Supporting service consumption : advanced discovery and recommendation techniques

Walid Gaaloul

Université Pierre et Marie Curie

Supporting Service Consumption: Advanced Discovery and Recommendation Techniques

submitted by

Walid Gaaloul

in fulfillment of the requirements for the degree of

Habilitation à Diriger des Rechecrhes

defended on the 19^{th} of September 2014 in front of the panel composed of

Reviewers

Fabio Casati	Professor	University of Torento, Italy
Marlon Dumas	Professor	University of Tartu, Estonia
Lionel Seinturier	Professor	University of Lille 1, France

Examiners

Bernd Amann	Professor	Pierre and Marie Curie University, France
Claude Godart	Professor	University of Lorraine, France
Mohand-Said Hacid	Professor	Claude Bernard University, France

Abstract

Abstract

The tremendous growth in the amount of available (Web) services impulses many researchers on proposing discovery, recommendation and management tools and techniques to help users retrieve services efficiently. Services can be consumed in different contexts: published in distributed registries; invoked as individual services which provide interfaces to receive inputs and return outputs; or composed and integrated into service-based processes as new value added composite services. In our work, we aim at facilitating service discovery and management in these three consumption contexts. First, we propose a functionality-driven approach by clustering and organizing registries according to the functionalities of the service they advertise. Second, to recommend services for individual use, we propose a usage-driven approach that takes into account the user usage data which reflect the user interest. Third, to recommend services for process use, we propose a composition-driven recommendation approach that takes into account the relations between services in service-based processes. We develop applications, as a proof of concept, to validate our techniques. We also perform experiments on the data collected by our applications and on large public datasets. Experimental results show that our techniques are feasible, accurate and have good performance in real use-cases.

Keywords: Service Recommendation, Service Discovery, Implicit Knowledge, Service-based Process Modeling, Process Model Reuse, Configurable Process Models, Process Mining.

Contents

Chapter 1

Introduction

1.1 Preamble

My research interests mainly focus on supporting **process (re)modeling** and **service consumption** using techniques and methods such us: mining, verification, discovery, recommendation, collaborative filtering, mediation, and configuration. This report focuses mainly on the latter topic (i.e. service consumption), summarizes parts of the results of my research over the past six years (2008 - 2014), and presents some ideas for future work that I consider interesting to explore in collaboration with my colleagues and students. My research activities were a natural consequence of my PhD thesis which I defended at the University of Lorraine in November 2006 [56]. Through this thesis I proposed to analyze process logs to discover workflow transactional behavior and to subsequently improve and correct related recovery mechanisms. Thereafter I was interested in the semantically-enabled management of service-based business processes [22]. This latter work was conducted between 2006 and 2008 while I was a postdoctoral researcher at DERI-NUIG (Galway, Ireland).

In October 2008, I joined TELECOM SudParis as an associate professor. I continued to work with my colleagues in Galway on developing mediators for supporting service interactions [178, 177]. Then I started a research topic around the management and discovery of data providing services [179, 137]. Between 2008 and 2012, I supervised two PhD students that worked on the development of techniques to support service consumption in different contexts: distributed registries, individual service, service-based process. While continuing to work on facilitating service consumption, the last two years I have started a research activity around (semantic) service-based process configuration [172, 6].

I would like to note that I have had the chance to work in a research team leaded by Samir Tata, and a department that was directed by Bruno Defude until 2013 then by Djamel Belaid who have always have confidence on me and give me great autonomy not only at supervising students but also the management and proposal of research projects. In fact, I conducted my research in conjunction with real application requirements, which are often concretized by research projects (Orange S3P (2012-1014), emundus GreenIT (2012-2014), ANR PAIRSE 2010-2012, and FUI CompatibleOne (2010-2012)).

1.2 Context

The last few years have seen a democratization in the use of Internet technologies, mainly Web services, for electronic B2B transactions. The evolution of communication networks and technologies have led to the explosion of services over the Internet. The importance of this market triggered an increase in the use of e-services to ensure electronic

B2B transactions. In this context, more and more companies are using Web services to achieve transactions with their partners and/or offer on-line services. For instance, in a Mckinsey Quarterly survey conducted on more than 2800 companies worldwide, 80% are using or planning to use Web services. Among these companies, 78% says that the Web service technology is among the three most important technologies to their business [152].

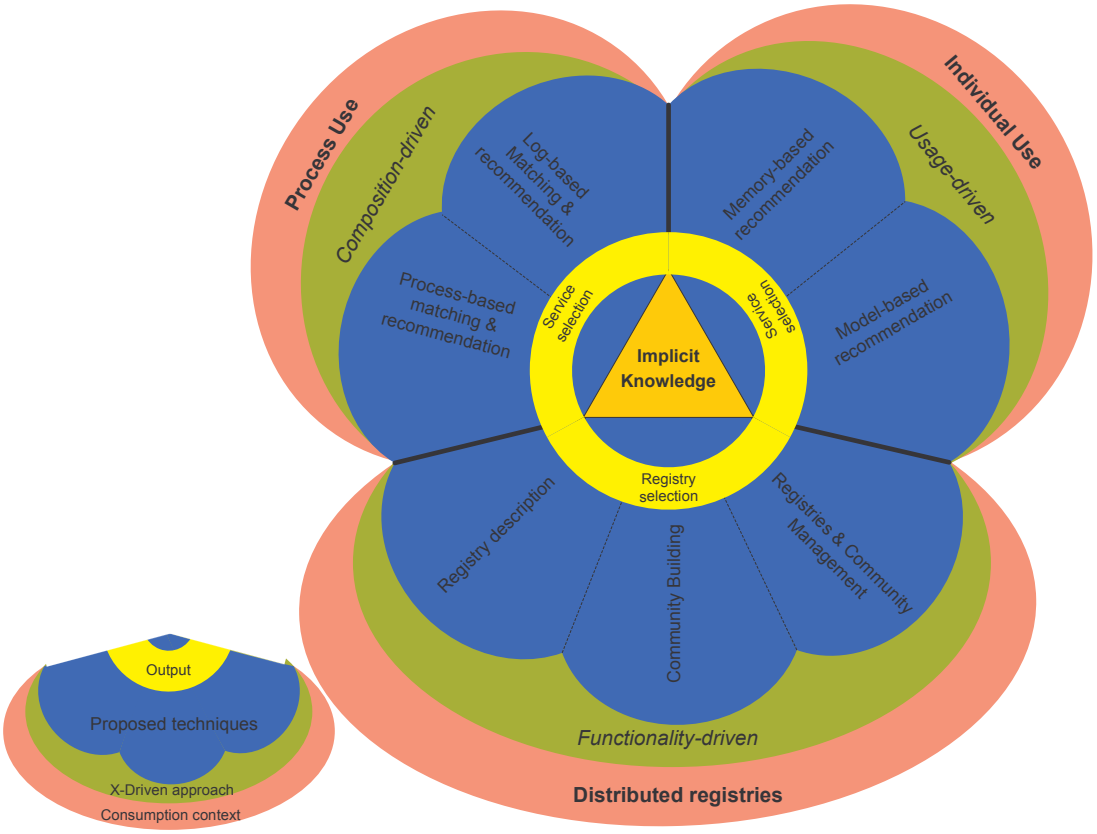
Service providers always compete to rapidly provide the best services to service requesters. This circumstance requires the development of service oriented technologies. Web services appeared as an attractive paradigm for publishing and consuming services. They have been developed as a standard technology to deliver services over the Internet. The Web service technology has been proven as an efficient mean to delivering services to users. 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 service based processes. For instance, individual Web services can be services for city codes, local temperatures, up-to-date news; and composite services (i.e. service-based processes) can be flight booking processes that compose other services such as customer authentication, online check-in, car rental, and payment to accomplish a flight booking transaction.

Service discovery is at the heart of SOA, enabling visibility between service providers and service consumers. The role of service discovery is to provide a mechanism that allows a service requester to locate a service offering an expected functionality. 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 requesters to search and invoke individual services. 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 the process design phase. However, service requesters (1) can easily get confused by the number of services returned by search engines and special crawlers and (2) do not know about the advantage of a service (i.e. decide which is the best) in comparison with others.

Currently, companies can make their services available for consultation through their own service registries. As a result, the number of service registries that are made available for use can be as many as the large number of companies. This raises an old, search engine, problem in a new form: discovery mechanisms of services are not efficient both in response times and quality of results [144]. Basically, a company interested in a service has to screen several registries to discover the service that best suits its needs. This task can be very cumbersome since the number of available registries, and also the services they advertise can be very large [18]. In this context, if appropriate solutions are not considered, "traditional" services discovery mechanisms that consist of scanning all the registries would for instance slow down the democratization of services.

Moreover, 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 business process can be implemented as a service composition that execute a set of services to achieve a business goal. The process design is the initial and key phase [48] of business process development as it helps to design the business process model, plan resources, identify new opportunities and foresee risks. However, designing a business process from scratch is always a labor-intensive and time-consuming task. Process analysts need tools and mechanisms to

Figure 1.1: Flower-shaped overview



facilitate the discovery of services composing their processes [147, 162].

Summarizing up, services can be published in a **distributed registry environment**. They can be consumed in two different ways as: (i) individual services which provide interfaces to receive inputs and return outputs; (ii) components to be integrated into business processes. We call the first consumption case **individual use** and the second case **(business) process use**. In such environment, one can consider service discovery as two successive steps: (1) registry selection and (2) service selection. Therefore, the requirement for specific tools to assist consumers in these different consumption contexts involves many researches in both academics and industry [67, 98]. Figure 1.1 gives an overview of the research activities described in this manuscript. Basically, I have developed with my students and colleagues a set of techniques and models to discover and recommend the most appropriate registry and service for different consumption contexts: distributed registries, individual use and process use. We propose respectively three different approaches: functionality-driven, usage-driven, and composition-driven. Our purpose is threefold: (i) providing a functionality-driven approach to manage and discover services in a distributed registry environment, (ii) recommending to users services that are close to their interest, (ii) recommending to process designers services that are relevant to a given composition context. All the proposed techniques share the following principle **exploiting implicit knowledge** : (i) our approach extract and utilize implicit knowledge hidden in usage data, service descriptions, process models, and process logs; (ii) our approach does not bother service consumers by asking them additional or complementary information. The proposed techniques make also a compromise between the computational complexity and the quality of results.

1.3 Contributions

1.3.1 Functionality-driven registry discovery and management

As the number of registries can be very large, registry discovery acts as an initial filter to detect the adequate registries to a requester query and thus reduce his/her search space. To deal with this problem and to address the large number of service registries and their poorly organized network, we propose to organize service registries into communities according to the functionalities of the services they advertise.

We use communities as a means for a functionality-driven organization of a distributed service registry environment. In the Web services research field, [17] define a Web service community as "*a collection of Web services with a common functionality although different non-functional properties*". [92] consider a community as "*a means for providing a common description of a desired functionality without explicitly referring to any concrete Web service that will implement this functionality at run-time*". In our approach, a service registry community is defined as a set of registries offering services providing similar functionalities.

To build communities of registries, we propose to associate to each registry a signature, that we call Web Service Registry Description (WSRD) [135], reflecting its functionalities. These descriptions are the basis for building the communities of registry. To build a network of communities of service registries, we propose to use a clustering technique rather than a classification approach. We also provide the management operations needed to guarantee the consistency of communities during their life-cycles. Below, we detail our motivations by answering these two questions: *Why do we use a functionality driven organization?* and *Why do we need to manage communities after their creation?*

Functionality-driven organization: In a service discovery process, a service requester is usually interested by a functionality that a service can offer. Therefore, it is more obvious and appropriate to organize service registries based on their offered functionalities [1]. Based on a user query, representing the required functionality, we can filter and reduce the search space to the community of registries advertising services offering the specified functionalities.

Need of management: Communities and service registries operate within a dynamic environment where changes are frequent. In fact, a new service description can be published in a registry and others can be unpublished at any time. In the same way, a registry can join a community or leave it according to its convenience. However, we cannot re-run our community clustering approach when changes occur mainly for cost reasons. We should only use our communities building approach as a "cold starter" for a registry network organization and so we have to define management operations to handle the dynamic aspects of communities.

1.3.2 Usage-driven service recommendation

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 services* returned by search engines or service crawlers. Moreover, they may not be aware about the functionality and quality of the returned services.

Intuitively, users need support to understand their interests and suggest them appropriate services. In this case, recommender systems (RS) [114, 167] can be a good solution as

1. Functionalities of a service registry and functionalities of the services it advertise are interchangeably used.

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. Most of them take into account data from *provider side* (such as service descriptions, QoS, semantic annotations, etc.). Few of them consider data from consumer side (profile, rating, comments, etc). Basically, very few take into account user's behavior which is an important parameter for finding services that are close to a given user interest. They mostly exploit *explicit knowledge* which is either represented by user ratings, semantic descriptions or service's QoS.

We aim at recommending services that are close to **user interest**. We propose a solution from the **consumer side**. We target to exploit **implicit knowledge** hidden in (service) usage data. We **do not ask** users **any effort** such as rating or comments. 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.

The recommendation in our algorithms is generated based on the similarities among WS operations and users. We store the usage data in terms of "user ID", "WS operation ID" and "number of used times". The historical data can be represented by an operation-user matrix $A_{m\times n}$, where m is the number of WS operations and n is the number of users in the system. Each entry $A[i, j]$ in this matrix presents the number of times that the user U_j used the WS operation O_i. It presents also the interest degree of U_j on O_i, which is very important data for our recommendation strategy. Our operation-user matrix is equivalent to the term-document matrix used to compute the similarities among users (and WS operations). Based on these similarities, we proposed algorithms to extract the suitable WS operations for each user.

1.3.3 Composition-driven service recommendation

From a process use perspective, process designers need specific tools that can understand the business context in order to rapidly find the most relevant services to integrate into the ongoing designed process. 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 [148, 35, 119], or finding existing similar models to inspire the process design [170, 1, 83, 40, 47].

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* [158]. Indeed, recommending *entire process models* costs much *computation time* and it can make business analysts *confused*, especially when the number of components services is large, e.g. hundreds of services and transition flows.

Process analysts *may need recommendations* for some *selected positions* instead of entire processes. For example, a process analyst is designing a service-based process as shown in Figure 1.2. The process designer is looking for services that are suitable to the missing position (i.e. share the same composition context described through the interactions with its service neighbors) 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 may also help 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.

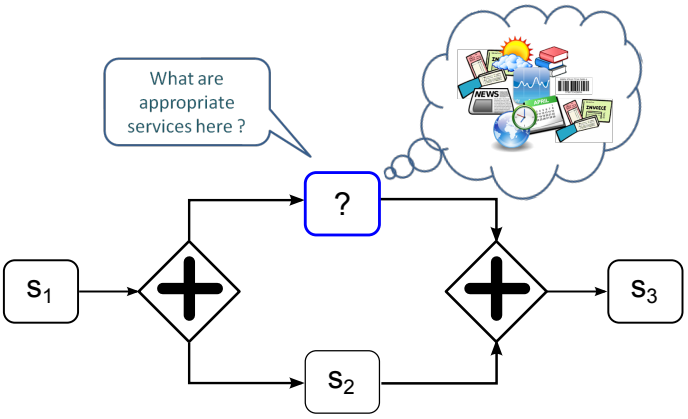


Figure 1.2: Composition context example

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. 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 composition context with the selected one. This context is defined as a business process fragment around the service and represents the composition context [2] of the component service. For a selected component service, we match its composition context with the composition contexts of other services from existing designed processes. The matching between two composition 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 target to exploit **implicit knowledge** hidden in business process models or logs. We use **existing data** (process models and logs) to make service recommendations. We *do not ask users any effort* to provide additional information. We take only into account relations between services in business processes.

1.4 Manuscript's organization

This thesis includes 5 chapters: In chapter 2, we present our model for formally representing a Web service registry, a community and a community network promoting a complete, independent and dynamic approach for service registries organization. In chapter 3, we present our solution to recommend services based on past usage data. In chapter 4, we elaborate our approach to recommend services to a chosen position in a business process based on their composition context matching. Finally, in chapter 5, we give an outlook to my research perspectives.

2. Neighborhood context and composition context are interchangeably used.

Chapter 2

Functionality-driven registry management

2.1 Introduction

More and more companies are using Web services for achieving transactions with their partners and/or offering on-line services. The widely used solution is to publish them using Web service registries available to Web service consumers. As a consequence, the number of Web service registries that are made available for use can be as many as the large number of companies. Basically, a company that requests a service has to screen registries of several companies to discover the needed service that best suits its request. This task can be very cumbersome since the number of available registries, and also the services they advertise can be very large [160, 18, 77]. In this context, if appropriate solutions are not considered, "traditional" Web service discovery mechanisms that consist of scanning all the registries would slow down service discovery and consequently the usability of Web services. The aim of the work we propose here is to tackle this issue by providing a new approach for Web service discovery in a distributed registry environment [141, 143].

To enhance response time and precision of Web service discovery, several approaches dealing with distributed registries [142, 145, 168] have already been proposed to structure their registry networks into several groups. As the number of registries can be very large, this organization can be used as an initial filter to target adequate registries or group of registries for a given query and thus to reduce the search space. Our work is inline with these approaches. It consists in (1) describing Web service registries using the functionalities of the Web services they advertise; (2) grouping Web service registries according to their descriptions and (3) using Web service registry groups to route queries to a limited number of adequate and interesting registries.

Broadly speaking, the description of Web service registries, that we call Web Service Registry Description (WSRD for short) is only based on the WSDL descriptions of their advertised Web services. Our WSRD computing is **automatic** and doesn't ask for any additional knowledge from registry providers. WSRD descriptions are then used for **functionality-driven** organization of Web service registries into registry groups that we call communities. Doing so allows grouping Web service registries into communities according to the functionalities of the Web services they advertise.

Since it is difficult to properly define in advance classes categorizing Web service registries in a distributed environment, we propose to organize Web service registries into communities using a **clustering** technique (where the different classes are deduced from the registry descriptions) rather than a classification technique (where the different classes

have to be defined in advance). In addition, the clustering we use is fuzzy (i.e. a registry can belong to more than one class) rather than hard clustering (where a registry is associated with only one class). Given a query for service discovery, our registry selection process is based on the matching of the query with the description of representatives of the different communities. This matching will results in the selection of a community of web service registries (the selection of more than one community is possible). Then the query will be matched against the Web services of the registries that belong to the selected community.

The main steps of our approach are summarized as follows:

1. Step 1: We characterize each Web service registry with a semantic WSRD description. This description is based on the descriptions of the Web services belonging to the considered registry and "semantically aggregate" its Web service functionalities. The registry description computing process is automatic and doesn't ask for any additional knowledge from a registry provider. A registry description is implicitly created using as only input Web service descriptions of that registry.
2. Step 2: We use WSRD registry descriptions to build communities of registries. Indeed, using the WSRD descriptions allows us to group into communities different Web service registries according to their offered functionalities.
3. Step 3: To handle the dynamic nature of communities and their members (i.e. Web service registries), we define the management mechanisms to monitor changes and reconcile potential conflicts. We identify the different operations of a registry (joining a community, updating functionalities,...) and a community (creation, dismantling, merging,...) life-cycle and we specify the associated management operations.

The rest of the chapter is structured as follows: Section 2.2 presents our registry description model (WSRD) and its computing process (Step 1). In Section 2.3, we show how we use the WSRD descriptions to organize a registry network as communities (Step 2). In Section 2.4 we introduce the management algorithms and operations for the registry and community life cycles (Step 3). The implementation, experimentation and usability of our approach are shown in Section 2.5. Section 2.6 discusses related work and Section 2.7 concludes the chapter.

2.2 Description of service registries

Our idea to describe a Web service registry consists in aggregating the WSDL descriptions of the Web services it advertises into one description called Web Service Registry Description (WSRD for short) [135]. Hence a WSRD of a registry can give an overview of the functional properties of its Web services.

In Figure 2.1, we introduce a graph representation of a WSRD description. Since a WSRD registry description is based on Web service descriptions published in that registry, the different nodes composing this graph are inspired by the WSDL format. As we provide a description of a registry (not of a service), we are only interested in the abstract section of the Web services description. To provide the semantic WSRD model, semantic descriptions of Web services are the only input that we use. In this work, we choose to use semantic Web service descriptions written in SAWSDL [79]. Other semantic languages, such as OWL-S [95], WSMO [5] or YASA [29] can be adopted.

