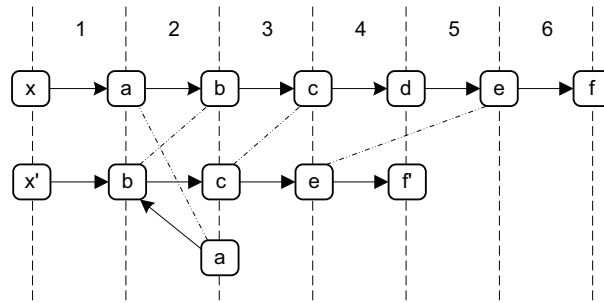


Figure B.5: Example: case 3

$$M^+ = \frac{17}{6} \times M({}_x^a f_{P_1}, {}_x^a f_{P_2}) + \frac{17}{3} \times M({}_x^b f_{P_1}, {}_x^b f_{P_2}) + \frac{9}{6} \times M({}_b^c f_{P_1}, {}_b^c f_{P_2}) + \frac{9}{3} \times M({}_c^e f_{P_1}, {}_c^e f_{P_2})$$

Second example (Figure. B.6):

$$M^+ = \frac{17}{6} \times M({}_x^b f_{P_1}, {}_x^b f_{P_2}) + \frac{9}{6} \times M({}_b^c f_{P_1}, {}_b^c f_{P_2}) + \frac{12}{6} \times M({}_b^d f_{P_1}, {}_b^d f_{P_2}) + \frac{1}{2} \times \left(\frac{16}{4} \times M({}_c^e f_{P_1}, {}_c^e f_{P_2}^{(d_1)}) + \frac{16}{4} \times M({}_c^e f_{P_1}, {}_c^e f_{P_2}^{(d_2)}) \right)$$

Third example (Figure. B.7):

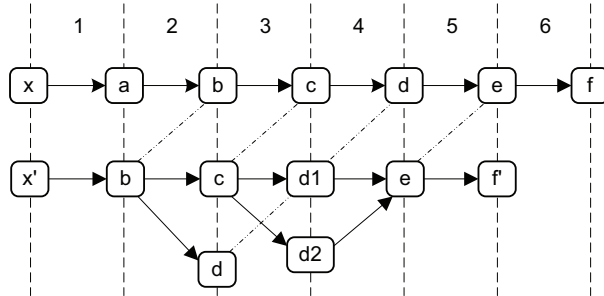


Figure B.6: Example: case 4

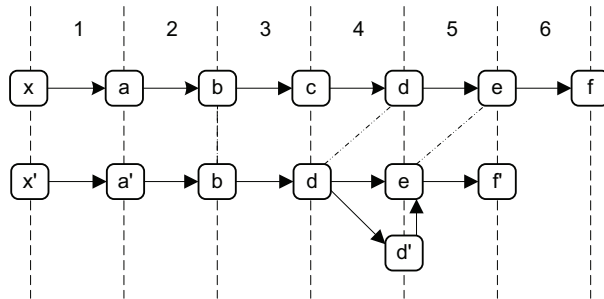


Figure B.7: Example: case 5

$$\begin{aligned}
 M^+ &= \frac{22}{6} \times \frac{1}{4} \times M_{(x f_{P_1}, x' f_{P_2})}^{(b f_{P_1}, b f_{P_2})} + \frac{11}{6} \times \frac{1}{3} \times M_{(b f_{P_1}, b f_{P_2})}^{(d f_{P_1}, d f_{P_2})} \\
 &+ \frac{1}{2} \times \left(\frac{5}{6} \times M_{(d f_{P_1}, d f_{P_2})}^{(e f_{P_1}, e f_{P_2})} + \frac{7}{6} \times M_{(d f_{P_1}, d f_{P_2})}^{(e f_{P_1}, e f_{P_2}^{(d')})} \right)
 \end{aligned}$$

C.1 List of publications

Journal article

1. Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. A recommender system based on historical usage data for web service discovery. *Service Oriented Computing and Applications*, 6(1):51-63, March 2012.

