Equivalence of Web Services in Process-Aware Service Compositions

Stefanie Rinderle-Ma, Manfred Reichert
Institute DBIS, Ulm University, Germany
{stefanie.rinderle, manfred.reichert}@uni-ulm.de

Martin Jurisch
AristaFlow GmbH, Germany
martin.jurisch@aristaflow.de

Abstract

Deciding on web service equivalence in process-aware service compositions is a crucial challenge throughout the composition life cycle. Restricting such decisions to (activity) label equivalence constitutes a simplification for many practical applications: if two web services have equivalent labels, does this necessarily mean they are equivalent as well? In many scenarios other factors play an important role. Examples include context information (e.g., input and output messages) and information on the position of web services within compositions. In this paper, we introduce the composition life cycle and discuss specific requirements for web service equivalence along its different phases. We define adequate equivalence notions for design, execution, analysis, and evolution of service compositions. Main focus is put on attribute and position equivalence. Altogether this paper is a first step towards a new understanding and treatment of equivalence notions in service compositions.

1. Introduction

The challenge of finding and checking adequate notions for semantic equivalence is an important research subject in many areas. In federated databases or data warehouses, for example, data from heterogeneous sources need to be integrated within one schema. Among other problems, schema integration has to deal with *synonyms* and *homonyms* [14]; i.e., equivalent attribute labels describing different data, or attributes with different labels but describing the same data. The capability of the integration process to handle such cases is crucial for its success.

Similar problems emerge when designing distributed processes as can be found in inter-organizational partner settings. As example consider a collaboration between partners from the US, India and Germany, which shall be implemented as web service choreography. At least, we need to translate web service labels between the service compositions of the partners. Still, such language-related aspects can be easily handled using ontologies (e.g., based on OWL-S [12]) since it can be taken for granted that service labels `confirm` and `bestaetigen` (German term for confirm) are homonyms. However, in many applications such knowledge is not at hand. One important use case is the mining of (completed) executions of service compositions; i.e., techniques to derive composition schemas from execution logs [27]. Since there is no a-priori knowledge on such logs, basically, it cannot be taken for granted that web services with equivalent label are equivalent themselves. Regarding our example, service `confirm` might have a different "meaning" depending on whether it is performed by a manager or a secretary. Obviously, we also need to consider information about the context of a service execution when defining equivalence notions for web services.

Another use case is the evolution of service compositions. Here, a composition schema is structurally changed by adding, deleting or moving web service executions. The ability to adequately support such changes has become crucial since a turbulent market and a constantly changing environment force any enterprise to quickly and correctly adapt their running business processes and service compositions, respectively. In this context, a composition runtime environment must enable both ad-hoc changes of single instances of a composition schema (e.g., to react on exceptions) and changes of a composition schema itself. The latter may have to be propagated to already running instances of a composition schema and become necessary, for example, when new regulations come into effect or the composition schema is redesigned. If changes are concurrently applied at composition schema and composition instance level, however, it becomes crucial to carefully decide on the equivalence of the affected web services. If, for example, two web services with equivalent label are inserted at both schema and instance level, we have to decide on their actual equivalence in order to avoid duplicate insertions at the instance level afterwards [19].

From the above examples we can conclude that different notions of equivalence are needed for comparing the services within compositions; this concerns all phases of the composition life cycle, i.e., design, execution, analysis, and

evolution. First, we summarize the service composition life cycle. Along it we discuss different requirements for web service equivalence notions. Based on these requirements we provide notions for label equivalence, attribute equivalence, and position equivalence, which significantly extend the existing restricted view on unique labels. We further show that these equivalence notions are extensible for the particular needs of other use cases.

Section 2 presents the composition life cycle. In Section 3 label equivalence is defined and discussed along the design and execution phase of a service composition. Attribute equivalence is introduced in Section 4 followed by position equivalence in Section 5. Section 6 discusses related work and Section 7 concludes with a summary.

2. Web service composition life cycle

Web service composition based on languages like BPEL or BPMN enable process-aware *orchestration* of web services [2, 24, 13]. The basic idea behind this is illustrated by Fig. 1. At composition level, two *activities* are connected in a sequence: first activity A invokes web service S and then activity B invokes web service T. The basic challenge addressed in this paper is how to decide at both composition and web service level when two activities and services respectively (e.g., S and T) can be considered as being equal. This section sketches the life cycle of web service compositions since the specific challenges for *web service equivalence* within compositions can be discussed along the different phases of the life cycle. Further, we show that for different life cycle phases different notions of equivalence between services are needed and thus have to be defined.