WSRD defines a registry using the following WSDL elements: `interface`, `operation`, `Input` and `Output`. These elements give an abstract description of the mean functionalities offered by the Web services of a registry. We associate each WSRD element with a

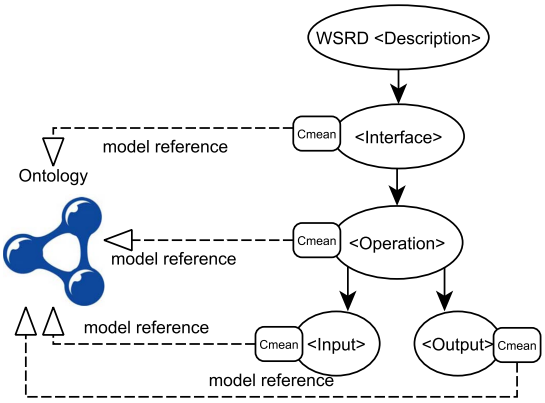


Figure 2.1: The WSRD model

concept taken from a semantic model (domain ontology (DO)) using the SAWSDL modelReference extension attribute. In our work, we suppose that service requesters share the same semantic stack. This is done through common ontologies or ontology mediation mechanisms if different ones are used. And since we are dealing with private registries belonging to a specific company, we assume that all the services advertised by a registry are homogenous in term of their business domain and semantics. In addition, since those services have the same provider, we suppose that they use the same semantic description language. Computing a registry's WSRD description goes through three steps:

- Step 1.1 : We first extract the annotating concepts and the number of times they occur from the Web service descriptions published in the registry (Section 2.2.1).
- Step 1.2 : We compute the groups of potential concepts, taken from the DO, to annotate the WSRD description (Section 2.2.2). To each concept we associate a value indicating its similarity degree to the whole set of extracted concepts.
- Step 1.3 : Finally, we reduce the computed concepts' groups to only keep the concept(s) that will be used to annotate a registry's WSRD description (Section 2.2.3).

2.2.1 Extracting the annotating concepts

The first step in a WSRD computing process is to extract the DO concepts annotating the different SAWSDL descriptions elements (i.e. `<interface>`, `<Operation>`, `<Input>` and `<Output>`) in the Web service registry (see Figure2.2). These concepts, as well as their number of occurrence constitute the "initial" WSRD description (Definition 2.2.1).

Definition 2.2.1 ("initial" WSRD)**.** *We define an "initial" WSRD as a quadruple* (I, O, In, Out) *of* hash maps[1]*. We call I (resp. O, In and Out) the* hash map *containing the extracted concepts of* `<interface>` *(resp.* `<operation>`*,* `<input>` *and* `<output>`*). These hash maps associate a value nb_i to a concept C_i where:*

- *C_i is the extracted annotating concept of an element (i.e.* `<interface>`*,* `<Operation>`*,* `<Input>` *or* `<Output>`*) of a SAWSDL Web service description.*
- *nb_i is the number of times the concept C_i was found in the corresponding description element of the Web service descriptions of the registry.*

1. A hash map is a data structure mapping some identifiers or keys to some associated values.
2. A hash map is a data structure mapping some identifiers or keys to some associated values.

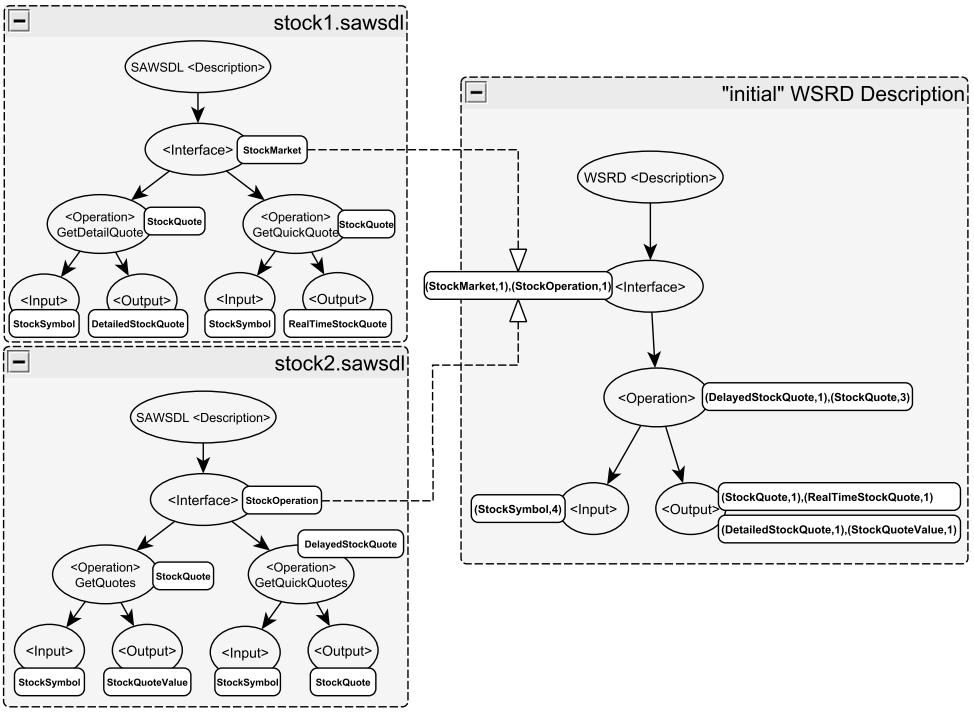


Figure 2.2: Concept Extraction

2.2.2 Computing groups of potential concepts

In this second step, we create a group of weighted concepts for each element of a WSRD description (i.e. `<interface>`, `<Operation>`, `<Input>` and `<Output>`). These groups contain all the semantic concepts of the used DO that we consider as candidate concepts for annotating a WSRD element. Each of these concepts is associated to a value s reflecting its similarity to the concepts extracted in step 1.1.

A value s_j, associated to a concept C_j in a group of weighted concepts, is computed using the couples (C_i,nb_i), stored in the "initial" WSRD, and the similarity degrees between the different semantic concepts (see formula 2.1). The formula, we propose, computes the average of the similarity factors ($Similarity[C_j, C_i]$) of C_j to the set of semantic concepts C_i extracted in step 1.1 weighted by nb_i (i.e. the number of times the concept C_i was identified). We use nb_i to weight the similarity factors since a concept identified many times is semantically more significant than a concept identified few times. A non identified concept is weighted by zero. Based on these data, we create the "intermediate" WSRD (Definition 2.2.2).

Definition 2.2.2 ("intermediate" WSRD)**.** *An "intermediate" WSRD is a quadruple* $(H_I, H_O, H_{In}, H_{Out})$*. We call* H_I *(resp.* H_O,H_{In} *and* H_{Out}*) the hash map used to store the set of potential* C_{mean} *for the* `<interface>` *(resp.* `<operation>`, `<input>` *and* `<output>`*) element. These hash maps will contain the different concepts* C_j *of the used ontology mapped to a value* s_j *representing the sum of the similarity factors between* C_j *and the different concepts associated to an element of the "initial" WSRD.* H_I *(resp.* H_O*,* H_{In} *and* H_{Out}*) is computed using the following formula:*

$$H_I[C_j] = s_j = \frac{\sum_{i=1}^{t}(I[C_i] \times Similarity[C_j, C_i])}{\sum_{i=1}^{t} I[C_i]}, For\ j = 1 \cdots t \tag{2.1}$$

Where:

- *t: is the number of concepts in the used ontology.*
- C_j, $j = 1 \cdots t$: *are the concepts of that used ontology.*
- *I (resp. O, In and Out): a hash map representing the extracted concepts of the* `<Interface>` *(resp.* `<operation>`, `<input>` *and* `<output>`*) element in step 1.1.*
- *Similarity: a matrix containing the similarity factors between all the concepts of the used ontology.*

In order to compute the similarity matrix, we are using the enhanced edge counting similarity measure proposed by [115] as it is simple and widely used. This measure computes the similarity between two concepts based on the number of edges found on the path between them. We are aware of the impact of the choice of the similarity computing method on our approach and we are studying this issue by testing other similarity methods [73, 84].

2.2.3 Reducing the concepts

In the third step, we aim at selecting the median concept(s) C_{mean} (from the group of weighted concepts of step 1.2) which are the most similar to the ones identified in step 1.1 (stored in the "initial" WSRD). The selected C_{mean} will be used to annotate the corresponding WSRD element. To each selected C_{mean} we associate a weight indicating its relevance in the annotated WSRD element vis-à-vis the other C_{mean}. The resulting WSRD is called the "final" WSRD and is defined in Definition 2.2.3.

Definition 2.2.3 ("final" WSRD)**.** *The "final" WSRD is a quadruple* (CM_{H_I} , CM_{H_O}, $CM_{H_{In}}$, $CM_{H_{Out}}$)*.* CM_{H_I} *(resp.* CM_{H_O}, $CM_{H_{In}}$, $CM_{H_{Out}}$*) is a hash map containing the* C_{mean} *annotating the WSRD* `<interface>` *(resp.* `<operation>`, `<input>` *and* `<output>`*) element associated with the value(s)* $s = H_I[C_{mean}]$ *(resp.* $H_O[C_{mean}]$, $H_{In}[C_{mean}]$ *and* $H_{Out}[C_{mean}]$*) indicating its representativeness.*

This reduction step can be achieved in two different ways : strong or weak.

- **Strong reduction**: This method consists in choosing a unique C_{mean} to annotate the associated WSRD element. A simple way of realizing strong reduction is to choose from its representative hash map $H_e = \{< C_i, s_i >\}$ the concept C_i having the highest value s_i. The chosen concept corresponds to the mean concept of the WSRD element.
- **Weak reduction**: By employing weak reduction a WSRD element will be annotated using more than one C_{mean}. The issues here are to define which and how many concepts to choose. In our work, we choose the C_{mean} using Algorithm 1. Following this algorithm, we first select from the groups of potential C_{mean} (H_e) the concept having the highest value $s{=}s_{start}$ as a C_{mean} (line 1-4). Then, we compute the absolute deviation σ between the values s of H_e (line 5). Then we choose another concept $C_{current}$ as C_{mean} such that its weight $s_{current}$ is less or equal than s (the weight of the previous chosen C_{mean} (line 6)) minus σ (line 10). This condition (the gap between the values of the chosen concepts is largest than the absolute deviation between them) allows us to avoid choosing two semantically similar concepts as C_{mean}. We then assign to s the value of the chosen concept and repeat the previous steps (line 7-12) until $s{-}\sigma \leq 0$. Algorithm 1 allows us to efficiently choose the WSRD annotating C_{mean} by eliminating the less representative concepts while ensuring that no lost of knowledge happened.

Algorithm 1 WeakReduction(H_e)

Require: H_e {The group of potential C_{mean} of a WSRD element e (i.e. e=interface, operation, input or output).}
Ensure: CM {Hash map of the weighted selected C_{mean}}

1: $H_e = Order(H_e)$ {Order: a function sorting in descending order the element C_i of H_e according to s_i}
2: $C_{start} = H_e.getElement()$
3: s_{start}=$H_e[C_{start}]$;
4: CM.add((C_{start}, s_{start}));{Select the concept having the highest value as C_{mean}}
5: σ =deviation(s_i) ;
6: $s = s_{start}$;{s is the weight associated to the last chosen C_{mean}}
7: **while** $(s - \sigma \geq 0)$ **do** {Find in H_e the concept having the highest value and distanced from s by σ}
8: $C_{current} = H_e.getElement()$
9: $s_{current} = H_e[C_{current}]$
10: **if** $(s_{current} \leq s - \sigma)$ **then**
11: CM.add($(C_{current}, s_{current})$);
12: $s = s_{current}$;
13: **end if**
14: **end while**
15: **return** CM

Strong reduction is a simple technique for concepts reduction. However, weak reduction better reflects the functionalities offered by the Web services of a registry. We recommend using weak reduction as the computation load is almost the same for both techniques [135].

2.3 Building of communities

In this section, we present the second step of our approach: based on WSRD descriptions computed in the first step, we organize a registry network into communities. We start by describing our model for representing a Web service registry, a community and a community network in Section 2.3.1. Then, we introduce our communities building approach in Section 2.3.2.

2.3.1 Modeling of communities

We define our distributed registry network based on the notations and concepts offered by graph theory. Indeed, graphs are highly flexible models for analyzing a wide range of practical problems, especially networks, through a collection of nodes and connections between them. A network is then formalized with a graph G, defined as a pair of sets $G = (V, E)$. V is the set of vertices (or nodes) and E is the edge set representing the network connections. Also, a graph can be weighted by a weight function $w : E \longrightarrow \mathbb{R}$ assigning a weight on each edge. A weighted graph is then denoted $G = (V, E, w)$. Throughout the rest of the chapter, we use these notations.

Modeling a Web Service Registry. In this work, we refer to each WSRD description of a Web service registry by f. A registry can belong to different communities at the same time. Thus, we assign to a registry a set of membership degrees that we call MEM. This set contains its membership degrees to each community in the network of communities (see Definition 2.3.1).

Definition 2.3.1 (Web service Registry)**.** *A registry is defined as a triple* $r = (id, f, MEM)$ *where:*

– *id is the registry identifier,*

- *f is a vector representing the average functionality offered by the advertised Web services within the registry,*
- *MEM represents the registry membership degrees to the different communities in the network. It is represented by a binary relation defined as follows.* $MEM = \{(c, d) | c \in C, d \in [0, 1]\}$ *where:*
  - *C is the community set,*
  - *d is the membership degree of the registry r to the community c.*

We define the domain and range of $MEM \subseteq C \times [0, 1]$ *as:*
- $dom(MEM) = \{c | (c, d) \in MEM \text{ for some } d \in [0, 1]\}$
- $ran(MEM) = \{d | (c, d) \in MEM \text{ for some } c \in C\}$

We also define the function U_j that computes the degrees of membership of a registry r_i to a community c_j as follows:
- $U_j(r_i) = d \Rightarrow (d, c_j) \in r_i.MEM$

Modeling a Community. A community is mainly characterized by its mean functionality f which represents the aggregation of the community registries functionalities. We distinguish two kinds of registries (*leader* and *follower*) based on their role inside a community. Therefore, the set of community members (nodes) can be divided into a singleton $L = \{l\}$ representing the *leader* and a set $Fl = \{fl_i | i : 1..n\}$ where n is the number of the community *followers*. Thus, the community nodes are modeled as a star graph G where nodes are registries and each edge represents the functional similarity between the *leader* and a *follower* fl, $fl \in Fl$. The similarity between the functionalities offered by the *leader* and a *follower* can be computed using the cosine function (see Section 2.3.2, formula (2.2)). A definition of a community of registries is given in Definition 2.3.2.

Definition 2.3.2 (Community of Registries)**.** *A community is a triple* $c = (id, f, G)$ *where:*
- *id is the community identifier,*
- *f is a vector representing the mean functionality of the community c,*
- *$G = (L \cup Fl, E, w)$ is an undirected weighted star graph where:*
  - *L is the community* leader *(the registry having the highest membership degree inside the community c),*
  - *Fl is the set of community* followers *such as* $L \cap Fl = \{\}$,
  - *$E \subseteq L \times Fl$ is the set of edges,*
  - *$w : E \longrightarrow [0, 1]$ is a weighting function, each weight represents the functional similarity between nodes.*

Modeling the Community Network. So far, our distributed registry environment which is a set of communities is modeled by a set of star graphs. As the number of registries (nodes) can be very large and a single registry can belong to many communities, the community management is a cumbersome task. To deal with this problem and to have a global view of the network, we define another graph CG, called *Community Graph*, in which nodes represent communities and edges are the relationships between them. If two communities have at least one registry in common, then there is an edge joining them. In this case, we compute the distance between the vectors f of these communities. This distance represent the weight of the edge relating these two communities and can be computed using formula 2.3. We present the model of our community network in Definition 2.3.3.

Definition 2.3.3 (Community network)**.** *The community network is represented by an undirected weighted graph* $CG = (C, E, w)$

- *C is a finite set of nodes. Each single node represents a community of registries,*
- *$E \subseteq C \times C$ is the set of edges (representing the relationships between communities),*
- *$w : E \longrightarrow [0, 1]$ is a weighting function representing the distance between two given nodes.*

2.3.2 Organizing a registry network as communities

A community of Web service registries will bring together registries offering similar functionalities. Since a Web service registry generally offer services proposing different functionalities, it is difficult to properly define in advance classes categorizing the functionalities of the different registries. To organize Web service registries into communities, we use a clustering technique (where the different communities will be deduced from the registry descriptions) rather than a classification technique (where the different communities have to be defined in advance). When using a dynamic clustering technique, the different clusters (i.e. the communities of Web service registries) will be identified from the given input data (i.e. the WSRD descriptions) and each data point (i.e. Web service registry) will be assigned to one community.

Since a registry can belong to different communities at the same time, the use of an exclusive clustering is inadequate for building communities of registries. In exclusive clustering, data are grouped exclusively. Thus, if a certain datum belongs to one cluster then it is automatically excluded from the others. Therefore we propose to use an overlapping clustering method, also known as fuzzy clustering [173], to organize our distributed registries into communities. In the following, we present our fuzzy clustering approach to organize our distributed registries into communities which was inspired from the fuzzy C-means method [49, 21]. The fuzzy C-means is a method of unsupervised clustering often used in the fields of data analysis and pattern recognition. In several works [74, 130], this method is employed for document clustering. A document contains terms and is represented by a vector which dimensions are the document's terms. By using fuzzy C-means to cluster a set of documents, each document belongs to two or more clusters. Concretely, we proceed in three steps to build communities of registries:

1. Step 2.1: WSRD descriptions are processed to map to the term-document structure.
2. Step 2.2: we propose a method to measure the distance between two registry vectors as needed by the fuzzy C-means Algorithm.
3. Step 2.3: using the results of the two previous steps we apply a fuzzy clustering technique to build communities of registries.

Step 2.1: Defining the registry vector space

In order to apply the fuzzy C-means method to our context, we consider each *registry as a document* and the set of its *ontology annotating concepts as terms* in the document. The data points are Web service registries r_i represented by their WSRD descriptions. We use the vector space model [128] to represent these descriptions as vectors. Each WSRD description will be represented by a vector $r_i.f = [w_1, w_2, \ldots, w_t]$ where t is the number of concepts in the ontology. The weights of the different w_i are computed as follows:

1. For the `<interface>` (resp. `<Operation>`, `<Input>` and `<Output>`) element we extract the associated weight (see Section 2.2) with each C_{mean} and we store these values in a vector $v_I = [e_1, e_2, \ldots, e_t]$ (resp. v_O, v_{In} and v_{Output}). These vectors have the same size t as the vector $r_i.f$. If a concept from the common ontology does not occur in the WSRD description, its value in the vectors is zero.

2. Since v_I, v_O, v_{In} and v_{Out} have different meanings, they should be associated with weights. We define four weights: α for v_I, β for v_O, δ for v_{In} and λ for v_{Out} such that $\alpha + \beta + \delta + \lambda = 1$.
3. Finally, a registry WSRD vector $r_i.f$ is computed as follows: $r_i.f = \alpha \times v_I + \beta \times v_O + \delta \times v_{In} + \lambda \times v_{Out}$

The coefficients α, β, δ and λ depend on the weights that the service consumer or provider want to give to the interface, the operation, the input or the output of a service. Bigger the coefficient is, more important the related element is considered for the WSRD computation. Normally, different value setting for α, β, δ and λ should impact the clustering result to some extent. Therefore service consumer and provider should agree on these coefficients based on the given environment. Indeed, if we have a capability centric service [175] then a greater weight should be given to the interface. Else, a greater weight should be given to the input/output if we have a data-centric service [180].

Step 2.2: Computing registry distance

A distance measure is used to establish the degrees of membership of each data point to the different clusters. In our work, the set of data points to cluster (the Web service registries) are represented by vectors computed from the WSRD descriptions. To compute the distance between two vectors, we use the cosine similarity measure [126] to establish the similarity between two vectors since it is adequate for high dimensional data [111]. The cosine function ranges between 0 (no similarity between vectors) and 1 (identical vectors). Given two vectors $r_1.f$ and $r_2.f$ that represent the functionalities of two Web service registries, the cosine similarity is computed using formula (2.2):

$$cosine(r_1.f, r_2.f) = \frac{r_1.f \times r_2.f}{\|r_1.f\| \times \|r_2.f\|} \tag{2.2}$$

Since the cosine function leads to the similarity between two vectors, and not to a distance as needed for the fuzzy C-means algorithm, we use Formula (2.3) to deduce the distance from the cosine similarity function:

$$distance(r_1.f, r_2.f) = 1 - cosine(r_1.f, r_2.f) \tag{2.3}$$

Step 2.3: Building communities

To build communities of Web service registries, we use a clustering technique. We proceed in two repetitive phases: (i) computing the communities' centers also called centroids (represented by f the mean functionality of the community) and (ii) assigning the different registries to these centers. Each registry is assigned to a community with a various degree of membership. The total sum of the membership's values for a registry should be equal to 1.