Conference proceeding

1. Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Assisting business process design by activity neighborhood context matching. In *10th International Conference on Service Oriented Computing (ICSOC)*, Lecture Notes in Computer Science, Shanghai, China, Nov. 2012. Springer.
2. Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Composition context matching for web service recommendation. In *Proceedings of the 2011 IEEE International Conference on Services Computing (SCC)*, pages 624-631, Washington, DC, USA, 2011. IEEE Computer Society.
3. Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Context-based service recommendation for assisting business process design. In *E-Commerce and Web Technologies (EC-Web)*, volume 85 of Lecture Notes in Business Information Processing, pages 39-51. Springer Berlin Heidelberg, 2011.
4. Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. A web service recommender system using vector space model and latent semantic indexing. In *IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 602-609, march 2011.
5. Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Web services recommendation based on user's behavior. In *IEEE 7th International Conference on e-Business Engineering (ICEBE)*, pages 214-221, Shanghai, China, Nov. 2010.
6. Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Collaborative filtering technique for web service recommendation based on user-operation combination. In *OTM Conferences (CoopIS)*, OTM'10, pages 222-239, Berlin, Heidelberg, 2010. Springer-Verlag.

Bibliography

- [1] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. W3c note, World Wide Web Consortium, March 2001.
- [2] Bob Atkinson, Tom Bellwood, Maud Cahuzac, Luc Clément, John Colgrave, Ugo Corda, Alexandru Czimbor, Matthew J. Dovey, Daniel Feygin, Shishir Garg, Rajul Gupta, Andrew Hately, Brad Henry, Aikichi Kawai, Paul Macias, Anne Thomas Manes, Claus von Riegen, Tony Rogers, Alok Srivastava, Paul Thorpe, Alessandro Triglia, Max Voskob, and George Zagelow. Uddi spec technical committee specification. Technical report, 2003.
- [3] Henrik F. Nielsen, Noah Mendelsohn, Jean J. Moreau, Martin Gudgin, and Marc Hadley. Soap version 1.2 part 1: Messaging framework. W3c recommendation, W3C, June 2003.
- [4] Neal Leavitt. Are web services finally ready to deliver? *Computer*, 37:14–18, 2004.
- [5] The Mckinsey Quarterly. How businesses are using web 2.0: A mckinsey global survey. Technical report, 2007.
- [6] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating web services on the world wide web. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 795–804, New York, NY, USA, 2008. ACM.
- [7] Bachlechner Daniel, Siorpaes Katharina, Lausen Holger, and Dieter Fensel. Web service discovery ? a reality check. *3rd European Semantic Web Conference, Budva, Montenegro*, June 2006.
- [8] C. Platzer and S. Dustdar. A vector space search engine for web services. *Web Services, 2005. ECOWS 2005. Third IEEE European Conference on*, pages 9 pp.–, Nov. 2005.
- [9] M. Brian Blake and Michael F. Nowlan. A web service recommender system using enhanced syntactical matching. In *ICWS*, pages 575–582, 2007.
- [10] Chen Wu, Vidyasagar Potdar, and Elizabeth Chang. Latent semantic analysis - the dynamics of semantics web services discovery. In Tharam Dillon, Elizabeth Chang, Robert Meersman, and Katia Sycara, editors, *Advances in Web Semantics I*, volume 4891 of *Lecture Notes in Computer Science*, pages 346–373. Springer Berlin / Heidelberg, 2009.

-
- [11] Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Similarity search for web services. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 372–383. VLDB Endowment, 2004.
- [12] Jiangang Ma, Yanchun Zhang, and Jing He. Web services discovery based on latent semantic approach. In *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*, pages 740–747, Washington, DC, USA, 2008. IEEE Computer Society.
- [13] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. Web*, 1, May 2007.
- [14] Zibin Zheng, Hao Ma, Michael R. Lyu, and Irwin King. Wsrec: A collaborative filtering based web service recommender system. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 437–444, Washington, DC, USA, 2009. IEEE Computer Society.
- [15] Yechun Jiang, Jianxun Liu, Mingdong Tang, and Xiaoqing (Frank) Liu. An effective web service recommendation method based on personalized collaborative filtering. In *IEEE International Conference on Web Services, ICWS 2011, Washington, DC, USA, July 4-9, 2011*, pages 211–218, 2011.
- [16] Umardand Shripad Manikrao and T. V. Prabhakar. Dynamic selection of web services with recommendation system. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, page 117, Washington, DC, USA, 2005. IEEE Computer Society.
- [17] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 333–347, London, UK, 2002. Springer-Verlag.
- [18] Aabhas V. Paliwal, Nabil R. Adam, and Christof Bornhövd. Web service discovery: Adding semantics through service request expansion and latent semantic indexing. In *2007 IEEE International Conference on Services Computing (SCC 2007), 9-13 July 2007, Salt Lake City, Utah, USA*, pages 106–113, 2007.
- [19] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, October 2003.
- [20] Adam Barker, Christopher D. Walton, and David Robertson. Choreographing web services. *IEEE Trans. Serv. Comput.*, 2(2):152–166, April 2009.
- [21] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte,