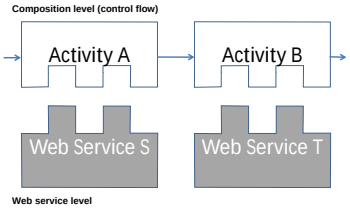

Figure 1. Web service composition

Similar to business processes, web service compositions undergo a life cycle [31] (cf. Fig. 2). Within the *design phase* the web service composition of interest is constructed resulting in a *web service composition schema*. As discussed in [23], for example, the web service composition is designed at a semantically rather high level in this phase. A common specification language for web service compositions at design time is BPMN [10]. As example take web service composition schemes S and S' as depicted in Fig. 3: S consists of three web services A, B, and C to be executed in sequence. In addition to such simple control flow structures, BPMN supports more complex control flow patterns (e.g., parallelism, alternative branchings, and loops). Typically, web service composition schemes capture message flow aspects as well; in composition schema S', for example, web service X is sending a message d to subsequent service Y (stored in flow variable $data$).

After the design of a composition, the resulting schema is deployed to web service flow engine(s) (e.g., IBM WebSphere Process Server or ADEPT2 Process Engine [3]). Usually, this includes the translation to an executable language such as BPEL.[1] However, it is also common that an explicit design phase is omitted and the stateful service is directly composed within the service flow engine. After its deployment the composition schema can be instantiated multiple times. Each of the resulting stateful *composition instances* is then executed, i.e., the underlying engine invokes the right service at its activation time with the right input messages and, if necessary, assigns it to the right users (e.g., based on BPEL4People). When finishing the execution of a web service, its status is set to completed and the next service(s) to be executed are determined. Finally, execution behavior of composition instances is captured in *execution traces* [8].

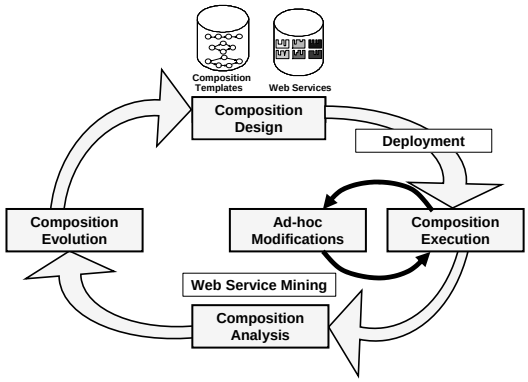

Figure 2. Web service composition life cycle

When using adaptive flow engines like ADEPT2 [3], the execution phase also enables *ad-hoc modifications* of composition instances (i.e., to structurally adapt the composition schema for one particular instance). Note that such instance-specific adaptations often become relevant in real-world scenarios [9, 28]. We refer to ad-hoc modified composition instances as *biased* instances in the following.

To ensure continuous improvement of a web service composition during its life cycle (cf. Fig. 2), its instances should steadily undergo a performance analysis. Similar to

[1] Comparable to business processes, there is a difference between languages used at design time and those specifying executable process. Existing mappings (e.g., from BPMN to BPEL [11]) are helpful here.

the analysis of executed workflows in process-aware information systems, relevant techniques comprise performance analysis and mining [6, 27, 1]. In the context of web service equivalence, particularly, mining is interesting. Mining techniques focus on deriving process structures (and service composition schemes respectively) from execution traces (*composition mining*), on checking different properties of composition instances, and on analyzing ad-hoc changes applied at instance level (*change mining* [4, 6]). With mining, deviations from the original composition schema can be detected, i.e., we can check whether or not composition instances "behave" as specified in the composition schema. This information can be used to improve composition quality (e.g., by exterminating design flaws) [29] and to discover a generic composition schema out of a collection of previously adapted composition instances [6].