The Web service registry clustering is based on the minimization of the following objective function (2.4):

$$J = \sum_{j=1}^{C} \sum_{i=1}^{N} [U_j(r_i)]^m distance(r_i.f, c_j.f)^2 \tag{2.4}$$

where C is the number of communities, N the number of registries to organize, U_j the membership function of the community j, r_i is the i^{th} registry of our registries set, m (the fuzziness coefficient) is any real number greater than 1 and *distance* is our defined distance measure giving the similarity between a registry r_i and a community c_j. The clustering is carried out through an iterative optimization of the function J until it tends to be stabilized. The update of the membership $U_j(r_i)$ vectors and the community centroids $c_j.f$ is done by respectively using Formulas (2.5) and (2.6).

$$U_j(r_i) = \frac{1}{\sum_{k=1}^{C} (\frac{distance(r_i.f, c_j.f)}{distance(r_i.f, c_k.f)})^{\frac{2}{m-1}}} \tag{2.5}$$

$$c_j.f = \frac{\sum_{i=1}^{N} U_j(r_i)^m \cdot r_i}{\sum_{i=1}^{N} U_j(r_i)^m} \tag{2.6}$$

After this step, we get C communities represented by their centroids. These centroids represent the mean functionality f of these communities in accordance with Definition 2.3.2. In addition, to each community c_j are associated N membership vectors $U_j(r_i), i = 1 \dots N$ indicating the degrees of membership of the N registries to the community. These data allow us to infer the membership degrees MEM (see Definition 2.3.1) of the N registries to organize them into communities.

However, using this technique, the membership degrees of some registries to some communities may be very low. We thus define a threshold th for the membership degrees. If the membership degree of a registry to a community is below this threshold, it will not be considered as a member. This threshold is necessary to preserve the "reputation" of a community and to ensure that the functionalities of its members are not too different than those announced by the community.

2.4 Management of communities

Communities and Web service registries operate within a dynamic environment where changes are mainly initiated by service and registry providers. A service provider can publish or delete a Web service. Similarly, a registry provider can register its Web service registry or dismantle it at any moment. To keep the consistency of our communities' network against these events, management operations are needed (step 3 of our approach) to handle Web service registries (Section 2.4.1) and communities of Web service registries (Section 2.4.2) during their life-cycles. Due to space limitation, we refer to [26] for more details about the different algorithms we proposed to implement theses management operations related to the the registry and community life-cycle.

2.4.1 Registry life-cycle

Figure 2.3 illustrates the communities and registries management process. This process does not include communities building as it is a "cold starter" step and executed once in our approach. A registry life-cycle can start when a registry provider decides to register its Web service registry in the community network (operation (1) in Figure 2.3).

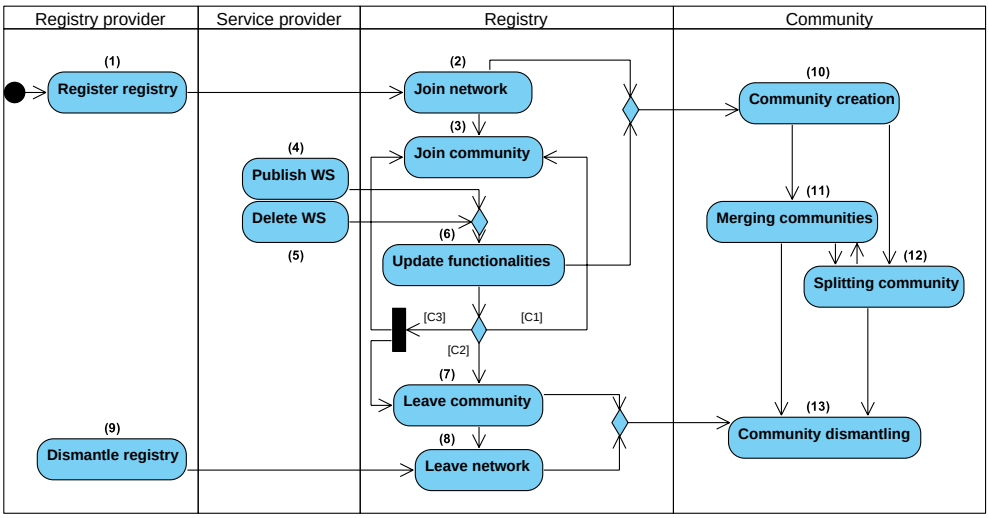


Figure 2.3: Communities and registries management process

In addition, service providers can publish (operation (4)) or delete (operation (5)) Web services within their registries thus leading to an update in the registry's mean functionality (operation (6)). In such scenario, the membership degrees of the updated registry to the existing communities can change and a suitability check of the registry memberships should be done. Finally, a registry can leave the whole network (operation (8)) if its provider decides to dismantle it (operation (9)).

2.4.2 Community life-cycle

The main steps describing a community life-cycle revolve around community creation, dismantling, merging and splitting. A community will be dismantled (operation (13)) if it becomes empty. Throughout a registry life-cycle, we check the similarity inside and between communities to ensure the principle goal of clustering: minimizing the similarity between clusters while maximizing it within each cluster. To guarantee this goal, a community can be merged (operation (11)) to another one or split (operation (12)) into two communities.

2.5 Validation

Our communities construction and management approach has been adopted in the French ANR funded research project PAIRSE[2] [179]. The PAIRSE project deals with issues (heterogeneities, query processing ...) related to data sharing in P2P environments by using data providing (DP) services. A DP service is particular type of Web services that only allows data access.

DP services clustering and categorization is important for locating an appropriate service, when a user needs to (i) discover a service that can fulfill her requirements, or (ii) to replace a service involving a given interaction when this service disappears or is unavailable. For this purpose, we adapted our communities construction and management approach to organize DP services described by RDF views [180]. Concretely, DP services were represented in terms of vectors while considering the composite relation between input, output, and semantic relations between them. Thereafter, DP service vectors are

2. `https://picoforge.int-evry.fr/cgi-bin/twiki/view/Pairse/Web/`

clustered using a refined fuzzy C-means algorithm. In addition, we have adopted some community/registry management operations for managing service clusters and the cluster network when handling the following situations: new service emergence, and existing service disappearance or unavailability.

In the following, we present the experimentation efforts made to validate the different steps of our approach. Our experimentation work was achieved in two stages: preparing the test-bed, and experimenting the efficiency of our approach by selecting the right communities for Web services queries.

Creating the Test collection. We start by creating a test collection of semantic Web service descriptions written in SAWSDL. We adapted a semantic Web services description generator introduced in [29] and adopted it to create a collection of SAWSDL service descriptions. A Web services description generator introduced in [105] has characterized realistic Web services. We have followed this work to generate a corpus of services with a similar distribution of occurrences of signatures. These service descriptions are annotated using concepts from an ontology described in OWL. For our experimentation, we generated a collection of 1400 SAWSDL descriptions. The generated test collection also includes seven queries $Q_i, i = 1 \dots 7$ and the relevance sets (queries responses) associated to these queries. Each relevance set contains 100 SAWSDL descriptions. We split up the generated SAWSDL descriptions into 7 Web service registries $R_j, j = 1 \dots 7$. Each registry R_m contains the 100 descriptions of the relevance set of the query Q_m in addition to 100 randomly generated descriptions.

Computing the WSRD descriptions. We compute for each registry the associated WSRD descriptions using the WSRDGen tool [3] that we implemented. WSRDGen performs the three steps (see Section 2.2) of our WSRD computing approach and experiments show that our approach is usable in realistic situations. For example, processing the WSRD description of a registry advertising 1400 Web services is done in 0.65 s (see Figure 2.4).

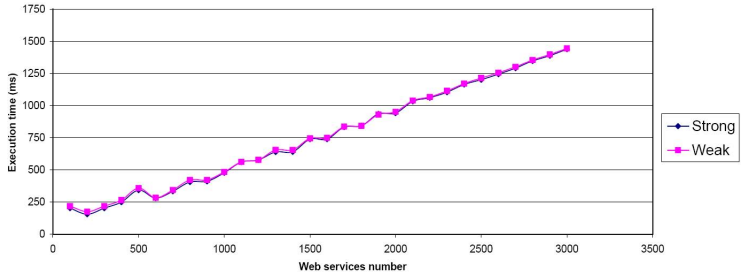


Figure 2.4: Execution time for WSRD description computing

Building the communities. After that, we applied our clustering approach to build the communities of registries. We implemented a *Community Builder* using the Fuzzy Clustering and Data Analysis Toolbox [4]. We used the 7 WSRD descriptions and transformed them into 5-dimensional vectors according to our vector space (Section 2.3.2). After that, we partitioned them into 4 communities.

We pushed further our experiments by testing our community building approach on a greater test collection. We used a set of 100 generated WSRD descriptions and transformed them into 6-dimensional vectors. We partitioned our data set into 5 communities and we present the clustering results in Figure 2.5. Since we can only graphically visualize our

3. `http://www-inf.it-sudparis.eu/SIMBAD/tools/WSRDGen/`
4. `http://www.abonyilab.com/clustering`

results in 3 or 2 dimensional graphs, the N (6 in our case) dimensions were devised in two 3-dimensions graphs (Figure 2.5) [5].

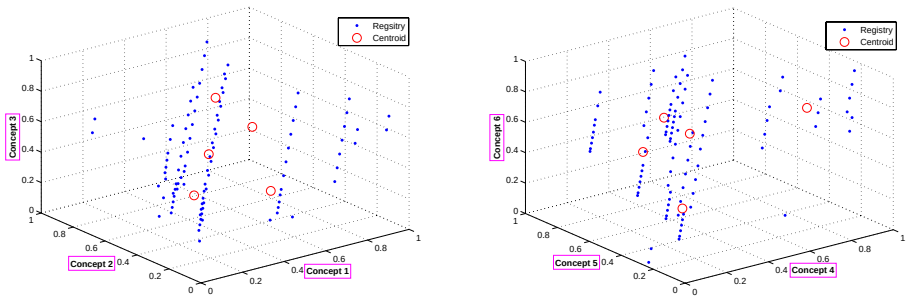


Figure 2.5: Fuzzy clustering of 100 WSRD registry descriptions into 5 clusters: Axis XYZ represent respectively the weights of (w_1,w_2, w_3) and (w_4,w_5, w_6) in a registry's vector

While creating the vector representation of a WSRD description, we considered all the semantic concepts of the used DO to provide an accurate representation. This choice can theoretically lead to "relatively" [6] high dimensional vectors (number of dimensions equal to the size of the DO). However, these vectors are sparse and do not lead to scalability issues since the WSRD's information that we represent are not related to all the semantic concepts. To test the scalability of our clustering approach while dealing with high dimensional data, we applied our clustering approach on the 100 generated WSRD descriptions and considered the execution time. In this experiment, the size of the WSRD vectors was increased from 10 to 100. Figure 2.6 shows that the clustering time is linear with respect to the vectors' size.

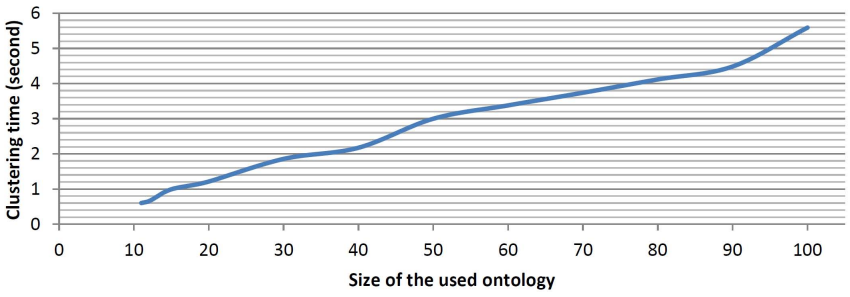


Figure 2.6: Relation between size of WSRD vectors and clustering time

It is worth mentioning that in a real world scenario, the size of WSRD vectors (i.e. equal to the size of the used DO) can be hundreds, but should not be thousands normally. In addition, in our work, we use the clustering technique only once as "cold starter" to build the communities and we use management operations to guarantee the evolution of the organization. Consequently, the performance of our clustering approach is satisfactory in real situations.

5. Further details on our clustering results can be found at `http://www-inf.it-sudparis.eu/SIMBAD/tools/FCM_results/`

6. Very high dimensional vectors can be observed while clustering text documents where, if a word frequency vector is used, the number of dimensions is equal to the size of a dictionary.

2.6 Related Work

In this section, we divide our literature review into the following parts: services organization (Section 2.6.1), registries organization (Section 2.6.2) and communities management (Section 2.6.3).

2.6.1 Services organization

Some Web services discovery approaches have used Web services WSDL descriptions to group Web services in order to enhance the discovery process. In [89], homogenous Web service communities are built where, a community contains a set of services providing either similar operations or potentially composable operations, with respect to a given request. Concretely, concepts are clustered from terms appeared in inputs and outputs. The similarity of service operations is computed as the combined similarity of (i) the text description and (ii) the parameters of inputs and outputs. Service Aggregation Graph (SAG) is constructed where: (i) each vertex represents an operation, and (ii) each edge reflects an output and input link between operations. To summarize, this approach aims to apply SAG to represent potential links between operations of Web services. So service discovery can be conducted by searching SAG and returning either a single operation or the shortest path of operations. Whereas in our work we group Web service registries into communities, where a Web service registry may represent several Web service operations. We aim to group similar registries into a same community, rather than to explore the links between Web services.

Another Web services organization approach is proposed in [46]. In this work, Web services parameter names defined in service operations are grouped into semantically meaningful concepts. Then, these concepts are used to measure the similarity of service operations by applying TF/IDF (i.e., term frequency/inverse document frequency) on the bag of words. This approach considers mostly parameter names, and does not deal with the composition problem. Generally, this approach aims to discover services based on their parameter names.

To summarize, compared to these techniques, we share the same methodology through using the Web services functional descriptions as an organization criterion. However, the main differences are:

- We are not organizing single services but rather atomic sets of Web services (i.e. Web service registries). These registries can advertise many functionally different Web services. So, in order to ensure this organization, we proposed a semantic model (WSRD) reflecting the mean functionality of a set of Web services and used it as a criterion unlike the work in [89, 46] where the organization is based on WSDL descriptions.
- We consider semantically annotated descriptions and uses the semantic concepts annotating the Web service descriptions to compute the similarity between registries. In [89, 46] the similarity measures are based on syntactic terms from WSDL descriptions specified by different Web services providers and can thereby be ambiguous.
- We consider the combined relation between service elements, and provide methods for handling the cases of new service/registry emergence or existing service/registry disappearance.

2.6.2 Registries organization

Several Web services discovery systems rely on a distributed registry environment to overcome the problems related to single centralized registry based discovery (bottlenecks, single point of failure ...). Our literature review yielded a good number of research projects that used distributed registries such as [160, 141, 11, 108, 169], but as far as we know, none of them have promoted the idea of using a functionality based semantic description model to organize Web service registries.

[160] propose a distributed Web service registry infrastructure for Web services discovery. In this work, the registry network is structured into federations using a specialized ontology called the *Registries Ontology*. This ontology allows grouping different Web service registries based on their business domain as the authors map each registry to one or several nodes in the *Registries Ontology*. Associating with each registry a specific domain or a group of domains from the *Registries Ontology* enhances Web services discovery. A Web service requester query will not be spread across all the registries since the query can be directly routed to the more adequate registry according to the requester's domains of interest. In [108], the authors also propose a distributed registry infrastructure called PYRAMID-S, where Web service registries are categorized by concepts token from a specialized ontology. This categorization allows selecting the adequate registry for a service requester query.

In previous work [141], we introduced the concept of requester characterization: a data structure containing a service requester's areas of interests, invocation history and non-functional requirements. A global registry characterization, which is the fusion of past requester characterizations who successfully discovered a service in that registry, is associated with a Web service registry. Using recommendation techniques, one or several registries can be advised to the service requester according to his characterization. A global registry characterization describes a Web service registry based on the consumers who used services it advertise but do not give an idea to the real functionalities it offers.

Authors in [11] present a super peer Web service discovery architecture. This architecture is based on a semantically clustered P2P network of *registry peers* (repository of Web service descriptions). Each *registry peers* cluster is indexed by a super peer called the *index peer* that stores the index information of the *registry peers* in a tree-based search data structure. In addition, the *index peers* are connected to each other with the CAN [112] routing protocol. In this architecture, a received search query will be routed by the *index peers* to the adequate *registry peer*. Using the CAN routing protocol avoids useless messages flooding the P2P network.

In [169], the authors use the match history to route a requester's query to an adequate registry. A service routing table is introduced into each registry node to store the history of matches. A services requester's query will be routed to a registry according to past requests and their matches, stored in a registry's routing table.

To sum up, our literature review yielded a good number of research projects that used distributed registries, but as far as we know, none of them have promoted the idea of using a functionality-based semantic description model to organize Web service registries. Compared to the above literature review, our contributions can be summarized in three main points:

1. Our registry description and organization is based only on implicit knowledge using existing advertised service descriptions. Thus, our approach is self-contained within the web service discovery process, independent from any explicit or human centric or error prone knowledge. Whereas other classification approaches [108, 160] ask for

additional explicit knowledge such as user's classification or service reputation and rating which can be hard to be captured.

2. Our approach proposes a functionally-based organization of Web service registries into communities. Such an organization enhances the Web services discovery process since the registries will be grouped according to the functionalities proposed by their advertised services, and thus a service requester's query can be guided to the adequate registry cluster for his needs. Such a query routing mechanism is radically different from those used in the previous approaches. Indeed, their query routing is based on registry users characterizations [141], the registry's business domains [160, 108], Web services business domains [11] or previous discovery results [169].
3. Due to the inherent autonomy, continuous and unforeseeable evolution of Web services description, Web service registries operate within a dynamic environment. By fuzzy clustering Web service registries according to their service descriptions, our approach has intrinsically the means for a dynamic, flexible and automatic management of Web service evolutions. Indeed, WSRD can be updated whenever the service provider can publish, delete or modify a web service description in order to be in line with the services they advertise. Whereas existing distributed registry environments [108, 141, 160] are not suitable for the management mechanisms. Indeed most of them are characterized by a rigid and a priori registry classification and organization which hamper dynamic management mechanisms.

2.6.3 Communities management

Several Web service discovery approaches in distributed registry environments (see Section 2.6.2) organize their networks as groups but did not provide the management mechanisms for these groups [16]. In this section, we overview some related efforts in the field of managing e-catalogs communities [107] and Web services communities [96, 92] that helped us tailor our approach.

[107] present the WS-catalogNet framework allowing to group e-catalogs into communities, build relationships between them and manage them constantly. An e-catalog is defined as a set of products organized based on a categorization. An e-catalog community is a set of e-catalogs having similar domain. The system offer monitoring functionalities and managing operations to restructure a community network according to the user interaction. Therefore, authors model the community network and then specify preconditions and effects for each operation based on the model they have defined [106]. However, the specified management operations, in particular merging and splitting communities, are not applicable in our context. Indeed, the members of a community (catalogs, Web services) have exclusive memberships while in our work a registry can belong to one or more communities.

[96] propose an approach to organize Web services into communities depending on their domain of interest. A community is an instance of an ontology meta-data called community ontology and is described by a set of generic operations. In this context, community providers can add, delete or modify some generic operations. Service providers, in turn, can delete a WS from a community or make its operations temporarily unavailable. Thus, authors propose a P2P approach to manage these changes.

[92] discuss the dynamic nature of Web service community and focus on potential conflicts. They propose an approach to engineer Web services communities in order to reconcile these potential conflicts. This approach is based on two protocols. The first one is called Community development protocol (WSCDProtocol) and is interested in managing

communities in term of attracting and retaining Web services, creating and dismantling communities.

The aforementioned research works employ a classification technique to organize communities, while we use a dynamic clustering technique. Furthermore, only [92] use the functionality criterion to structure communities. The other works use rather the business domain. While studying these research works, we noticed that the community management is generally established after a change initiated by end-users or service or community providers. To address conflicts that may result due to these changes, all these works propose managing operations. To facilitate the specification of these operations, [107] model their community network based on graph theory. The other works didn't propose a model and their descriptions are rather informal.

2.7 Conclusion

In this chapter, we presented an approach for building and managing communities of Web service registries. These communities are implicitly and automatically created using the registries WSRD descriptions. Compared to existing distributed registries organization approach, our approach uses a functionality-driven clustering technique and organizes registries according to the functionalities of the service they advertise. This functionality-driven organization of a registry network enhances Web service discovery. In fact, the search space can be reduced to the community of registries advertising Web services offering the required functionalities. We defined the required communities and community member management operations to maintain the consistency of the communities. We also experimented the efficiency of our approach in selecting the right communities for a Web service query, simulated a network of registry communities to test our management approach and provided a use case to show the feasibility of our communities' construction and management approach.

