International Journal of Web Services Research, 8(1):41-67, 2011

On Utilizing Web Service Equivalence for Supporting the Composition Life Cycle

Stefanie Rinderle-Ma[1], Manfred Reichert[2], Martin Jurisch[3]
[1]Workflow Systems and Technology Group, University of Vienna, Austria
stefanie.rinderle-ma@univie.ac.at
[2]Institute of Databases and Information Systems, University of Ulm, Germany
manfred.reichert@uni-ulm.de
[3]AristaFlow GmbH, Ulm, Germany
martin.jurisch@aristaflow.com

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, however, is not sufficient for many practical applications: if two activities and web services respectively have equivalent labels, does this necessarily mean they are equivalent as well? In many scenarios (e.g., evolution of a composition schema or mining of completed composition instances) other factors may play an important role as well. 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 whole composition life cycle and discuss specific requirements for web service equivalence along its different phases. We define adequate equivalence notions for the design, execution, analysis, and evolution of service compositions. Main focus is put on *attribute* and *position* equivalence. Altogether this paper shall contribute a new understanding and treatment of equivalence notions in service compositions.

KEY WORDS:
Web Service Composition, Life Cycle of Service Compositions, Web Service Equivalence

1 INTRODUCTION

The challenge of providing adequate notions for semantic equivalence constitutes 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. Amongst other problems, schema integration has to deal with *synonyms* and *homonyms* (Rahm & Bernstein, 2001); i.e., equivalent attribute labels describing different data, or attributes with different labels but describing same data. The capability of the integration process to handle such cases is crucial for its success.

1.1 Problem Statement

Similar problems emerge when designing distributed processes as can be found in inter-organizational partner settings. As an 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 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 and service orchestrations respectively; i.e., techniques to derive composition schemas from execution logs (van der Aalst & Verbeek, 2008). Since there is no a-priori knowledge on such logs, basically, it cannot be taken for granted that web services with equivalent labels 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 concerns the evolution of service compositions. Here, a composition schema is structurally changed by adding, deleting or moving steps (i.e., activities representing web service executions). The ability to adequately support such changes has become crucial since a turbulent market and a continuously changing environment force any enterprise to quickly and correctly adapt their running business processes and service compositions, respectively (Mutschler, Reichert, & Bumiller, 2008). In this context, a service orchestration engine must enable both ad-hoc changes of single instances of a composition schema at runtime (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 (e.g., due to business reengineering efforts). 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 (Rinderle, Reichert & Dadam, 2004).

1.2 Contribution

From the above examples we can conclude that different notions of equivalence are needed for comparing the services within compositions. Particularly, this concerns all phases of the composition life cycle, i.e., design, execution, analysis, and evolution. In this paper, first, we summarize the *service composition life cycle*. Along it we discuss practically relevant 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 discuss and compare the application of attribute and position equivalence in the context of evolving service composition schemes. Here, particular focus is put on how to utilize web service equivalence in order to correctly evolve service composition schemes even if changes have been concurrently applied at the level of single composition instances (e.g., to deal with exceptional situations). In addition to the consideration of correctness issues, a performance analysis is conducted to estimate the efforts of applying the different equivalence notions in order to support the evolution of large-scale service compositions. The service composition life cycle is illustrated by a powerful implementation we realized within the AristaFlow process management technology.

This paper constitutes a significant extension of the work we presented in (Rinderle-Ma, Reichert & Jurisch, 2009). Specifically, the results of discussing and comparing the applicability of the different equivalence notions for web services as well as the results on performance and implementation aspects are completely novel in this paper. Altogether this paper shall contribute to a better understanding and treatment of equivalence notions in service compositions.

Section 2 introduces 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 adressed in Section 4 followed by position equivalence in Section 5. Section 6 compares applicability of attribute equivalence with the one of position equivalence based on issues related to the evolution of service composition schemes. Section 7 provides the respective performance analysis and Section 8 illustrates how we practically support the service composition life cycle based on the service-aware process management technology AristaFlow. Section 9 discusses related work and Section 10 concludes with a summary.

2 Web Service Composition Life Cycle

Web service compositions are based on languages like WS-BPEL or BPMN and they enable the process-aware *orchestration* of the composed web services (Benatallah, Sheng & Dumas, 2003; Peltz, 2003; van der Aalst, 2003). The basic idea behind this is illustrated by Figure 1: At the service 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.

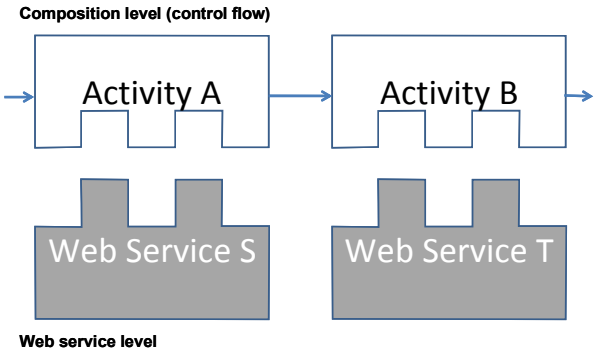


Figure 1: Web Service Compostion

This section sketches the life cycle of a web service composition. In particular, the specific challenges for *web service equivalence* within compositions can be discussed along the different phases of this 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.

Similar to business processes, web service compositions undergo a life cycle (Yang & Papazoglou, 2004) (cf. Figure 2). In the *design phase* the web service composition of interest is constructed resulting in a *web service composition schema*. As discussed in (Terai, Izumi & Yamaguchi, 2003), 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 (OMG, 2008). As example take web service composition schemes *S* and *S'* as depicted in Figure 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*).

Definition 1 (Composition Schema) *Let $\mathcal{W}$ be the set of all web services. Then a composition schema S is defined as tuple S=(N, E, D, DE) where*

- *N is the set of composition activities; for each $n \in N$ a corresponding web service $w \in \mathcal{W}$ is invoked at runtime when n is executed.*
- *$E \subseteq N \times N$ is the set of precedence relations setting out the order between activities*
- *D is the set of all composition data objects*
- *$DE \subseteq N \times E$ is the set of all data connectors (representing read and write accesses to data objects from D)*

The set of all web service composition schemata is denoted by $\mathcal{S}$.

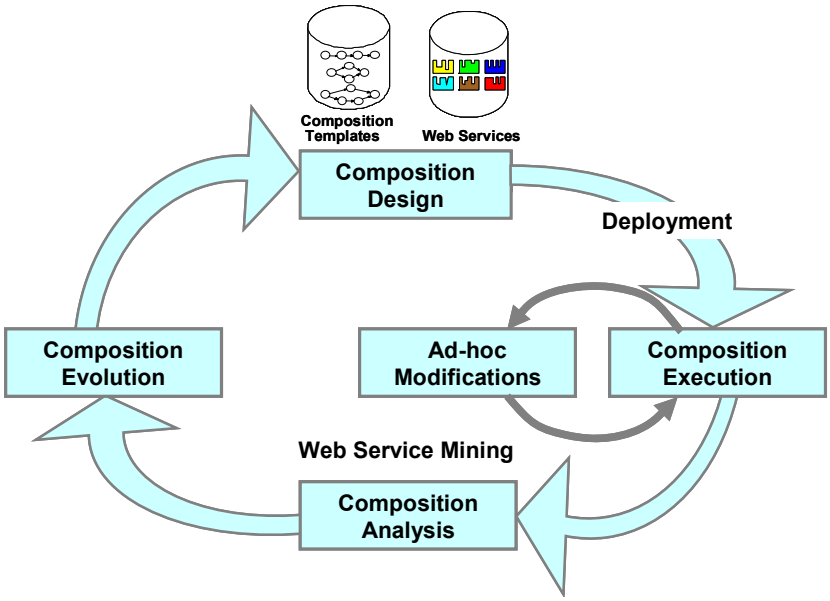


Figure 2: Web Service Composition Life Cycle

After the design of a composition, the resulting schema is deployed to a web service flow engine, like, for example, WebSphere Process Server (Kloppmann et al, 2004) or ADEPT2 Process Engine (Dadam et al, 2008; Dadam & Reichert, 2009). Usually, this includes the translation to an executable language such as WS-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 orchestrated and 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* (Ly, Rinderle & Dadam, 2008). Typically, execution traces log the start and end events of activity executions, possibly enriched by further information such as timestamps or actor assignments.