In the evolution phase, the improvements discovered in the analysis phase are applied to service composition schema S resulting in new schema version S' (cf. Fig. 3). One challenge is how to deal with already running composition instances (cf. Fig. 3). In existing systems, so far, optimizations can only be applied to newly started composition instances, i.e., running instances have to finish according to the old composition schema S. Though this is sufficient for instances of short duration, for long-running instances it is crucial to *propagate* schema changes to already running instances as well [20, 18]. In other words, we need to be able to *migrate* running composition instances to the new compositions schema version. However, such migration should not be done in an uncontrolled manner; i.e., system robustness must never be harmed. In Fig. 3, for example, migrating composition instance $I2$ to new schema version S' would result in an inconsistent instance state where newly added activity Y is not correctly supplied with input message d at runtime (e.g., leading to an erroneous service invocation or to a deadlock depending on the flow engine). Thus adequate correctness criteria for composition evolution and subsequent instance migration are needed. We have introduced a framework for the evolution of business processes in [20, 18]. As shown in [15, 21], the concepts can be transferred to the evolution of web service compositions.

3. Composition design and execution: label equivalence

During composition design (cf. Fig. 2) – regardless whether a distinct design phase is taking place or the (executable) composition is directly constructed – equivalence of composed web services is mostly defined based on the label of the activities invoking them [1]. In Fig. 1, for example, web services S and T are considered as being not equivalent, since the labels of activities A and B are different. The same kind of equivalence can be defined based on

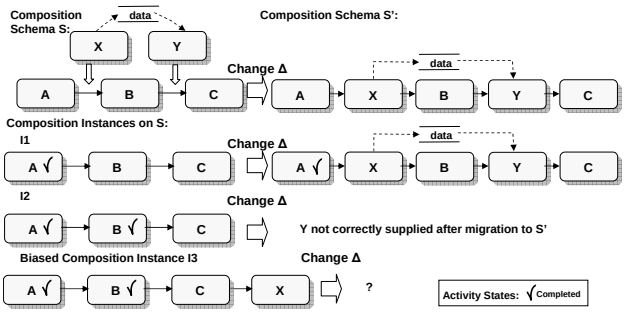

Figure 3. Evolution of composition schemas

the web service labels themselves. However, without loss of generality, we can focus on *label equivalence* between composition activities:

Definition 1 (Label equivalence) *Let $\mathcal{PS}$ be the set of all web service composition schemas, let $\mathcal{A}$ be the set of activities based on which web service composition schemas $S \in \mathcal{PS}$ are specified, and let W be the set of web services which are invoked by any activity in A. Then two web services w_1, $w_2 \in \mathcal{W}$ are called label equivalent if $a_1.l = a_2.l$ where a_1 is the activity invoking w_1 and a_2 is the one invoking w_2 (l denotes the generic label attribute of an activity).*

If two web services are label-equivalent, does this necessarily mean that they are also equivalent themselves? Here the general answer is *no* since label equivalence between services does not enforce their equivalence regarding other aspects (e.g., equivalence of input/output messages). Assume, for example, that activity `write_letter` is used twice within a composition. Assume further that one of these activities invokes service `W1` requiring input message `form` and the other one invokes service `W2` without any input message. Though `W1` and `W2` are label-equivalent (cf. Def. 1), *execution context* of `W1` and `W2` is different.

Obviously, we have to dig beyond the notion of label equivalence. Intuitively, equivalence of services within a composition also depends on *what* they are doing (e.g., what kind of input message is processed and what kind of output message is produced), and on *how* they are used within the composition. Consider Fig. 4: Web service S is plugged into a service composition. More precisely, S is connected to activity A of this composition. While S provides its specific functionality (i.e., the *web service context*), the corresponding composition activity is assigned its specific *activity context*. Service S is then executed within the activity context of A (e.g., specifying which actors are allowed to process A in the given context). In the following, we use the term "context" to summarize the attributes linked to a web service or composition activity, which specify the

conditions the service or activity may be executed in. Regarding a web service context, it is definitely necessary to specify correct input and output messages. This requires an adequate mapping to the associated composition activity; i.e., it must be ensured that the input message of the underlying web service can be supplied by the input data of the associated composition activity (i.e., out of the flow data context), and that the output message of the web service is correctly mapped onto output data of the associated composition activity. Thus, input and output messages are context attributes relevant for both services and composition activities. Context attributes specifically relevant for services are quality attributes such as quality of service or service level agreements. Composition activities have specific context attributes like actor assignment or duration. Intentionally, we do not give a "formal definition" here since we believe that the set of context attributes for services as well as for activities should be configurable and extensible.