- Ivana Trickovic, and Sanjiva Weerawarana. Bpel4ws, business process execution language for web services version 1.1. Technical report, 2003.
- [22] Martin Chapman, Duncan Johnston-Watt, Nickolas Kavantzias, Yves Lafon, Jeff Mischkinsky, and Greg Ritzinger. Web services choreography description language version 1.0. Technical report, 2005.
- [23] W.M.P. van der Aalst, A. H. M. Ter Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of the 1st International Conference on Business Process Management*, pages 1–12. Springer-Verlag, 2003.
- [24] Scott Stephens. Supply chain operations reference model version 5.0: A new tool to improve supply chain efficiency and achieve best practice. *Information Systems Frontiers*, 3:471–476, December 2001.
- [25] Thomas Curran, Gerhard Keller, and Andrew Ladd. *SAP R/3 business blueprint: understanding the business process reference model*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.
- [26] Remco Dijkman, Marlon Dumas, and Luciano Garcia-Banuelos. Graph matching algorithms for business process model similarity search. In *Proceedings of the 7th International Conference on Business Process Management, BPM '09*, pages 48–63, Berlin, Heidelberg, 2009. Springer-Verlag.
- [27] Zhiqiang Yan, Remco Dijkman, and Paul Grefen. Fast business process similarity search with feature-based similarity estimation. In *Proceedings of the 2010 international conference on On the move to meaningful internet systems - Volume Part I, OTM'10*, pages 60–77, Berlin, Heidelberg, 2010. Springer-Verlag.
- [28] W.M.P. Aalst, A.K.Alves Medeiros, and A.J.M.M. Weijters. Process equivalence: Comparing two process models based on observed behavior. In Schahram Dustdar, JosÁ©Luiz Fiadeiro, and AmitP. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 129–144. Springer Berlin Heidelberg, 2006.
- [29] Remco M. Dijkman. A classification of differences between similar business processes. In *11th IEEE International Enterprise Distributed Object Computing Conference, 15-19 October 2007, Annapolis, Maryland, USA*, pages 37–50, 2007.
- [30] Chen Li, Manfred Reichert, and Andreas Wombacher. On measuring process model similarity based on high-level change operations. In *Proceedings of the 27th International Conference on Conceptual Modeling, ER '08*, pages 248–264, Berlin, Heidelberg, 2008. Springer-Verlag.
- [31] Boudewijn Dongen, Remco Dijkman, and Jan Mendling. Measuring similarity between business process models. In *Proceedings of the 20th international*

- conference on Advanced Information Systems Engineering, CAiSE '08*, pages 450–464, Berlin, Heidelberg, 2008. Springer-Verlag.
- [32] Marc Ehrig, Agnes Koschmider, and Andreas Oberweis. Measuring similarity between semantic business process models. In *Proceedings of the fourth Asia-Pacific conference on Conceptual modelling - Volume 67*, APCCM '07, pages 71–80, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.
- [33] Ahmed Awad. Bpmn-q: A language to query business processes. In *EMISA*, pages 115–128, 2007.
- [34] Ahmed Awad, Artem Polyvyanyy, and Mathias Weske. Semantic querying of business process models. In *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 85–94, Washington, DC, USA, 2008. IEEE Computer Society.
- [35] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using bpmn-q and temporal logic. In *Proceedings of the 6th International Conference on Business Process Management, BPM '08*, pages 326–341, Berlin, Heidelberg, 2008. Springer-Verlag.
- [36] Sherif Sakr and Ahmed Awad. A framework for querying graph-based business process models. In *Proceedings of the 19th international conference on World wide web, WWW '10*, pages 1297–1300, New York, NY, USA, 2010. ACM.
- [37] Sherif Sakr, Emilian Pascalau, Ahmed Awad, and Mattias Weske. Partial process models to manage business process variants. *International Journal of Business Process Integration and Management (IJBPIIM)*, 6(2):20, September 2011.
- [38] A. J. M. M. Weijters and W. M. P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integr. Comput.-Aided Eng.*, 10(2):151–162, April 2003.
- [39] Wil van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, September 2004.
- [40] Fabrizio M. Maggi, R.P.Jagadeesh Chandra Bose, and Wil M.P. Aalst. Efficient discovery of understandable declarative process models from event logs. In Jolita Ralyte, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza, editors, *Advanced Information Systems Engineering*, volume 7328 of *Lecture Notes in Computer Science*, pages 270–285. Springer Berlin Heidelberg, 2012.

