Service Querying to Support Process Variant Development

Ngoc Chan Nguyen, Nattawat Nonsung, Walid Gaaloul

Service Querying to Support Process Variant Development

Nguyen Ngoc Chan[a,*], Nattawat Nonsung[b], Walid Gaaloul[c]

[a] *Université de Lorraine, LORIA UMR 7503, France*
[b] *SiteMinder, Sydney NSW, 2000 Australia*
[c] *Télécom SudParis, Samovar UMR 5157, France*

Abstract

Developing process variants enables enterprises to effectively adapt their business models to different markets. Existing approaches focus on business process models to support the variant development. The assignment of services in a business process, which ensures the process variability, has not been widely examined. In this paper, we present an innovative approach that focuses on component services instead of process models. We target to recommend services to a selected position in a business process. We define the service composition context as the relationships between a service and its neighbors. We compute the similarity between services based on the matching of their composition contexts. Then, we propose a query language that considers the composition context matching for service querying. We developed an application to demonstrate our approach and performed different experiments on a public dataset of real process models. Experimental results show that our approach is feasible and efficient.

Keywords:
service composition, service-based business process, process variant, service querying, composition context matching

*Corresponding author. Phone: +33-3-5495-8523; Fax: +33-3-8327-8319
Email addresses: ngoc-chan.nguyen@loria.fr (Nguyen Ngoc Chan), nattawat.nonsung@siteminder.com (Nattawat Nonsung), walid.gaaloul@mines-telecom.fr (Walid Gaaloul)

1. Introduction

Variability has been considered as a key factor that enables software systems to be extended, changed, customized, or configured for use in a specific context [1, 2]. Enterprises or organizations usually need to support variability to adapt their business models to different markets. For example, car rental companies, such as Hertz, Avis or Sixt, need to customize their reservation process to follow 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 [3].

In service-based process modeling and execution, variability plays an important role as it enables a dynamic environment in which services can be replaced or reconfigured to adapt to different circumstances [4]. It not only supports the development of business process variants, but also brings out several advantages in process configuration. Concretely, it enhances system availability (by replacing an unavailable service by another), supports runtime configuration (by rebinding services at runtime), optimizes performance (by service replacement if necessary) or optimizes the quality attributes (by changing the configuration of the system) [4, 5]. Two contexts that require service variability include: finding alternatives of a service in a business process (see Figure 1a) and retrieving adequate services that can be plugged into some selected positions (see Figure 1b). These requirements have raised research challenges in service matching and querying. These challenges are amplified by the continuous increasing of the number of services and process models along with the maturity of business process management [6, 7].

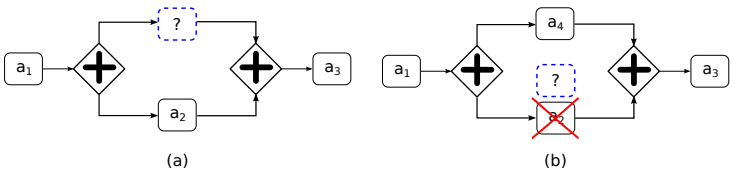


Figure 1: Finding services to be composed in a business process

Early approaches focus on service characteristics to support service discovery. Some of them analyze service descriptions [8, 9, 10, 11], study the QoS of services [12, 13, 14], while others are based on semantic concepts[15, 16, 17, 18]. Different methods in information retrieval, data mining and artificial intelligence domains have been experimented, such as collaborative filtering [9, 12, 13, 17], associated rules [19, 20, 21], clustering [8, 11, 22],

divide and conquer [11]. In contrast, recent approaches pay much attention on business process models. Instead of retrieving services, they attempt to evaluate the behavior equivalence of services in business processes [23, 24], compare process models [25, 26, 27, 28, 29, 30, 31], develop configurable process models [32, 33, 34], or build a query language that supports searching for execution paths in business processes [35, 36, 37, 38, 39].

In this paper, we address the research problem of finding suitable services for selected positions in a business process. We propose an approach that takes into account *service composition context*, which is defined as relations between a given service and its neighbors. These relations are exposed by sequences of flows and connection elements between them, i.e., AND, OR, XOR, etc. They are labeled by the names of services and connection elements. We take existing process models as input in order to learn from the past design. We exploit the knowledge acquired for previous designed process to infer service similarity. Concretely, we present the service composition context as a graph in which the considered service is positioned at the center. We compute the similarity between services based on the matching of their context graphs. We also propose a query language as a tool to search for similar services.

It is worthy to notice that the service composition context can be represented by any of existing process modeling languages, such as Petri net, UML diagram, EPC, BPMN and YAWL [39]. Business process ontology [40] and semantic annotation [41, 42] can also be applied. The most importance issue is that the relation between services, which is defined as sequences of connection elements, is represented. In our approach, we use BPMN and graph theory to model the service composition context as they are one of the most popular tools for process modeling [43] and suitable for service relation formalization.

The objective of our approach is twofold: (i) to propose a new approach for the computation of similarity between component services in services-based processes and (ii) to provide a useful query tool for process variant development. By identifying adequate services for selected positions, our approach not only supports the process creation during the design time, but also enables the service configuration during the runtime.

The work presented in this paper is an extension of our previous work [44], in which the composition context is improved to take into account parallel relations between services and similarity between connection elements. Richer experiments and deeper discussion are also provided.

Our paper is organized as follows. The next section presents a motivating example. Section 3 introduces definitions and notations. Detail of the service similarity computation is presented in section 4. A query language is proposed in section 5. An implementation and different experimental results are shown in section 6. Section 7 presents related work. Finally, section 8 concludes our work and provides an outline on the future work.

2. Motivating Example

Consider a travel agency. Suppose that they have had in their information system two business process models named 'flight-reservation' and 'hotel-reservation', which correspond to flight and hotel booking functions (see Figure 2).

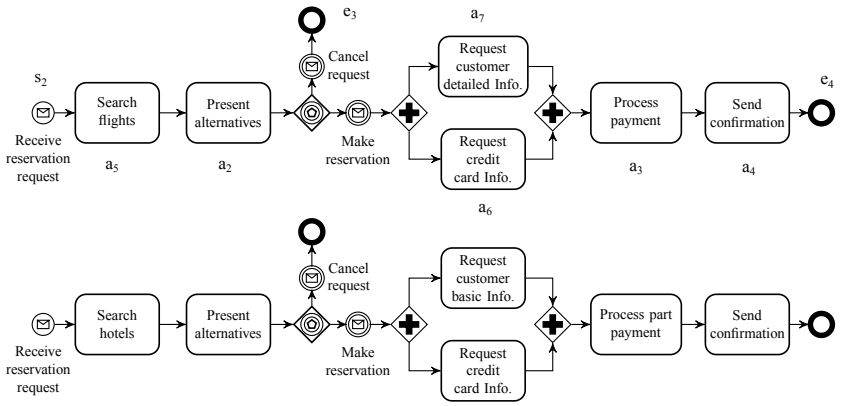


Figure 2: Flight & hotel reservation processes

The travel agency decides to design a new process to offer a new train booking function. The process designer can rapidly sketch out the train-reservation process with some basic services as shown in Figure 3. Suppose that he is looking for a service, which is annotated by a round-corner rectangle with a '?' symbol. This requested service has to fulfill some composition constraints, such as: it has to be executed before 'Process payment' and/or executed after 'Present alternatives' and/or has similar connection flows with the missing position, etc.

Existing approaches, such as text-matching mechanism and process equivalence, are not applicable for this inquiry as they do not provide a way to express composition constraints. Text-matching approaches do not consider

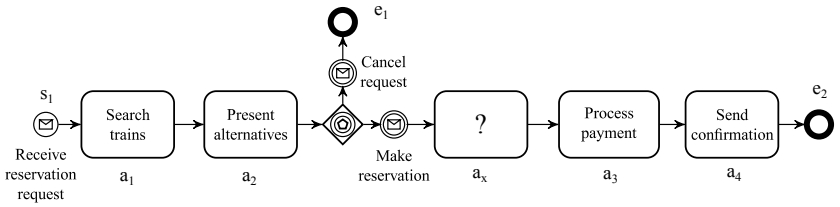


Figure 3: An incomplete train reservation process

the interactions between services in business processes.Whereas, process-querying approaches return similar process models instead of services. In this case, our approach can be applied as it focuses on the composition context around services. Concretely, it detects that the 'unknown' service has similar context with the 'Request customer detailed Info.', 'Request customer basic Info.' and 'Request credit card Info.' in the existing flight & hotel reservation processes (Figure 2) because they have similar connections from/to the same services, which are 'Present alternatives' and 'Process payment'. So, we recommend these services for the missing position and the process designer may select the 'Request credit card Info.' service for this position as it is the most suitable (Figure 4).

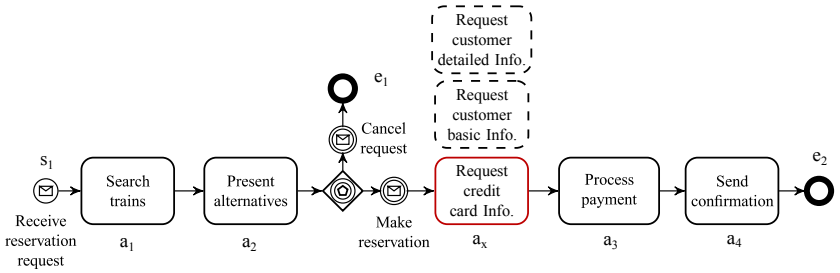


Figure 4: The complete train reservation process

We provide also a query language which allows the process designer to express his requests with flexible constraints (see details in section 5). By querying services based on composition context, our approach not only assists the process designer during the design time (completing a model or developing new process variants) but also helps to guarantee the completion of business process instances during the execution time by finding alternatives of a service in case of failure.