This work was conducted during the PhD thesis of Mohamed Sellami [134]. The community management has been extended and refined in the context of the Master thesis of Olfa Bouchaala [25]. This work was mainly published in the following conference proceedings and journals [136, 179, 138, 137, 140, 139, 141].

Chapter 3

Usage-driven Recommendation

3.1 Introduction

Web service discovery in a distributed registry environment have been most often tackled as two distinctive steps: (1) registry discovery and (2) Web service selection. In Chapter 2, I described our work for functionality-driven discovery and management of distributed service registry. In this chapter, I detail our work for recommending services based on consumer usage. The tremendous growth in the amount of available web services impulses many researchers on proposing recommender systems to help users discover services. However, most of the proposed solutions analyze query strings and web service descriptions to generate recommendations. They take into account data from provider side such as Web service descriptions, QoS and semantic concepts of services. However, these text-based recommendation approaches depend mainly on languages and notations which may decrease recommendation's efficiency. They require explicit knowledge presented by service descriptions 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.

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. The term "collaborative filtering" (CF) was firstly coined by the developers of the Tapestry recommender system [58]. 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 [59]. 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. Basically, CF techniques can be classified into three basic categories: *memory-based CF*, *model-based CF* and *hybrid CF* [150].

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.

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 [127]. Detail of our algorithms is presented in section 3.2.1 and section 3.2.2. We propose also an algorithm based on hybrid CF technique that combines the two previous memory-based CF methods: *user-based top-N* and *item-based top-N*(see section 3.2.3).

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. 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) [70, 37] 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 (see section 3.3).

3.2 Memory-based recommendation

3.2.1 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. The key step of the algorithm is finding the similarity between a service s_i and another service s_x. To compute this similarity, we apply the vector space model (VSM). VSM is firstly introduced by Gerard Salton et al. [128]. It is developed to compute the similarity between two individual documents. It presents documents in a k 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 3.1.

$$\begin{aligned} w_{i,j} &= tf_{i,j} \times idf_{j,S} \\ &= \frac{a_{i,j}}{\sum_{k=1}^{n} a_{i,k}} \times \log \frac{m}{|s_t \in S : a_{t,j} > 0|} \end{aligned} \tag{3.1}$$

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.

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 3.2).

$$sim(s_a, s_b) = \frac{\overrightarrow{w_a^u} \times \overrightarrow{w_b^u}}{|\overrightarrow{w_a^u}| \times |\overrightarrow{w_b^u}|} \tag{3.2}$$

where $\overrightarrow{w_a^u}$, $\overrightarrow{w_b^u}$ are the weight vectors of services s_a and s_b respectively, $\overrightarrow{w_a^u} = \{ w_{a1}, w_{a2}, \ldots, w_{an} \}$, $\overrightarrow{w_b^u} = \{ w_{b1}, w_{b2}, \ldots, w_{bn} \}$, $w_{ak}, w_{bk} \in W^u_{[m \times n]}, k = 1..n$, and n is the total number of users.

To generate recommendation for a a given WS operation, we apply (3.2) to find its most similar WS operations. Then, we sort the similarities in descending order and select the l WS operations which have the highest similarities values for the recommendation. 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.

3.2.2 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 3.3.

$$\begin{aligned} w_{i,j} &= tf_{i,j} \times idf_{i,U} \\ &= \frac{a_{i,j}}{\sum_{k=1}^{m} a_{k,j}} \times \log \frac{n}{|u_t \in U : a_{i,t} > 0|} \end{aligned} \tag{3.3}$$

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 3.3 on the usage matrix, we get a weight matrix $W^s_{[m\times n]}$ 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 3.4.

$$sim(u_x, u_y) = \frac{\overrightarrow{w^s_x} \times \overrightarrow{w^s_y}}{|\overrightarrow{w^s_x}| \times |\overrightarrow{w^s_y}|} \tag{3.4}$$

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

We generate recommendations in three steps algorithm. Firstly, we compute the similarity between the active user and others based on their usage data. Secondly, we sort other users in descending order of similarity and select the top-k users in the list. Finally, for each selected user, we select the t-most-frequently-used services that were not used by the active user to make recommendations.

In the last step of our algorithm, we select the t-mostly 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.

3.2.3 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.

Concretely, the *service-user combination algorithm* 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. 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. 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.

3.3 Model-based recommendation

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.

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 [38, 37]. It applies the Singular Value Decomposition (SVD), which is a factorization algorithm to decompose a rectangle matrix into three matrices. They transform the original user-matrix to a new approximate matrix by removing unrepresentative or insignificant users or items. The original matrix is equal to the multiplication of these matrices. The mathematical fundamental of SVD and its computation are explained in [37, 19, 20, 60].

Basically, a matrix $A_{[m\times n]}$ can be decomposed into three matrices $U_{[m\times n]}$, $\Sigma_{[n\times n]}$ and $V^T_{[n\times n]}$ using SVD. This decomposition is given by Equation 3.5.

$$A_{[m\times n]} = U_{[m\times n]}\Sigma_{[n\times n]}V^T_{[n\times n]} \tag{3.5}$$

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 $\overrightarrow{\sigma}$, we will have $\overrightarrow{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, $\sigma_i > 0$ for $1 \leq i \leq r \leq n$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_n = 0$. $r > 0$ is called *rank* of A.

As $\sigma_{r+1} = \ldots = \sigma_n = 0$, the rows and columns $(r+1)^{th}, \ldots, n^{th}$ 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 $V^T_{[n\times n]}$, the multiplication of these matrices also yields to the original matrix (Equation 3.6).

$$A_{[m\times n]} = U_{[m\times r]}\Sigma_{[r\times r]}V^T_{[r\times n]} \tag{3.6}$$

On the other hand, as elements on the diagonal of $\Sigma_{[n\times n]}$ are sorted in descending order of their values ($\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$), 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 $V^T_{[r\times n]}$, the multiplication of $U_{[m\times k]}$, $\Sigma_{[k\times k]}$ and $V^T_{[k\times n]}$ will yield a matrix $A^k_{[m\times n]}$ that is approximated to $A_{[m\times n]}$ (Figure 3.1, Equation 3.7).

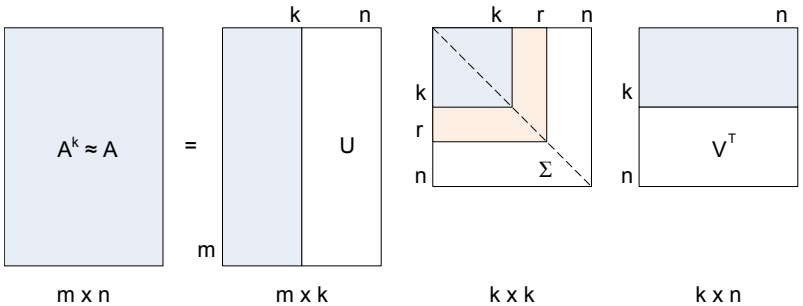


Figure 3.1: Decomposition in k dimensions [20]

$$A_{[m\times n]} \approx A^k_{[m\times n]} = U_{[m\times k]}\Sigma_{[k\times k]}V^T_{[k\times n]} \tag{3.7}$$

Assume that $A_{[m\times n]}$ is a service-user usage matrix. The derived $A^k_{[m\times n]}$ 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 $\overrightarrow{q}_{[m]}$ in the k-dimensional space is represented by Equation 3.8 [20].

$$\overrightarrow{q}_{[k]} = \overrightarrow{q}^{T}_{[m]} \times U_{[m \times k]} \times \Sigma^{-1}_{[k \times k]} \tag{3.8}$$

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. 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 3.5). 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 3.1). The original matrix $A_{[m \times n]}$ is approximated to a matrix $A^{k}_{[m \times n]}$ (by Equation 3.7). 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 3.8). 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.

It is noticed that the mathematical computation of SVD is elaborated in [19] and SVD has been implemented in different languages such as C, C++ or Java [1]. 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 $\overrightarrow{q}_{[m]}$, in which only the element q_i has value $a_{i,j}$, other elements are equal to 0. m is the number of services (Equation 3.9).

$$q_t = \begin{cases} a_{i,j} & \text{if } t = i,\ t \in [1..m] \\ 0 & \text{if } t \neq i,\ t \in [1..m] \end{cases} \tag{3.9}$$

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 a 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.

3.4 Validation

In this section, we present the implementation efforts we have done to validate our approach. We developed a web application [2] which allows users to register, create their profiles and use the WS operations and providers to upload their web service description files. The application also provides a simple search engine for finding WS operations. The recommendation is presented in four lists respective to the proposed algorithms and it is created whenever a user click on a WS operation provided by the search engine or the recommendation components.

1. `http://web.eecs.utk.edu/research/lsi/soft.html`
2. Our tool can be found at `http://www-inf.it-sudparis.eu/SIMBAD/tools/WSRS/`.

We also performed experiments on a dataset collected by our application. 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*. 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 [116]. 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.

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 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 3.1 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

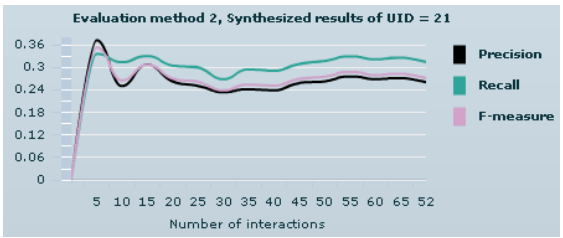
Table 3.1: Experiments with the two relevant sets

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.

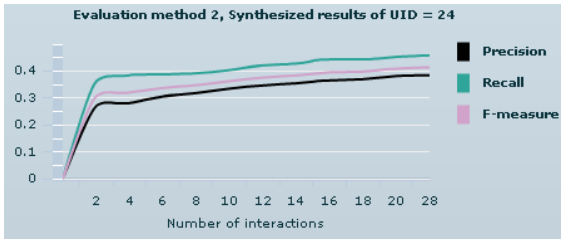
They also show that the user-based algorithm has the highest Precision and Recall values. Indeed, the usage data is collected from the usage of PhD students who have somehow similar behavior. Hence, the user-based algorithm can return good results and becomes the most suitable algorithm for this context.

Experiments with the last-used relevant data show that the algorithms which take into account the relations of all services (service-based and LSI-based) achieved the best results. The user-based algorithm and service-user combination algorithm 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 3.2 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 3.2a) 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 3.2b) keeps or slightly changes his interest, our recommendations are of higher quality.



(a) UID=21



(b) UID=24

Figure 3.2: Synthesized results of particular users

Summarizing up, our experiments 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*.

We also used AudioScrobbler [3], which is a large public dataset, for another set of experimentations. 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) [66] metric, which is a metric to evaluate the performance of prediction system.

Experiments on the 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*

3. `http://www.audioscrobbler.net/development/`

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 3.2.3 for details), especially when the number of users and services are very large. More details about this experimentation can be foun in [33].

3.5 Related Work

There were various researches on how to lead users to the short and accurate list of web services. Vector space model and cosine coefficient were applied to find the similarities between user's query strings, the services descriptions [109] to return the most closed services to users. They were also applied together with a splitting/merging technique to cluster the services into categories [46]. Birukou, Blanzieri et.al [23] used VSM and TF-IDF to implement a recommender system for web services. But they applied them on terms of the query string to find the similarity between requests, not similarity between terms in requests and terms in the database. They designed a System for Implicit Culture Support (SICS) and used WordNet to supplement the synonyms of words in query strings. Also using the SICS, the authors in [76] recorded user's behaviors by the way that: if a user sends the request X, he will invoke the WS operation Y. Then, they stored the user's requests and compare the similarity between users based on the used requests and WS operations. In this research, user's requests were considered as a parameter in generating a recommended list, therefore, the performance of algorithm still depended on user's query strings. These approaches encountered the synonym and polysemy problems. In fact, each user has his own language and it is possible that a user either types incorrect words or uses different notations and short written words in his query strings with the expectation that the recommender system is intelligent enough to provide him the accurate services.

Manikrao and Prabhakar [93] proposed a dynamic web service selection framework and an architecture of a semantic matcher. They also proposed a matching algorithm used for web service recommendations based on user's ratings. However, their solution is based on explicit knowledge as input which can be hard to be captured. In another context, Blake and Nowlan [24] focused on underlying searching and ranking algorithms that enable recommendations. They applied Levenshtein distance and Letter Pairing algorithms to propose a new algorithm for calculating similarity of customer's files and WS operations. However, they just evaluated some WS operations of five typical web services and stopped at comparing results of matching methods. It was also really hard to determine appropriate recommendations for a specific case.

The previous approaches can be classified in the following categories: clustering [46], rating based [93], words analysis [24] and vector space model [109, 23, 76]. They mainly analyzed the query strings and web service descriptions. Other solutions applied Latent Semantic Analysis on service descriptions to find the implicit relationships between terms and documents describing the services [166, 90]. The LSA solution could avoid the problem of lexical analysis. It, however, captures the relationships of elements in the text documents which could not present user's interests.

Different from them, our approach is inspired by the ideas of making recommendations based on user's WS operations [76] and the application of collaborative filtering (CF). The CF has been applied efficiently by Amazon in finding the most related products to the one selected by current user [88, 87]. This popular technique is used in building recommender system, especially for video, music or books recommendation [151]. In our approach, we apply this technique on a new domain to support web service discovery.

By applying the CF technique on user's behaviors, our solution can avoid the problems

of string-based algorithms. Moreover, our proposed algorithms generate recommendations based only on the user's usage data within the discovery process. Thus, our approach is *self-contained* within the web service discovery process, *independent* from any explicit or human centric or error prone knowledge. Whereas other information such as user's rating or service reputation can be hard to be captured, our approach is a good solution as it does not use such kind of explicit knowledge as inputs to propose recommendations.

3.6 Conclusion

In this chapter, We developed four algorithms to make individual service recommendations based on past usage, three of them are based on the memory-based CF technique and one algorithm is based on the model-based CF technique. There are few approaches 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. We do not ask users any effort to provide additional information such as profiles, ratings and comments. Our recommendations are made based on the correlation between users and services. This knowledge is implicitly presented in usage data. The computation time in our approach is polynomial.

This work was conducted during the Master and PhD thesis of Nguyen N. Chan [80, 103], and was mainly published in the following conference proceedings and journals [33, 32, 31].

Chapter 4

Composition-driven recommendation

4.1 Introduction

Process model design is the initial and key phase of service-based (business) process lifecycle where component services are chosen[97]. Prior research has emphasized the advantages of recommendations during process model design [71, 117]. Meanwhile, recommending entire process models costs much computation time, especially when the number of services is large. Large models are also not handy for a designer who needs to pick a specific piece of functionality from them. In this context, it is desirable to recommend only a small but well-selected set of services in order to help the designer. In contrast to recommending entire process models, recommending services during the design allows process designers to flexibly adjust and improve their designed process. It helps to interactively extend an existing process or create more process variants. Concretely, when a process designer is looking for suitable services to complete a designed process, to extend an existing process, or to replace a service, they need service recommendations for some selected positions. In these cases, a short list of recommended services will make recommendations clearer, more focused and helpful.

In this chapter, we propose an approach that recommends suitable services based on explicit process models for a selected position in a service-based process model (see section 4.2). To do so, we define the *neighborhood (or composition) context* of a service which captures its relations with its neighbors in a given process model. We match composition contexts to infer similarity between services. We make recommendations based on the matching results.

We realize that business processes do not often reflect the reality of service execution. Furthermore, in some cases, business processes are not explicitly presented [154], even when process logs are available. Moreover, process models do not explicitly show the importance of services or connection flows, which can be a valuable parameter to compute more precisely the similarity between two services. Meanwhile, this information is recorded in process event logs in the form of traces and their frequency. Process event logs contain some information that cannot be reflected by business process models. We propose therefore another solution that takes as unique input process event logs for service recommendation (see section 4.3).

4.2 Process-based service recommendation

We start the chapter by presenting some preliminaries that help to formally define business processes and service composition contexts (section 4.2.1). Next, we present our approach to compute the similarity between services based on their composition context matching (section 4.2.2). Then, we show how to make recommendations for a selected position using its composition context matching (section 4.2.3).

4.2.1 Preliminaries

service-based process graph

There are a number of graph-based business process modeling languages, e.g. BPMN, EPC, YAWL, and UML service 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 [125]. We use BPMN to present service-based processes as it is one of the most popular business process modeling language.

We consider termination events (such as start or end events) 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 Figure 4.1, s_1, a_1, a_2, e_1 are services; and 'flow-transition', 'event-based-gateway' and 'message-catching' are connection elements.

Relations between services in a service-based process are presented by the execution orders between them. We take into account both *causal* relations (e.g. "Search flights" and "Present alternatives" in Figure 4.1) and *parallel flow* relations (e.g. "Request customer detailed Info." and "Request credit card Info.") between services. In the following, we present definitions that are used to present service relations, service-based processes and the original concept of composition context.

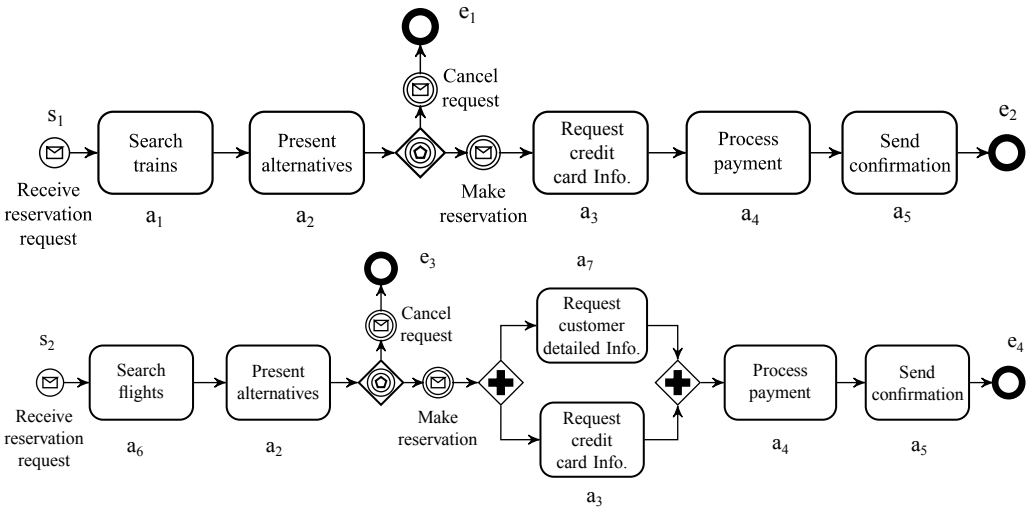


Figure 4.1: Train & flight reservation processes

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

Definition 4.2.1 (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 4.2.2 (connected relation). *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 $e_j \rightarrow_P e_i$.*

Definition 4.2.3 (connection flow). *A connection flow from a_i to a_j, $a_i, a_j \in A_P$, denoted by ${}^{a_j}_{a_i}f_P$, is a sequence of connection elements $c_1, c_2, \ldots, c_n \in C_P$ satisfying: $a_i \leftrightarrow_P c_1$, $c_1 \leftrightarrow_P c_2, \ldots, c_{n-1} \leftrightarrow_P c_n, c_n \leftrightarrow_P a_j$ [1]. ${}^{a_j}_{a_i}f_P \in C_P^*$, C_P^* is set of sequences of connection elements in P.*

Definition 4.2.4 (connected relation 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:*

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

Definition 4.2.5 (connection flow label). *The label of a connection flow ${}^{a_j}_{a_i}f_P$, denoted by $l({}^{a_j}_{a_i}f_P)$, is defined as following:*

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

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

For example, label of the connection flow from "Search flights" to "Present alternatives" in Figure 4.1 is: a_6'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:

- A connection flow is labeled by a sequence of connected relation labels. A connected relation label is an ordered pair of services and connection elements. This order is not changed when we label the connection flow in both directions. In both cases, we can represent the unique connection flow based on constituted connected relations labels. So, we can label *an edge connecting two services $a_i, a_j \in A_P$ by either $l({}^{a_j}_{a_i}f_P)$ or $l({}^{a_i}_{a_j}f_P)$*. For example, the edge connecting a_6 to a_2 in Figure 4.1 can be label by $l({}^{a_2}_{a_6}f_P)$=a_6'sequence':'sequence'a_2 or $l({}^{a_6}_{a_2}f_P)$= 'sequence'a_2.a_6'sequence'.
- *There can be more than one connection flow between two services.* For instance, in the case that two services are connected by an AND-split and an AND-join (parallel relation). In this case, we *number* these connection flows to distinguish them. For example, there are two connection flows from a_7 to a_3 in Figure 4.2 and we number them as follows: $l({}^{a_3}_{a_7}f^1_P)$='parallel-split'a_7:'parallel-split'a_3 and $l({}^{a_3}_{a_7}f^2_P)$=a_7'synchronization'.a_3'syn-chronization'.