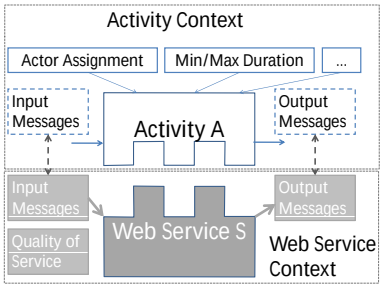

Figure 4. Plugging web service into composition activity context

Consequently, we distinguish the following three cases:

Definition 2 (Equivalence, synonyms, homonyms) *Let the assumptions be as in Def. 1. Let* $w_1, w_2 \in \mathcal{W}$ *be web services. Then:*

1. w_1 *and* w_2 *are called equivalent if* w_1 *and* w_2 *are label-equivalent and have equivalent semantics (formally denoted by* $w_1 \equiv_{sem} w_2$*)*
2. w_1 *and* w_2 *are called synonyms if* w_1 *and* w_2 *are label-equivalent, but do not have equivalent semantics; i.e.,* $\neg(w_1 \equiv_{sem} w_2)$
3. w_1 *and* w_2 *are called homonyms if* w_1 *and* w_2 *are not label-equivalent, but have equivalent semantics; i.e.,* $w_1 \equiv_{sem} w_2$

What are the specific problems occurring in the context of equivalent, synonymous, or homonymous web services within compositions? As we show in the following sections, equivalent web services require a different treatment in comparison to synonymous or homonymous ones, particularly in the context of mining and change. Thus, if we knew exactly that two web services are equivalent, synonymous or homonymous, we could take the right decision on how to react within a particular situation (e.g., when adding two equivalent services at schema and instance level). However, in practice the exact relation between services within a composition is not always clear. Reasons are that compositions might evolve over time or have been not designed in a rigor manner (e.g., using a controlled vocabulary). Hence it will be crucial to provide means of how to decide on equivalence of web services if no a-priori knowledge is available. Obviously, problems might arise in connection with synonyms and homonyms. In addition, we have to take care of multiple occurrences of equivalent web services, particularly in the context of composition changes and evolution (cf. Section 5). However, before discussing solutions for the different phases of the composition life cycle, we finish our considerations on the design phase of compositions.

Focusing on **composition design**, we can abstract from synonyms and homonyms if an ontology [12] is used; e.g., in combination with a service repository (similar to activity repositories in process-aware information systems [3]). Still, there might be equivalent services occurring multiple times within a composition schema [5]. Note that for realistic applications, the multiple usage of the same web service from the repository might be desirable (e.g., obtaining product information in different stages of an orchestration).

During **composition execution** (cf. Fig. 2) multiple occurrences of a particular web service can be realized based on specific composition patterns like *multi instantiation* [26]. As described in [26, 17], multi-instantiation adds an instance of the same (web) service multiple times to a given composition instance. This pattern has been realized, for example, in BPEL and UML. Note that when using multi-instantiation pattern, the multiple runtime occurrence of instances of the same activity is desired and happens in a controlled manner. Specifically, in this case we can consider these multiple activity instances as being equivalent according to Def. 2 since they constitute some kind of copies of each other. Thus there will be no side-effects caused by unknown equivalent activities at runtime.

During the execution of a composition instance, **ad-hoc modifications** might become necessary (e.g., to react on an exceptional situation). If the ad-hoc change is conducted based on a controlled vocabulary and repository, respectively (i.e., new web services to be inserted at instance level are chosen from the repository), the occurrence of synonyms, homonyms, and equivalent web services (cf. Def. 2) can be controlled as well; i.e., it will be clear which kind of equivalent notion holds for two web services. However, equivalent web services might be inserted leading to their multiple occurrence within the affected composition

instance. This requires no specific action at execution time. However, as we show in Section 5, in conjunction with **composition evolution**, side-effects between web services inserted at schema and at instance level might occur; i.e., another notion of equivalence between services becomes necessary.

4. Composition analysis: attribute equivalence

We now focus on the analysis of executed composition instances; i.e., we look at mining techniques which construct the composition schema out of a given set of *composition instances traces* (traces for short). Informally, a trace contains all events happening during execution of a composition instance; i.e., the start and end events of composition activities executed in the run of the instances [27].

Which challenges do occur regarding the equivalence of web services when applying mining techniques? Currently, most (process) mining algorithms assume that equivalence of web services (or process activities respectively) is constituted by their label equivalence. This assumption will hold if the usage of a controlled vocabulary (e.g., a web service repository) can be ensured when designing, executing and changing compositions. However, this often constitutes a non-valid simplification in practice. In most cases, there is no information about whether or not a controlled vocabulary has been used for the design of the composition schema whose instances have produced the corresponding traces.