3. Preliminaries

In this section, we present some notations and definitions that are used to formally define a business process (section 3.1) and the composition context of a service (section 3.2). We also explain how we handle loop cases (section 3.3). We use the 'train-reservation' process (Figure 3) and the 'flight-reservation' process (Figure 2) to illustrate our approach.

3.1. Business Process Graph

As the structure of a business process can be mapped to a graph, we choose graph theory to present a business process. Indeed, there are a number of graph-based business process modeling languages, e.g., BPMN, EPC, YAWL, and UML activity diagram. Despite their variances in expressiveness and modeling notations, they all share the common concepts of tasks, events, gateways, artifacts and resources, as well as relations between them, such as transition flows [39]. In our approach, we use BPMN in our approach to present business 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), or an *intermediate event* (e.g., error message, message-catching) (Figure 5). For example, in Figure 3, s_1, a_1, a_2, e_1 are services; and 'flow-transition', 'event-based-gateway' and 'message-catching' are connection elements. Although services and connection elements in our approach are defined using BPMN notations, they are easily mapped to equivalent notations in other business process modeling and workflow languages, e.g., tasks and workflow control patterns [45].

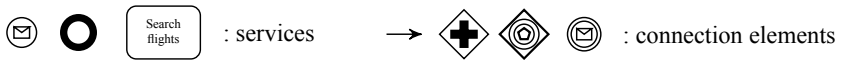


Figure 5: Services and connection elements

Relations between services in a business process are presented by the execution orders between them. In our previous work [44], we considered only the *causal* relation between services, i.e., a service is situated next to another service (such as connection between a_5 and a_2 or a_2 and a_7 in Figure 2). In this work, we improve our definitions to take into account both *causal* and

parallel relations, i.e., the relations between services that belong to parallel flows, e.g. relation between a_7 and a_6 in Figure 2.

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

Definition 3.1 (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 follows e_i in P.*

Definition 3.2 (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 3.3 (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: $a_i \leftrightarrow_P c_1$, $c_1 \leftrightarrow_P c_2, \ldots, c_{n-1} \leftrightarrow_P c_n, c_n \leftrightarrow_P a_j$. ${}^{a_j}_{a_i}f_P \in C_P^*$, C_P^* is set of sequences of connection elements in P*[1]*.*

Definition 3.4 (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 3.5 (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, the label of the connection flow from 'Search flights' to 'Present alternatives' in Figure 2 is: a_5'sequence'.'sequence'a_2; from 'Present alternatives' to "Request customer detailed Info." is: a_2'event-based-gateway'.'event-based-gateway''message-catching'.'message-catching''parallel-split'.'parallel-split'a_7.

We notice that:

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

- *An edge connecting two services $a_i, a_j \in A_P$ can be labeled by either* $l(^{a_j}_{a_i} f_P)$ *or* $l(^{a_i}_{a_j} f_P)$. For example, the edge connecting a_5 to a_2 in Figure 2 can be labeled by $l(^{a_2}_{a_5} f_P)$=a_5'sequence'.'sequence'a_2 or $l(^{a_5}_{a_2} f_P)$= 'sequence'a_2.a_5'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_6 in Figure 2 and we number them as follows: $l(^{a_6}_{a_7} f^1_P)$='parallel-split'a_7.'parallel-split'a_6 and $l(^{a_6}_{a_7} f^2_P)$=a_7'synchronization'.a_6'synchronization'.

We consider each service as a node, each connection flow as an edge. We define business process as a multigraph, in which, set of edges is a multiset (Definition 3.6).

Definition 3.6 (Business Process graph)**.** *A business process graph of P is an undirected labeled multigraph $G_P = (V_P, E_P, L_P, l)$ in which V_P is a set of nodes, E_P is a multiset of edges, L_P is a set of edge labels, and l is a mapping function that maps edges to labels, where:*

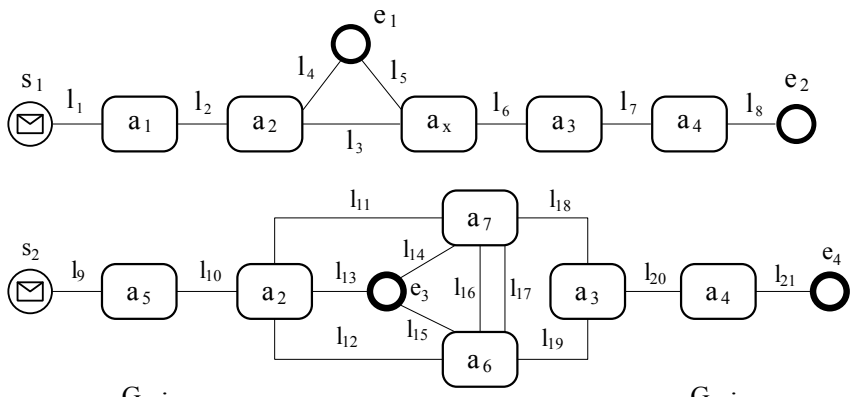
- $V_P = A_P$,

- $E_P \subseteq \langle A_P \times A_P, g \rangle, \quad g : A_P \times A_P \longrightarrow N$
  $g((a_i, a_j))$ *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:*

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

For example, the business process graphs of the 'train-reservation' process (Figure 3) and the 'flight-reservation' process (Figure 2) are presented in Figure 6.



G_{P_1}:

Nodes: s_1, e_1, e_2, a_1, a_2, a_3, a_4, a_x

Edges: <(s_1,a_1),1>, <(a_1,a_2),1>, <(a_2,a_x),1>, <(a_2,e_1),1>, <(e_1,a_x),1>, <(a_x,a_3),1>, <(a_3,a_4),1>, <(a_4,e_2),1>

G_{P_2}:

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

Edges: <(s_2,a_5),1>, <(a_5,a_2),1>, <(a_2,a_7),1>, <(a_2,a_6),1>, <(a_2,e_3),1>, <(e_3,a_7),1>, <(e_3,a_6),1>, <(a_7,a_6),2>, <(a_7,a_3),1>, <(a_6,a_3),1>, <(a_3,a_4),1>, <(a_4,e_4),1>

Edge labels:

l_1="s_1S.Sa$_1$"
l_2="a_1S.Sa$_2$"
l_3="a_2E.EM.Ma$_x$"
l_4="a_2E.EM.Me$_1$"
l_5="Me$_1$.EM.EM.Ma$_x$"
l_6="a_xS.Sa$_3$"
l_7="a_3S.Sa$_4$"
l_8="a_4S.Se$_4$"
l_9="s_2S.Sa$_5$"
l_{10}="a_5S.Sa$_2$"
l_{11}="a_2E.EM.MP.Pa$_7$"
l_{12}="a_2E.EM.MP.Pa$_6$"
l_{13}="a_2E.EM.Me$_3$"
l_{14}="Me$_3$.EM.EM.MP.Pa$_7$"
l_{15}="Me$_3$.EM.EM.MP.Pa$_6$"

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

Figure 6: Business process graphs of the 'train-reservation' process (Figure 3) and the 'flight-reservation' process (Figure 2)

3.2. Service Composition Context

We define the composition context of a service as a business 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 composition context of a service can be considered as a business fragment that presents the behavior of the associated service.

We present in the following some definitions that are used to formally define the composition context.

Definition 3.7 (connection path)**.** *A connection path from a_i to a_j in a business 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 3.7, a connection path in a business process graph is *undirected*. It means that the edges in a connection path can be oriented in different directions. For example, in Figure 2, a connection path from 'Search flights' (a_5) to 'Request customer detailed Info.' (a_7) can be either 'a_5, a_2, a_7' or 'a_5, a_2, a_6, a_7' or 'a_5, a_2, a_6, a_3, a_7', etc.

Definition 3.8 (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 3.9 (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 2, the shortest path from a_5 to a_7 is 'a_5, a_2, a_7' and its length is 2.

Definition 3.10 (k^{th}-layer neighbor)**.** a_j *is a* k^{th}*-layer neighbor of* a_i *in a business 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* $N_P^k(a_i)$*.* $N_P^0(a_i) = \{a_i\}$.

For example in Figure 2, s_2 and a_2 are the 1^{st}-layer neighbors of a_5; a_5, e_3, a_7 and a_6 are the 1^{st}-layer neighbors of a_2; a_6 is one of the 2^{nd}-layer neighbors of a_5 and so on.

As the distance from a service a_i to its k^{th}-layer neighbors is k, we can imagine that the k^{th}-layer neighbors of a service a_i are located on a circle whose center is a_i and k is the radius. The circle is latent since it exists but it is not explicitly represented in the business process graph. We call this latent circle *connection layer* and the area limited by two adjacent latent circles *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 3.11).

Definition 3.11 (k^{th}-zone flow)**.** ${}^{a_v}_{a_u}f_P$ *is a* k^{th}*-zone flow of* a_i *iff* $\exists^{a_v}_{a_u}f_P : (a_u, a_v \in N_P^{k-1}(a_i)) \vee (a_u \in N_P^{k-1}(a_i) \wedge a_v \in N_P^k(a_i)) \vee (a_v \in N_P^{k-1}(a_i) \wedge a_u \in N_P^k(a_i))$*. The set of all* k^{th}*-zone flows of a service* $a_i \in P$ *is denoted by* $Z_P^k(a_i)$*.* $Z_P^0(a_i) = \emptyset$ *and* $|Z_P^k(a_i)|$ *is the number of connection flows in the* k^{th} *connection zone of* a_i.

For example in Figure 2, the connection from a_2 to a_7 is the 2^{nd}-zone flow of a_5 while the connection from a_7 to a_3 is its 3^{rd}-zone flow. $|Z^2_{P_2}(a_5)| = 3$ as in the 2^{nd}-zone of a_5, there are three connection flows, which are from a_2 to a_6, a_7 and e_3.