We consider each service as a node, each connection flow as an edge. We define service-based process as a multigraph, in which, the set of edges is a multiset (Definition 4.2.6).

Definition 4.2.6 (Service-based process graph). *A service-based 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:*

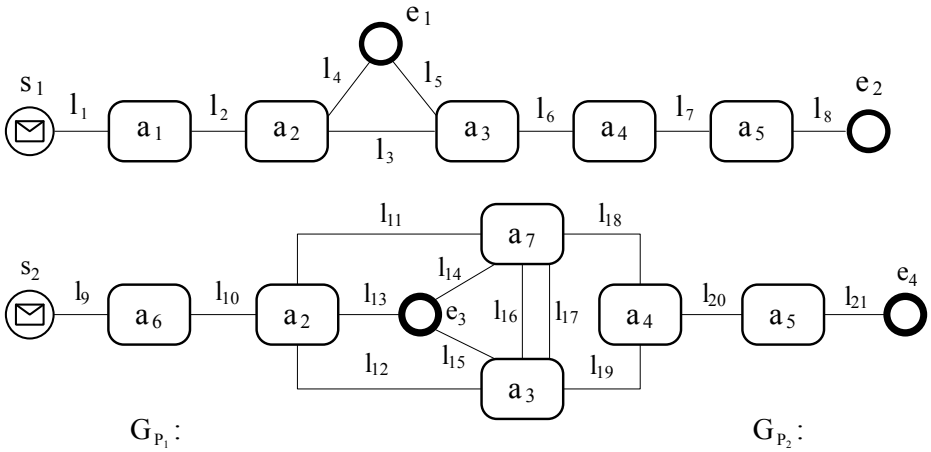
- $V_P = A_P$,

1. The connection flow from a_j to a_i is the inverse of the connection flow from a_i to a_j

- $E_P \subseteq \langle A_P \times A_P, g \rangle, \quad g: A_P \times A_P \longrightarrow N$
  $g((a_i, a_j))$ *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 by* $(a_i, a_j)^t$, $t = 1..k$, $k > 1$.
- $L_P = l(E_P)$, *where:*

$$\begin{array}{lccl} l: & E_P & \longrightarrow & L_P \\ & (a_i, a_j) & \mapsto & l(^{a_j}_{a_i} f_P) \text{ , if } g((a_i, a_j)) = 1 \\ & (a_i, a_j)^t & \mapsto & l(^{a_j}_{a_i} f^t_P) \text{ , if } g((a_i, a_j)) = k > 1,\ t = 1..k \end{array}$$

For example, the service-based process graphs of the 'train-reservation' process and the 'flight-reservation' process (Figure 4.1) are presented in Figure 4.2.



Nodes: $s_1, e_1, e_2, a_1, a_2, a_3, a_4, a_5$

Edges: $\langle(s_1,a_1),1\rangle, \langle(a_1,a_2),1\rangle, \langle(a_2,a_3),1\rangle, \langle(a_2,e_1),1\rangle, \langle(e_1,a_3),1\rangle, \langle(a_3,a_4),1\rangle, \langle(a_4,a_5),1\rangle, \langle(a_5,e_2),1\rangle$

Nodes: $s_2, e_3, e_4, a_2, a_3, a_4, a_5, a_6, a_7$

Edges: $\langle(s_2,a_6),1\rangle, \langle(a_6,a_2),1\rangle, \langle(a_2,a_7),1\rangle, \langle(a_2,a_3),1\rangle, \langle(a_2,e_3),1\rangle, \langle(e_3,a_7),1\rangle, \langle(e_3,a_3),1\rangle, \langle(a_7,a_3),2\rangle, \langle(a_7,a_4),1\rangle, \langle(a_3,a_4),1\rangle, \langle(a_4,a_5),1\rangle, \langle(a_5,e_4),1\rangle$

Edge labels:

l_1="s_1S.Sa_1"
l_2="a_1S.Sa_2"
l_3="a_2E.EM.Ma_3"
l_4="a_2E.EM.Me_1"
l_5="Me_1.EM.EM.Ma_3"
l_6="a_3S.Sa_4"
l_7="a_4S.Sa_5"
l_8="a_5S.Se_4"
l_9="s_2S.Sa_6"
l_{10}="a_6S.Sa_2"
l_{11}="a_2E.EM.MP.Pa_7"
l_{12}="a_2E.EM.MP.Pa_3"
l_{13}="a_2E.EM.Me_3"
l_{14}="Me_3.EM.EM.MP.Pa_7"
l_{15}="Me_3.EM.EM.MP.Pa_3"
l_{16}="Pa_7.Pa_3"
l_{17}="a_7Y.a_3Y"
l_{18}="a_7Y.Ya_4"
l_{19}="a_3Y.Ya_4"
l_{20}="a_4S.Sa_5"
l_{21}="a_5S.Se_4"

Annotation:
S: 'sequence'
E: 'event-based-gateway'
M: 'message-catching'
P: 'parallel-split'
Y: 'synchronization'

Figure 4.2: Service-based process graphs of 'train-reservation' and 'flight-reservation' processes (Figure 4.1)

Composition context graph

We define the composition context of a service as a process fragment that includes the associated service and the closest relations to its neighbors. A composition context is presented as a graph in 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 presents the behavior of the associated service within the process. We present in the following some definitions that are used to formally define the composition context.

Definition 4.2.7 (connection path). *A connection path from a_i to a_j in a service-based process graph G_P, denoted by ${}^{a_j}_{a_i}\mathcal{P}_P$, is a sequence of services $a_1, a_2, \ldots, a_k$ where $a_1 = a_i$, $a_k = a_j$ and $\exists({}^{a_{t+1}}_{a_t} f_P \in C^*_P \ \vee \ {}^{a_t}_{a_{t+1}} f_P \in C^*_P) \ \forall 1 \leq t \leq k-1$.*

According to Definition 4.2.7, a connection path in a service-based process graph is *undirected*. It means that the edges in a connection path can be oriented in different directions. For example, in Figure 4.1, a connection path from 'Search flights' (a_6) 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 4.2.8 (connection path length). *The length of a connection path ${}^{a_j}_{a_i}\mathcal{P}_P$, denoted by $\mathcal{L}({}^{a_j}_{a_i}\mathcal{P}_P)$ is the number of connection flows in the path.*

Definition 4.2.9 (shortest connection path). *The shortest connection path between a_i and a_j, denoted by ${}^{a_j}_{a_i}\mathcal{S}_P$, is the connection path between them that has the minimum connection path length.*

For example, in Figure 4.1, 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 4.2.10 (k^{th}-layer neighbor). *a_j is a k^{th}-layer neighbor of a_i in a service-based process P iff $\exists {}^{a_j}_{a_i}\mathcal{P}_P : \mathcal{L}({}^{a_j}_{a_i}\mathcal{P}_P) = k$. The set of k^{th}-layer neighbors of a service a_i is denoted by $N^k_P(a_i)$. $N^0_P(a_i) = \{a_i\}$.*

For example in Figure 4.1, 'Receive reservation request' and 'Present alternatives' are the 1^{st}-layer neighbors of 'Search flights'; 'Search flight', 'end-event', 'Request customer detail Info.' and 'Request credit card Info.' are the 1^{st}-layer neighbors of 'Present alternatives'; 'Request credit card Info.' is one of the 2^{nt}-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 is not explicitly represented in the service-based 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 4.2.11).

Definition 4.2.11 (k^{th}-zone flow). *${}^{a_v}_{a_u} f_P$ is a k^{th}-zone flow of a_i iff $\exists {}^{a_v}_{a_u} f_P : (a_u, a_v \in N^{k-1}_P(a_i)) \vee (a_u \in N^{k-1}_P(a_i) \wedge a_v \in N^k_P(a_i)) \vee (a_v \in N^{k-1}_P(a_i) \wedge a_u \in N^k_P(a_i))$. The set of all k^{th}-zone flows of a service $a_i \in P$ is denoted by $Z^k_P(a_i)$. $Z^0_P(a_i) = \emptyset$ and $|Z^k_P(a_i)|$ is the number of connection flows in the k^{th} connection zone of a_i.*

For example in Figure 4.1, 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. $|Z^2_{P_2}(\text{'Search flights'})| = 3$ 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 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 service-based process, we define the *composition context graph* (formally defined in Definition 4.2.12) which presents all the shortest paths from a service to others. Each service in a service-based process has a composition context graph. Each vertex in the composition 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 composition context graph the *layer number*. The area limited between two adjacent layers is called zone. The edge connecting two vertexes in a composition context graph belongs to a zone. We assign to each edge in the composition 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 composition 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.

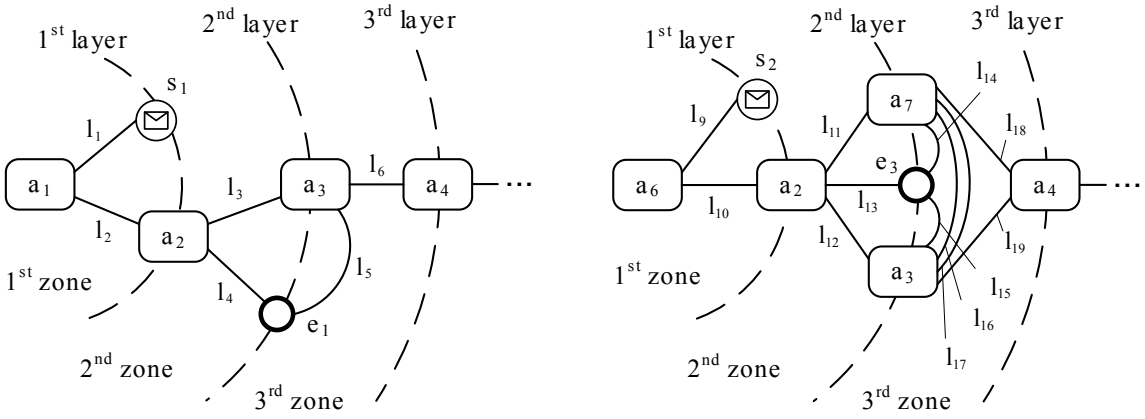
Basically, assume that e_{ij} is the edge connecting two vertexes a_i and a_j in the composition context graph of a service a_x. The lengths of the shortest connection paths connecting a_i and a_j to a_x are $l_1 = \mathcal{L}(^{a_x}_{a_i}\mathcal{S}_P)$ and $l_2 = \mathcal{L}(^{a_x}_{a_j}\mathcal{S}_P)$ respectively. Let $d = |l_1 - l_2|$, d has only two possible values, which are 0 and 1. 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 a 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 composition context graph of a_x the value $Min(\mathcal{L}(^{a_x}_{a_i}\mathcal{S}_P), \mathcal{L}(^{a_x}_{a_j}\mathcal{S}_P)) + 1$. The maximum zone value of all connection flows in the context graph of a_x will be $Max(\mathcal{L}(^{a_x}_{a_t}\mathcal{S}_P)) + 1 \ \forall a_t \in P$.

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

Definition 4.2.12 (Composition context graph)**.** *The composition context graph of a service $a_x \in P$, denoted by $G_P^{a_x} = (V_P^{a_x}, E_P^{a_x}, L_P, l)$, is an undirected labeled multigraph created from $G_P = (V_P, E_P, L_P, l)$. $V_P^{a_x}$ is a set of vertexes associated to their layer numbers and $E_P^{a_x}$ is a set of edges associated to their zone numbers. $V_P^{a_x}$ and $E_P^{a_x}$ are defined as following:*

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

For example, an excerpt of 'Search trains' composition context graph created from "train-reservation" process and an excerpt of the "Search flights" composition context graph created from "flight-reservation" process (Figure 4.1) are presented in Figure 4.3. In these graphs, all causal and parallel flow relations are presented.



Edge labels:

	l_8="a_5S.Se_4"	l_{16}="Pa_7.Pa_3"	Annotation:
l_1="s_1S.Sa_1"	l_9="s_2S.Sa_6"	l_{17}="a_7Y.a_3Y"	S: 'sequence'
l_2="a_1S.Sa_2"	l_{10}="a_6S.Sa_2"	l_{18}="a_7Y.Ya_4"	E: 'event-based-gateway'
l_3="a_2E.EM.Ma_3"	l_{11}="a_2E.EM.MP.Pa_7"	l_{19}="a_3Y.Ya_4"	M: 'message-catching'
l_4="a_2E.EM.Me_1"	l_{12}="a_2E.EM.MP.Pa_3"	l_{20}="a_4S.Sa_5"	P: 'parallel-split'
l_5="Me_1.EM.EM.Ma_3"	l_{13}="a_2E.EM.Me_3"	l_{21}="a_5S.Se_4"	Y: 'synchronization'
l_6="a_3S.Sa_4"	l_{14}="Me_3.EM.EM.MP.Pa_7"		
l_7="a_4S.Sa_5"	l_{15}="Me_3.EM.EM.MP.Pa_3"		

Figure 4.3: An excerpt of the composition context graphs of a_1 and a_6 (in Figure 4.1)

4.2.2 Composition 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 composition contexts. This matching is used for service recommendation. To compute the composition context matching, we propose to *match all connection flows that belong to the same connection zone and have the same ending services.*

To illustrate the computation process, we demonstrate the matching between the composition context of the 'Search trains' service in the 'train-reservation' process (a_1 in Figure 4.1) and the 'Search flights' service in the 'flight-reservation' process (a_6 in Figure 4.1). The composition context graphs of these services are shown in Figure 4.3.

Connection flow matching

To compute the matching between two connection flows, we propose to use the Levenshtein distance (LD for short) [82]. We consider each connection element in a connection flow as a character and label connection flows as a sequence of characters. Then, the similarity between two connection flows can be computed based on the similarity of their labels using LD. However, as an edge connecting a_u and a_v in P_1 can be labeled by either ${}^{a_v}_{a_u}f_{P_1}$ or ${}^{a_u}_{a_v}f_{P_1}$ and there can be multiple connection flows between two services, we compute the matching between ${}^{a_v}_{a_u}f_{P_1}$ and ${}^{a_n}_{a_m}f_{P_2}$ as following.

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

$$M({}^{a_v}_{a_u}f_{P_1}, {}^{a_n}_{a_m}f_{P_2}) = 1 - \frac{Diff(st_1, st_2)}{Max(length(st_1), length(st_2))} \quad (4.1)$$

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

For example in Figure 4.3, we have $M_p(^{a_3}_{a_2}f_{P_1}, ^{a_3}_{a_2}f_{P_2}) = M(\text{“}a_2\text{E.EM.M}a_3\text{”}, \text{“}a_2\text{E.EM.MP.P}a_3\text{”}) = 1 - \frac{2}{8} = 0.75$.

We prove that LD of two strings is equal to LD of their inverse strings [2]. So, whatever the edges (a_u, a_v) and (a_m, a_n) are labeled by $l(^{a_v}_{a_u}f_{P_1})$ or $l(^{a_u}_{a_v}f_{P_1})$ and $l(^{a_n}_{a_m}f_{P_2})$ or $l(^{a_m}_{a_n}f_{P_2})$, Equation 4.1 gives a unique value.

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.

Composition context graph matching

To compute the composition context matching between two services, we propose to sum up the matching of the connection flows in the two contexts. There are two cases to consider: the *first zone* and the *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 the 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.

We apply Equation 4.1 to compute the composition context matching in either the first and other zones. However, Equation 4.1 considers only connection flows that connecting the same services in two service-based processes. So, to adapt it in the first zone, we assume that the two associated services have the same name, so-called a_0. Then, we match connection flows connecting a_0 to the same services in the first layer.

Formally, let a_i, a_j are two *associated services*. We change a_i, a_j to a_0. Then, $\forall a_c \in N^1_{P_1}(a_i) \cap N^1_{P_2}(a_j)$, we compute the similarity between $^{a_c}_{a_i}f_{P_1}$ and $^{a_c}_{a_j}f_{P_2}$ based on the similarity between $^{a_c}_{a_0}f_{P_1}$ and $^{a_c}_{a_0}f_{P_2}$. Concretely, let $st_1 = l(^{a_c}_{a_0}f_{P_1})$, $st_2 = l(^{a_c}_{a_0}f_{P_2})$, then:

$$Diff(st_1, st_2) = LD(l(^{a_c}_{a_0}f_{P_1}), l(^{a_c}_{a_0}f_{P_2})) \tag{4.2}$$

and we apply Equation 4.1 to compute this matching.

For example, in Figure 4.3, we have $M(^{a_2}_{a_1}f_{P_1}, ^{a_2}_{a_6}f_{P_2}) = M(\text{“}a_0\text{S.S}a_2\text{”}, \text{“}a_0\text{S.S}a_2\text{”}) = 1$, and so on.

Basically, the composition context matching between $a_i \in P_1$ and $a_j \in P_2$ within k zones, denoted by $\mathbb{MC}^k(G^{a_i}_{P_1}, G^{a_j}_{P_2})$, is computed by Equation 4.3.

$$\mathbb{MC}^k(G^{a_i}_{P_1}, G^{a_j}_{P_2}) = \frac{\sum_{t=1}^{k} \sum_{^{a_v}_{a_u}f_{P_1} \in Z^t_{P_1}, ^{a_n}_{a_m}f_{P_2} \in Z^t_{P_2}} \mathsf{MF}^t(^{a_v}_{a_u}f_{P_1}, ^{a_n}_{a_m}f_{P_2})}{\sum_{t=1}^{k} |Z^t_{P_1}(a_i)|} \tag{4.3}$$

where k is the number of considered zones, $|Z^t_{P_1}(a_i)|$ is the number of connection flows in the t^{th} zone of $G^{a_i}_{P_1}$, and $\mathsf{MF}^t(^{a_v}_{a_u}f_{P_1}, ^{a_n}_{a_m}f_{P_2})$ is the matching value of $^{a_v}_{a_u}f_{P_1}$ and $^{a_n}_{a_m}f_{P_2}$ in

2. http://www-inf.it-sudparis.eu/SIMBAD/tools/BPAR/ld-inverse-strings.pdf

zone t:

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

For example, the composition context matching between a_1 and a_6 (Figure 4.3) within 3 zones computed by Equation 4.3 is:

$$\begin{aligned} \mathbb{MC}^3(G^{a_x}_{P_1}, G^{a_6}_{P_2}) &= \frac{\begin{array}{r} M({}^{a_1}_{s_1}f_{P_1},{}^{a_6}_{s_2}f_{P_2}) + M({}^{a_2}_{a_1}f_{P_1},{}^{a_2}_{a_6}f_{P_2}) + M({}^{a_3}_{a_2}f_{P_1},{}^{a_3}_{a_2}f_{P_2}) \\ +M({}^{e_1}_{a_2}f_{P_1},{}^{e_3}_{a_2}f_{P_2}) + M({}^{a_3}_{e_1}f_{P_1},{}^{a_3}_{e_3}f_{P_2}) + M({}^{a_4}_{a_3}f_{P_1},{}^{a_4}_{a_3}f_{P_2}) \end{array}}{2+2+2} \\ &= \frac{1+1+\frac{3}{4}+1+\frac{9}{10}+\frac{1}{2}}{6} = 0.86 \end{aligned}$$

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 a zone-weight value computed by a polynomial function which is $w_t = \dfrac{k+1-t}{k}$, 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 ($w_k = \frac{1}{k}$).