- [41] Robert Engel, Wil van der Aalst, Marco Zapletal, Christian Pichler, and Hannes Werthner. Mining inter-organizational business process models from edi messages: A case study from the automotive sector. In Jolita Ralytė, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza, editors, *Advanced Information Systems Engineering*, volume 7328 of *Lecture Notes in Computer Science*, pages 222–237. Springer Berlin / Heidelberg, 2012.
- [42] Wil M. P. van der Aalst. Process discovery: capturing the invisible. *Comp. Intell. Mag.*, 5(1):28–41, February 2010.
- [43] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, March 1997.
- [44] Cheng-Ting Wu and Hsiao-Fan Wang. Recent development of recommender systems. *Industrial Engineering and Engineering Management, 2007 IEEE International Conference on*, pages 228–232, Dec. 2007.
- [45] A. Birukou, E. Blanzieri, V. D’Andrea, P. Giorgini, and N. Kokash. Improving web service discovery with usage data. *Software, IEEE*, 24(6):47–54, Nov.-Dec. 2007.
- [46] Natallia Kokash, Aliaksandr Birukou, and Vincenzo D’Andrea. Web service discovery based on past user experience. In *Proceedings of the 10th international conference on Business information systems, BIS’07*, pages 95–107, Berlin, Heidelberg, 2007. Springer-Verlag.
- [47] Zhenqiang Wang, Kaiyin Liu, Guoying Lv, and Xiaoyan Hao. Study of an algorithm of web service matching based on semantic web service. In *ALPIT ’07: Proceedings of the Sixth International Conference on Advanced Language Processing and Web Information Technology (ALPIT 2007)*, pages 429–433, Washington, DC, USA, 2007. IEEE Computer Society.
- [48] Shaofei Wu. A new web services matching algorithm. In *IUCE ’09: Proceedings of the 2009 International Symposium on Intelligent Ubiquitous Computing and Education*, pages 414–416, Washington, DC, USA, 2009. IEEE Computer Society.
- [49] Wil M. P. van der Aalst, Marlon Dumas, Florian Gottschalk, Arthur H. M. ter Hofstede, Marcello La Rosa, and Jan Mendling. Preserving correctness during business process model configuration. *Formal Asp. Comput.*, 22(3-4):459–482, 2010.
- [50] Matthias Weidlich, Remco Dijkman, and Jan Mendling. The icop framework: identification of correspondences between process models. In *Proceedings of the 22nd international conference on Advanced information systems engineering, CAiSE’10*, pages 483–498, Berlin, Heidelberg, 2010. Springer-Verlag.

-
- [51] Anne Yun-An Chen and Dennis McLeod. Collaborative filtering for information recommendation systems. *Encyclopedia of E-Commerce, E-Government, and Mobile Commerce, IGI Global*, pages 118–123, January 2006.
- [52] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, January 2004.
- [53] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. GroupLens: applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, March 1997.
- [54] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". pages 210–217. ACM Press, 1995.
- [55] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '95*, pages 194–201, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.
- [56] B.J. Dahlen, J.A. Konstan, J.L. Herlocker, N. Good, A. Borchers, and J. Riedl. Jump-starting movielens: User benefits of starting a collaborative filtering system with "dead-data". In . University of Minnesota TR 98-017, 1998.
- [57] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work, CSCW '00*, pages 241–250, New York, NY, USA, 2000. ACM.
- [58] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.
- [59] Wil M. P. van der Aalst. Challenges in business process analysis. In *9th International Conference on Enterprise Information Systems (Selected Papers)*, pages 27–42. Springer, 2007.
- [60] R Birukou, Enrico Blanzieri, Paolo Giorgini, Natallia Kokash, and Alessio Modena. Ic-service: A service-oriented approach to the development of recommendation systems. In *In: Proceedings of ACM Symposium on Applied Computing. Special Track on Web Technologies, ACM Press*, pages 1683–1688. Press, 2007.
- [61] Khalid Elgazzar, Ahmed E. Hassan, and Patrick Martin. Clustering wsdl documents to bootstrap the discovery of web services. In *Proceedings of the 2010 IEEE International Conference on Web Services, ICWS '10*, pages 147–154, Washington, DC, USA, 2010. IEEE Computer Society.