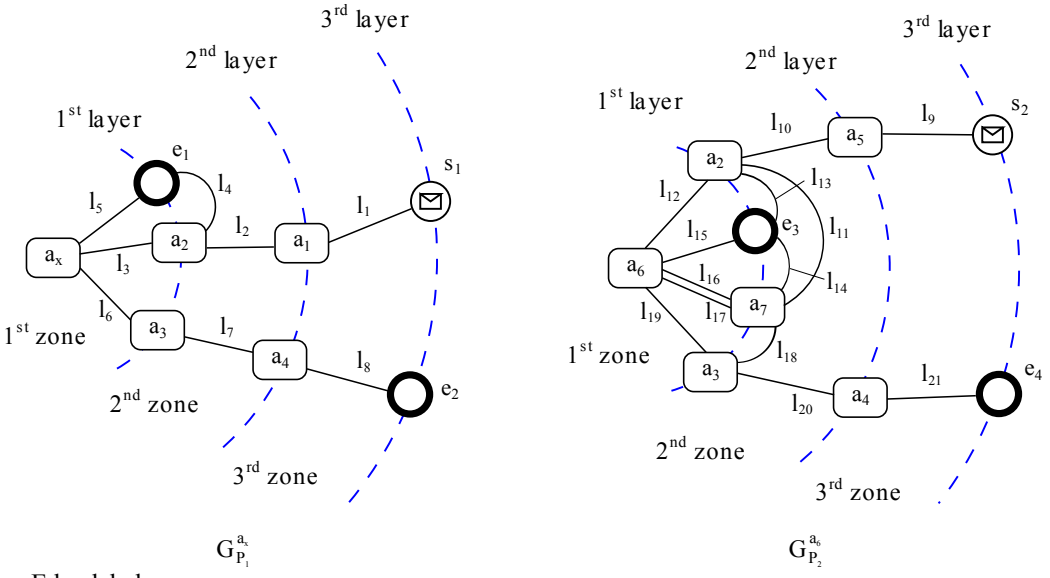
Intuitively, the connection paths between two services present their relation in term of closeness. The longer the connection path is, the weaker their relation is and the shortest connection path between two services presents their best relation. To illustrate the best relations of a service to others services in a business process, we define the *service composition context graph* (formally defined in Definition 3.12) which presents all the shortest paths from a service to others. Each service in a business 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.

Concretely, 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 (see our proofs [46], section A). In the case that $d = 0$ ($l_1 = l_2$), i.e., a_i and a_j are both l_1^{th}-layer neighbors of a_x, we assign to e_{ij} $l_1 + 1$ as zone value. In the case that $d = 1$, i.e., a_i and a_j belong to two adjacent layers, e_{ij} is the k^{th}-zone flow connecting a_i and a_j and we assign to e_{ij} the zone value k, i.e., $min(l_1, l_2) + 1$. Consequently, we assign to the connection flow connecting a_i and a_j in the 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 linked business process graph, 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.



$G^{a_x}_{P_1}$ $G^{a_6}_{P_2}$

Edge labels:

l_1="s_1S.Sa_1"
l_2="a_1S.Sa_2"
l_3="a_2E.EM.Ma_x"
l_4="a_2E.EM.Me_1"
l_5="Me_1.EM.EM.Ma_x"
l_6="a_xS.Sa_3"
l_7="a_3S.Sa_4"
l_8="a_4S.Se_4"
l_9="s_2S.Sa_5"
l_{10}="a_5S.Sa_2"
l_{11}="a_2E.EM.MP.Pa_7"
l_{12}="a_2E.EM.MP.Pa_6"
l_{13}="a_2E.EM.Me_3"
l_{14}="Me_3.EM.EM.MP.Pa_7"
l_{15}="Me_3.EM.EM.MP.Pa_6"
l_{16}="Pa_7.Pa_6"
l_{17}="a_7Y.a_6Y"
l_{18}="a_7Y.Ya_3"
l_{19}="a_6Y.Ya_3"
l_{20}="a_3S.Sa_4"
l_{21}="a_{54}S.Se_4"

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

Figure 7: Composition context graphs of a_x (in Figure 3) and a_6 (in Figure 2)

Definition 3.12 (Composition context graph)**.** *The composition context graph of a service $a_x \in P$, denoted by $G^{a_x}_P = (V^{a_x}_P, E^{a_x}_P, L_P, l)$, is an undirected labeled multigraph created from $G_P = (V_P, E_P, L_P, l)$. $V^{a_x}_P$ is a set of vertexes associated to their layer numbers and $E^{a_x}_P$ is a set of edges associated to their zone numbers. $V^{a_x}_P$ and $E^{a_x}_P$ are defined as following:*

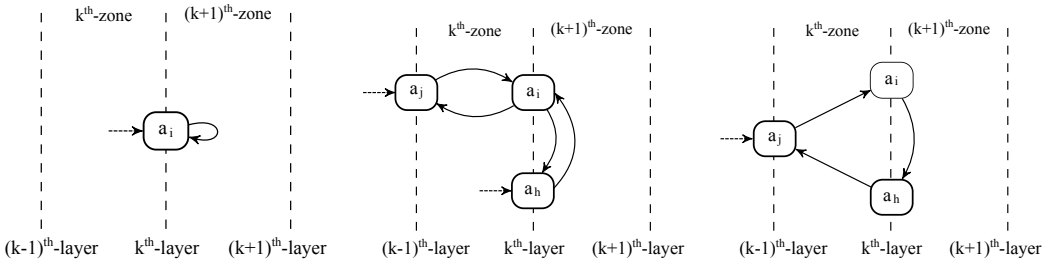
- $V^{a_x}_P = \{(a_i, \mathcal{L}(^{a_x}_{a_i}\mathcal{S}_P)) : \ a_i \in V_P\}$

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

For example, the composition context graphs of the 'unknown' service (a_x in Figure 3) and the 'Request credit card Info.' (a_6 in Figure 2) are presented in Figure 7. In these graphs, all causal and parallel flow relations are presented.

3.3. Loop Cases

There are three typical loop cases in a business process: *self-loop*, *loop via another service* and *loop via other services*. By applying Definition 3.12, the possible layers to which the loop services can belong are depicted in Figure 8.



(a) Self loop (b) Loop via another service (c) Loop via other services

Figure 8: Connection flows in loop cases

In the self-loop case (Figure 8a), if a service a_i is located on the k^{th}-layer, then its self-loop edge belong to the zone $(k+1)^{th}$ because according to Definition 3.12, the zone number of this edge is ${}^{a_i}_{a_i} z_P^{a_x} = Min(\mathcal{L}({}^{a_i}_{a_x}\mathcal{S}_P), \mathcal{L}({}^{a_i}_{a_x}\mathcal{S}_P)) + 1 = Min(k, k) + 1 = k + 1$.

In the loop-via-another-service case (Figure 8b), there are two possibilities: (i) the two services are located on adjacent layers (e.g., a_j and a_i) and (ii) the two services are located on the same layer (e.g., a_i and a_h). In the first possibility, assume that a_i and a_j are respectively located on the $(k-1)^{th}$-layer and k^{th}-layer, the edges connecting them are assigned the zone number ${}^{a_j}_{a_i} z_P^{a_x} = {}^{a_i}_{a_j} z_P^{a_x} = Min(\mathcal{L}({}^{a_i}_{a_x}\mathcal{S}_P), \mathcal{L}({}^{a_j}_{a_x}\mathcal{S}_P)) + 1 = Min(k-1, k) + 1 = k$, i.e., they are on the zone limited by these adjacent layers. In the second possibility, assume that a_i and a_h are located on the same k^{th}-layer, the edges connecting them are assigned the zone number ${}^{a_h}_{a_i} z_P^{a_x} = {}^{a_i}_{a_h} z_P^{a_x} = Min(\mathcal{L}({}^{a_i}_{a_x}\mathcal{S}_P), \mathcal{L}({}^{a_h}_{a_x}\mathcal{S}_P)) + 1 = Min(k, k) + 1 = k + 1$, i.e., they are presented on the outer zone of the layer where the services are located.

In the loop-via-other-services case (for example, Figure 8c shows the loop created by 3 services), the edges between services are assigned their zone numbers following Definition 3.12. For the two services located on adjacent layers (e.g., a_j and a_i, or a_j and a_h in Figure 8c), the edge connecting them belongs to the zone limited by these layers. For the two services located on the same layer (e.g., a_i and a_h in Figure 8c), the edge connecting them belongs to the outer zone of their layer.

So, in any business 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 rest services are located. Consequently, we can always assign a layer number to a service and thus, a zone number to a connection flow, in a composition context graph.

4. Composition Context Similarity

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. We firstly present how we match two connection flows (section 4.1). Then, we elaborate the matching between two composition contexts (section 4.2). Finally, we show how we integrate the similarity between connection elements into our composition context similarity computation (section 4.4).

To illustrate the computation process, we will consider the composition context of the 'unknown' service in the 'train-reservation' process (a_x in Figure 3) and the 'Request credit card Info.' service in the 'flight-reservation' process (a_6 in Figure 2). The composition context graphs of these services are shown in Figure 7.

4.1. Connection Flow Matching

To compute the matching between two connection flows, we propose to use the Levenshtein distance (LD for short) [47]. In information theory, the LD is a metric for measuring the difference between two sequences of characters. The LD is defined as the minimum number of edits needed to transform one sequence of characters into the other, with the allowable edit operations being *insertion*, *deletion*, or *substitution* of a single element. For

example, the LD between "gumbo" and "gambol" is 2, between "kitten" and "sitting" is 3, etc. Inspired by this, we consider each connection element in a connection flow as a character and we 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.