Then, the composition contexts matching between $G^{a_i}_{P_1}$ and $G^{a_j}_{P_2}$ within k zones and with zone weight consideration, denoted by $\mathbb{MW}^k(G^{a_i}_{P_1}, G^{a_j}_{P_2})$, is given by Equation 4.4.

$$\mathbb{MW}^k(G^{a_i}_{P_1}, G^{a_j}_{P_2}) = \frac{2}{k+1} \times \sum_{t=1}^{k} \frac{k+1-t}{k} \times \frac{\sum\limits_{\substack{{}^{a_v}_{a_u}f_{P_1}\in Z^t_{P_1} \\ {}^{a_n}_{a_m}f_{P_2}\in Z^t_{P_2}}} \mathsf{MF}^t({}^{a_v}_{a_u}f_{P_1},{}^{a_n}_{a_m}f_{P_2})}{|Z^t_{P_1}(a_i)|} \tag{4.4}$$

For example, the composition context matching between a_1 and a_6 (Figure 4.3) computed by Equation 4.4 is:

$$\begin{aligned} \mathbb{MW}^3(G^{a_x}_{P_1}, G^{a_6}_{P_2}) &= \frac{2}{3+1} \times \Big(\frac{3}{3} \times \frac{M({}^{a_1}_{s_1}f_{P_1},{}^{a_6}_{s_2}f_{P_2}) + M({}^{a_2}_{a_1}f_{P_1},{}^{a_2}_{a_6}f_{P_2})}{2} + \\ &\quad \frac{2}{3} \times \frac{M({}^{a_3}_{a_2}f_{P_1},{}^{a_3}_{a_2}f_{P_2}) + M({}^{e_1}_{a_2}f_{P_1},{}^{e_3}_{a_2}f_{P_2})}{3} + \\ &\quad \frac{1}{3} \times \frac{M_p({}^{a_3}_{e_1}f_{P_1},{}^{a_3}_{e_3}f_{P_2}) + M_p({}^{a_4}_{a_3}f_{P_1},{}^{a_4}_{a_3}f_{P_2})}{2}\Big) \\ &= \frac{2}{4} \times \Big(\frac{1+1}{2} + \frac{2}{3} \times \frac{\frac{3}{4}+1}{2} + \frac{1}{3} \times \frac{\frac{9}{10}+\frac{1}{2}}{2}\Big) = 0.91 \end{aligned}$$

4.2.3 Recommendation

We make recommendations based on the composition context matching. For a selected service, we compute its composition context graph matching with other services in other service-based 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 composition 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 where a process analyst needs service recommendation: *discovering services* or *improving the ongoing designed service-based process.*

In the first case, the process analyst wants to discover services that are suitable to a position in a service-based process, have the same composition contexts, and therefore can be easily plugged into a position in the ongoing designed service-based process. She marks this position as an 'unknown' service (a round rectangle with a '?' symbol). Our approach will capture the composition context of the 'unknown' service. Then, it matches the captured context with others and retrieves relevant services to the selected position.

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 alternatives for a specific position in the process, she will select the service at this position. Our approach will recommend her relevant services. With these recommendations, the process analyst can create different process variants from the current designed process.

In addition, our approach 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.

4.3 Log-based service recommendation

In this section, we propose an approach that builds upon process event logs for making service recommendations. We examine the relation between services based on their execution order and frequency as recorded in logs. We define the notion of a *log-based composition context* of a service as a fragment of the log-based model that contains the considered service and relations to its neighbors. Relations between services and their occurrence frequency provide the basis for the computation of the similarity between services. In the following, we firstly present some definitions related to process logs (section 4.3.1). Then, we present definitions of the log-based process (section 4.3.1) and the service composition context (section 4.3.1). Finally, we detail our approach with log-based composition context matching and activity recommendation (section 4.3.1).

4.3.1 Preliminaries

According to [154] and [132], a process log is defined as follows.

Definition 4.3.1 (Log trace, process log, L)**.** *Let A be a set of services. A^* denotes the set of finite sequences over A and $\sigma = a_1 a_2 \dots a_n \in A^*$ is a log trace. $L \in \mathcal{P}(A^*)$ is a*

process log[3].

As explained in [154, 132], a process log does not consider the repetition of a trace. For example, in Table 4.1, which contains the Event logs of a liability claim process, L includes only traces 1, 2, 3, 6 and 7. Traces 4 and 5 are excluded by L as they repeat traces 2 and 1. In our approach, we extend Definition 4.3.1 to define the *full* process log (see Definition 4.3.2) that includes all log traces.

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

Table 4.1: Event logs of a liability claim process ([123])

Traces	Log traces	Repeat
1	KBJFH	
2	KBJGH	
3	KBJFH	trace 1
4	KBJFH	trace 1
5	KBJGH	trace 2

Table 4.2: Event logs of a customer subscription process

Definition 4.3.2 (Full process log, L^*)**.** *A full process log is the process log that includes all executed traces. The full log is denoted by* L^*, $L^* \in \mathcal{P}^*(A^*)$. $L \subseteq L^*$.

For example, in Table 4.1, L^* includes all traces from 1 to 7. In Table 4.2, L^* includes all traces from 1 to 5, while L includes only trace 1 and 2.

Definition 4.3.3 (Log-based ordering relation, $>_L$)**.** *Let L be a process log over A, i.e., $L \in \mathcal{P}(A^*)$. Let $a, b \in A$. $a >_L b$ iff $\exists \sigma = a_1 a_2 \dots a_n$, $i \in \{1, 2, \dots, n-1\}$: $\sigma \in L \wedge a_i = a \wedge a_{i+1} = b$.*

For example, from the logs given in Table 4.1, we have $A >_L B$, $A >_L C$, $C >_L D$, $B >_L D$, and so on.

Log-based process

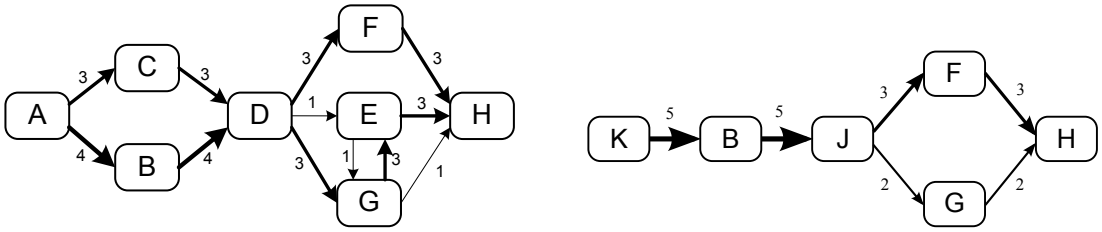
The sequence of services in a log trace $\sigma = a_1 a_2 \dots a_n \in A^*$ presents their ordering relations. A relation between a service a_i and its followed service a_{i+1} in the trace σ, $1 \leq i \leq n-1$ can be presented as a *directed edge* from a_i to a_{i+1}. The service relations in a process log L can be presented as 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 process graph* (Definition 4.3.4).

Definition 4.3.4 (Log-based process graph)**.** *A log-based process graph is a weighted directed graph $G_L = (V_L, E_L, w)$ built from a process log $L^* \in \mathcal{P}^*(A^*)$ where:*

- $V_L = A = \{a_1, a_2, \dots, a_n\}$,
- $E_L = \{(a_i, a_j) \in A \times A : a_i >_L a_j\} \subseteq A \times A$,
- *w is a weight function from E_L to N:*

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

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



(a) the liability claim process (b) the customer subscription process

Figure 4.4: Log-based process graphs

$|a_i >_L a_j|$ *is number of times that* $a_i >_L a_j$ *comes about in the log* L^*
$w(a_i, a_j) = 0$ *if* $\nexists \sigma = a_1 a_2 \dots a_n$, $k \in \{1, 2, \dots, n-1\} : a_k = a_i \wedge a_{k+1} = a_j$

For example, the log-based process graphs of the event logs given in Table 4.1 and Table 4.2 are depicted in Figure 4.4. The weight of each flow is the number of times that the flow is executed. It is emphasized by the arrow's thickness.

The log-based graph presents the execution of a business process in reality, regardless its conceptual model. The weights of edges present their execution frequency which indicates the strength of relations between services. In the following, we present our approach to build the composition context of a service based on these relations.

Log-based composition context

We define the log-based composition 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 composition context of a service presents the best relations between the service and its neighbors.

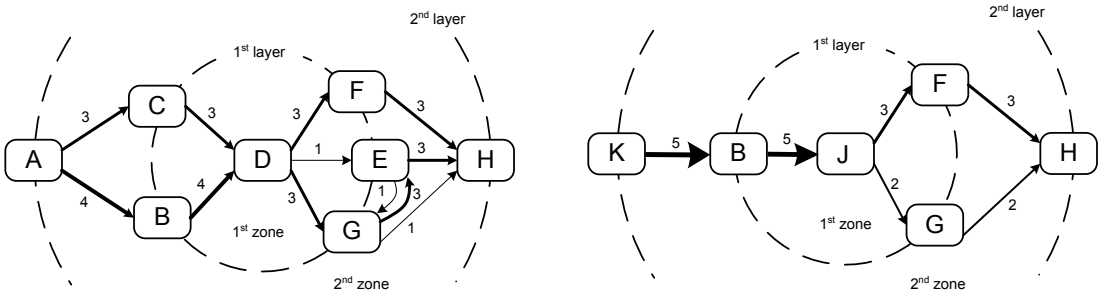
In a log-based composition context graph, each vertex is associated to a number that indicates the shortest path length from it to the associated service. Vertexes that have the same shortest path length are considered to be located on the same *layer* around the associated service. Similarly to section 4.2, we name the number associated to each service in a log-based composition 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 log-based composition context graph belongs to a zone as the vertexes are on the same or adjacent layers. We assign to each edge a number, so-call *zone number*, which determines the zone that the edge belongs to.

Definition 4.3.5 (Log-based composition context graph)**.** *The log-based composition 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* $l(a_j)_{G_C(a_i)}$*, is the shortest path length from* a_j *to* a_i *and the zone number of an edge* (a_j, a_k)*, denoted by* $z(a_j, a_k)_{G_C(a_i)}$*, has value* $min(l(a_j)_{G_C(a_i)}, l(a_k)_{G_C(a_i)}) + 1$*:*

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

For example, the composition context graphs of service D and J in Figure 4.4 are depicted in Figure 4.5.

Definition 4.3.6 (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* $N^k(a_i)$*.* $N^k(a_i) = \{a_j : l(a_j)_{G_C(a_i)} = k\}$.



(a) the liability claim process (b) the customer subscription process

Figure 4.5: Composition context graphs of the given event logs

For example, in Figure 4.4a, $N^1(A) = \{B, C\}$, $N^2(A) = \{D\}$, $N^3(A) = \{F, E, G\}$, $N^1(D) = \{C, B, F, E, G\}$, $N^2(D) = \{A, H\}$, $N^3(D) = \varnothing$, and so on.

4.3.2 Matching and recommendation

To compute the matching between two log-based composition contexts, we (1) compute the matching of their edges in each zone, (2) multiply this matching with a zone-weight value and (3) sum up the matching in all zones.

We apply the vector space model (VSM) to compute the matching of edges in each zone of two composition context graphs. We present each zone as a vector of which elements are edges and values are their corresponding weights. Then, we align elements that connect the same services in the same layers. Next, we present these vectors in the same space by filling 0 values in corresponding positions of the unaligned elements. Finally, we compute the cosine value of these two zone-vectors.

Particularly, in the first zone, we match the edges that connect the two associated services to the same services in the first layer. To do so, we define the two associated services as *root services* and name them r_0.

Concretely, assume that P_p and P_q are two log-based processes constructed from event logs L_p and L_q. Let A_p, A_q be sets of services of P_p and P_q respectively. We compute the similarity between services $a \in A_p$ and $b \in A_q$ by applying VSM as following.

Let $E^k_{P_p}(a)$ and $E^k_{P_q}(b)$ be 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 corresponding zone vectors.

$$\begin{aligned}
E^k_{P_p}(a) &= \{(x, y) : z(x, y) = k, x, y \in A_p\} \\
&= \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \\
\overrightarrow{e(a)} &= (w(x_1, y_1), w(x_2, y_2), \ldots, w(x_m, y_m)) \\
E^k_{P_q}(b) &= \{(e, f) : z(e, f) = k, e, f \in A_q\} \\
&= \{(e_1, f_1), (e_2, f_2), \ldots, (e_n, f_n)\} \\
\overrightarrow{e(b)} &= (w(e_1, f_1), w(e_2, f_2), \ldots, w(e_n, f_n))
\end{aligned}$$

Let $N^{k-1}_c(a, b)$ and $N^k_c(a, b)$ be the sets of common neighbors of a and b on layers $k-1$ and k, $k > 0$. We have:

$$\begin{aligned}
N^{k-1}_c(a, b) &= N^{k-1}_{P_p}(a) \cap N^{k-1}_{P_q}(b) \\
N^k_c(a, b) &= N^k_{P_p}(a) \cap N^k_{P_q}(b)
\end{aligned}$$

As we define the two associated services as *root services* and name them r, we have: $N^0_{P_p}(a) = a = r_0$, $N^0_{P_q}(b) = b = r_0$ and $N^0_c(a,b) = r_0$.

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

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

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

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

By applying VSM, the similarity between a and b in the k^{th} zone is given by Equation. 4.5.

$$M^k(a,b) = \frac{\overrightarrow{e_c(a)} \cdot \overrightarrow{e_c(b)}}{|\overrightarrow{e(a)}| \times |\overrightarrow{e(b)}|} \tag{4.5}$$

For example, we have: the common neighbors of D and J in the 1^{st}-layer are $N^1_c(D,J) = \{F,G,B\}$. So, $\overrightarrow{e_c(D)} = (w(D,F), w(D,G), w(B,D)) = (3,3,4)$, $\overrightarrow{e_c(J)} = (w(J,F), w(J,G), w(B,J)) = (3,2,5)$ and their matching in the 1^{st}-zone is:

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

In the 2^{nd}-zone, we have the common edges of these two context graphs are: (F,G) and (G,H). So, their matching in this zone is:

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

The behavior of a service is strongly reflected by the connections to its closet neighbors. Therefore, we propose to consider a zone weight in our matching computation. 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 the polynomial function $w^z_j = \frac{k+1-j}{k}$ given in section Similarly to section 4.2. The final matching formula integrating the zone weight consideration is given in Equation 4.6.

$$M^*(a,b) = \frac{2}{k+1} \times \sum_{i=1}^{k} \frac{k+1-i}{k} \times M^i(a,b) \tag{4.6}$$

For example, the matching between the composition contexts of D and J (in 2 zones) with zone weights is:

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

Basically, the steps to make recommendations based on log-based composition context matching are: (i) We represent the execution logs in a log-based graph. (ii) For each service in the log-based graph, we build its composition context graph. (iii) We compute

the matching between composition context graphs using vector space model. (iv) Finally, for a selected service, we sort other services in descending order of similarity and pick up top-n services for recommendation.

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 log-based process are not great [4], our algorithm can run fast in computing the composition context matching of two services. The worst case of this algorithm's computation time is $\mathcal{O}(n_A \times n_P \times n \times k)$, where n_A is the number of services, n_P is the number of service-based 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 service-based processes in the system are entirely matched. In addition, the performance of the algorithm can be improved by processing the composition context matching periodically off-line.

4.4 Validation

As a proof of concept, we implemented two applications that provide recommendations to process designers during the design phase based respectively on process models or logs. The first application [5] was developed based on Signavio [6], which is a platform for business process design. The second application was implemented as plug-in and integrated into ProM. This plug-in [7] uses log files to recommend services for selected positions in an ongoing designed process.

We also performed experiments on a large public collection of business processes. Our goal is three fold: (i) to evaluate the feasibility of our approach; (ii) to measure its efficiency and to (iii) evaluate the performance of our algorithm. The dataset used in our approach is shared by the Business Integration Technologies (BIT) research group [8] at the IBM Zurich Research Laboratory. It was presented in [54]. It contains business process models designed for financial services, telecommunications, and other domains. It is presented in XML format following BPMN 2.0 standard.

We performed experiments to show that our approach is *feasible*, *accurate* and of *good performance*. 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 the number of selected zones (k^{th}-zone number) and zone weight on the number of recommended services. We evaluate the accuracy of our algorithms based on Precision and Recall values and we measure the performance of our algorithms based on computation time.

In the first experiment, we set k^{th}-zone = 1 (i.e. we take into account only the first zone) and match the service composition context graphs of all services in the repository. We obtain that 2938 (77.7%) services match at least one service with a matching value greater than 0. This result shows that our approach can provide recommendations for a majority services as we can retrieve similar composition contexts for more than 3/4

4. We found in a public process dataset that in average there are 11.36 services in a service-based process (see section 4.4)

5. Our tool is published at `http://www-inf.it-sudparis.eu/SIMBAD/tools/WebRec/`.

6. `http://www.signavio.com/`

7. This plug-in is published at `http://www-inf.it-sudparis.eu/SIMBAD/tools/LogRec/`.

8. `http://www.zurich.ibm.com/csc/bit/`

number of services. It means that our approach is *feasible*.

In the second experiment, we examine the impact of k^{th}-zone values and zone weight. We run our algorithms with k from 3 to 1. Figure 4.6 shows the number of services that have matching values greater than or equal to 0.5. It shows that when k decreases, the number of recommended services increases. Indeed, when k decreases, the number of unmatched services in further layers decreases, consequently, the matching values between composition contexts increase, thereafter the number of services increases. Figure 4.6 also shows that zone weight increases the number of recommended services ($k = 2$ and $k = 3$).

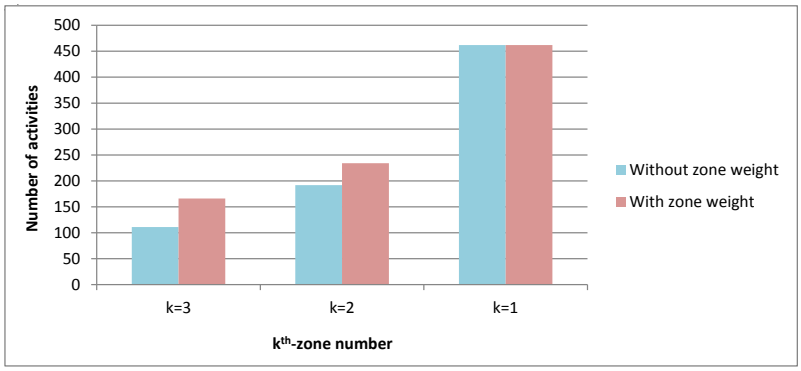


Figure 4.6: Number of services with different k^{th}-zone values

In the third experiment, we evaluate the efficiency of our approach based on Precision values. We also compare our approach to randomly generated recommendations. Concretely, consider a selected service a in a business process P. Assume that a appears in n business processes. The recommendations for this selected position consist of l services, in which $t(t \leq l)$ services are a.

In this experiment, we tune the number of recommended services for each position from 5 to 1. We take into account the matching in only the first zone ($k = 1$). To ignore the noise of the irrelevant processes, we compute Precision for only the services that appear in at least 10 business processes. Consequently, 29 services and 267 business processes are considered for our experiment.

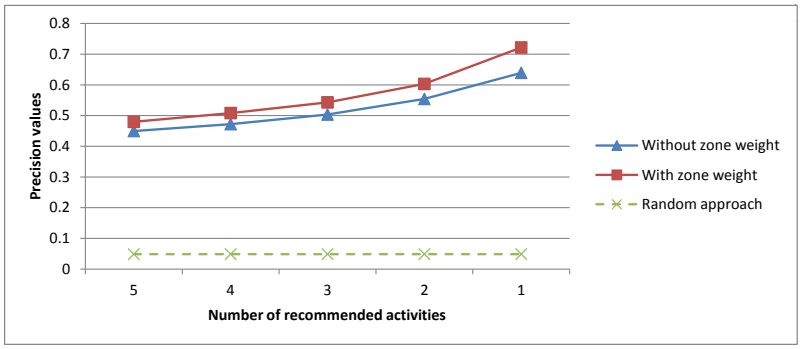


Figure 4.7: Precision values computed by taking into account the first zone

Figure 4.7 shows the average Precision values obtained in this experiment. It shows that our recommendations achieved good Precision values. These values are much more higher than a random approach. Our approach is more precise if we take into account zone weight in our computation.

Figure 4.7 also shows that the Precision values increase when the number of recom-

mended 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. It also show that our approach is much more efficient than a random approach (at least 10 times better).

In the forth experiment, we measure the performance of our algorithm based on computation time. We run our algorithm on a computer running Ubuntu 11.10 with the configuration: Pentium 4 CPU 2.8GHz, cache 512KB, RAM 512MB, HDD 80GB. We match each service in a business process with all services in other business processes.

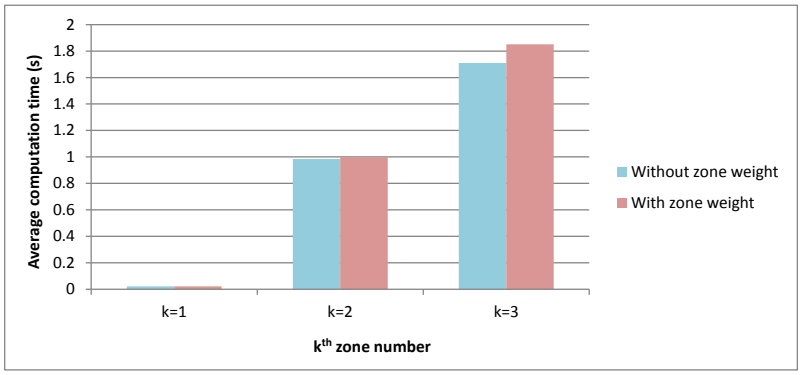


Figure 4.8: Average computation time with $k = 3$

Figure 4.8 shows the average computation time with different k^{th}-zone values. With $k = 1$, our algorithms take about *2 milliseconds* to compute the matching between a service and the other 6362 services. This computation time increases when k increases. In average, it is about *1 seconds* with $k = 2$ and *1.7 seconds* with $k = 3$. This means that our algorithms have *good performance* as they can make recommendations in a very short time by considering a large number of services. To shorten the response time for recommendations, the matching computation in our approach can be done offline.