[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 (Ouyang, van der Aalst, Dumas & ter Hofstede, 2006)) are helpful in this context.

When using adaptive flow engines like ADEPT2 and AristaFlow BPM Suite respectively (Dadam & Reichert, 2009), the execution phase also enables *ad-hoc modifications* of composition instances (i.e., to structurally adapt the composition schema for one particular flow instance). Note that such instance-specific adaptations often become relevant in real-world scenarios (Mutschler, Reichert & Bumiller, 2008; Weber, Reichert & Rinderle-Ma, 2008) to deal with exceptional or changed situations. We refer to ad-hoc modified composition instances as *biased* instances in the following.

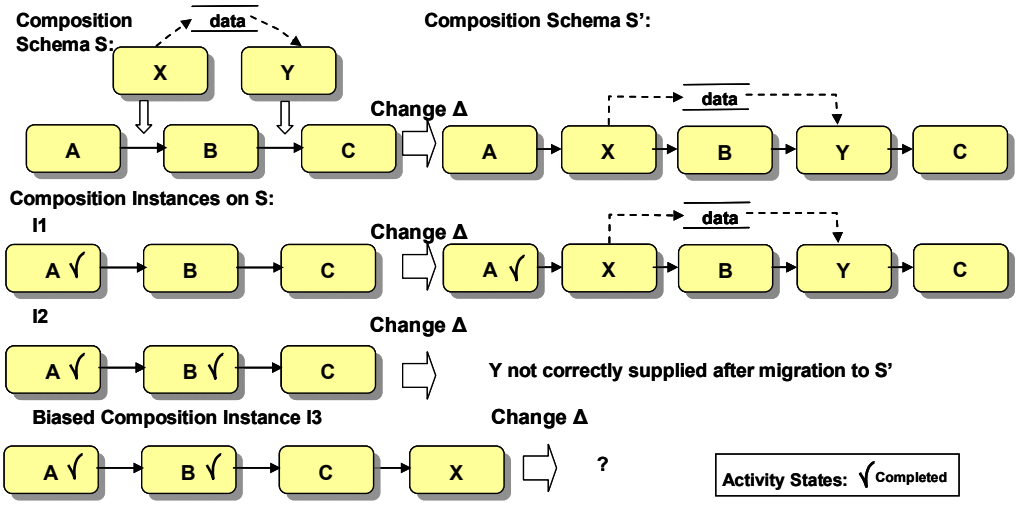


Figure 3: Evolution of Composition Schemas

To ensure continuous improvement of a web service composition during its life cycle (cf. Figure 2), its instances should steadily undergo a performance analysis. Similar to the analysis of workflows in process-aware information systems, relevant techniques comprise performance analysis and mining (Li, Reichert Wombacher, 2008a; van der Aalst & Verbeek, 2008; de Medeiros, van der Aalst & Pedrinaci, 2008). In the context of web service equivalence, particularly, flow mining is of high relevance. Mining techniques focus on deriving process structures (i.e., service composition schemes) from execution traces (*composition mining*) (van der Aalst & Verbeek, 2008), on checking different properties of composition instances (van der Aalst, de Beer & van Dongen, 2005), and on analyzing ad-hoc changes applied at instance level (*change mining*) (Günther et al, 2008; Li, Reichert Wombacher, 2009a). 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 the quality of composition schemas (e.g., by exterminating design flaws) (Weber, Reichert, Wild & Rinderle-Ma, 2009) and to discover a generic composition schema out of a collection of previously adapted composition instances (Li, Reichert & Wombacher, 2008a & 2009b).

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. Figure 3). One challenge is how to deal with already running composition instances (cf. Figure 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* while new ones are executed according to *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 (Rinderle, Reichert & Dadam, 2004; Rinderle, Reichert & Dadam, 2004a). 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 Figure 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 comprehensive framework for the evolution of business processes in (Rinderle, Reichert & Dadam, 2004; Rinderle, Reichert & Dadam, 2004a). As shown in (Reichert, Rinderle & Dadam, 2004; Rinderle, Wombacher, & Reichert, 2006), in principle, these concepts can be transferred to the evolution of web service compositions as well.

3 Composition design and execution: label equivalence

During composition design (cf. Figure 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 (de Medeiros, van der Aalst & Pedrinaci, 2008). In Figure 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 the web service labels themselves. However, without loss of generality, we can focus on *label equivalence* between composition activities:

Definition 2 (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 are specified, and let $\mathcal{W}^o$ be the set of web services which are invoked by any activity in $\mathcal{A}$. Then: Two web services w_1, $w_2 \in \mathcal{W}^o$ are called label-equivalent if $a_1.l = a_2.l$ where $a_1 \in \mathcal{A}$.is the activity invoking w_1 and $a_2 \in \mathcal{A}$. is the one invoking w_2 (a.l denotes the generic label attribute of activity a).*

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. Definition 2), the *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 Figure 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 associated with its specific *activity context*. Service *S* is then executed within the activity context of *A* (e.g., specifying which actors are allowed to process activity *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 (Bodenstaff, Reichert, Wombacher & Jaeger, 2008, 2009). Finally, composition activities have specific context attributes like duration or actor assignment (Rinderle-Ma & Reichert, 2008).

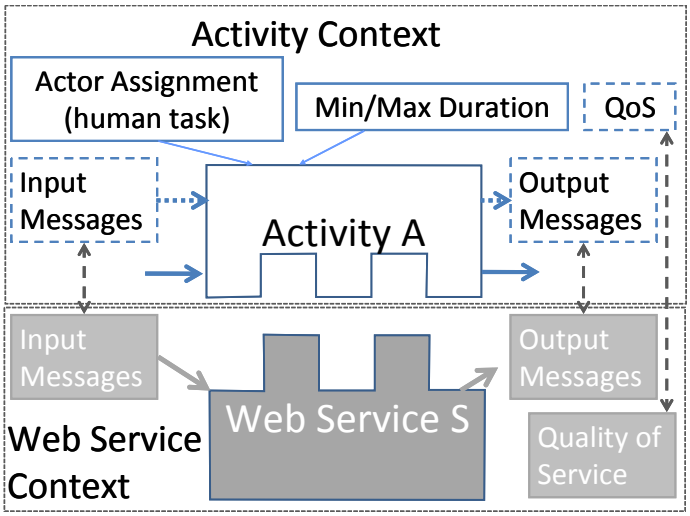


Figure 4: Plugging Web Service into Composition Activity Context

Definition 3 (Activity context) *Let $S = (N, E, D, DE)$ be a composition schema and let activity $a \in N$ invoke web service $w \in \mathcal{W}^{\circ}$ ($\mathcal{W}^{\circ}$ denotes the set of all web services). Then the activity context of a (denoted as a.ctxt) is defined as union of all attributes of a and all attributes of w (constituting the context of web service w). Note that in case there exists a mapping between web services attributes and activity attributes, only the latter are taken into consideration.*

Consequently, we distinguish the following three cases:

Definition 4 (Equivalence, synonyms, homonyms) *Let $w_1, w_2 \in \mathcal{W}^{\circ}$ 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. Equivalent semantics of two web services can be specified in different ways; e.g., on attribute or position equivalence as defined in the following sections, 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 composition schema changes or mining composition executions. Thus, if we know exactly that two web services are equivalent, synonymous or homonymous, we are able to 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 might have been not designed in a rigor manner (i.e., using a controlled vocabulary and ontology respectively). 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 schema changes and composition schema 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, based on a standard framework like OWL-S (OLW-S, 2004), is used; e.g., in combination with a service repository (similar to activity repositories in process-aware information systems (Dadam et al, 2008)). Still, there might be equivalent services occurring multiple times within a composition schema (Koehler & Vanhatalo, 2007). 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. Figure 2) multiple occurrences of a particular web service can be realized based on specific composition patterns like *multi instantiation* (van der Aalst, ter Hofstede, Kiepuszewski & Barros, 2003; Rinderle & Reichert, 2006). As described in (van der Aalst, ter Hofstede, Kiepuszewski & Barros, 2003; Rinderle & Reichert, 2006), 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 WS-BPEL 2.0 and BPMN 2.0. Note that when using multi-instantiation pattern, the multiple runtime occurrence of instances of the same activity is desired and therefore happens in a controlled manner. Specifically, in this case we can consider these multiple activity instances as being equivalent according to Definition 4 since they constitute some kind of copies of each other. Thus, no side-effects caused by unknown equivalent activities occur at runtime.