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

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

where:

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

For example in Figure 7, we have $M_p(\binom{a_4}{a_3} f_{P_1}, \binom{a_4}{a_3} f_{P_2}) = M($"$a_3$S.S$a_4$", "$a_3$S.S$a_4$") $= 1 - \frac{0}{4} = 1$; $M_p(\binom{a_2}{a_1} f_{P_1}, \binom{a_2}{a_5} f_{P_2}) = 0$ and so on.

We prove that LD of two strings is equal to LD of their inverse strings (see our proofs [46], section B). So, whatever the edges (a_u, a_v) and (a_m, a_n) are labeled by $l(\binom{a_v}{a_u} f_{P_1})$ or $l(\binom{a_u}{a_v} f_{P_1})$ and $l(\binom{a_n}{a_m} f_{P_2})$ or $l(\binom{a_m}{a_n} f_{P_2})$, Equation 1 gives the same 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.

4.2. Composition Context 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 *other zones*. In the first zone, we match the connection flows that connect the two associated services and same services in the first layer. In other zones, we match the connection flows that connect the same services. We sum all matching values then divide them by the number of connection flows in the considered zones of the first service.

We apply Equation 1 to compute the composition context matching in either the first and other zones. However, Equation 1 considers only connection flows that connecting the same services in two business 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 ${}^{a_c}_{a_0}f_{P_2}$.

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 2.

$$\mathbb{MC}^k(G^{a_i}_{P_1}, G^{a_j}_{P_2}) = \frac{\sum_{t=1}^{k} \sum_{{}^{a_v}_{a_u}f_{P_1} \in Z^t_{P_1}, {}^{a_n}_{a_m}f_{P_2} \in Z^t_{P_2}} \mathsf{MF}^t({}^{a_v}_{a_u}f_{P_1}, {}^{a_n}_{a_m}f_{P_2})}{\sum_{t=1}^{k} |Z^t_{P_1}(a_i)|} \tag{2}$$

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 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_x and a_6 (Figure 7) computed by Equation 2 is:

$$\begin{aligned} \mathbb{MC}^3(G^{a_x}_{P_1}, G^{a_6}_{P_2}) &= \frac{\begin{array}{c} M({}^{a_x}_{a_2}f_{P_1}, {}^{a_6}_{a_2}f_{P_2}) + M({}^{a_x}_{e_1}f_{P_1}, {}^{a_6}_{e_3}f_{P_2}) + M({}^{a_3}_{a_x}f_{P_1}, {}^{a_3}_{a_6}f_{P_2}) \\ + M({}^{e_1}_{a_2}f_{P_1}, {}^{e_3}_{a_2}f_{P_2}) + M({}^{a_4}_{a_3}f_{P_1}, {}^{a_4}_{a_3}f_{P_2}) + M({}^{e_2}_{a_4}f_{P_1}, {}^{e_4}_{a_4}f_{P_2}) \end{array}}{3+3+2} \\ &= \frac{\frac{3}{4} + \frac{4}{5} + \frac{1}{2} + 1 + 1 + 1}{8} = 0.63 \end{aligned}$$

4.3. Zone Weight

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 $w_t = \frac{k+1-t}{k}$, where t is the zone number ($1 \leq t \leq k$) and k is the number of considered zones around the service. All connection flows connecting the associated service have the greatest weight ($w_1 = 1$) and the connection flows connecting services in the furthest zone have the smallest weight ($w_k = \frac{1}{k}$).

Then, the composition context matching between $G^{a_i}_{P_1}$ and $G^{a_j}_{P_2}$ within k zones and with zone weight, denoted by $\mathbb{MW}^k(G^{a_i}_{P_1}, G^{a_j}_{P_2})$, is given by Equation 3.

$$\mathbb{MW}^k(G^{a_i}_{P_1}, G^{a_j}_{P_2}) = \frac{2}{k+1} \times \sum_{t=1}^{k} \frac{k+1-t}{k} \times \frac{\sum\limits_{\substack{{}^{a_v}_{a_u} f_{P_1} \in Z^t_{P_1} \\ {}^{a_n}_{a_m} f_{P_2} \in Z^t_{P_2}}} \mathsf{MF}^t({}^{a_v}_{a_u} f_{P_1}, {}^{a_n}_{a_m} f_{P_2})}{|Z^t_{P_1}(a_i)|} \quad (3)$$

where:

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

For example, the composition context matching between a_x and a_6 (Figure 7) computed by Equation 3 is:

$$\begin{aligned} \mathbb{MW}^3(G^{a_x}_{P_1}, G^{a_6}_{P_2}) &= \frac{2}{3+1} \times \Big(\frac{3}{3} \times \frac{M({}^{a_x}_{a_2} f_{P_1}, {}^{a_6}_{a_2} f_{P_2}) + M({}^{a_x}_{e_1} f_{P_1}, {}^{a_6}_{e_3} f_{P_2}) + M({}^{a_3}_{a_x} f_{P_1}, {}^{a_3}_{a_6} f_{P_2})}{3} + \\ &\quad \frac{2}{3} \times \frac{M({}^{e_1}_{a_2} f_{P_1}, {}^{e_3}_{a_2} f_{P_2}) + M({}^{a_4}_{a_3} f_{P_1}, {}^{a_4}_{a_3} f_{P_2})}{3} + \frac{1}{3} \times \frac{M({}^{e_2}_{a_4} f_{P_1}, {}^{e_4}_{a_4} f_{P_2})}{2}\Big) \\ &= \frac{2}{4} \times \Big(\frac{3}{3} \times \frac{\frac{3}{4} + \frac{4}{5} + \frac{1}{2}}{3} + \frac{2}{3} \times \frac{1+1}{3} + \frac{1}{3} \times \frac{1}{2}\Big) = 0.65 \end{aligned}$$

4.4. Similarity Between Connection Elements

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

In the previous sections (4.2 and 4.3), we consider that the matching between two connection elements has only two values: 0 (in the case that they do not have the same type) and 1 (in the case that they have the same type). In this section, we propose a metric to compute the similarity between connection elements in terms of execution properties. This metric allows evaluating their similarity by a real value between 0 and 1.

We firstly identify matching rules that the similarity has to satisfy (section 4.4.1). Then, we present our proposition to compute the similarity between typical connection elements (section 4.4.2). Finally, we describe how to integrate the computed similarity into the composition context matching (section 4.4.3).

4.4.1. Matching Rules

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

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

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

① If two connection elements are *identical*, their similarity is 1.
For example, similarity of two 'sequence' is 1; similarity of two 'AND-split' elements is 1 and so on.

② Similarity between two connection elements that *have the same types but different number of output flows* is 1. Similarity between two *different* connection elements is less than 1.
For example, similarity between two 'AND-split' elements that have different number of output flows is 1 whereas similarity between an 'AND-split' and an 'OR-split' is less than 1.

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

4.4.2. Similarity Computation

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

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

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

Let x, y and z be the number of output flows that an 'AND-split', an 'OR-split' and an 'XOR-split' respectively have. The numbers of possible cases of these connection elements are given in the 4^{th} column of Table 1. The last column of Table 1 presents the *probabilities that an output flow is executed*. We explain in the following how we compute these values.

For the 'sequence' and 'AND-split' cases, there is only one possible execution case, in which all services are executed. Hence, the probability that

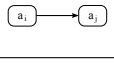
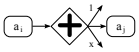
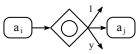
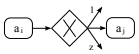
Element	Presentation	No. paths	No. possible cases	Probability that an output flow is executed
Sequence		1	1	1
AND-split		x	1	1
OR-split		y	$2^y - 1$	$\frac{2^{y-1}}{2^y - 1}$
XOR-split		z	z	$\frac{1}{z}$

Table 1: Probability that a_j appears in the possible cases

an output flow is executed is 1.

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

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

To compute the similarity between these connection elements, we propose to combine the *weight* of an output flow and the probability that it is executed. The weight of an output flow is specified by the inverse number of output flows. For example, consider an 'AND-split' that derives 3 output flows. The weight of each output flow is $\frac{1}{3}$.

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

$$S(c_u, c_v) = w^*_{c_u} \times w^*_{c_v} \times P^+_{c_u} \times P^+_{c_v} \tag{4}$$

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

The similarities between aforementioned connection elements are given in Table 2. In the section C of our proofs [46], we prove that the similarities computed by our approach satisfy the aforementioned matching rules.

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

Table 2: Similarities between typical connection elements

For example, with $x = 2$, $y = 3$, $z = 4$, S(AND-split,sequence)=0.5, S(OR-split,sequence)=0.19, S(XOR-split,sequence)=0.06, S(AND-split,OR-split)=0.10, S(AND-split,XOR-split)=0.03 and S(OR-split,XOR-split)=0.01.

4.4.3. Integration into the Composition Context Matching

The composition context matching is computed from the matching of connection flows (section 4). A connection flow is a sequence of connection elements that connect two services (Definition 3.3). It can contain one or more connection elements. We propose to integrate the similarity between connection elements in the case where connection flows contain only one connection element. For example, consider two connection flows ${}^{a_y}_{a_x}f_{P_1}$ and ${}^{a_v}_{a_u}f_{P_2}$ of two composition context graph $G^{a_i}_{P_1}$ and $G^{a_j}_{P_2}$. Assume that they are located on the same connection zone and connect the same ending services. Each of them contains only one connection element, which is c and c' respectively. Similarity between them is inferred by the similarity between the connection elements, which is computed by Equation 4.

In the case that connection flows contain more than one connection element, we transform them to strings of characters and reuse Levenshtein distance to compute their similarity (section 4.1).