In summary, the statistics on the number of recommended services showed that our approach is *feasible*. The Precision values showed that our approach was *accurate*. Finally, experiments on the computation time showed that our approached had *good performance*. So, our approach can be applied in real use-cases to facilitate process design.

We performed also experiments for our log-based service recommendation approach (section 4.3). A big challenge of our approach is the availability of real process logs. We attempted to search and contact other research groups for both public and private logs. However, process event logs are not published or they are not under a disclosure agreement. There are very few logs that are shared for the competition of the BPI challenge [9]. But they are not usable in our approach as they are just logs of one business process while we need logs of several different processes. We performed experiments on logs generated from the previously presented dataset shared by the IBM Business Integration Technologies (BIT) team. Due to space limitation, more details about these experimentation can be found in [34].

4.5 Related work

Some existing approaches [170, 1, 42, 57] target to fasten the design phase by retrieving similar process to the current designed process from repositories [121]. They proposed

9. `http://www.win.tue.nl/bpi/`

either to rank existing business process models for similarity search [170, 40, 161], or to measure the similarity between them [1, 83, 50, 65] for creating new process models. Business processes in reality consist of large number of services and flow connections, therefore, recommending the designer a list of business processes can make him confused because it is hard to detect how the business processes are similar and which parts should be considered from the recommended processes to use for his current design. In addition, computation on the whole process leads the existing approaches to the graph-matching problem which is NP-complexity [3] and they have to deal with the trade-off among the complexity, accuracy (efficiency) and system performance [170, 40]. In our approach, we focus partially on the business process and take into account only the service composition context for recommendations. Consequently, we recommend services without facing the computational complexity problem.

Similarity metrics, including label matching similarity, structural similarity and behavioral similarity, have been synthesized by R. Dijkman et. al. [41]. They used Levenshtein distance to compare the service labels; graph edit distance and vector space model to determine the similarity between business process structures. They also proposed the ICoP framework [163] and a semantic matching method [81] to identify the match between parts of process models using these metrics. Different from them, we focused on service composition contexts with layers and zones. We compute the similarity between composition contexts based on the matching of connection flows in zones with zone weight consideration instead of matching service labels or matching virtual documents.

Recommendation-based approaches were proposed by M. Lincoln et. al. [86] and T. Hornung et. al. [71]. In [86], the authors aimed at suggesting a generic method for designing new business process models encoded by the Process Descriptor Catalog notations. Meanwhile, in [71], the authors interpreted process descriptions in tags and retrieved related processes by using the open source Java search engine Lucene. Although they can recommend parts of business process, the process parts are defined by themselves. In our work, we recommend the relevant services based on the connections between the associated services and their neighbors.

S. Sakr et. al. [125] proposed a query language that takes into account the process models to manage business process variants. They, however, retrieved parts of processes based on strictly mapping to a structured input without considering the service similarity. Beheshti et. al. [15] proposed a query language to analyze Business processes execution from various user perspectives. A search framework that aims at retrieving process segments was proposed by M. Lincoln et. al. [85]. In their work, they defined the object grouping model (OGM) which includes the relationship between a primary object and others in a process segment. The weight of each edge is the number of its repetition in the related process model segments and term frequency-inverse document frequency (TF-IDF) is applied for the OGM-segment matching. In our approach, 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. In their work, they dynamically created business process segments based on the search phrase. They also transformed the search phrase into a process descriptor with object, action, object qualifier and action qualifier. For the object and action in the search phrase, they construct the action scope model, object grouping model and action influence model based on the existence of these items in the repositories. Then they matched the created segments with the constructed models to find the relevant segments.

Thomas Gschwind et. al. [64] applied workflow patterns for business process design. They aimed at helping business users understand the context and apply patterns during

the editing process. In our work, we help designers better design a business process by automatically recommending services that have similar contexts instead of patterns.

Different techniques have been defined for automatically discovering whole models from logs, e.g. [154, 94, 113, 51]. The automatic matching between event logs and process models is discussed in [13] showing that logs on the execution level are often much more detailed than models. The challenge of process mining is the observation that process models often turn out to be overwhelmingly complex, so-called spaghetti models [157]. Behavioral abstractions such as trace adjacency [12, 176] and weak order relations [52, 164] provide a means to compare process behavior [40, 78, 44]. These notions are applied, e.g., for identifying connections between actions [146] comparable to our notion of service composition. Our approach builds on this observation to make recommendations for executable processes. The approach reported in this chapter helps to present correlations between services in a context-specific way, which allows us to hide the complexity of the behavior. Hidden knowledge in process event logs are discovered for assisting process design [102, 99] or supporting the dynamic evolution of service protocols [124]. In this way, we complement log-based recommendation approaches to support process designers at runtime [14] or at configuration time [101, 100, 133].

4.6 Conclusion

In this chapter, we addressed the challenge of supporting process designer during the act of modeling, even in cases where no comparable process models exist. We present an approach that effectively utilizes knowledge extracted from process models or logs for recommending services. This approach is based on a notion of service composition and a corresponding calculation of similarity. Our approach helps process designers to flexibly adjust and interactively improve a designed process. Indeed, there are rationale and benefits behind introducing the concept of service composition context as it informs us about service behavior and thereafter can unveil its business context. By using this context, our objective is two-fold: (i) taking into account the process fragment surrounding a service as an input which would help to focus on specific parts of the business process and can avoid the computational complexity problem of the business process structure matching and (ii) benefiting from the existing process models and logs by extracting the implicit knowledge which is process fragments.

It is worthy to notice that our approach does not aim at finding services that have similar functions or capabilities with a selected one. Instead, we aim at retrieving services that have similar composition contexts. By recommending services, our approach not only helps to find suitable services for a missing position but also helps to find other alternatives for a selected service. These alternatives can be useful in either expanding a designed process to provide new functionality, replacing existing services or creating process variants. Moreover, we have also integrated our composition matching technique with a specific query language to help to retrieve services that have similar contexts based on a requested service context [30].

Due to the general limitation of public business process datasets, which provide only and service identifiers without any further information, the validation of our approach so far was done with only the perfect match of service identifiers. However, our approach can be easily improved to deal with other comparison metrics and the validation can be extended for the imperfect matching. In future work, we aim to extend the similarity calculation with other service properties such capability, QoS, consumed resources, etc..

This work was conducted during the PhD thesis of Nguyen Ngoc Chan [174]. The

service composition context mining approach has been proposed in the context of the Master thesis of Nattawat Nonsung [104]. Whereas the composition query language has been refined in the context of the Master thesis of Karn Yongsiriwit [171]. This work was mainly published in the following conference proceedings and journals [34, 33, 32, 32].

Chapter 5

Research Perspective

Today's fast changing environment imposes new challenges for effective management of business processes [68]. In such a highly dynamic environment, the business process design becomes time-consuming, error-prone, and costly [2]. Therefore, seeking reuse [55] and adaptability [131] is a strong requirement for a successful business process design. As mentioned in the preamble, a key element of my research activities is supporting business process modeling. My future and current work share the same general objectives as my previous activities namely: (i) supporting variability in configurable process models and (ii) semantically-enabled management of multi-tenants processes in the cloud.

5.1 Supporting variability in configurable processes

Configurable reference models introduced in [122, 63] were a step toward enabling *process design by reuse* while providing *flexiblility*. A configurable process model is a generic model that integrates multiple process variants of one business process in a given domain through variation points. These variation points are referred to as *configurable elements* and allow for multiple design variants. A process variant is an adjustment of a business process to flexibly adapt the business model to a specific context [45]. Enterprises or organizations usually need to support many variants of the same process due to constraints from regulations, geography, religion, etc. For example, car rental companies, such as Hertz, Avis or Sixt, need to customize their reservation process to follows laws in a country or culture of a region. Suncorp, one of the largest Australian insurance group, has developed more than 30 different variants of the process of handling an insurance claim [155].

In recent years, there have been many efforts on facilitating the development of business process variants such as using available reference models to be individualized to fit the requirements [148, 35], or finding existing similar models to inspire process design [170, 1, 40]. To build configurable process models, some approaches propose to merge existing process variants [118, 39, 62], others try to mine one configurable process model from execution logs [28, 61, 27]. To derive individual variants, some works propose to use questionnaires [120] or ontologies [72] in order to get business requirements and guide the configuration process. Others propose to use non functional requirements to assess configuration decisions on the process performance [129]. Despite the considerable advances achieved by exiting works on business process configuration, I identify three serious challenges:

1. First, configurable process models provided for reuse face two main issues. On one hand, merging or mining entire configurable process models can encounter the

problem of managing the complexity of the merged model when input variants are large and manifold [43]. In addition, building and recommending entire configurable processes cost much computation time, especially when a large number of process variants needs to be merged or mined. On the other hand, configurable process models provide only the possibility to configure and (re)use the entire process model; while in some circumstances, business designers may be interested in only *some parts*. For instance, a business designer may look for *process fragments* that are suitable to a missing part or that can replace some parts causing efficiency degradation.

2. Second, existing decision supports for the configuration of the process model rely heavily on the domain expert. The main difficulty in a configuration approach is to find the *interrelationship between configuration decisions*. That is, a decision support system must be able to predict next suitable configurations given a selected one. Existing approaches manually link business decisions to variation points by questioning a domain expert. Relying only on the knowledge of a domain expert, which may be helpful, is nevertheless error-prone and time-consuming.

3. Third, existing configuration approaches fail to derive optimal configurations according to individual user requirements. An optimal configuration results in an individualized process model that maximizes some performance metrics as for example, the most frequently used configuration, the configuration with a minimal execution time, minimal cost, or a combination of these three metrics, etc. This issue, identified in [53] as a serious limitation, has not been addressed before.

In light of the identified shortcomings, my main objectives are: (i) assist business designers to complete their processes or to create new variants; (ii) assist business analysts to develop a better understanding and reasoning on the variability in their configurable process models; (iii) automate the derivation of optimal configurations.

In order to realize the first objective I propose to use the notion of *composition context graph* defined in the chapter 4 as our process fragment model. Therefore, for a selected activity in an ongoing designed process, I propose to merge the neighborhood context graphs of the same activity in different business processes to create a merged fragment. Process mining techniques can be also considered in order to mine a configurable process fragment from multiple process event logs. Eventually, I can also propose a frequency-based approach that guides the configuration of the mined fragment. Concretely, I can explore the (in)frequent executions in the event logs in order to derive ranked configuration guidelines. Early results related to this objective have been already published [7, 8, 10].

The second objective can be achieved by introducing a frequency-based approach for extracting configuration rules that describe the interrelationships between the frequently selected configurations. In fact, I can take advantage of machine learning techniques [165], in particular association rule mining, and use the Apriori algorithm [4], one of the earliest and relevant proposed algorithms, in order to extract the interdependencies among the variation points. Early results related to this objective have been already published [9].

For the third contribution, I propose to use existing works in process mining for performance analysis [159, 110, 36] in order to mine performance metrics for the variation points. Basically, I can leverage the business configuration to a *Constraint Satisfaction Problem* (CSP) [153] in order to derive a configuration that satisfies a user defined performance constraints. We use a CSP solver along with a user specified objective function to find all the optimal configurations.

This ongoing and future work is being conducted as part of the PhD thesis of Nour Assy [6].

5.2 Semantically-enabled management of processes in the cloud

Cloud Computing allows companies to optimize their processes by providing dynamically scalable and often virtualized resources on demand. It is changing the way in which business processes are managed and supported [156]. Basically, cloud computing can provide a framework where different companies essentially perform the same process, while sharing best practices, knowledge, or a common infrastructure as a set of services deployed in a shared cloud [155]. Multi-tenant processes are organization-specific variants of the same process running in a cloud infrastructure. Since many organizations may work on similar processes with some variations, *configurable processes* have been proposed as a key technique for a flexible process design in cloud environment. Indeed, as tenants may have different needs and preferences, and cloud providers may present different offers, using configurable process models to support variability is straightforward in cloud environment. By configuring a configurable process model in the cloud, one can expect to obtain a concrete process variant and a concrete resource allocation. Consequently, the configurable model needs to be able to support meaningful variations of the same process according to the available cloud offers. However, a systematic approach to support and analyze multi-tenant processes and related cloud offers is missing [156].

Current configurable models focus mainly on the control flow and ignore other aspects (resources, data). Particularly, resource perspective and allocation policies are currently missing in (configurable) process models. The resource perspective remained poorly described [69]. Many works have been realized for providing support and enhancement to the resource management in business processes development. Nevertheless, they have basically focused on human resources [91, 149] and have neglected other types of resources particularly cloud resources [75]. As cloud environments are heterogeneous, the need for a common vocabulary in order to share knowledge is a crucial issue. Therefore, seeking for defining resource perspective, in a semantic way, is an important issue for a successful business process management in the cloud.

The general idea is to link (configurable) process models to cloud resources and allocation policies to express such integrated specification in a format understandable by the domain experts and at the same time processable by the machine. Concretely, I aim at extending configurable process with the semantic capability descriptions of their tasks and consumed cloud resources. In order to augment the synergy between cloud providers and process owners, I aim at building a semantic knowledge base as a first step towards the optimization of multi-tenant process and cloud resource management.

The first objective of my future work under this research dimension is specifying a semantically-enabled configurable process model which subsumes the behaviour of all variants models and most importantly captures the variation points that represent differences between different multi-tenants processes. I aim at developing an ontology enabling to abstract from configurable processes languages. I propose to specify also a frame-based semantic meta-model for describing the capabilities of the cloud resources by capturing their functional and non-functional aspects. A resource refers to what a cloud service achieves. We enhance configurable processes with the descriptions of the capabilities of their resources defined according to our semantic meta-model. Our model explicitly features resource properties which configurations are conducted based on. This allows for comparing processes within different organizations and resources from different cloud providers. Consequently, multiple organizations can share information about their similar processes. Once the semantic model is defined, it becomes relatively easy to compare

different variants of the same configurable model running in the cloud, share and optimize their consumed resources. The goal is to let organizations learn from each other and establish proven best practices. Of course privacy issues may again complicate such analysis, however, it may be sufficient to compare things at an aggregate level or to anonymize the results. This is of course only possible in a non-competitive environment [156], e.g., different branches of some multinational organization, franchises, municipalities, courts, etc..

The second objective of my future work under this research dimension is developing algorithms to automatically generate optimal process variants from a cloud resource consumption perspective. Using a set of rules performed over the previously specified knowledge base, I aim to proactively/provisionally allocate resources based on business constraints, service level agreement (SLA), QoS parameters such as performance, security and privacy. I also intend to specify monitoring technique to manage the dynamic change of cloud resources to match new business requirements. We need also to ensure the correctness of a process family and all of its configurations according to privacy, security and optimization perspectives.

This ongoing and future work is being conducted as part of the PhD thesis of Karn Yongsiriwit [172].

Bibliography

[1] W. Aalst, A. Medeiros, and A. Weijters. Process equivalence: Comparing two process models based on observed behavior. In *BPM*, pages 129–144, 2006.

[2] W. M. P. v. d. Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 161–183, London, UK, UK, 2000. Springer-Verlag.

[3] M. A. Abdulrahim. *Parallel algorithms for labeled graph matching.* PhD thesis, Golden, CO, USA, 1998. AAI0599838.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.

[5] S. Arroyo, E. Cimpian, J. Domingue, C. Feier, D. Fensel, B. König-Ries, H. Lausen, A. Polleres, and M. Stollberg. Web Service Modelling Ontology Primer. World Wide Web Consortium Member Submission, 2005.

[6] N. Assy. *Supporting variability in configurable business process.* PhD thesis, Telecom SudParis, Evry, France, expected 2015.

[7] N. Assy, N. N. Chan, and W. Gaaloul. Assisting business process design with configurable process fragments. In *IEEE SCC*, pages 535–542, 2013.

[8] N. Assy, N. N. Chan, W. Gaaloul, and B. Defude. Deriving configurable fragments for process design. *International Journal of Business Process Integration and Management*, 7(1):2–21, 2014.

[9] N. Assy and W. Gaaloul. Configuration rule mining for variability analysis in configurable process models. In *International conference on service oriented computing ICSOC. To appear*, 2014.

[10] N. Assy, W. Gaaloul, and B. Defude. Mining configurable process fragments for business process design. In *Advancing the Impact of Design Science: Moving from Theory to Practice - 9th International Conference, DESRIST 2014, Miami, FL, USA, May 22-24, 2014. Proceedings*, volume 8463, pages 209–224, 2014.

[11] E. Ayorak and A. B. Bener. Super peer web service discovery architecture. In *International Conference on Data Engineering, 2007, Istanbul, Turkey*, 2007.

[12] J. Bae, L. Liu, J. Caverlee, L.-J. Zhang, and H. Bae. Development of distance measures for process mining, discovery and integration. *Int. J. Web Service Res.*, 4(4):1–17, 2007.

[13] T. Baier and J. Mendling. Bridging abstraction layers in process mining by automated matching of events and activities. In *BPM*. 2013.

[14] I. Barba, B. Weber, C. D. Valle, and A. J. Ramirez. User recommendations for the optimized execution of business processes. *Data Knowl. Eng.*, 86:61–84, 2013.

[15] S.-M.-R. Beheshti, B. Benatallah, H. R. M. Nezhad, and S. Sakr. A query language for analyzing business processes execution. In *BPM*, pages 281–297, 2011.

[16] B. Benatallah, M.-S. Hacid, H.-Y. Paik, C. Rey, and F. Toumani. Peering and querying e-catalog communities. In *ICDE*, page 846, 2004.

[17] B. Benatallah, Q. Z. Sheng, and M. Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, 7:40–48, 2003.

[18] J. Bentahar, Z. Maamar, D. Benslimane, and P. Thiran. An argumentation framework for communities of web services. *IEEE Intelligent Systems*, 22(6):75–83, 2007.

[19] M. W. Berry. Large scale sparse singular value computations. *International Journal of Supercomputer Applications*, pages 13–49, 1992.

[20] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995.

[21] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, 1981.

[22] S. Bhiri, W. Gaaloul, M. Rouached, and M. Hauswirth. Semantic web services for satisfying soa requirements. In *Advances in Web Semantics I*, pages 374–395. 2009.

[23] 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.

[24] M. B. Blake and M. F. Nowlan. A web service recommender system using enhanced syntactical matching. In *ICWS*, pages 575–582, 2007.

[25] O. Bouchaala. managing communities of web services registries. Master's thesis, Telecom SudParis, Evry, France, 2011.

[26] O. Bouchaala, M. Sellami, W. Gaaloul, S. Tata, and M. Jmaiel. Modeling and managing communities of web service registries. In *Web Information Systems and Technologies*, volume 101 of *Lecture Notes in Business Information Processing*, pages 88–102. Springer Berlin Heidelberg, 2012.

[27] J. C. A. M. Buijs and H. A. Reijers. Comparing business process variants using models and event logs. In *BMMDS/EMMSAD*, pages 154–168, 2014.

[28] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. Mining configurable process models from collections of event logs. In *BPM '13*.

[29] Y. Chabeb, S. Tata, and A. Ozanne. YASA-M: A semantic web service matchmaker. In *AINA 2010, April 20-23, Perth, Australia*, 2010.

[30] N. N. Chan and W. Gaaloul. Querying services based on composition context. In *WETICE*, 2014.

[31] N. N. Chan, W. Gaaloul, and S. Tata. Collaborative filtering technique for web service recommendation based on user-operation combination. In *OTM Conferences (1)*, pages 222–239, 2010.

[32] N. N. Chan, W. Gaaloul, and S. Tata. Composition context matching for web service recommendation. In *IEEE SCC*, pages 624–631, 2011.

[33] N. N. Chan, W. Gaaloul, and S. Tata. A recommender system based on historical usage data for web service discovery. *Service Oriented Computing and Applications*, 6(1):51–63, 2012.

[34] N. N. Chan, K. Yongsiriwit, W. Gaaloul, and J. Mendling. Mining event logs to assist the development of executable process variants. In *CAiSE*, pages 548–563, 2014.

[35] T. Curran, G. Keller, and A. Ladd. *SAP R/3 business blueprint: understanding the business process reference model*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[36] G. C.W. *Process Mining in Flexible Environments*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2008.

[37] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Informatioin Science*, 41(6):391–407, 1990.