During the execution of a composition instance, structural **ad-hoc modifications** of its composition schema (Weber et al., 2009) might become necessary for this particular instance; 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. Definition 4) can be controlled as well; i.e., it is 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 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 completed composition instances; i.e., we look at mining techniques which construct the composition schema out of a given set of *composition instance traces* (traces for short). Informally, a trace contains all events happening during the execution of a composition instance; i.e., the start and end events of composition activities being executed in the run of the instances (van der Aalst & Verbeek, 2008).

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 (van der Aalst & Weijters, 2004). 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 Figure 5a where – at first glance – web service `sign` is used twice within web service composition *S*. Possible traces on *S* are σ1 and σ2 with

- σ1 = <`apply, **sign**, appoint, prepare, exam, **sign**, inform`>
- σ2 = <`apply, **sign**, prepare, appoint, exam, **sign**, inform`>

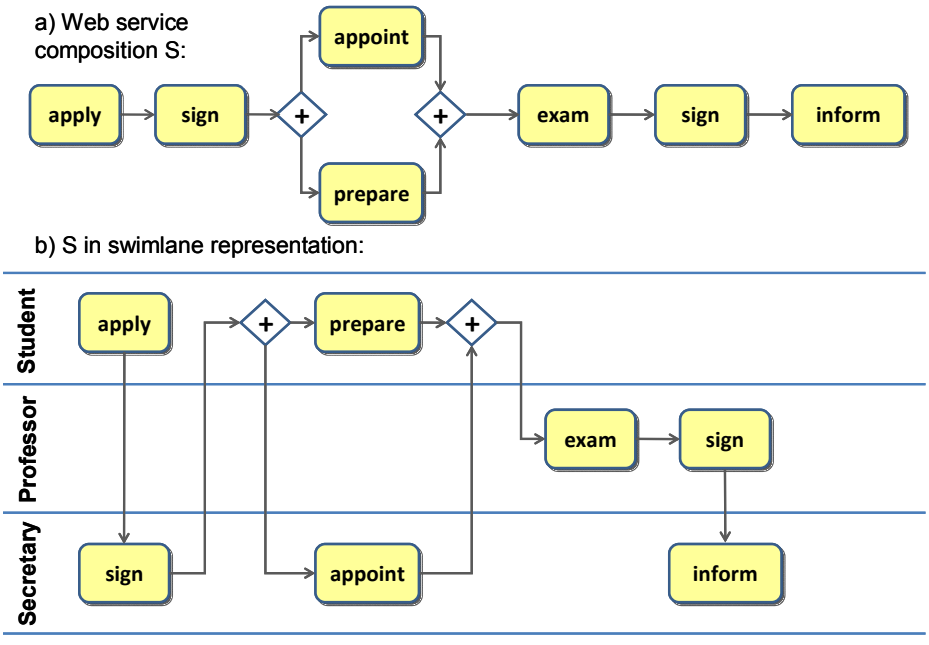


Figure 5: Example (in BPMN Notion)

Based on label information any mining algorithm counting frequencies of activity (label) occurrences within the traces would fail to detect the actual composition structure (i.e., schema). As can be seen from Figure 6, application of the Alpha++-Algorithm (de Medeiros, van Dongen, van der Aalst & Weijters, 2004), for example, leads to a completely different composition structure as depicted in Figure 5. This statement will be leveraged, if unique node identifiers are stored in addition to activity labels; e.g., for traces produced by the ADEPT2 process management system (Dadam et al, 2008) and mined by the ProM framework (Günther et al, 2008). 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)`.

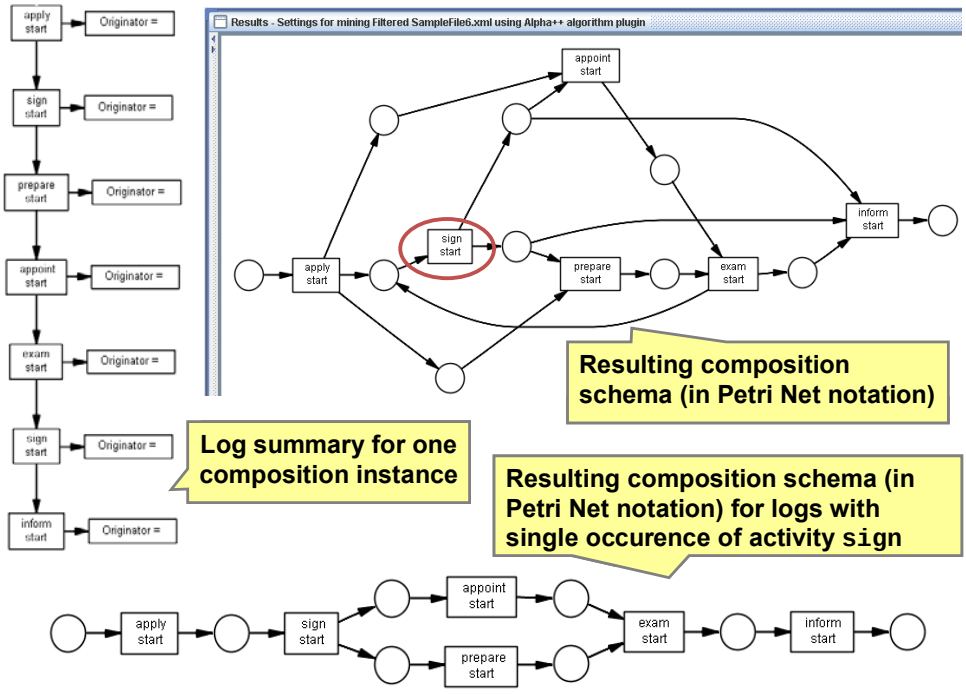


Figure 6: Application of α++-Algorithm to Example Logs, ProM Framework (PROM, 2009)

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 Figure 5 and assume that composition activities having label `sign` are both connected with the same web service. Still they can be distinguished by their composition activity context. Specifically, as can be seen from the swim lane representation in Figure 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`.

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 from Figure 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* which are relevant in the context of human activities; 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 behavior 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 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 (i.e., instances with modified composition schemes). – As example consider the evolution scenario depicted in Figure 7. Initially, two (biased) instances are running on composition schema *S*. Instance I_1 has been individually biased by inserting web service (activity) *X* between *B* and *C*. Instance I_2, in turn, has been modified by inserting *X* between *A* and *B*.

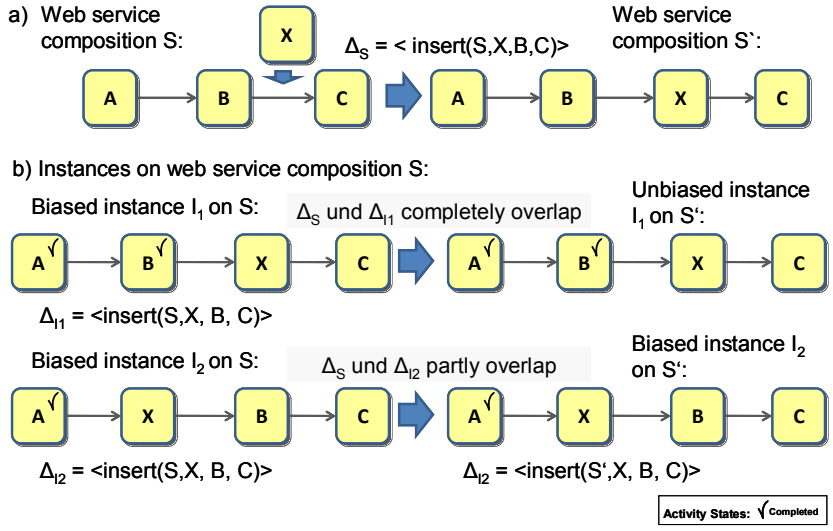


Figure 7: Evolution with Biased Instances

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 (Weber, Reichert & Rinderle-Ma, 2008; Weber et al., 2009; Li, Reichert & Wombacher 2009b). Consider again Figure 7. Assume, for example, that composition schema *S* is optimized by inserting new web service (activity) *X* between *B* and *C*. Then the challenge is to decide on an adequate strategy for migrating the already running instances I_1 and I_2.