As the similarity between connection elements is applied to compute the similarity of connection flows, it impacts on the final result of the composition context matching. In our experiments, we analyze matching cases with and without the similarity between connection elements to assess its impact.

5. Service Querying

Each service in a business process has a composition context which presents the relations between the service and its neighbors. By matching composition contexts, we can find services that have similar relations to common neighbors. In this section, we present a query language used to retrieve relevant services to a selected position in a business process (section 5.1) and its execution (section 5.2).

5.1. Query Grammar

The query in our approach not only helps to search for relevant services based on a given composition context but also allows adding constraints to filter the searching results. It consists of three parameters, which are: an *associated service*, *connection constraints*, and a *radius*. The associated service is the service whose composition context is considered to be matched with other contexts. Connection constrains are services or connection flows to be included/excluded to filter the query's results. The radius is the number of connection layers taken into account for the composition context matching. It specifies the largeness of the considered composition contexts.

We present in the following our proposed query grammar using the Extended Backus-Naur Form (EBNF)[2] [48]. We use ';' to separate the input parameters; '(' and ')' to separate query's constrains; '<' and '>' to group services or connection flows; '[' and ']' to separate a connection flow and its ending services. We use '+' and '-' signs to include/exclude constraints; and '|' sign for multiple choice operator. Details of the grammar are presented in Table 3.

1. Query ::= ServiceID,':',[Constraint],':',Radius;
2. ServiceID ::= Character,{Character|Digit};
3. Constraint ::= ('+'|'-')Term | Constraint,'|',Term;
4. Term ::= Item | Term,'+',Item | Term,'-',Item;
5. Item ::= ServiceID| ConFlow | '(',Constraint,')';
6. ConFlow ::= '<',[ServiceID],'[',FlowString,']',[ServiceID]'>';

[2]the EBNF standard is adopted by ISO, no. ISO/IEC 14977

7.	FlowString	::=	ConElement,{ConElement};
8.	ConElement	::=	‘sequence’|‘AND-split’|‘AND-join’|‘OR-split’|‘OR-join’|‘XOR-split’|‘XOR-join’;
9.	Radius	::=	DigitNotZero,{Digit};
10.	Character	::=	‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘g’ | ‘h’ | ‘i’ | ‘j’ | ‘k’ | ‘l’ | ‘m’ | ‘n’ | ‘o’ | ‘p’ | ‘q’ | ‘r’ | ‘s’ | ‘t’ | ‘u’ | ‘v’ | ‘w’ | ‘x’ | ‘y’ | ‘z’;
11.	DigitNotZero	::=	‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’;
12.	Digit	::=	‘0’ | DigitNotZero;

Table 3: Query grammar

The query grammar is explained as follows:

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

services.

- The string of connection elements is defined in line 7. It includes at least a connection element which is defined in line 8.
- In line 9, we define the radius as a natural number greater than 0.
- Finally, lines 10, 11, 12 define literal characters and digits.

Examples of our query are given in Table 4. These queries are used to find services that have composition contexts similar to the composition context of s_x. These composition contexts are limited to 3 layers. In Table 4, we also explain the concerned constraints used to filter the query result.

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

Table 4: Query examples

In our motivating example (Figure 3), as the designer wants to find services whose composition contexts are similar to a_x, he selects a_x as the associated services. Then, he can specify two constraints as follows: (1)

the queried service should be executed right after 'Present alternatives' (a_2) and before 'Process payment' (a_3) and (2) it is connected from a_2 by a connection flow 'event-based-gateway''message-catching' and to a_3 by a 'sequence'. Finally, he specifies the radius, which is 1 in this case, because he wants to find the contexts in which a_2 and a_3 are connected directly to a_x. Consequently, his query is: a_x:+<a_2['event-based-gateway''message-catching']>+<[sequence]a_3>:1.

More examples about our query can be given as follows. If the process designer wants to find services that have similar context to 'Search trains' (a_1) within the first zone, he can make a query as following: a_1::1. If he wants to find services that are similar to 'Search trains' but not followed by 'Present alternatives', he will exclude the connection flow connecting 'Search trains' to 'Present alternatives' in its first layer. The query will be: a_1:-(<a_1[sequence]a_2>):1. In the case that he wants to know possible payment methods, he may select 'Process payment' service (a_3) and add a constraint which does not include a 'sequence' that connects to 'Send confirmation' service (a_4). He can also widen the considered composition context in two layers to get more results. So, his query is: a_3:-(<['sequence']a_4>):2. Or, if he wants to find services that are similar to 'Search trains' (a_1), and include the services 'Request payment Info.' (a_8) and 'Process payment' (a_3) but exclude the AND-join connection between them, his query is as follows: a_1:+a_8+a_3-(<a_8['AND-join']a_3>):5, etc.

5.2. Query Execution

In general, queries are processed as follows:

1. We capture the composition context of the associated service. The largeness of the composition context is specified by the radius parameter.
2. We match the composition context of the associated service to composition contexts of other services in other processes.
3. We refine the matching result by selecting only services whose composition contexts satisfy the query's constraints.
4. We sort the selected services based on the matching values and pick up top-N services[3] for the response.

[3]N can be flexibly tuned by the process designer

6. Implementation & Experiments

In this section, we present our implementation (section 6.1) and experiments (section 6.2) to validate our approach.

6.1. Implementation

To demonstrate our approach, we implemented an application that allows the process designer to create business processes and get recommendations during the design phase. Our application was developed based on Signavio[4], which is a platform for business process design. This platform provides a web-based graphical interface to design business processes. It uses BPMN notations. It has two versions: commercial and open source. The open source version with limited features is published[5] for free downloading and testing.

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

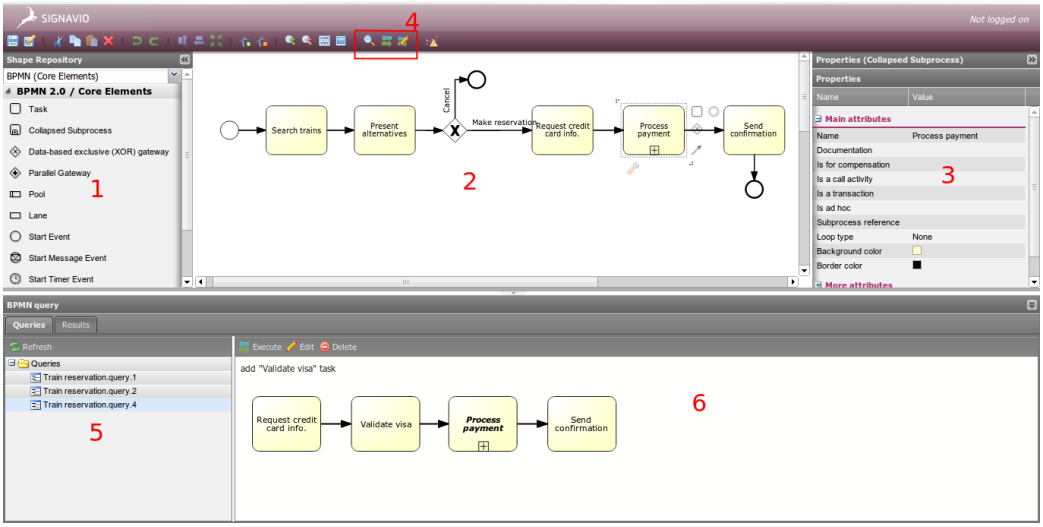


Figure 9: Application screen-shot

[4] `http://www.signavio.com/`

[5] `http://code.google.com/p/signavio-core-components/`

In this application, the process designer can design and store business processes. During the design, he can select a service and process a query. He needs to specify the number of layers/zones needed to be taken into account and choose an algorithm to be executed. The composition context matching is executed beforehand. Then, query constraints are applied on the matching result to filter unrelated services. Queries and their results can be saved and reloaded for a future use.

A screen-shot of the application is shown in Figure 9. The areas 1, 2 and 3 show the BPMN elements for design, canvas and property of the selected element. They are provided by the Signavio platform. We developed the areas 4, 5, and 6. Area 4 contains the buttons for launching the context matching and query, area 5 shows the list of previous queries and area 6 displays either the query design or its results.

A basic scenario[6] is as follows:

1. A process designer opens the application to design a new process. He can also load an existing process for editing.
2. During the design, he can select a service, create a query, and specify the number of layers (or zones) needed to be taken into account for the composition context matching.
3. He can select to run one of 4 prepared algorithms[7].
4. The application executes the query and returns results. When the designer selects a service from the result, it shows the corresponding process in which the selected service is highlighted.
5. The designer can copy services from the returned process and paste them on the active canvas to continue his design.
6. He can save the executed query for future usage. He can also load, modify and executed a saved query.

6.2. Experiments

We performed experiments on a large public collection of business processes. Our goal is three fold: (i) to evaluate the feasibility of our approach;

[6]Tutorial at: http://www-inf.it-sudparis.eu/SIMBAD/tools/WebRec/tutorial.html and video demo at: http://www-inf.it-sudparis.eu/SIMBAD/tools/WebRec/demo.html

[7]at the current stage, we have developed 4 algorithms for the composition context matching, which are with/without zone weight and with/without connection element similarity

(ii) to measure its accuracy and to (iii) evaluate the performance of our algorithm. Details of the dataset and experiments are given as following.

6.2.1. Dataset & Experimental Cases

The dataset used in our approach is shared by the Business Integration Technologies (BIT) research group[8] of the IBM Zurich [49]. It contains business process models designed for financial services, telecommunications, and other domains. It is presented in XML format following BPMN 2.0 standard. Each XML file stores the data of a business process, including elements' IDs, service names, and the sequence flows between elements. The dataset consists of 560 business processes with 6363 services. There are 3781 different services in which 1196 services appear in more than one process. In average, there are 11.36 services, 18.96 gateways (including parallel, exclusive and inclusive gateways) and 46.81 sequence flows in a process.