However, even if the usage of a controlled vocabulary can be assumed, this does not solve the problem of multiple occurrences of label-equivalent web services. Consider the example depicted in Fig. 5a where – at first sight – web service `sign` is used twice within web service composition S. Possible traces on S are σ_1 and σ_2 with

- σ_1 =<`apply, sign, appoint, prepare, exam, sign, inform`> and
- σ_2 =<`apply, sign, prepare, appoint, exam, sign, inform`>.

Based on label information any mining algorithm counting frequencies of activity (label) occurrences within the traces would fail to detect the actual process structure. This statement will be leveraged, if unique node identifiers are stored in addition to activity labels (e.g., for traces produced in ADEPT2 and mined by the ProM framework [4]). Reason is that, for example, the first occurrence of activity `sign` becomes distinguishable from the second occurrence, if both are additionally labeled as `(sign,2)` and `(sign,6)`.

However, even if we use a combination of activity labels and node identifiers for web services `(sign,2)` and `(sign,6)`, there is no statement on equivalence of their execution semantics. More precisely, what is the information we can obtain from the label and node identifier combination for `(sign,2)` and `(sign,6)`? It can be concluded that within the underlying composition there are two label-equivalent web services, which occur at different positions within the composition. However, there is no information on their execution semantics. Hence, it is not clear whether `(sign,2)` and `(sign,6)` are equivalent or homonymous. As example consider Fig. 5 and assume that composition activities having label `sign` are both connected with same web service. Still they can be distinguished by their composition activity context. Specifically, as can be seen from the swim lane representation in Fig. 5, actor assignments are different; i.e., first an actor with role `secretary` is authorized to perform `sign` and later in the composition execution, this activity may be performed by an actor with role `professor`.

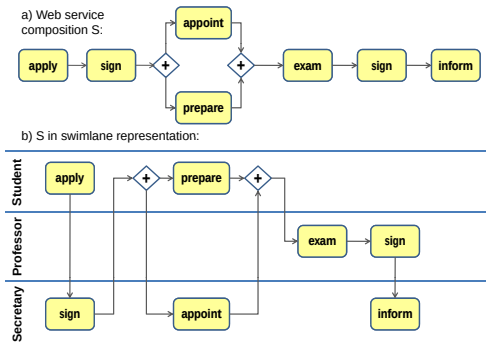

Figure 5. Example (in BPMN notation)

From the above considerations we can conclude that equivalence of web services within a composition can be specified on subsets of the context attributes of both, the web services and the composition activities. Based on this, it can be specified more precisely whether or not two web services within a composition are actually equivalent or used synonymously. Note that in the example depicted in Fig. 5, we consider the equivalence or distinction of web services within compositions based on *static* attribute values; i.e., values which are determined during design time. However, there are also *dynamic* attribute values, which might be used for deciding on the equivalence of web services. One example are message or data values, which are written during the execution time of a composition instance. Another example is provided by so called dynamic actor assignments (e.g., activity `Y` shall be processed by the same actor who worked on preceding activity `X`).

Of course, when using attribute-based equivalence, performance considerations have to be taken into account as well. Typically, it is not a big performance problem to check for attribute equivalence in case of label-equivalent web services when mining composition logs – their number is restricted. Theoretically, the case might occur that all services within the traces describing the execution behav-

ior of a certain composition schema might stem from labels which constitute homonyms. Then, for a multitude of possible traces over a complex service composition (where for all traces all entries are to be compared to all other entries), a performance penalty would result. However, in practical scenarios, this case is of rather theoretical nature.

5. Composition evolution: position equivalence

To be able to discover equivalent web services within a service composition is also important when composition schema and composition instances are changed concurrently; i.e., if composition schema changes (applied at type level) shall be propagated to biased composition instances. As example consider the evolution scenario depicted in Fig. 6. Initially, two (biased) instances are running on composition schema S. I_1 has been individually biased by inserting web service (activity) X between B and C. I_2, in turn, has been modified by inserting X between A and B. In a practical scenario, the composition designer might notice that composition instances again and again deviate from the original composition schema by having added web service (activity) X. As a consequence the designer might decide to lift up the instance-specific changes to the composition schema level. Such strategies can be supported by the use of intelligent mechanisms as discussed in [28]. Assume that S is optimized by inserting new web service (activity) X between B and C. Then the challenge is to decide on an adequate migration strategy for running instances I_1, I_2, etc.