In this scenario the changes of a composition schema and a composition instance *overlap* as typical when instances anticipate later schema optimizations. Assume, for example, that at instance level, again and again a certain activity is inserted in an ad-hoc manner. This can point to a process flaw and the insertion of this activity can be lifted up to composition schema level in order to "heal" this flaw. As a consequence all instances, for which the activity has been already inserted, possess an overlapping instance-specific bias when compared to the composition change. As can be seen from Figure 7, the *overlap degree* between instance-specific change Δ_{I1} and composition change Δ_S is different from the one between Δ_{I2} and Δ_S. Specifically, Δ_{I1} and Δ_S *completely overlap* whereas Δ_{I2} and Δ_S only *partly overlap*.

Figure 8 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 Figure 7, 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 (Rinderle, Reichert & Dadam, 2004b).

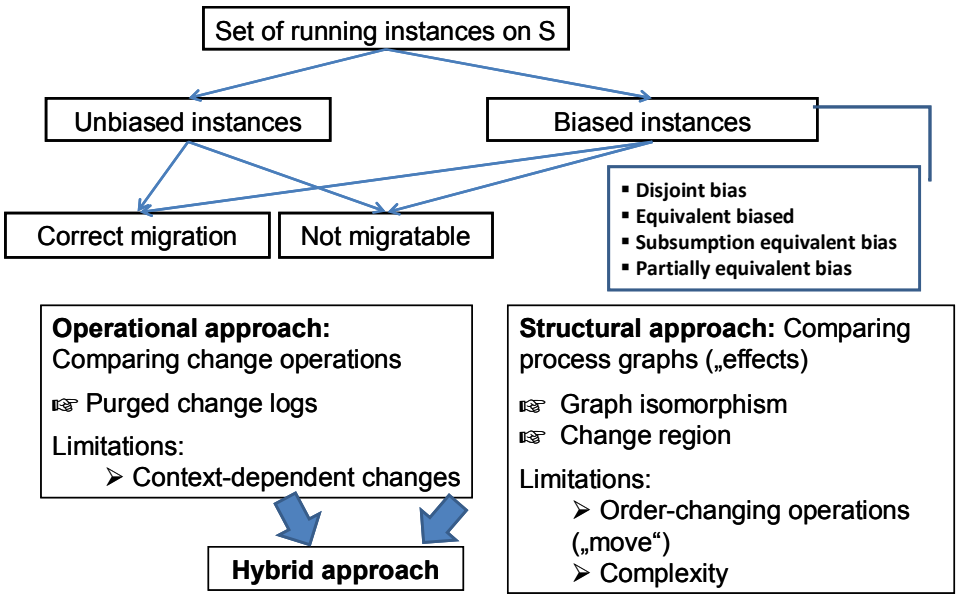


Figure 8: Possible Migration Strategies in the context of Composition Evolution

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 Definition 4, 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 process graphs. This leads us to another notion of equivalence for web services, which we denote as *position equivalence*. In Figure 7, for example, *X* is position-equivalent for *S* and I_1.

Basically, as described in Figure 8, we can decide on position equivalence of web services within composition graphs (i.e., composition schemes) in two ways: either the composition schema (s*tructural 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.

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, Δ) – can be determined as follows:*

```
PosIns(S, Δ) :=∅;
for all (X ∈ N_Δ^add) do
      find {(left, X), (X, right)} ∈ CtrlE';
      while (left ∈ N_Δ^add ∪ N_Δ^move) do
            find (leftleft, left) ∈ CtrlE';
            left = leftleft;
      od
      while (right ∈ N_Δ^add ∪ N_Δ^move) do
            find (right, rightright) ∈  CtrlE';
            right= rightright;
      od
      PosIns(S, Δ)=PosIns(S, Δ) ∪ {(left,X,right)};
od
```

Regarding Figure 7, we obtain the following position sets: $PosIns(S, \Delta_S)$ = {(B, X, C)}, $PosIns(S,\Delta_{I1})$ = {(B, X, C)} and $PosIns(S, \Delta_{I2})$ = {(A, X, B)}. Based on this and on the assumption that *X* is equivalent in all three cases, one can easily conclude that composition schema change Δ_S (at type level) and instance change Δ_{I1} are completely overlapping and Δ_S and instance change Δ_{I2} are partly overlapping.

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. As example consider Figure 9: 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

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

PosIns(S, Δ_S) = *PosIns(S, Δ_I)* = {(A,X,B), (A,Y,B)}. Based on this, Δ_S and Δ_{I1} can be correctly classified as completely overlapping.

If directly implemented as described above the complexity of Algorithm 1 is $O(m^2*n^2)$ for schema S and change Δ where m corresponds to the number of activities newly inserted by Δ and n corresponds to the number of activities contained in schema S. As we will show in Section 8, the use of a special implementation concept, the so called *delta layer* for storing composition changes leads to a significantly improved performance of Algorithm 1.

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 exactly this or that position, but within a certain *region* of the composition schema. Thus, in future work, we will relax the notion of position equivalence to *region equivalence*.

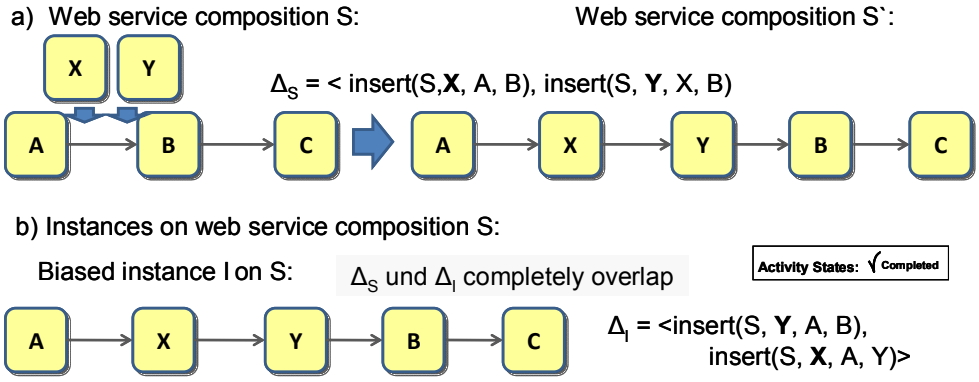


Figure 9: Evolution with multiple Occurrence of Equivalent Web Services

6 Discussion: Attribute vs. Position Equivalence

We evaluate the use of applying attribute equivalence and position equivalence respectively along the use case of *composition evolution* (Rinderle, Reichert & Dadam, 2004); i.e., adpatations of a composition schema at the type level that become necessary due to changes in laws, regulations or organizational rules. Composition evolution with its multifold interdependencies and outstanding practical impact yields the ideal "killer application" for advanced equivalence notions for web services being invoked within a composition.

6.1 Composition Evolution – Challenges and Objectives

As discussed in (Rinderle-Ma, Reichert & Weber, 2008) there are different objectives when evolving a composition schema and migrating already running composition instances to the new schema version. One is to preserve correctness of the composition schema and the composition instances at each point of time to ensure system robustness. In particular, this concerns composition instances which are migrated to the modified composition schema. Furthermore no composition instance should be needlessly excluded from being migrated to the new composition schema version; i.e., we want to migrate as many instances as possible. These objectives necessitate the existence of adequate correctness criteria for composition schema evolution. One prominent example is the *compliance criterion* as introduced in (Casati et al., 1998). Formally:

Definition 5 (Compliance) *Let S and S' be two composition schemas. Further let I be a composition instance running on S with trace* σ^I_S. *Then: I is compliant with S' iff* σ^I_S *can be replayed on S'; i.e., all events logged in* σ^I_S *could also have been produced by an instance on S' in the same order as set out by* σ^I_S.