-
- [62] Yilei Zhang, Zibin Zheng, and Michael R. Lyu. Wsexpress: A qos-aware search engine for web services. In *ICWS*, pages 91–98, 2010.
- [63] Zibin Zheng and Michael R. Lyu. Collaborative reliability prediction of service-oriented systems. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE '10*, pages 35–44, New York, NY, USA, 2010. ACM.
- [64] Joyce El Haddad, Maude Manouvrier, Guillermo Ramirez, and Marta Rukoz. Qos-driven selection of web services for transactional composition. In *Proceedings of the 2008 IEEE International Conference on Web Services, ICWS '08*, pages 653–660, Washington, DC, USA, 2008. IEEE Computer Society.
- [65] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, and Francesco Lo Presti. Flow-based service selection for web service composition supporting multiple qos classes. *2012 IEEE 19th International Conference on Web Services*, 0:743–750, 2007.
- [66] Yutu Liu, Anne H. Ngu, and Liang Z. Zeng. Qos computation and policing in dynamic web service selection. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, WWW Alt. '04*, pages 66–73, New York, NY, USA, 2004. ACM.
- [67] Mohammad Alrifai and Thomas Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *Proceedings of the 18th international conference on World wide web, WWW '09*, pages 881–890, New York, NY, USA, 2009. ACM.
- [68] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, 33(6):369–384, June 2007.
- [69] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, May 2004.
- [70] Tao Yu and Kwei-Jay Lin. Service selection algorithms for composing complex services with multiple qos constraints. In *Proceedings of the Third international conference on Service-Oriented Computing, ICSOC'05*, pages 130–143, Berlin, Heidelberg, 2005. Springer-Verlag.
- [71] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web, WWW '03*, pages 411–421, New York, NY, USA, 2003. ACM.

- [72] W3C OWL Working Group. Owl web ontology language. Technical report, 2004.
- [73] Marcello La Rosa, Marlon Dumas, Arthur H. M. ter Hofstede, and Jan Mendling. Configurable multi-perspective business process models. *Inf. Syst.*, 36(2):313–340, 2011.
- [74] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.
- [75] Nick Russell, Arthur H.M. ter Hofstede, Wil M.P. van der Aalst, and Nataliya Mulyar. Workflow control-flow patterns: A revised view. Technical report, BPM Center Report BPM-06-22, 2006.
- [76] Thomas Gschwind, Jana Koehler, and Janette Wong. Applying patterns during business process modeling. In *Proceedings of the 6th International Conference on Business Process Management, BPM '08*, pages 4–19, Berlin, Heidelberg, 2008. Springer-Verlag.
- [77] Uwe Zdun and Schahram Dustdar. Model-driven and pattern-based integration of process-driven soa models. In Frank Leymann, Wolfgang Reisig, Satish R. Thatte, and Wil van der Aalst, editors, *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
- [78] Lucineia Heloisa Thom, Jean Michel Lau, Cirano Iochpe, and Jan Mendling. Extending business process modeling tools with workflow pattern reuse. In *Proceedings of the Ninth International Conference on Enterprise Information Systems*, pages 447–452, 2007.
- [79] Steen Brahe and Behzad Bordbar. A pattern-based approach to business process modeling and implementation in web services. In *Proceedings of the 4th international conference on Service-oriented computing, ICSOC'06*, pages 166–177, Berlin, Heidelberg, 2007. Springer-Verlag.
- [80] Maya Lincoln, Mati Golani, and Avigdor Gal. Machine-assisted design of business process models using descriptor space analysis. In *Proceedings of the 8th international conference on Business process management, BPM'10*, pages 128–144, Berlin, Heidelberg, 2010. Springer-Verlag.
- [81] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, 18(9):689–694, August 1997.
- [82] VI Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966.