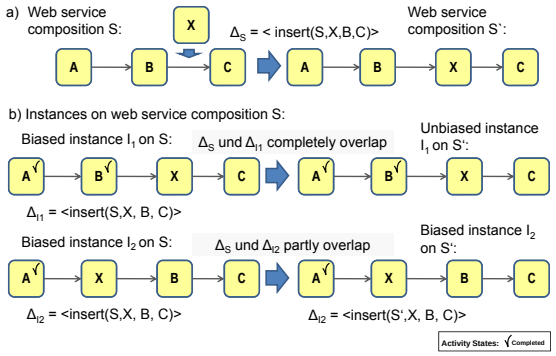

Figure 6. Evolution with biased instances

In this scenario the changes of a composition schema and a composition instance *overlap* as typical when instances anticipate later schema optimizations. As can be seen from Fig. 6, the *overlap degree* between instance-specific change Δ_{I_1} and composition change Δ_S is different from the one between Δ_{I_2} and Δ_S. Specifically, Δ_{I_1} and Δ_S *completely overlap* whereas Δ_{I_2} and Δ_S only *partly overlap*. Fig. 7 gives an overview of different overlap degrees between instance- and composition-specific changes. – Why is it important to distinguish between different kinds of overlap? As can be seen from Fig. 6, the strategies for migrating I_1 and I_2 to modified schema version S' are different. I_1 can be directly migrated to S' (i.e., without further checks) and then becomes unbiased again based on S'. In turn, for I_2 an instance-specific change has to be maintained after migrating it to S'. More details on migration strategies based on particular degrees of overlap can be found in [19, 28].

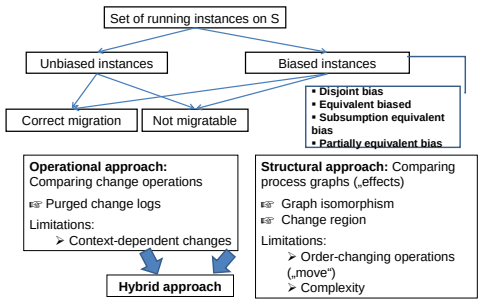

Figure 7. Possible migration strategies in the context of composition evolution

Regarding equivalence of web services, the challenge for composition evolution is as follows: even if we assume that web service X inserted at instance level and web service X inserted at composition (i.e., type) level are equivalent according to Def. 2, this information is not sufficient to decide on the degree of overlap between instance-specific change and composition change. We also need to know the particular position of the web service activities within the respective graphs. This leads us to another notion of equivalence for web services; i.e., *position equivalence*. In Fig. 6, for example, X is position-equivalent for S and I_1.

Basically, as described in Fig. 7, we can decide on position equivalence of web services within composition graphs in two ways: either the composition schema (*structural approach*)[2] or the applied changes (*operational approach*) are directly compared. Since both approaches show specific limitations (e.g., complexity of graph comparison), they have been combined to a *hybrid* approach. Specifically, the hybrid approach first compares sets of newly inserted or deleted composition activities (structural approach) and extracts the missing information on order-changing operations from the applied changes (operational approach). Starting from this idea, in the following we provide an algorithm to quickly determine the positions of newly inserted composition activities, which can be used to decide on position equivalence (details on other change patterns can be found in [16]).

[2]Here methods such as graph isomorphism can be used.

Algorithm 1 (Positions for insert operations) *Let S be a composition schema and let Δ be a change which transforms S into composition schema S'. Let further N_{Δ}^{add} be the set of newly added web service activities in S' and N_{Δ}^{move} be the set of moved ones. Let further CtrlE denote the set of all control links in S and CtrlE' the set of all control links in S' respectively. Then: The positions of the insert operations applied within Δ – $PosIns(S, \Delta)$ – can be determined as follows:*

$PosIns(S, \Delta) = \emptyset$;
for all (X $\in N_{\Delta}^{add}$) do
  find {(left, X), (X, right)} $\in$ CtrlE';
  while (left $\in N_{\Delta}^{add} \cup N_{\Delta}^{move}$) do
    find (leftleft, left) $\in$ CtrlE';
    left = leftleft;
  od
  while (right $\in N_{\Delta}^{add} \cup N_{\Delta}^{move}$) do
    find (right, rightright) $\in$ CtrlE';
    right= rightright;
  od
  $PosIns(S, \Delta) = PosIns(S, \Delta) \cup$ {(left,X,right)};
od

Regarding Fig. 6, we obtain the following position sets: $PosIns(S, \Delta_S)$ = {(B, X, C)}, $PosIns(S, \Delta_{I_1})$ = {(B, X, C)}, and $PosIns(S, \Delta_{I_2})$ = {(A, X, B)}. Based on this and the assumption that X is equivalent in all three cases, one can easily conclude that Δ_S and Δ_{I_1} are completely overlapping and Δ_S and Δ_{I_2} are partly overlapping.