In addition to these objectives it is crucial to provide means for efficiently checking on whether or nor changes of a composition schema may be propagated to running composition instances. This is particularly true for scenarios in which several thousands of composition instances are active at the same time (Rinderle, Reichert & Dadam, 2004). Thus existing approaches on process schema evolution have aimed at finding conditions based on which it can be quickly checked whether a particular instance can be correctly migrated to the changed process schema (e.g., by exploiting the semantics of the applied change operations) (Rinderle, Reichert & Dadam, 2004; Rinderle, Reichert & Dadam, 2004a). However, these approaches have been developed independently from the question of equivalence of the invoked activities and web services respectively.

Knowledge on the equivalence of composition activities becomes crucial in the context of overlapping changes at composition schema (i.e., type) and composition instance level (cf. Figure 8). Two changes are overlapping, for example, if they insert equivalent composition activities or refer to the same regions of the compositions schema at type and instance level. Consider Figure 10: Schema change Δ_S and instance change Δ_I overlap since they insert an activity at same position. The challenge is now to determine "how much" Δ_S and Δ_I overlap; i.e., if X is equivalent at composition schema (i.e., type) and composition instance level, Δ_S and Δ_I insert an equivalent activity at the same position and consequently they are equivalent (*complete overlap*). As opposed to this, if X is not equivalent for Δ_S and Δ_I, these changes insert a non-equivalent activity at the same position and thus are regarded as being partially equivalent. The distinction between equivalent and partially equivalent changes has significant effects on the migration strategy to be applied for composition instance I afterwards. Table 1 summarizes migration strategies for composition schema change Δ_S (transforming composition schema S into S') and composition instance change Δ_I (realizing an instance-specific ad-hoc change) in case both changes insert an activity X.

Degree of overlap Δ_S and Δ_I	**Checks for Insertion of Activity X[3]**	**Migration Strategy for I to S'**
Δ_S and Δ_I are *equivalent*	• X equivalent for Δ_S and Δ_S • X is inserted at the same position for Δ_S and Δ_S (using Algorithm 1) **→ X is attribute- and position-equivalent for Δ_S and Δ_S**	• I can be migrated to S' without any further correctness checks • After migration I is running based on S' • Δ_I becomes empty on S'
Δ_S and Δ_I are *partially equivalent*	• X is not equivalent for Δ_S and Δ_S, but is inserted at the same position for Δ_S and Δ_S **→ X is position-, but not attribute-equivalent for Δ_S and Δ_S** • X is equivalent for Δ_S and Δ_I, but inserted at different position **→ X is attribute-, but not position-equivalent for Δ_S and Δ_S**	• I can only be migrated to S' if I is compliant with S' (according to Definition 5 and further structural compliance conditions, see (Rinderle et al. 2004d) for details) • After migration I is running based on S' • Δ_I on S' is determined by Δ_I / Δ_S

Table 1: Degrees of Overlap between Composition Changes and Migration Strategies

[3] Other change operations (e.g., deleting or moving activities within service compositions) will be discussed at the end of Section 7).

6.2 On Utilizing Equivalence Notions for Determining the Degree of Overlap for Activity Insertions at Schema and Instance Level

As major conclusion from Table 1 we can draw that the position where X is inserted at schema and instance level is essential for determining the degree of overlap between schema change Δ_S and instance change Δ_I. If X is not position-equivalent, Δ_S and Δ_I can be directly rated as being partially equivalent. One might argue that the same holds for attribute equivalence. However, the difference here is that, first of all, attribute equivalence – contrary to position equivalence – can be determined in a "fine-tuned" way. Specifically, instead of checking equivalence for all attributes of X at schema and instance level, the user might want to consider only a subset of attributes or even decide that attribute equivalence is not important at all. Secondly, attribute equivalence can be ignored causing "bearable effects" whereas position equivalence cannot be ignored.

The following example shows why attribute equivalence can be neglected, but position equivalence cannot: Consider Figure 10. Obviously, X is inserted at same position at schema and instance level and thus is rated as position-equivalent. At this point in time we already know that Δ_S and Δ_I are at least partially equivalent in any case.

Case 1 (Neglecting attribute equivalence). In this case schema change Δ_S and instance change Δ_I are classified as equivalent solely based on position equivalence of activity X. According to the migration strategies from Table 1, composition instance `I` can be correctly migrated to the changed composition schema S' without further checks. After migration of instance `I`, attribute `maxDur` = 6 will be overwritten at instance level to `maxDur` = 5 (cf. Figure 10 ❶). Reason is that after migration of instances with equivalent bias, the respective instance is actually running based on S' without any further bias, i.e., Δ_I becomes empty.

Case 2 (Taking attribute equivalence into consideration). X is not attribute-equivalent for Δ_S and Δ_I due to the varying values of attribute `maxDur` at schema and instance level. Thus, Δ_S and Δ_I are considered as being partially equivalent (cf. Table 1). In this case, the migration of composition instance `I` to the modified composition schema S' has to be rejected by the system since instance `I` is not compliant with S' anymore: within the trace of `I` an end entry of activity `X` has been already written with annotation `maxDur` = 6 for the respective attribute. This event cannot be reproduced on composition schema S'. Hence, according to the compliance criterion (cf. Definition 5), `I` cannot be correctly migrated to S'. The compliance criterion might be bypassed by migrating `I` to S' and storing an instance-specific bias for `I` on S' which reflects the different attribute value for `I` when compared to S' (cf. Figure 10 ❷). Respective relaxations for compliance are discussed in (Rinderle & Reichert, 2008), but are outside the scope of this paper.

Altogether, from the above example, we see that neglecting attribute equivalence might result in overwriting attributes at instance level, but possibly enables a higher number of composition instances to migrate to the new composition schema version. This constitutes an important objective in the context of composition evolution. Contrary, neglecting position equivalence is not possible at all for determining overlap degrees between composition schemas. In summary, position equivalence is necessary and less restrictive than attribute equivalence in respect to the number of compliant instances. However, if a precise analysis of activity attributes is desired for instance migration, position equivalence can be combined with attribute equivalence. For example, composition designers might want to specify a subset of sensitive attributes for which their instance-specific deviation must not be overwritten by a schema attributes. This decision could be also specified in a semi-automated way, for example, by introducing priorities for different attributes.

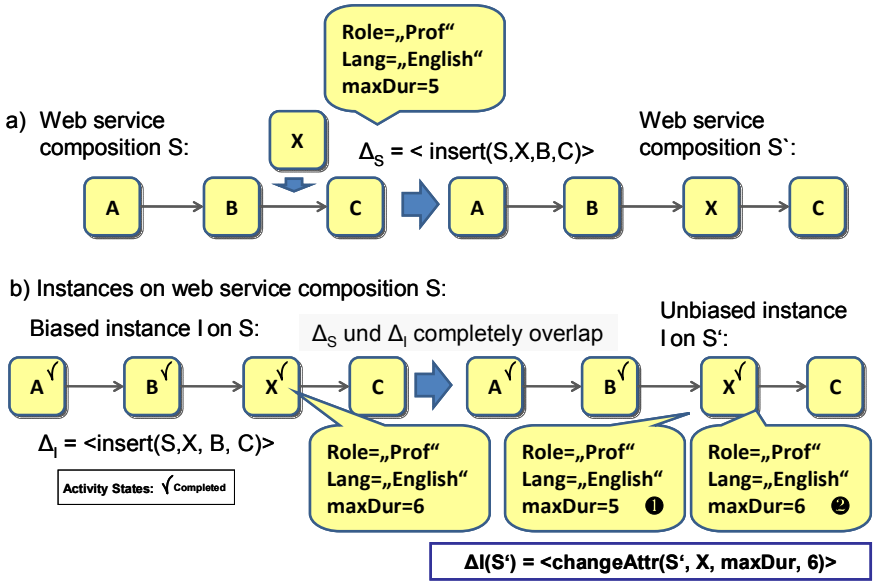


Figure 10: Degree of Overlap depends on Equivalence Notion

7 Performance Issues

In this section we have a closer look at performance issues in the context of checking attribute and position equivalence. Again we provide a detailed analysis for the insertion of activities into compositions. A discussion on further change operations can be found at the end of this section.