[38] S. C. Deerwester, S. T. Dumais, G. W. Furnas, R. A. Harshman, T. K. Landauer, K. E. Lochbaum, and L. A. Streeter. Computer information retrieval using latent semantic structure. *US Patent No. 4839853*, June 1989.

[39] W. Derguech and S. Bhiri. An automation support for creating configurable process models. In A. Bouguettaya, M. Hauswirth, and L. Liu, editors, *WISE*, volume 6997 of *Lecture Notes in Computer Science*, pages 199–212. Springer, 2011.

[40] R. Dijkman, M. Dumas, and L. 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.

[41] R. Dijkman, M. Dumas, B. van Dongen, R. Käärik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 36(2):498–516, Apr. 2011.

[42] R. M. Dijkman, M. L. Rosa, and H. A. Reijers. Managing large collections of business process models - current techniques and challenges. *Computers in Industry*, 63(2):91–97, 2012.

[43] R. M. Dijkman, M. L. Rosa, and H. A. Reijers. Managing large collections of business process models - current techniques and challenges. *Computers in Industry*, 2012.

[44] R. M. Dijkman, B. F. van Dongen, M. Dumas, L. García-Bañuelos, M. Kunze, H. Leopold, J. Mendling, R. Uba, M. Weidlich, M. Weske, and Z. Yan. A short survey on process model similarity. In *Seminal Contributions to Information Systems Engineering*. Springer, 2013.

[45] M. Döhring, H. A. Reijers, and S. Smirnov. Configuration vs. adaptation for business process variant maintenance: An empirical study. *Inf. Syst.*, 39:108–133, 2014.

[46] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 372–383, 2004.

[47] M. Dumas, L. García-Bañuelos, M. L. Rosa, and R. Uba. Fast detection of exact clones in business process model repositories. *Inf. Syst.*, 38(4):619–633, 2013.

[48] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013.

[49] J. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3:32–57, 1973.

[50] M. Ehrig, A. Koschmider, and A. Oberweis. Measuring similarity between semantic business process models. In *Proceedings of the fourth Asia-Pacific conference on Comceptual modelling - Volume 67*, APCCM '07, pages 71–80, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[51] C. C. Ekanayake, M. Dumas, L. García-Bañuelos, and M. L. Rosa. Slice, mine and dice: Complexity-aware automated discovery of business process models. In *BPM*, pages 49–64, 2013.

[52] R. Eshuis and P. W. P. J. Grefen. Structural matching of bpel processes. In *ECOWS 2007*, pages 171–180. IEEE Computer Society, 2007.

[53] G. F. *Configurable process models*. PhD thesis, Technical University of Eindhoven, Eindhoven, The Netherlands, 2009.

[54] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Instantaneous soundness checking of industrial business process models. In *7th BPM*, pages 278–293, 2009.

[55] P. Fettke and P. Loos. Classification of reference models: a methodology and its application. *Information Systems and eBusiness Management*, 2003.

[56] W. Gaaloul. *Transactional workflow mining for reliable executions*. PhD thesis, University of Lorraine, Nancy, France, 2006.

[57] L. García-Bañuelos, M. Dumas, M. L. Rosa, J. D. Weerdt, and C. C. Ekanayake. Controlled automated discovery of collections of business process models. *Inf. Syst.*, 46:85–101, 2014.

[58] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, Dec. 1992.

[59] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151, July 2001.

[60] G. H. Golub and C. F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[61] F. Gottschalk, W. Aalst, and M. Jansen-Vullers. Mining Reference Process Models and their Configurations. In *EI2N08, OTM 2008 Workshops*.

[62] F. Gottschalk, W. M. Aalst, and M. H. Jansen-Vullers. Merging event-driven process chains. In *OTM '08*.

[63] F. Gottschalk and et al. Configurable workflow models. *Int. J. Cooperative Inf. Syst.*, 2008.

[64] T. Gschwind, J. Koehler, and J. 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.

[65] M. Guentert, M. Kunze, and M. Weske. Evaluation measures for similarity search results in process model repositories. In P. Atzeni, D. Cheung, and S. Ram, editors, *ER '12*, volume 7532, pages 214–227. Springer Berlin Heidelberg, 2012.

[66] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, January 2004.

[67] G. Hermosillo, L. Seinturier, and L. Duchien. Creating context-adaptive business processes. In *ICSOC*, pages 228–242, 2010.

[68] G. Hermosillo, L. Seinturier, and L. Duchien. Using complex event processing for dynamic business process adaptation. In *IEEE SCC*, pages 466–473, 2010.

[69] P. Hoenisch and et al. Workflow scheduling and resource allocation for cloud-based execution of elastic processes. In *IEEE 6th International Conference on Service-Oriented Computing and Applications, Koloa, USA, December 2013*, pages 1–8.

[70] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, Jan. 2004.

[71] T. Hornung, A. Koschmider, and G. 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.

[72] Y. Huang, Z. Feng, K. He, and Y. Huang. Ontology-based configuration for service-based business process model. In *IEEE SCC*, pages 296–303, 2013.

[73] J. J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *The Computing Research Repository, CoRR*, cmp-lg/9709008, 1997.

[74] M. Jursic and N. Lavrac. Fuzzy clustering of documents. In *Conference on Data Mining and Data Warehouses, SiKDD 2008, Ljubljana, Slovenia*, 2008.

[75] S. Kächele and et al. Beyond iaas and paas: An extended cloud taxonomy for computation, storage and networking. In *Proceedings of the 6th IEEE/ACM International Conference Utility and Cloud Computing UCC*, USA, 12 2013. IEEE.

[76] N. Kokash, A. Birukou, and V. 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.

[77] P. Küngas, M. Dumas, S. Mokarizadeh, and M. Matskin. Analyzing web services networks: Theory and practice. In *Advanced Web Services*, pages 381–406. 2014.

[78] M. Kunze and M. Weske. Metric trees for efficient similarity search in large process model repositories. In M. zur Muehlen and J. Su, editors, *BPM Workshops*, volume 66 of *Lecture Notes in Business Information Processing*. Springer, 2010.

[79] H. Lausen and J. Farrell. Semantic annotations for WSDL and XML schema. W3C recommendation, W3C, Aug. 2007. http://www.w3.org/TR/2007/REC-sawsdl-20070828/.

[80] S. L. Leng. Recommender system for enhancing web services discovery. Master's thesis, Telecom SudParis, Evry, France, 2009.

[81] H. Leopold, M. Niepert, M. Weidlich, J. Mendling, R. Dijkman, and H. Stuckenschmidt. Probabilistic optimization of semantic process model matching. In *BPM*, 2012.

[82] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966.

[83] C. Li, M. Reichert, and A. 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.

[84] D. Lin. An information-theoretic definition of similarity. In *The Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconson, USA, July 24-27, 1998*, pages 296–304, 1998.

[85] M. Lincoln and A. 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.

[86] M. Lincoln, M. Golani, and A. 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.

[87] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, Jan/Feb 2003.

[88] G. D. Linden, J. A. Jacobi, and E. A. Benson. Collaborative recommendations using item-to-item similarity mappings, July 2001.

[89] X. Liu, G. Huang, and H. Mei. Discovering homogeneous web service community in the user-centric web environment. *IEEE T. Services Computing*, 2(2):167–181, 2009.

[90] J. Ma, Y. Zhang, and J. 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.

[91] Z. Maamar and et al. Towards a user-centric social approach to web services composition, execution, and monitoring. In *Web Information Systems Engineering,WISE, 13th International Conference, Paphos, Cyprus, November 2012. Proceedings*, pages 72–86.

[92] Z. Maamar, S. Sattanathan, P. Thiran, D. Benslimane, and J. Bentahar. An approach to engineer communities of web services - concepts, architecture, operation, and deployment. *International Journal of E-Business Research (IJEBR)*, 9(4), Dec. 2009.

[93] U. S. 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.

[94] R. Mans, H. A. Reijers, H. Berends, W. Bandara, and R. Prince. Business process mining success. In *ECIS*, page 89, 2013.

[95] D. Martin et al. Owl-s: Semantic markup for web services. Technical report, 2004.

[96] B. Medjahed and A. Bouguettaya. A dynamic foundational architecture for semantic web services. *Distributed and Parallel Databases*, 17(2):179–206, 2005.

[97] A. Mesmoudi, M. Mrissa, and M.-S. Hacid. Combining configuration and query rewriting for web service composition. In *ICWS*, pages 113–120, 2011.

[98] S. Mosser, G. Hermosillo, A.-F. L. Meur, L. Seinturier, and L. Duchien. Undoing event-driven adaptation of business processes. In *IEEE SCC*, pages 234–241, 2011.

[99] K. Musaraj, T. Yoshida, F. Daniel, M.-S. Hacid, F. Casati, and B. Benatallah. Message correlation and web service protocol mining from inaccurate logs. In *ICWS*, pages 259–266, 2010.

[100] H. R. M. Nezhad, B. Benatallah, F. Casati, R. Saint-Paul, P. Andritsos, and A. Guabtni. Exploration of discovered process views in process spaceship. In *ICSOC*, pages 724–725, 2008.

[101] H. R. M. Nezhad, B. Benatallah, R. Saint-Paul, F. Casati, and P. Andritsos. Process spaceship: discovering and exploring process views from event logs in data spaces. *PVLDB*, 1(2):1412–1415, 2008.

[102] H. R. M. Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah. Event correlation for process discovery from web service interaction logs. *VLDB J.*, 20(3):417–444, 2011.

[103] C. N. Nguyen. *Service Recommendation for Individual and Process Use.* PhD thesis, Telecom SudParis, Evry, France, 2012.

[104] N. Nonsung. Retrieving similar activities in business process. Master's thesis, Telecom SudParis, Evry, France, 2012.

[105] S.-C. Oh, H. Kil, D. Lee, and S. R. T. Kumara. WSBen: A Web Services Discovery and Composition Benchmark. In *ICWS, Chicago, Illinois, USA*, 2006.

[106] H.-Y. Paik, B. Benatallah, and R. Hamadi. Dynamic restructuring of e-catalog communities based on user interaction patterns. *World Wide Web*, 5(4):325–366, 2002.

[107] H.-Y. Paik, B. Benatallah, and F. Toumani. Toward self-organizing service communities. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 35(3):408–419, 2005.

[108] T. Pilioura and A. Tsalgatidou. Unified publication and discovery of semantic web services. *ACM Transactions on the Web (TWEB)*, 3(3), 2009.

[109] 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.

[110] H. P.T.G. Performance analysis of business processes through process mining, 2007.

[111] G. Qian, S. Sural, Y. Gu, and S. Pramanik. Similarity between euclidean and cosine angle distance for nearest neighbor queries. In *ACM symposium on Applied computing*, 2004.

[112] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, San Diego, CA, USA*, 2001.

[113] H. Reguieg, F. Toumani, H. R. M. Nezhad, and B. Benatallah. Using mapreduce to scale events correlation discovery for business processes mining. In *BPM*, pages 279–284, 2012.

[114] P. Resnick and H. R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, Mar. 1997.

[115] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *The 1995 International Joint Conference on AI, IJCAI-95, 20-25 August 1995,Montreal/Quebec, Canada*, pages 448–453, 1995.

[116] C. J. V. Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[117] C. Rodríguez, S. R. Chowdhury, F. Daniel, H. R. M. Nezhad, and F. Casati. Assisted mashup development: On the discovery and recommendation of mashup composition knowledge. In *Web Services Foundations*, pages 683–708. 2014.

[118] L. Rosa and et al. Business process model merging: An approach to business process consolidation. *ACM Trans. Softw. Eng. Methodol. '13*.

[119] M. L. Rosa, M. Dumas, R. Uba, and R. M. Dijkman. Business process model merging: An approach to business process consolidation. *ACM Trans. Softw. Eng. Methodol.*, 22(2):11, 2013.

[120] M. L. Rosa and et al. Questionnaire-based variability modeling for system configuration. *Software and System Modeling*, 8(2):251–274, 2009.

[121] M. L. Rosa, H. A. Reijers, W. M. P. van der Aalst, R. M. Dijkman, J. Mendling, M. Dumas, and L. García-Bañuelos. Apromore: An advanced process model repository. *Expert Syst. Appl.*, 38(6):7029–7040, 2011.

[122] M. Rosemann and W. M. P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 2007.

[123] 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, Dec. 2007.

[124] S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul. Supporting the dynamic evolution of web service protocols in service-oriented architectures. *TWEB*, 2(2), 2008.

[125] S. Sakr, E. Pascalau, A. Awad, and M. Weske. Partial process models to manage business process variants. *International Journal of Business Process Integration and Management (IJBPIM)*, 6(2):20, September 2011.

[126] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. Technical report, 1987.

[127] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[128] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11), 1975.

[129] E. Santos and et al. Configuring the variability of business process models using non-functional requirements. In *BMMDS/EMMSAD*, pages 274–286, 2010.

[130] R. Saraçoğlu, K. Tütüncü, and N. Allahverdi. A fuzzy clustering approach for finding similar documents using a novel similarity measure. *Expert Syst. Appl.*, 33(3), 2007.

[131] H. Schonenberg and et al. Towards a taxonomy of process flexibility. In *CAiSE Forum*, pages 81–84, 2008.

[132] H. Schonenberg, B. Weber, B. Dongen, and W. 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.

[133] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Softw., Pract. Exper.*, 42(5):559–583, 2012.

[134] M. Sellami. *Web services registries discovery in a distributed environment*. PhD thesis, Telecom SudParis, Evry, France, 2010.

[135] M. Sellami, O. Bouchaala, W. Gaaloul, and S. Tata. WSRD: A Web Services Registry Description. In *International conference on New Technologies of Distributed Systems, 2010, Tozeur, Tunisia*, 2010.

[136] M. Sellami, O. Bouchaala, W. Gaaloul, and S. Tata. Communities of web service registries: Construction and management. *Journal of Systems and Software*, 86(3):835–853, 2013.

[137] M. Sellami, W. Gaaloul, and B. Defude. Data mapping web services for composite daas mediation. In *WETICE*, pages 36–41, 2012.

[138] M. Sellami, W. Gaaloul, B. Defude, and S. Tata. Towards a unified marketplace for functionality-based cloud service discovery. In *CLOSER*, pages 252–257, 2012.

[139] M. Sellami, W. Gaaloul, and S. Tata. Functionality-driven clustering of web service registries. In *IEEE SCC*, pages 631–634, 2010.

[140] M. Sellami, W. Gaaloul, and S. Tata. Implementation of communities of web service registries. In *ICWS*, pages 690–691, 2011.

[141] M. Sellami, W. Gaaloul, S. Tata, and M. Jmaiel. Using recommendation to limit search space in web services discovery. In *24th IEEE International Conference on Advanced Information Networking and Applications, Perth, Australia*, 2010.

[142] M. Sellami, S. Tata, and B. Defude. Service discovery in ubiquitous environments: Approaches and requirements for context-awareness. In *Proceedings of the Business Process Management Workshops, Milano, Italy*, pages 516–522. Springer Verlag, 2008.

[143] M. Sellami, S. Tata, Z. Maamar, and B. Defude. A recommender system for web services discovery in a distributed registry environment. In *International Conference on Internet and Web Applications and Services, 2009, May, Venice, Italy*, 2009.

[144] S. Sioutas, E. Sakkopoulos, C. Makris, B. Vassiliadis, A. Tsakalidis, and P. Triantafillou. Dynamic web service discovery architecture based on a novel peer based overlay network. *Journal of Systems and Software*, 82(5):809 – 824, 2009.

[145] K. Sivashanmugam, K. Verma, and A. P. Sheth. Discovery of web services in a federated registry environment. In *Proceedings of the IEEE International Conference on Web Services, San Diego, California, USA*, 2004.

[146] S. Smirnov, M. Weidlich, J. Mendling, and M. Weske. Action patterns in business process model repositories. *Computers in Industry*, 2012.

[147] S. Soi, F. Daniel, and F. Casati. Conceptual design of sound, custom composition languages. In *Web Services Foundations*, pages 53–79. 2014.

[148] S. 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.

[149] L. J. R. Stroppi and et al. Extending the ws-humantask architecture to support the resource perspective of bpel processes. *CLEI Electron. J.*, 2013.

[150] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

[151] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

[152] The Mckinsey Quarterly. How businesses are using web 2.0: A mckinsey global survey, 2007.

[153] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1995.

[154] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, Sept. 2004.

[155] W. M. P. Van Der Aalst. Configurable services in the cloud: supporting variability while enabling cross-organizational process mining. In *Proceedings of the 2010 international conference on On the move to meaningful internet systems - Volume Part I*, OTM'10, pages 8–25, Berlin, Heidelberg, 2010. Springer-Verlag.

[156] W. M. P. van der Aalst. Business process configuration in the cloud: How to support and analyze multi-tenant processes? In *ECOWS*, pages 3–10, 2011.

[157] W. M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes.* Springer, 2011.

[158] W. M. P. van der Aalst, M. Dumas, F. Gottschalk, A. H. M. ter Hofstede, M. L. Rosa, and J. Mendling. Preserving correctness during business process model configuration. *Formal Asp. Comput.*, 22(3-4):459–482, 2010.

[159] W. M. P. van der Aalst and B. F. van Dongen. Discovering workflow performance models from timed logs. In *Engineering and Deployment of Cooperative Information Systems, First International Conference, EDCIS 2002, Beijing, China, September 17-20, 2002, Proceedings*, pages 45–63, 2002.

[160] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. *Inf. Technol. and Management*, 6(1):17–39, 2005.

[161] B. Weber, M. Reichert, J. Mendling, and H. A. Reijers. Refactoring large process model repositories. *Computers in Industry*, 62(5):467–486, 2011.

[162] I. Weber, H.-Y. Paik, and B. Benatallah. Form-based web service composition for domain experts. *TWEB*, 8(1):2, 2013.

[163] M. Weidlich, R. Dijkman, and J. 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.

[164] M. Weidlich, J. Mendling, and M. Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Soft. Eng.*, 37:410–429, 2011.

[165] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., 2005.

[166] C. Wu, V. Potdar, and E. Chang. *Advances in Web Semantics I: Ontologies, Web Services and Applied Semantic Web*, chapter Latent Semantic Analysis — The Dynamics of Semantics Web Services Discovery, pages 346–373. Springer-Verlag, Berlin, Heidelberg, 2009.

[167] C.-T. Wu and H.-F. Wang. Recent development of recommender systems. *Industrial Engineering and Engineering Management, 2007 IEEE International Conference on*, pages 228–232, Dec. 2007.

[168] B. Xu and D. Chen. Semantic web services discovery in p2p environment. In *International Conference on Parallel Processing Workshops*. IEEE Computer Society, 2007.

[169] Y. Xu, S. Tang, Y. Xu, R. Xiao, and L. Fang. Discovering web services based on match history. In *Advances in Intelligent Web Mastering, Proceedings of the 5th Atlantic Web Intelligence Conference, Fontainbleau, France*, 2007.

[170] Z. Yan, R. Dijkman, and P. 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.

[171] K. Yongsiriwit. Process fragment querying. Master's thesis, Telecom SudParis, Evry, France, 2013.

[172] K. Yongsiriwit. *Modelling and mining process variants in Cloud environments*. PhD thesis, Telecom SudParis, Evry, France, expected 2016.

[173] L. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.

[174] M. Zaremba. *Semantically-enabled Service Discovery and Late Binding*. PhD thesis, Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland, 2010.

[175] M. Zaremba, T. Vitvar, S. Bhiri, and M. Hauswirth. Preference-based discovery of dynamically generated service offers. In *International Conference on Services Computing, SCC 2011, Washington, DC, USA, 4-9 July*, pages 338–345, 2011.

[176] H. Zha, J. Wang, L. Wen, C. Wang, and J. Sun. A workflow net similarity measure based on transition adjacency relations. *Computers in Industry*, 61(5):463–471, 2010.

[177] Z. Zhou, S. Bhiri, H. Zhuge, and W. Gaaloul. Assessment of service protocol adaptability based on novel walk computation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 42(5):1109–1140, 2012.

[178] Z. Zhou, W. Gaaloul, L. Shu, S. Tata, and S. Bhiri. Assessing the replaceability of service protocols in mediated service interactions. *Future Generation Comp. Syst.*, 29(1):287–299, 2013.

[179] Z. Zhou, M. Sellami, W. Gaaloul, M. Barhamgi, and B. Defude. Data providing services clustering and management for facilitating service discovery and replacement. *IEEE T. Automation Science and Engineering*, 10(4):1131–1146, 2013.

[180] Z. Zhou, M. Sellami, W. Gaaloul, and B. Defude. Clustering and managing data providing services using machine learning technique. In *International Conference on Semantics Knowledge and Grid (SKG 2011), Beijing, China, October 24-26*, pages 225–232, 2011.