-
- [83] W.M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639 – 650, 1999.
- [84] Remco Dijkman, Marlon Dumas, Boudewijn van Dongen, Reina Käärrik, and Jan Mendling. Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 36(2):498–516, April 2011.
- [85] Mohammad Abdulkader Abdulrahim. *Parallel algorithms for labeled graph matching*. PhD thesis, Golden, CO, USA, 1998. AAI0599838.
- [86] Thomas Hornung, Agnes Koschmider, and Georg Lausen. Recommendation based process modeling support: Method and user experience. In *Proceedings of the 27th International Conference on Conceptual Modeling*, ER '08, pages 265–278, Berlin, Heidelberg, 2008. Springer-Verlag.
- [87] Maya Lincoln and Avigdor Gal. Searching business process repositories using operational similarity. In *Proceedings of the 2011th Confederated international conference on On the move to meaningful internet systems - Volume Part I*, OTM'11, pages 2–19, Berlin, Heidelberg, 2011. Springer-Verlag.
- [88] Mariusz Momotko and Kazimierz Subieta. Process query language: A way to make workflow processes more flexible. In *Advances in Databases and Information Systems*, volume 3255 of *Lecture Notes in Computer Science*, pages 306–321. Springer Berlin / Heidelberg, 2004.
- [89] Catriel Beeri, Anat Eyal, Simon Kamenkovich, and Tova Milo. Querying business processes. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 343–354. VLDB Endowment, 2006.
- [90] Daniel Deutch and Tova Milo. Querying structural and behavioral properties of business processes. In *Proceedings of the 11th international conference on Database programming languages*, DBPL'07, pages 169–185, Berlin, Heidelberg, 2007. Springer-Verlag.
- [91] Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg. Monitoring business processes with queries. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 603–614. VLDB Endowment, 2007.
- [92] Seyed-Mehdi-Reza Beheshti, Boualem Benatallah, Hamid Motahari-Nezhad, and Sherif Sakr. A query language for analyzing business processes execution. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 281–297. Springer Berlin / Heidelberg, 2011.
- [93] A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, March 2008.

-
- [94] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rev.: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
- [95] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *Proceedings of the 2011 IEEE 15th International Enterprise Distributed Object Computing Conference, EDOC '11*, pages 55–64, Washington, DC, USA, 2011. IEEE Computer Society.
- [96] Dirk Fahland, Massimiliano De Leoni, Boudewijn F. Van Dongen, and Wil M. P. Van Der Aalst. Conformance checking of interacting processes with overlapping instances. In *Proceedings of the 9th international conference on Business process management, BPM'11*, pages 345–361, Berlin, Heidelberg, 2011. Springer-Verlag.
- [97] Jan Mendling, Gustaf Neumann, and Wil Van Der Aalst. Understanding the occurrence of errors in process models based on metrics. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I, OTM'07*, pages 113–130, Berlin, Heidelberg, 2007. Springer-Verlag.
- [98] J. Mendling, H. M. W. Verbeek, B. F. van Dongen, W. M. P. van der Aalst, and G. Neumann. Detection and prediction of errors in eps of the sap reference model. *Data Knowl. Eng.*, 64(1):312–329, January 2008.
- [99] WilM.P. Aalst and Minseok Song. Mining social networks: Uncovering interaction patterns in business processes. In Jorg Desel, Barbara Pernici, and Mathias Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer Berlin Heidelberg, 2004.
- [100] Wil M. P. Van Der Aalst, Hajo A. Reijers, and Minseok Song. Discovering social networks from event logs. *Comput. Supported Coop. Work*, 14(6):549–593, December 2005.
- [101] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '98*, pages 469–483, London, UK, UK, 1998. Springer-Verlag.
- [102] W. M. P. van der Aalst, H. A. Reijers, A. J. M. M. Weijters, B. F. van Dongen, A. K. Alves de Medeiros, M. Song, and H. M. W. Verbeek. Business process mining: An industrial application. *Inf. Syst.*, 32(5):713–732, July 2007.