7.1 Single Insertion of Activities

We start with single insertion of activities at schema and instance level since for single insertions positions as determined by Algorithm 1 are always unique. The more complex case of multiple activity insertions is elaborated on in Section 7.2.

Case1 (Checking attribute equivalence (without position equivalence)). Consider again the example depicted in Figure 10. Preliminarily, we assume that schema change Δ_S takes place after instance change Δ_I. Consequently, when Δ_S inserts X into S we have to check instance schema $S_I := S + \Delta_I$ for activities which are attribute-equivalent to X. In more detail, this means that we have to compare the attributes of X with the ones of all activities contained in S_I. For activities $a^I_i \in N_I$ ($S^I = (N_I, E_I, \ldots)$) this results in $\Sigma_{i=1,\ldots,|N_I|}$ ($|X.ctxt| * |a^I_i.ctxt|$) comparisons (X.ctxt denotes the set of attributes associated with activity X, cf. Definition 3). Assuming three attributes for each activity of S_I (cf. Figure 10), checking attribute equivalence results in 36 comparisons. However, in realistic scenarios the number of comparisons might increase significantly when checking attribute equivalence; e.g., in the automotive domain there are composition schemes consisting of several thousands of activities of which each has a large number of attributes (Müller, Reichert & Herbst. 2007).

Case2 (Checking attribute equivalence by utilizing position equivalence). When utilizing position equivalence, the number of comparisons for attribute equivalence is reduced to |X.ctxt| if no fine-

tuning by the composition designer is desired (or, if the designer wants to consider only a subset $A^X \subseteq X.ctxt$ of the attributes of X, the number of comparisons turns out as $|A^X|$). However, the effort of applying Algorithm 1 to decide on position equivalence has to be taken into account as well. The complexity of the "naive" implementation as given in Section 5 can be optimized significantly by the implementation concepts we present in Section 8.

7.2 Multiple Insertions of Activities

When inserting multiple activities we have to distinguish the following cases: (1) multiple activities are inserted at different positions and (2) multiple activities are inserted "in one row".

Consider Figure 11: Schema change Δ_S (on the top of Figure 11) and instance change Δ_I respectively (at the left on the bottom of Figure 11) insert multiple activities at different positions into S. Applying Algorithm 1 leads to the corresponding positions. Note that Algorithm 1 currently works with unique predecessor and successor nodes for activities, which is realized by using specific structuring nodes like AND-Split, AND-Join, XOR-Split, or XOR-Join. Such structuring nodes are supported by composition modeling languages like BPMN and composition execution languages like WS-BPEL. Regarding schema S_I in Figure 11 it can be seen that all newly added activities have been inserted at different positions. Thus the positions as contained in sets PosIns(S, Δ_S) and PosIns(S, Δ_I) (cf. Figure 11) are unique for each activity. For the given scenario we can conclude that – if we abstract from attribute equivalence – PosIns(S, Δ_S) and PosIns(S, Δ_I) have to be compared in order to state that Δ_S and Δ_I are equivalent with respect to the insertion of activity X. Figure 11 (at the right on the bottom) also shows the result we obtain when migrating composition instance `I` to the new composition schema version S' (assuming that X is being rated equivalent in the context of Δ_S and Δ_I).

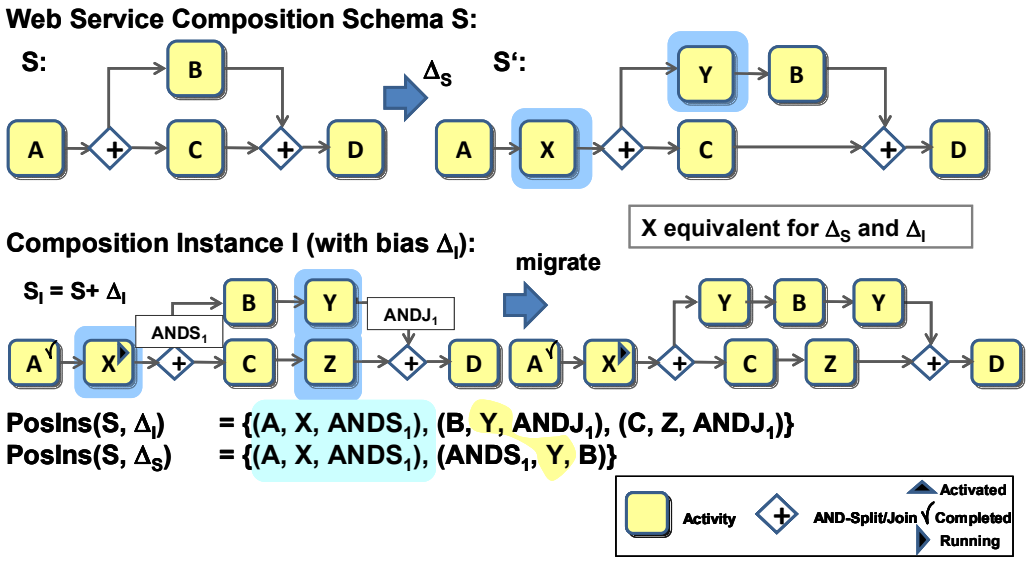


Figure 11: Position Equivalence for Insertion of Multiple Activities

An example for the insertion of multiple activities "in one row" (2) is depicted in Figure 9. Algorithm 1 determines the following position sets for Δ_S and Δ_I respectively: PosIns(S, Δ_S) =

{(A,X,B), (A,Y,B)} and PosIns(S, Δ_I) = {(A,X,B), (A,Y,B)}. Obviously, the positions for the newly inserted activities are not unique. In this case position equivalence can be augmented by attribute equivalence for activities X and Y at schema and instance level. Altogether, for the insertion of multiple activities "in one row" position equivalence as determined by Algorithm 1 yields subsets of activities for which attribute equivalence can be checked afterwards.

7.3 Other Change Operations

The reason why equivalence notions become necessary in the context of insert operations is that we might not have any knowledge on the activities to be inserted. If schema and instance have to checked on their overlap, equivalence notions can be used in order to decide on the relation between activities and related insertion positions, and thus on the overlap degree between schema and instance changes. As opposed to activity insertions, for **delete** operations, equivalence notions are not primarily used for deciding on the degree of overlap between changes at schema level and instance level, but become necessary in order to be able to specify the delete operations in a unique way. To support this statement, first of all, different cases have to be distinguished:

(a) Deletion of activities with single occurrence at schema and instance level can be handled without any further equivalence analysis. Assume that activity `C` occurs in a single way in schema S (cf. Figure 12). Then activity `C` is also unique for all composition instances started on S. If `C` is deleted at schema and instance level (expressed by change `<delete(S, C)>`) the respective changes can be rated as equivalent to each other.

(b) Assume now that activities occur in a multiple way, for example, activity `B` occurs twice at schema and instance level. Then change operation `<delete(S, B)>` is not unique since it is not clear to which `B` the change refers. For multiple occurrences of activities at different positions, as it is the case for composition schema S in Figure 12, deleting `B` can be made unique based on position equivalence again. Respective delete operations are Δ_S = `<delete(S, B, [, A, C])>` and Δ_{I1} = `<delete(S, B, [,C, D])>`. (The first delete operation refers to activity `B` positioned between activities `A` and `C`, and the second one to activity `B` positioned between `C` and `D`.)

(c) Finally, for multiple occurring activities "in one row" specifying positions is not unique. Additionally addressing the activities to be deleted by utilizing attribute equivalence is another way to achieve uniqueness. Finally, if two position- and attribute-equivalent activities are to be deleted, the delete operations might be finally conducted by specifying orders on these activities (*order equivalence*). Due to lack of space we do not elaborate on this aspect in the following.