We performed experiments to measure the *feasibility*, the *accuracy* and the *performance* of our approach. We evaluate the feasibility based on 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 returned services. We evaluate the accuracy based on the Precision and Recall and we evaluate the performance based on the computation time.

Our algorithm was experimented with 5 different cases which are presented in Table 5. In Case 1, we considered only the zone-weight. The parallel flow relations and connection element similarity were not taken into account in the composition context matching. In Case 2, we considered only the parallel relations, regardless the zone-weight and connection element similarity. Case 3 took into account both parallel flow relations and zone-weight, whereas Case 4 considered parallel relations and connection element similarity. In Case 5, the algorithm was performed with all parameters above.

In our previous work [44], we presented some preliminary results of the first case, which considers only zone weight and consecutive relations between services. In this work, apart from the additional results of the first case, we present and discuss about the experimental results of the other cases which take into account parallel relations between services and similarity between connection elements.

[8] http://www.zurich.ibm.com/csc/bit/

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

Table 5: Examined cases

6.2.2. Results

- *Approach feasibility and parameter impact* :

In the first experiment, we aim at evaluating the feasibility of the approach and the impact of parameters. The feasibility is evaluated based on the number of services that can have recommendations. Figure 10 shows the percentages of services whose matching values with other services are greater than or equal to 0.5. It shows that cases 2, 3, 4, 5, which take into account the parallel flow relations, retrieve greater number of services than case 1. The zone-weight parameter has no impact in the first zone. Hence, with $k = 1$, case 2 and case 3 has the same number of services (similar to case 4 and 5). However, with $k = 2$, the number of services and similarity values decreased. Case 3 and case 5, which take into account zone weight in their computation, retrieve more services than case 2 and case 4 respectively. It means that *zone weight impacts on the number of retrieved services*. When we take into account zone weight, we retrieve more services. Similarly, *the similarity between connection elements impacts on the number of retrieved services*. When we take into account similarity between connection elements (case 4 and case 5 compared to case 2 and case 3 respectively), we retrieve more services. Case 5, which takes into account both zone-weight and connection element similarity, retrieves the highest number of services.

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

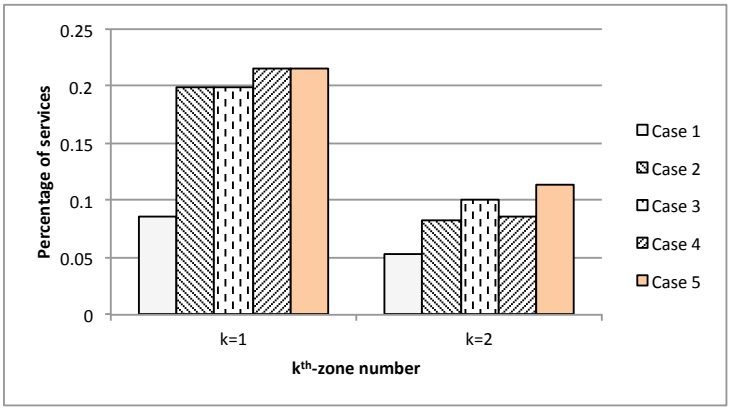


Figure 10: Percentage of services whose matching value >= 0.5

To examine the impact of k^{th}-zone values, we run our algorithms with k from 5 to 1. The experimental results (Figure 11) show that when k decreases, the number of recommended services increases. It is because when k decreases that the number of unmatched services in further layers decreases, the matching values between composition contexts increase, the number of services increases.

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

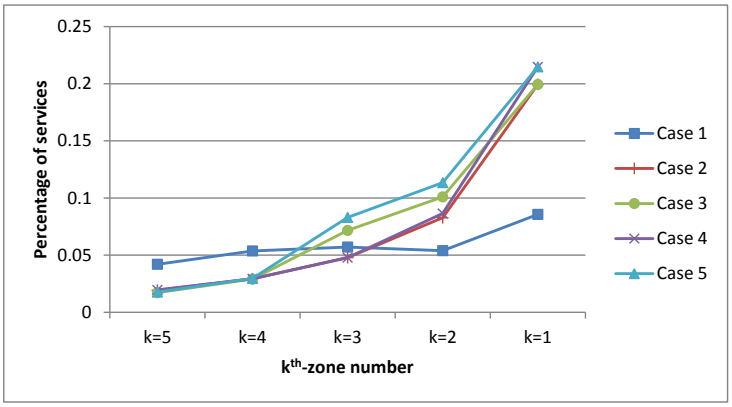


Figure 11: Percentage of services with different k^{th}-zone values

Figure 11 also shows the impact of *parallel flow relations*. When we take into account the parallel flow relations between services, many services located on the further layers of a composition context graph are relocated on the nearer layers. Therefore, the matching values in the first zone are high and these values decrease quickly when we consider further zones. Figure 11 shows that in Cases 2, 3, 4 and 5, which take into account parallel flow relations, the number of services is very low with great k and very high with small k. Meanwhile, in Case 1, the number of services slightly change when k decreases.

- *Algorithms accuracy* :

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

Concretely, consider a selected service s in a process P. Assume that s appears in n processes. The recommendations for this selected position consist of l services, in which $t(t \leq l)$ services are s. Precision and Recall of these recommendations are given by Equation 5.

$$Precision = \frac{t}{l}; \quad Recall = \frac{t}{n} \tag{5}$$

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

The average Precision and Recall values are shown in Figure 12. The Precision and Recall values of the examined cases are not so different, as the different parameters that distinguish these cases has a slight impact if we consider just the *the first zone* (as explained in the previous section). The Precision values increase when the number of recommended services decreases. This means that the relevant services mostly appear at the top of the recommendation list. In other words, when we shorten the recommendation

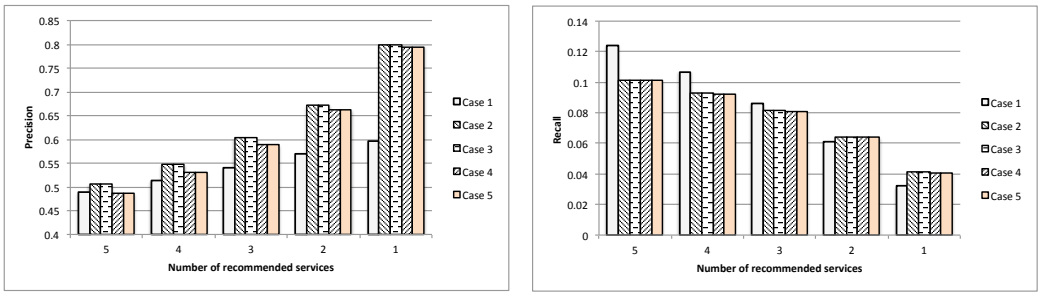


Figure 12: Precision and Recall values computed by taking into account the first zone

list, the recommendations generated by our approach are more focused and precise.

Currently, there are few approaches [50, 51] that consider the matching between services in processes. They however they use the matching results to search for relevant processes. In addition, they do not provide experiments with Precision and Recall values. So, we can not compare the accuracy of our approach to their experiment. Instead, we consider the *random case*, where a system recommends randomly services for a selected position. We compare the simplest case of our approach, which make recommendations without considering the concurrent relation and similarity between connection elements, with the *random case*.

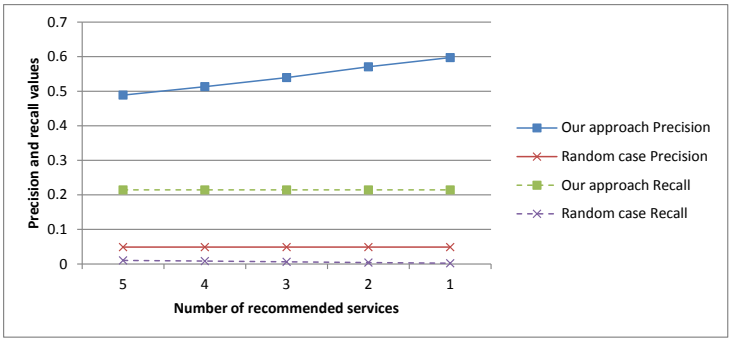


Figure 13: Comparing the simplest case of our approach to the random case

Figure 13 shows the result of our experiment. In this experiment, we make recommendations randomly for each service which appears in at least 10 business processes and compute the average Precision and Recall values. Figure 13 shows that the *worst* result of our approach is still much better

than the *best* result of the random case (with 5 recommended services, our approach achieves 10.01 times higher Precision value and 20.62 times higher Recall value).

- *Algorithms performance* :

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

We measure the time that each algorithm consumes to perform the matching between a service and all other services in the dataset. Figure 14 shows the average computation time of all algorithms in the case that the k^{th}-zone value is 3. In average, our algorithms spend less than *2 seconds* to compute the matching between a service and the other 6362 services. This means that these algorithms have *acceptable* computation time as they can make recommendations in a very short time by considering a large number of services. The result also shows that Case 1, which does not take into account parallel flow relations, is the least time-consuming. Other cases, which consider more parameters, are more time-consuming. To shorten the response time for recommendations, the matching computation in our approach can be done offline.

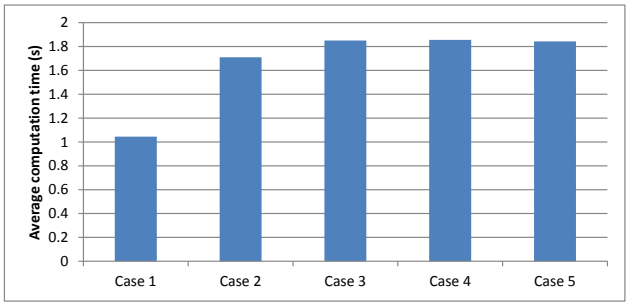


Figure 14: Average computation time with $k = 3$

On synthesis, the statistics on the number of recommended services showed that our approach is *feasible*. The Precision and Recall values showed that our approach is *accurate*. Finally, experiments on the computation time showed that our approached is of *good performance*.