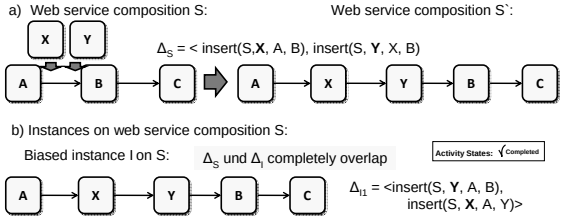

Figure 8. Evolution with multiple occurrence of equivalent web services

The algorithm for determining position equivalence between insert operations at different levels also works if newly added web services "use" other newly added ones as insertion context. Consider Fig. 8: web services X and Y are inserted at composition and instance level at same position, but in different order; i.e., at schema level first X is inserted followed by Y, whereas at instance level first Y is inserted and then X. In this case, Algorithm 1 is "looking" for the first occurrence of a service which has been already present within the composition. Within the example, for schema as well as instance change, the positions would be determined as $PosIns(S, \Delta_S)$ = $PosIns(S, \Delta_S)$ = {(A,X,B), (A,Y,B)}. Based on this, Δ_S and Δ_I can be correctly classified as completely overlapping.

In practical scenarios it might become necessary to allow for more "fuzzy" position equivalence notions. For example, new web services do not always have to be inserted at exact this or that position, but within a certain *region* of the composition schema. Then the notion of position equivalence can be relaxed to *region equivalence*.

6. Related Work

We have already referred to some related work in previous sections. Generally, the treatment of synonyms and homonyms is important for any schema integration problem (as can be found, for example, in federated databases or data warehouses) [14]. Obviously, there are similarities to the equivalence of web services as described in this paper. In both cases, it might be necessary to decide on equivalence beyond label equivalence. However, characteristics of data and web services with respect to equivalence are different; i.e., web services embrace much more information, which might become relevant for equivalence checks, than data: web services have an execution context, offer process logic, and so forth. These issues are taken into consideration when reasoning, for example, about semantic matchmaking of web services. Describing web services semantically by using, for example, OWL-S, these approaches apply label and/or attribute equivalence [22]. Position equivalence, i.e., information on the specific position within a composition, has not been considered so far, but can be used to enhance information as basis for matchmaking.

In the process management area there are several approaches for deciding on composition equivalence and similarity based on equivalence notions such as graph isomorphism, trace equivalence, and bi-simulation [25, 30, 7]. Specifically, it is decided whether two compositions reflect the same process behavior, whereas in this paper, we focus on equivalence notions between single web services. However, particularly attribute and position equivalence could be used to support the aforementioned equivalence notion for compositions, since most of them assume unique labeling of process activities.

7. Summary and Outlook

We discussed requirements for equivalence notions of web services within compositions along the phases of the composition life cycle. Considered equivalence notions include label equivalence, attribute equivalence, and position equivalence. Attribute equivalence focuses on the context, in which web services are executed. Context, in turn, refers

to the web service context itself (e.g., input messages) as well as to activity context. The latter is determined by the composition activity into which the web service is plugged and its specific attributes (e.g., actor assignments). Attribute equivalence is particularly important for composition analysis (e.g., composition mining). Finally, position equivalence refers to the position where a service is added to a composition. Respective information is useful for deciding on position equivalence in connection with composition evolution. The different equivalence notions refer to single phases of the composition life cycle and are generic. Nevertheless, the framework presented in this paper needs to be extensible; i.e., users must be able to specify their own equivalence notions if necessary. In future work we will elaborate equivalence notions in the context of composition evolution. A first step is to introduce a more general notion of region equivalence. Furthermore, we want to investigate which kind of equivalence notions are necessary for more complex evolution scenarios.

References

[1] A. Alves de Medeiros, W. van der Aalst, and C. Pedrinaci. Semantic process mining tools: Core building blocks. In *Proc. ECIS'08*, 2008.