Move operations (i.e., to shift an activity from its current position to a new one within a composition schema) can be logically considered as combination of delete and insert operations. If an activity X is moved from its current position pos_c to another position pos_n within a composition schema, we can express this operation as a sequence of deleting X first and inserting X at pos_n afterwards. For "deleting" X at pos_c the above considerations on the delete operation hold. For "inserting" X afterwards at pos_n, there is a slight difference to insert operations as discussed in Section 7.2. X is not a "black-box" for the composition since X has already been part of the schema. However, regarding new position $pos_{n,}$ position equivalence is useful to compare the target of two move operations at schema and instance level.

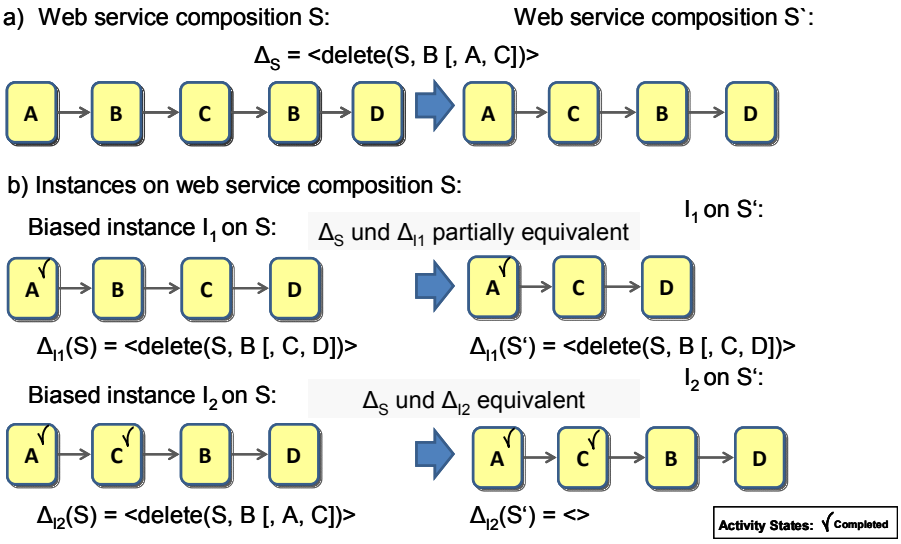


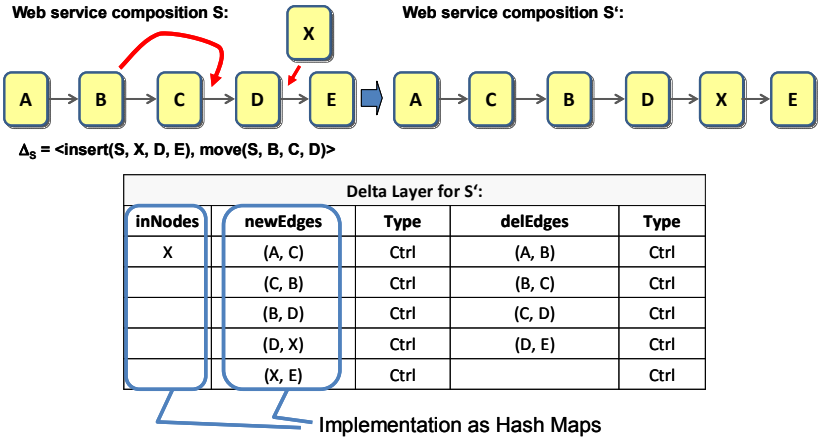
Figure 12: Deleting Activities with Multiple Occurrence at Schema and Instance Level

8 Implementation Issues

Position equivalence has many advantages when comparing it with checking attribute equivalence. For example, it enables us to quickly determine candidates for equivalence with the option to precisely specify relevant attributes and to abstract from insignificant ones. Aside these more qualitative considerations, in this section we dig deeper into implementation issues. In Section 7, performance considerations for checking attribute and position equivalence have been presented. One open issue was how to implement Algorithm 1 in such a way that its performance is significantly improved. Recall that the "naive" implementation of Algorithm 1 leads to complexity $O(m^2*n^2)$ where m corresponds to the number of newly inserted activities and n to the number of activities already contained within the composition schema of interest.

The key factor to reduce complexity of Algorithm 1 is to optimize the implementation of the used sets; i.e., the sets of newly inserted activities N_Δ^{add} and moved activities N_Δ^{move}. In (Rinderle, Reichert, Jurisch & Kreher, 2006; Rinderle, Jurisch & Reichert, 2007) we have provided a compact representation form for these sets, the so called *delta layer*, which can be also applied in the given context. Consider Figure 14: Composition change Δ_S is represented by a delta layer storing the differences between original schema S and transformed schema S'. The delta layer is constructed in such a way that changes being overwritten are automatically purged. Assume, for example, that activity B is first moved to a position between activities C and D and afterwards moved to a position between activities D and E. Then the delta layer automatically overwrites the outdated information; i.e., it overwrites the change caused by the first move operation whose effects are no longer visible within the composition schema.

For newly inserted activity X an entry is stored within the set of inserted nodes whereas for moved activity B only inserted and deleted control edges are stored (cf. Figure 14). The sets themselves are implemented as hash maps. Following this approach, finding entries within each of the hash maps has expected complexity of *O(1)* (Cormen, Leiserson, Rivest & Stein, 2001). Thus the complexity of Algorithm 1 can be reduced to *O(m*n)* in total. For supporting this



Delta Layer for S':

inNodes	newEdges	Type	delEdges	Type
X	(A, C)	Ctrl	(A, B)	Ctrl
	(C, B)	Ctrl	(B, C)	Ctrl
	(B, D)	Ctrl	(C, D)	Ctrl
	(D, X)	Ctrl	(D, E)	Ctrl
	(X, E)	Ctrl		Ctrl

Implementation as Hash Maps

Figure 14: Implementation as Delta Layer

statement look again at Algorithm 1: the outer loop iterates on the *m* entries of N_Δ^{add}. For the first `find` statement the set of control edges in *S'* has to be searched resulting in *O(n)* worst case where *n* is the number of activities in composition schema *S'*; note that in our AristaFlow system composition schemes are represented as serial-parallel, not fully-connected graphs (cf. Rinderle, Reichert & Dadam, 2004). The first inner loop iterates over all newly inserted or moved activities. Since the respective sets are implemented as hash map the resulting complexity results in O(1). Inside this loop we get a complexity of *O(n)* again for searching the set of control edges of *S'*. The same holds for the second loop. Altogether this results in a complexity of *O(m*n)*.