- [103] Wil M. P. van der Aalst and B. F. van Dongen. Discovering workflow performance models from timed logs. In *Proceedings of the First International Conference on Engineering and Deployment of Cooperative Information Systems, EDCIS '02*, pages 45–63, London, UK, UK, 2002. Springer-Verlag.
- [104] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, November 2003.
- [105] Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, July 1998.
- [106] W. M. P. Van Der Aalst, B. F. Van Dongen, C. W. Günther, R. S. Mans, A. K. Alves De Medeiros, A. Rozinat, V. Rubin, M. Song, H. M. W. Verbeek, and A. J. M. M. Weijters. Prom 4.0: comprehensive support for real process analysis. In *Proceedings of the 28th international conference on Applications and theory of Petri nets and other models of concurrency, ICATPN'07*, pages 484–494, Berlin, Heidelberg, 2007. Springer-Verlag.
- [107] A. Rozinat and W. M. P. van der Aalst. Decision mining in prom. In *Proceedings of the 4th international conference on Business Process Management, BPM'06*, pages 420–425, Berlin, Heidelberg, 2006. Springer-Verlag.
- [108] J. M. Werf, B. F. Dongen, C. A. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. In *Proceedings of the 29th international conference on Applications and Theory of Petri Nets, PETRI NETS '08*, pages 368–387, Berlin, Heidelberg, 2008. Springer-Verlag.
- [109] Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Collaborative filtering technique for web service recommendation based on user-operation combination. In *OTM Conferences (CoopIS)*, OTM'10, pages 222–239, Berlin, Heidelberg, 2010. Springer-Verlag.
- [110] Nguyen Ngoc Chan, W. Gaaloul, and S. Tata. Web services recommendation based on user's behavior. In *IEEE 7th International Conference on e-Business Engineering (ICEBE)*, pages 214 –221, Shanghai, China, nov. 2010.
- [111] Nguyen Ngoc Chan, W. Gaaloul, and S. Tata. A web service recommender system using vector space model and latent semantic indexing. In *IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 602 –609, march 2011.
- [112] Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. A recommender system based on historical usage data for web service discovery. *Service Oriented Computing and Applications*, 6(1):51–63, March 2012.

-
- [113] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, December 1992.
- [114] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151, July 2001.
- [115] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work, CSCW '94*, pages 175–186, New York, NY, USA, 1994. ACM.
- [116] Gregory D. Linden, Jennifer A. Jacobi, and Eric A. Benson. Collaborative recommendations using item-to-item similarity mappings, July 2001.
- [117] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.
- [118] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, NY, USA, 2001. ACM.
- [119] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.
- [120] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning, ICML '98*, pages 46–54, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
- [121] Koji Miyahara and Michael J. Pazzani. Collaborative filtering with the simple bayesian classifier. In *Proceedings of the 6th Pacific Rim international conference on Artificial intelligence, PRICAI'00*, pages 679–689, Berlin, Heidelberg, 2000. Springer-Verlag.
- [122] Xiaoyuan Su and Taghi M. Khoshgoftaar. Collaborative filtering for multi-class data using belief nets algorithms. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, ICTAI '06*, pages 497–504, Washington, DC, USA, 2006. IEEE Computer Society.
- [123] L. Ungar and D. Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*. AAAI Press, Menlo Park California, 1998.

-
- [124] Sonny Han Seng Chee, Jiawei Han, and Ke Wang. Rectree: An efficient collaborative filtering method. In *Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery, DaWaK '01*, pages 141–151, London, UK, UK, 2001. Springer-Verlag.
- [125] Gui-Rong Xue, Chenxi Lin, Qiang Yang, WenSi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '05*, pages 114–121, New York, NY, USA, 2005. ACM.
- [126] Slobodan Vucetic and Zoran Obradovic. Collaborative filtering using a regression-based approach. *Knowl. Inf. Syst.*, 7(1):1–22, January 2005.
- [127] Guy Shani, David Heckerman, and Ronen I. Brafman. An mdp-based recommender system. *J. Mach. Learn. Res.*, 6:1265–1295, December 2005.
- [128] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, January 2004.
- [129] Thomas Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Mach. Learn.*, 42(1-2):177–196, January 2001.
- [130] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.
- [131] Chris H. Q. Ding. A similarity-based probability model for latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '99*, pages 58–65, New York, NY, USA, 1999. ACM.
- [132] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 158–167, New York, NY, USA, 2000. ACM.
- [133] Dmitry Pavlov, Eren Manavoglu, David M. Pennock, and C. Lee Giles. Collaborative filtering with maximum entropy. *IEEE Intelligent Systems*, 19(6):40–48, November 2004.
- [134] Daniel Nikovski and Veselin Kulev. Induction of compact decision trees for personalized recommendation. In *Proceedings of the 2006 ACM symposium on Applied computing, SAC '06*, pages 575–581, New York, NY, USA, 2006. ACM.