7. Related Work

Approaches supporting service querying have focused on the text matching between the query string and service descriptions [9, 10, 17]. Some of them clustered services [8, 11], or examined the quality of services [14, 12, 13], whereas others relied on the service semantic descriptions [15, 16, 18]. These approaches have exploited explicit knowledge exposed by services themselves, e.g., text-based description, and/or communication environment, e.g., throughput or response time. They have not yet considered the business context into which services are integrated. In our approach, we complement this existing work by exploiting the composition context around each service.

Recent research on querying to support process design and execution pays much attention on business process models. To support the design, a query language, named BPMN-Q, has been proposed [36, 38, 39]. This language allows retrieving partial process fragments that start from a given activity (corresponding to service in our approach) and end by another given activity. To support the execution, Momotko et. al. [52] proposed a query language to retrieve the status of service instance during the process execution time. Meanwhile, Beeri et. al. [53, 54, 55] proposed a query language and Balan et al. [56] proposed a tool for business process monitoring. Different from them, our approach can be applied in both design and execution phases. We compute the similarity between composition contexts instead of the perfect matching of service names and their connections. We allow including/excluding services and connection flows. In addition, we exploit the relations between services in the design instead of the attributes of a service instance.

Deutch et. al. [57] were inspired by [53] and proposed a model for querying the structural and behavioral properties of business processes. While [58] proposed a query language for process specifications and Markovic et. al [59] proposed an ontoloty-based framework for process querying. In our approach, we focus on the service's 'behavior', i.e., relations between services, instead of the process 'behavior', and we match the service (composition context) graphs instead of the process graphs.

Other approaches that aim at facilitating the process design without building a query language include business process searching [26, 25] and process similarity measuring [27, 29, 31, 60, 28, 30, 61, 62]. Their objective is to help the process designer find similar processes to a selected process. Related to them, our work also helps to facilitate process design. However,

instead of comparing process models, we focus on comparing services in process models.

The concept of composition context has been introduced in our previous work [44, 63], in which we consider only the consecutive relation between services. In this work, we upgrade our model to take into account both consecutive and parallel relations. We also evaluate the similarity between connection elements and examine its impact on the context matching. Finally, we develop a context free grammar language and graphical user interface application for service querying. Richer experimental results of different matching cases have been also provided and analyzed.

8. Conclusion

In this paper, we present a service recommendation approach that takes into account existing process models for assisting the development of process variants. We introduce the service composition context and propose an algorithm to compute the similarity between services. We also evaluate the similarity between connection elements and measure its impact on the composition context matching. A query language is also proposed as a useful tool for service querying.

Our approach is autonomic and independent. It can be combined with other approaches for better matching or filtering. Indeed, there are rationales and benefits behind the service composition context as it informs us about the service's behavior and thereafter can unveil its business context. By using this context, we not only focus on specific parts of the business process but also avoid the computational complexity problem of the business process structure matching.

Due to the general limitation of public business process datasets, which provide only elements' identifier and services' names without any further information such as the one we used in our experiment, the validation of our approach so far is done with only the perfect match of services' names. However, our approach can be easily improved to deal with other comparison metrics and the validation can be extended for the imperfect matching. For example, we can take into account neighbors if their similarities on another comparison metric are greater than a certain threshold.

In our future work, we intend to investigate other service properties and study the co-existence of connection flows in business processes, as well as the number of times that a service is used in order to refine our matching

algorithm. We also aim at extending our approach to extract the hidden knowledge existing in business processes logs as another input.

References

[1] M. Sinnema, S. Deelstra, Classifying variability modeling techniques, Inf. Softw. Technol. 49 (7) (2007) 717–739. doi:10.1016/j.infsof.2006.08.001.
URL http://dx.doi.org/10.1016/j.infsof.2006.08.001

[2] M. Sinnema, S. Deelstra, J. Nijhuis, J. Bosch, Covamof: A framework for modeling variability in software product families, in: R. Nord (Ed.), Software Product Lines, Vol. 3154 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 197–213. doi:10.1007/978-3-540-28630-1_12.
URL http://dx.doi.org/10.1007/978-3-540-28630-1_12

[3] W. van der Aalst, Intra- and inter-organizational process mining: Discovering processes within and between organizations, in: P. Johannesson, J. Krogstie, A. Opdahl (Eds.), The Practice of Enterprise Modeling, Vol. 92 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, 2011, pp. 1–11. doi:10.1007/978-3-642-24849-8_1.
URL http://dx.doi.org/10.1007/978-3-642-24849-8_1

[4] C.-a. Sun, R. Rossing, M. Sinnema, P. Bulanov, M. Aiello, Modeling and managing the variability of web service-based systems, J. Syst. Softw. 83 (3) (2010) 502–516. doi:10.1016/j.jss.2009.10.011.
URL http://dx.doi.org/10.1016/j.jss.2009.10.011

[5] M. Koning, C. ai Sun, M. Sinnema, P. Avgeriou, Vxbpel: Supporting variability for web services in bpel, Information and Software Technology 51 (2) (2009) 258 – 269. doi:http://dx.doi.org/10.1016/j.infsof.2007.12.002.
URL http://www.sciencedirect.com/science/article/pii/S0950584908000207

[6] M. Dumas, L. García-Bañuelos, M. L. Rosa, R. Uba, Fast detection of exact clones in business process model repositories, Information Systems 38 (4) (2013) 619 – 633, special section on {BPM} 2011

conference. doi:http://dx.doi.org/10.1016/j.is.2012.07.002.
URL http://www.sciencedirect.com/science/article/pii/S0306437912000993

[7] R. Dijkman, M. L. Rosa, H. A. Reijers, Managing large collections of business process models—current techniques and challenges, Computers in Industry 63 (2) (2012) 91 – 97, managing Large Collections of Business Process Models Managing Large Collections of Business Process Models. doi:http://dx.doi.org/10.1016/j.compind.2011.12.003.
URL http://www.sciencedirect.com/science/article/pii/S0166361511001369

[8] X. Dong, A. Halevy, J. Madhavan, E. Nemes, J. Zhang, Similarity search for web services, in: VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases, VLDB Endowment, 2004, pp. 372–383.

[9] C. Platzer, S. Dustdar, A vector space search engine for web services, Web Services, 2005. ECOWS 2005. Third IEEE European Conference on (2005) 9 pp.–doi:10.1109/ECOWS.2005.5.

[10] M. B. Blake, M. F. Nowlan, A web service recommender system using enhanced syntactical matching, in: ICWS, 2007, pp. 575–582.

[11] J. Ma, Y. Zhang, J. He, Web services discovery based on latent semantic approach, in: ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services, IEEE Computer Society, Washington, DC, USA, 2008, pp. 740–747. doi:http://dx.doi.org/10.1109/ICWS.2008.135.

[12] Z. Zheng, H. Ma, M. R. Lyu, I. King, Wsrec: A collaborative filtering based web service recommender system, in: ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services, IEEE Computer Society, Washington, DC, USA, 2009, pp. 437–444.

[13] Y. Jiang, J. Liu, M. Tang, X. F. Liu, An effective web service recommendation method based on personalized collaborative filtering, in: IEEE International Conference on Web Services, ICWS 2011, Washington, DC, USA, July 4-9, 2011, 2011, pp. 211–218.

[14] T. Yu, Y. Zhang, K.-J. Lin, Efficient algorithms for web services selection with end-to-end qos constraints, ACM Trans. Web 1. doi:http://doi.acm.org/10.1145/1232722.1232728.

[15] U. S. Manikrao, 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, IEEE Computer Society, Washington, DC, USA, 2005, p. 117. doi:http://dx.doi.org/10.1109/NWESP.2005.32.

[16] M. Paolucci, T. Kawamura, T. R. Payne, K. P. Sycara, Semantic matching of web services capabilities, in: ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web, Springer-Verlag, London, UK, 2002, pp. 333–347.

[17] C. Wu, V. Potdar, E. Chang, Latent semantic analysis - the dynamics of semantics web services discovery, in: T. Dillon, E. Chang, R. Meersman, K. Sycara (Eds.), Advances in Web Semantics I, Vol. 4891 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2009, pp. 346–373.
URL http://dx.doi.org/10.1007/978-3-540-89784-2_14

[18] A. V. Paliwal, N. R. Adam, C. Bornhövd, Web service discovery: Adding semantics through service request expansion and latent semantic indexing, in: 2007 IEEE International Conference on Services Computing (SCC 2007), 9-13 July 2007, Salt Lake City, Utah, USA, 2007, pp. 106–113.

[19] N. Kokash, A. Birukou, V. D'Andrea, Web service discovery based on past user experience, in: Proceedings of the 10th international conference on Business information systems, BIS'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 95–107.

[20] A. Birukou, E. Blanzieri, V. D'Andrea, P. Giorgini, N. Kokash, Improving web service discovery with usage data, Software, IEEE 24 (6) (2007) 47–54. doi:10.1109/MS.2007.169.

[21] R. Birukou, E. Blanzieri, P. Giorgini, N. Kokash, A. Modena, Ic-service: A service-oriented approach to the development of recommendation systems, in: In: Proceedings of ACM Symposium on Applied Computing.

Special Track on Web Technologies, ACM Press, Press, 2007, pp. 1683–1688.