[2] B. Benatallah, Q. Sheng, and M. Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, 7(1):40–48, 2003.

[3] P. Dadam, M. Reichert, S. Rinderle, M. Jurisch, H. Acker, K. Göser, U. Kreher, and M. Lauer. Towards truly flexible and adaptive process-aware information systems. In *Int'l Conference UNISCON*, pages 72–83, 2008.

[4] C. Günther, S. Rinderle-Ma, M. Reichert, W. van der Aalst, and J. Recker. Using process mining to learn from process changes in evolutionary systems. *Int'l J of Business Process Integration and Management*, 3(1):61–78, 2008.

[5] J. Koehler and J. Vanhatalo. Process anti-patterns: How to avoid the common traps of business process modeling, part 1. *IBM WebSphere Developer Technical Journal*, 2007.

[6] C. Li, M. Reichert, and A. Wombacher. Discovering reference process models by mining process variants. In *Proc. ICSW'08*, pages 45–53, 2008.

[7] C. Li, M. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. In *Proc. ER'08*, LNCS 5231, pages 248–264, 2008.

[8] L. Ly, S. Rinderle, and P. Dadam. Integration and verification of semantic constraints in adaptive process management systems. *Data and Knowledge Eng.*, 64(1):3–23, 2008.

[9] B. Mutschler, M. Reichert, and J. Bumiller. Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors and implications. *IEEE Transactions on Systems, Man, and Cybernetics*, 38(3):280–291, 2008.

[10] OMG. *Business Process Modeling Notation, V1.1*, omg document number: formal/2008-01-17 edition, 2008.

[11] C. Ouyang, W. van der Aalst, M. Dumas, and A. ter Hofstede. Translating BPMN to BPEL. Technical Report BPM-06-02, BPM Center, 2006.

[12] OWL-S. Home page., 2003.

[13] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[14] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *Int'l Journal Very Large Databases*, 10(4):334–350, 2001.

[15] M. Reichert, S. Rinderle, and P. Dadam. On the modeling of correct service flows with BPEL4WS. In *Proc. EMISA'04*, pages 117–128, Luxembourg, 2004.

[16] S. Rinderle. *Schema Evolution in Process Management Systems*. PhD thesis, Ulm University, 2004.

[17] S. Rinderle and M. Reichert. Data-driven process control and exception handling in process management systems. In *Proc. CAiSE'06*, LNCS 4001, pages 273–287, 2006.

[18] S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems – a survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.

[19] S. Rinderle, M. Reichert, and P. Dadam. Disjoint and overlapping process changes: Challenges, solutions, applications. In *CoopIS'04*, LNCS 3290, pages 101–120, 2004.

[20] S. Rinderle, M. Reichert, and P. Dadam. Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases*, 16(1):91–116, 2004.

[21] S. Rinderle, A. Wombacher, and M. Reichert. Evolution of process choreographies in DYCHOR. In *Proc. CoopIS'06*, LNCS 4275, pages 273–290, 2006.

[22] G. Shua, O. Ranab, N. Avisb, and C. Dingfanga. Ontology-based semantic matchmaking approach. *Advances in Engineering Software*, 38(1):59–67, 2006.

[23] K. Terai, N. Izumi, and T. Yamaguchi. Coordinating web services based on business models. In *Int'l Conference on Electronic Commerce*, pages 473–478, 2003.

[24] W. van der Aalst. Dont go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 2003.

[25] W. van der Aalst, A. Alves de Medeiros, and A. Weijters. Process equivalence: Comparing two process models based on observed behavior. In *Proc. BPM'06*, LNCS 4102, pages 129–144, 2006.

[26] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.

[27] W. van der Aalst and H. Verbeek. Process mining in web services: The websphere case. *IEEE Bulletin of the Technical committee on Data Engineering*, 33(3):46–49, 2008.

[28] B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3):438–466, 2008.

[29] B. Weber, M. Reichert, W. Wild, and S. Rinderle-Ma. Providing integrated life cycle support in process-aware information systems. *Int. J Coop. Inf. Sys.*, 18(1), 2009.

[30] A. Wombacher and A. Martens. Soundness and equivalence of petri nets and annotated finite state automate. In *Proc. IEEE DEST'07*, 2007.

[31] J. Yang and M. Papazoglou. Service components for managing the life-cycle of service compositions. *Information Systems*, 29(2):97–125, 2004.