- [135] Charu C. Aggarwal, Joel L. Wolf, Kun-Lung Wu, and Philip S. Yu. Horting hatches an egg: a new graph-theoretic approach to collaborative filtering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '99, pages 201–212, New York, NY, USA, 1999. ACM.
- [136] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. pages 43–52. Morgan Kaufmann, 1998.
- [137] Alexandrin Popescul, Lyle H. Ungar, David M. Pennock, and Steve Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, UAI '01, pages 437–444, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- [138] David M. Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 473–480, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [139] Kai Yu, Anton Schwaighofer, Volker Tresp, Xiaowei Xu, and Hans-Peter Kriegel. Probabilistic memory-based collaborative filtering. *IEEE Trans. on Knowl. and Data Eng.*, 16(1):56–69, January 2004.
- [140] Marko Balabanovic and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, March 1997.
- [141] Michael J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev.*, 13(5-6):393–408, December 1999.
- [142] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.
- [143] Scott C. Deerwester, Susan T. Dumais, George W. Furnas, Richard A. Harshman, Thomas K. Landauer, Karen E. Lochbaum, and Lynn A. Streeter. Computer information retrieval using latent semantic structure. *US Patent No. 4839853*, June 1989.
- [144] Michael W. Berry. Large scale sparse singular value computations. *International Journal of Supercomputer Applications*, pages 13–49, 1992.
- [145] Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995.

-
- [146] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.
- [147] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, (25):259–284, 1998.
- [148] April Kontostathis and William M. Pottenger. A framework for understanding latent semantic indexing (lsi) performance. *Inf. Process. Manage.*, 42(1):56–73, January 2006.
- [149] L.G. Perez, M. Barranco, and L. Martinez. Building user profiles for recommender systems from incomplete preference relations. *Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International*, pages 1–6, July 2007.
- [150] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.
- [151] Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Composition context matching for web service recommendation. In *Proceedings of the 2011 IEEE International Conference on Services Computing (SCC)*, pages 624–631, Washington, DC, USA, 2011. IEEE Computer Society.
- [152] Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Context-based service recommendation for assisting business process design. In *E-Commerce and Web Technologies (EC-Web)*, volume 85 of *Lecture Notes in Business Information Processing*, pages 39–51. Springer Berlin Heidelberg, 2011. 10.1007/978-3-642-23014-1_4.
- [153] Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. Assisting business process design by activity neighborhood context matching. In *10th International Conference on Service Oriented Computing (ICSOC)*, Lecture Notes in Computer Science, Shanghai, China, Nov. 2012. Springer.
- [154] Niklaus Wirth. What can we do about the unnecessary diversity of notation for syntactic definitions? *Commun. ACM*, 20(11):822–823, November 1977.
- [155] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst. Discovering colored petri nets from event logs. *Int. J. Softw. Tools Technol. Transf.*, 10(1):57–74, December 2007.
- [156] Helen Schonenberg, Barbara Weber, Boudewijn Dongen, and Wil Aalst. Supporting flexible processes through recommendations based on history. In *Proceedings of the 6th International Conference on Business Process Management, BPM '08*, pages 51–66, Berlin, Heidelberg, 2008. Springer-Verlag.

-
- [157] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.
- [158] James Bennett, Charles Elkan, Bing Liu, Padhraic Smyth, and Domonkos Tikk. Kdd cup and workshop 2007. *SIGKDD Explor. Newsl.*, 9(2):51–52, dec 2007.
- [159] James Bennett and Stan Lanning. The netfix prize. In *Proceedings of KDD Cup and Workshop*, New York, NY, USA, 2007. ACM.
- [160] Dirk Fahland, Cédric Favre, Barbara Jobstmann, Jana Koehler, Niels Lohmann, Hagen Völzer, and Karsten Wolf. Instantaneous soundness checking of industrial business process models. In *7th BPM*, pages 278–293, 2009.
- [161] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18, 2010.
- [162] Liang-Jie Zhang and Qun Zhou. Ccoa: Cloud computing open architecture. In *IEEE International Conference on Web Services (ICWS), 2009*, pages 607–616, july 2009.
- [163] Heather Kreger, Michael Behrendt, Bernard Glasner, Petra Kopp, Robert Dieckmann, Gerd Breiter, Stefan Pappé, and Ali Arsanjani. Ibm cloud computing reference architecture 2.0, 2011.
- [164] Ingo Weber, Jörg Hoffmann, and Jan Mendling. Beyond soundness: on the verification of semantic business process models. *Distrib. Parallel Databases*, 27(3):271–343, June 2010.
- [165] Y. Liu, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM Syst. J.*, 46(2):335–361, April 2007.