[22] K. Elgazzar, A. E. Hassan, P. Martin, Clustering wsdl documents to bootstrap the discovery of web services, in: Proceedings of the 2010 IEEE International Conference on Web Services, ICWS '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 147–154. doi:10.1109/ICWS.2010.31.
URL http://dx.doi.org/10.1109/ICWS.2010.31

[23] M. Weidlich, R. Dijkman, 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, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 483–498.
URL http://dl.acm.org/citation.cfm?id=1883784.1883832

[24] M. Weidlich, R. Dijkman, M. Weske, Behaviour equivalence and compatibility of business process models with complex correspondences, Comput. J. 55 (11) (2012) 1398–1418. doi:10.1093/comjnl/bxs014.
URL http://dx.doi.org/10.1093/comjnl/bxs014

[25] R. Dijkman, M. Dumas, 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, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 48–63. doi:10.1007/978-3-642-03848-8_5.
URL http://dx.doi.org/10.1007/978-3-642-03848-8_5

[26] Z. Yan, R. Dijkman, 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, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 60–77.
URL http://dl.acm.org/citation.cfm?id=1947725.1947737

[27] W. Aalst, A. Medeiros, A. Weijters, Process equivalence: Comparing two process models based on observed behavior, in: S. Dustdar, J. Fiadeiro, A. Sheth (Eds.), Business Process Management, Vol. 4102 of Lecture

Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 129–144. doi:10.1007/11841760_10.
URL http://dx.doi.org/10.1007/11841760_10

[28] R. M. Dijkman, A classification of differences between similar business processes, in: 11th IEEE International Enterprise Distributed Object Computing Conference, 15-19 October 2007, Annapolis, Maryland, USA, 2007, pp. 37–50.

[29] C. Li, M. Reichert, 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, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 248–264. doi:10.1007/978-3-540-87877-3_19.
URL http://dx.doi.org/10.1007/978-3-540-87877-3_19

[30] B. Dongen, R. Dijkman, J. Mendling, Measuring similarity between business process models, in: Proceedings of the 20th international conference on Advanced Information Systems Engineering, CAiSE '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 450–464.

[31] M. Ehrig, A. Koschmider, A. Oberweis, Measuring similarity between semantic business process models, in: Proceedings of the fourth Asia-Pacific conference on Comceptual modelling - Volume 67, APCCM '07, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 2007, pp. 71–80.
URL http://dl.acm.org/citation.cfm?id=1274453.1274465

[32] M. Rosemann, W. M. P. van der Aalst, A configurable reference modelling language, Inf. Syst. 32 (2007) 1–23. doi:10.1016/j.is.2005.05.003.
URL http://dl.acm.org/citation.cfm?id=1221586.1221839

[33] M. L. Rosa, M. Dumas, A. H. M. ter Hofstede, J. Mendling, Configurable multi-perspective business process models, Inf. Syst. 36 (2) (2011) 313–340.

[34] 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, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 8–25.
URL http://dl.acm.org/citation.cfm?id=1947725.1947733

[35] A. Awad, Bpmn-q: A language to query business processes, in: EMISA, 2007, pp. 115–128.

[36] A. Awad, A. Polyvyanyy, M. Weske, Semantic querying of business process models, in: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference, IEEE Computer Society, Washington, DC, USA, 2008, pp. 85–94. doi:10.1109/EDOC.2008.11.
URL http://dl.acm.org/citation.cfm?id=1437901.1438838

[37] A. Awad, G. Decker, M. Weske, Efficient compliance checking using bpmn-q and temporal logic, in: Proceedings of the 6th International Conference on Business Process Management, BPM '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 326–341. doi:10.1007/978-3-540-85758-7_24.
URL http://dx.doi.org/10.1007/978-3-540-85758-7_24

[38] S. Sakr, A. Awad, A framework for querying graph-based business process models, in: Proceedings of the 19th international conference on World wide web, WWW '10, ACM, New York, NY, USA, 2010, pp. 1297–1300.

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

[40] M. Hepp, D. Roman, An ontology framework for semantic business process management, in: eOrganisation: Service-, Prozess-, Market-Engineering: 8. Internationale Tagung Wirtschaftsinformatik - Band 1, WI 2007, Karlsruhe, Germany, February 28 - March 2, 2007, Universitaetsverlag Karlsruhe, 2007, pp. 423–440.
URL http://aisel.aisnet.org/wi2007/27

[41] C. Di Francescomarino, C. Ghidini, M. Rospocher, L. Serafini, P. Tonella, Reasoning on semantically annotated processes, in:

A. Bouguettaya, I. Krueger, T. Margaria (Eds.), Service-Oriented Computing – ICSOC 2008, Vol. 5364 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 132–146. doi:10.1007/978-3-540-89652-4_13.
URL http://dx.doi.org/10.1007/978-3-540-89652-4_13

[42] M. Dimitrov, A. Simov, S. Stein, M. Konstantinov, A bpmo based semantic business process modelling environment, in: Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management SBPM 2007, held in conjunction with the 3rd European Semantic Web Conference (ESWC 2007), Innsbruck, Austria, June 7, 2007, CEUR Workshop Proceedings, CEUR-WS.org, 2007.
URL http://ceur-ws.org/Vol-251/paper13.pdf

[43] R. K. Ko, S. S. Lee, E. Wah Lee, Business process management (bpm) standards: a survey, Business Process Management Journal 15 (5) (2009) 744–791. arXiv:http://dx.doi.org/10.1108/14637150910987937, doi:10.1108/14637150910987937.
URL http://dx.doi.org/10.1108/14637150910987937

[44] N. N. Chan, W. Gaaloul, Querying services based on composition context, in: IEEE International Conference on 23rd IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2014.

[45] W. van der Aalst, A. H. M. T. Hofstede, M. Weske, Business process management: A survey, in: Proceedings of the 1st International Conference on Business Process Management, Springer-Verlag, 2003, pp. 1–12.

[46] http://www-inf.it-sudparis.eu/simbad/tools/webrec/jss-proofs.pdf [online].

[47] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet Physics Doklady 10 (8) (1966) 707–710.

[48] N. Wirth, What can we do about the unnecessary diversity of notation for syntactic definitions?, Commun. ACM 20 (11) (1977) 822–823. doi:10.1145/359863.359883.
URL http://doi.acm.org/10.1145/359863.359883

[49] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, K. Wolf, Instantaneous soundness checking of industrial business process models, in: 7th BPM, 2009, pp. 278–293.

[50] M. Lincoln, M. Golani, 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, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 128–144.
URL http://dl.acm.org/citation.cfm?id=1882061.1882076

[51] M. Lincoln, 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, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 2–19.
URL http://dl.acm.org/citation.cfm?id=2074356.2074360

[52] M. Momotko, K. Subieta, Process query language: A way to make workflow processes more flexible, in: Advances in Databases and Information Systems, Vol. 3255 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2004, pp. 306–321.

[53] C. Beeri, A. Eyal, S. Kamenkovich, T. Milo, Querying business processes, in: Proceedings of the 32nd international conference on Very large data bases, VLDB '06, VLDB Endowment, 2006, pp. 343–354.
URL http://dl.acm.org/citation.cfm?id=1182635.1164158

[54] C. Beeri, A. Eyal, T. Milo, A. Pilberg, Monitoring business processes with queries, in: Proceedings of the 33rd international conference on Very large data bases, VLDB '07, VLDB Endowment, 2007, pp. 603–614.
URL http://dl.acm.org/citation.cfm?id=1325851.1325921

[55] S.-M.-R. Beheshti, B. Benatallah, H. Motahari-Nezhad, S. Sakr, A query language for analyzing business processes execution, in: S. Rinderle-Ma, F. Toumani, K. Wolf (Eds.), Business Process Management, Vol. 6896 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2011, pp. 281–297.

[56] E. Balan, T. Milo, T. Sterenzy, Bp-ex: a uniform query engine for business process execution traces, in: EDBT, 2010, pp. 713–716.

[57] D. Deutch, T. Milo, Querying structural and behavioral properties of business processes, in: Proceedings of the 11th international conference on Database programming languages, DBPL'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 169–185.
URL http://dl.acm.org/citation.cfm?id=1783534.1783552

[58] D. Deutch, T. Milo, A structural/temporal query language for business processes, J. Comput. Syst. Sci. 78 (2) (2012) 583–609.

[59] I. Markovic, A. Costa Pereira, D. Francisco, H. Muñoz, Querying in business process modeling, in: E. Nitto, M. Ripeanu (Eds.), Service-Oriented Computing - ICSOC 2007 Workshops, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 234–245.

[60] M. Guentert, M. Kunze, M. Weske, Evaluation measures for similarity search results in process model repositories, in: P. Atzeni, D. Cheung, S. Ram (Eds.), ER '12, Vol. 7532, Springer Berlin Heidelberg, 2012, pp. 214–227.

[61] M. Kunze, M. Weidlich, M. Weske, Behavioral similarity: A proper metric, in: Proceedings of the 9th International Conference on Business Process Management, BPM'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 166–181.
URL http://dl.acm.org/citation.cfm?id=2040283.2040301

[62] J. Wang, T. He, L. Wen, N. Wu, A. H. M. Ter Hofstede, J. Su, A behavioral similarity measure between labeled petri nets based on principal transition sequences, in: Proceedings of the 2010 International Conference on On the Move to Meaningful Internet Systems - Volume Part I, OTM'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 394–401.
URL http://dl.acm.org/citation.cfm?id=1947725.1947759

[63] N. N. Chan, W. Gaaloul, S. Tata, Composition context matching for web service recommendation, in: Proceedings of the 2011 IEEE International Conference on Services Computing (SCC), IEEE Computer Society, Washington, DC, USA, 2011, pp. 624–631. doi:10.1109/SCC.2011.68.
URL http://dx.doi.org/10.1109/SCC.2011.68