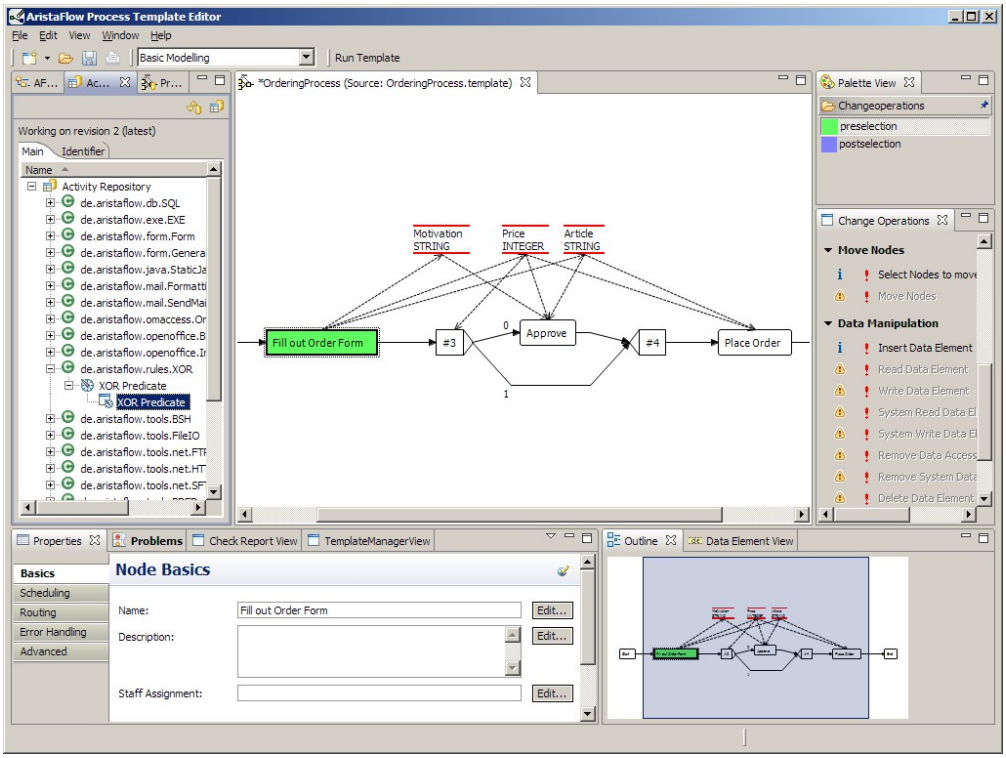


Figure 13: On Building Web Service Compositions in AristaFlow

Figure 13 shows an example of how to build web service compositions using the AristaFlow process management system[4]. Here, activities and services respectively can be composed and be connected in the desired execution order. To each activity a web service or general application component (e.g., a database application, a user form or another executable) can be assigned in a "drag & drop-like" fashion, and be invoked when exectuting corresponding composition instances. In addition to control flow, the composition schema also describes the message flow between the activities. Recall that input and output messages constitute activity attributes which can be used as basis for attribute equivalence. Another possibility for specifying relevant attributes in the context of activity execution is depicted in Figure . Using this dialogue, staff assignment rules for composition activities can be defined which control the users authorized to work on certain activities at runtime.

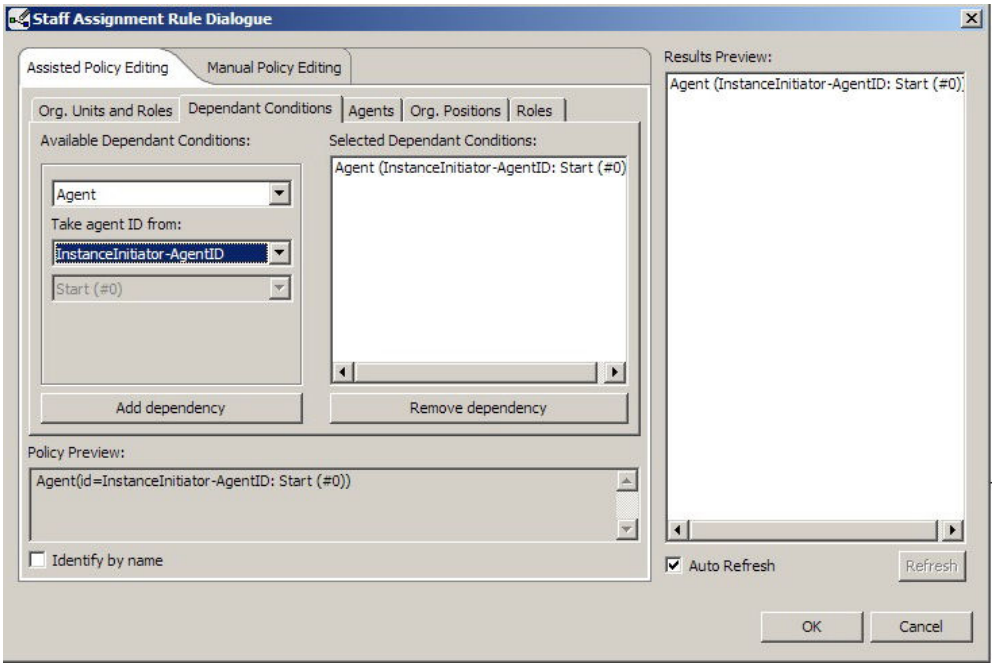


Figure 14: On Defining Staff Assignment Rules in AristaFlow

Figure 13 and Figure 14 show examples for composition design and execution phase of the composition life cycle. The AristaFlow process management system also provides comprehensive support for composition changes and evolution. Due to lack of space we refer to (Dadam et al, 2008) for further details here.

9 Related Work

We have already refered 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; e.g., (Rahm & Bernstein, 2001)). Obviously, there are similarities to the equivalence of web services as described in this paper. In both cases,

[4] AristaFlow is an industrial-strength BPM technology that has been implemented based on the results achieved within the ADEPT1 and the ADEPT2 research projects (www.aristaflow.com)

it might become 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 than data, which might become relevant for equivalence checks: 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 (OWL-S, 2004), these approaches apply label and/or attribute equivalence (Shua, Ranab, Avisb & Dingfanga, 2006). 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.

Various papers have studied similarity and equivalence issues that can be related to composition schemes as well. In graph theory, for example, graph isomorphism (Tan, Steinbach, Kumar, 2005) is used to measure equivalence or similarity between two graphs. Basically, corresponding techniques examine edges and nodes of a graph, but cannot deal with many of the issues being relevant in the context of composition schemes (e.g., guaranteeing soundness of a composition schema or distinguishing between AND- and XOR-splits). Algorithms for measuring tree edit distances (Bille, 2005) show similar drawbacks, i.e., syntactical issues being relevant for the composition life cycle are missing.

In the database field, the delta algorithm (Labio & Garcia-Molina, 1996) is suggested to measure the difference between two models. However, it only considers change primitives (i.e., node/edge insertions and deletions respectively), and is unable to cope with high-level change operations (e.g., to insert activities into a composition schema or to delete activities from it). Regarding Petri-nets and state automata, similarity of two models based on the changes needed to transform one model into another is difficult to measure since these formalisms are not very tolerant in respect to changes. Inheritance rules (van der Aalst & Basten, 2002) are one possible technique for reasoning about transformations between process models described in terms of a Petri-net. – Opposed to these approaches, this paper does not consider structural or behavioral similarity of composition schemes, but focusses on equivalence notions related to single web services.

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 (Wombacher & Martens, 2007). Trace equivalence is commonly used to compare whether two process models (i.e., composition schemes) are similar or identical (Hidders et al., 2005). Bisimulation further extends trace equivalence by providing a stronger equivalence notion (van der Aalst & Basten, 2002; van Glabbeek & Weijland, 1996). The approach presented in (van der Aalst, de Medeiros & Weijters, 2006) assigns weights to each trace based on execution logs which reflect the importance of a certain trace. The edit distance (Wombacher & Rozie, 2006) is also used to measure the difference between traces; the sum of them represents the differences of two process models. Some similarity measures use *precision* and *recall* to evaluate the difference between two process schemes (van der Aalst, de Medeiros & Weijters, 2006; Pinter & Golani, 2004). Finally, (Li, Reichert & Wombacher, 2008b) measure similarity between two composition schemes based on the effort (i.e., high-level changes) needed for transforming the one schema into the other. – Specifically, all these approaches allow to decide whether two composition schemes are structurally equivalent or similar, or whether they reflect same process behavior. By contrast, this has focused on equivalence notions between single web services. However, 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.

In general, matchmaking between web services (Wombacher, A., Mahleko, B. & Neuhold, E., 2004; Field, S. & Hoffner, Y., 2003) helps to find web services offering a specific functionality

requested by another web service. Obviously, one important challenge in this context is to describe the functionality of the web services in such a way that matchmaking can be conducted in a (semi-)automatic way. For this purpose different techniques have been proposed (e.g., ontologies being specified in OWL-S). The approaches supporting matchmaking of web services can be seen as orthogonal to the equivalence notions presented in this paper. Equivalence notions can make use of web service descriptions for supporting, for instance, attribute equivalence. The other way round, matchmaking could be supported by exploiting knowledge on equivalence relations between web services.

Finally, the presented equivalence notions are complementary to the contributions made by semantic web services. In this context, the Web Service Modeling Ontology (WSMO) provides a meta model for the Web Service Modeling Language (Wang, Gibbins, Payne, Saleh, & Sun, 2007); i.e., it defines a formal language which enables semantic descriptions of all relevant aspects of web services. Simlarly, OWL-S (OWL-S, 2004) provides an ontology for describing services for the purpose of discovery, composition and delivery. In OWL-S a service is formally described by a *Service Model* that defines the steps required to execute the service. Obviously, the use of an ontology also fosters the application of the discussed label equivalence.

10 Summary and Outlook

We discussed requirements for equivalence notions of web services within compositions along all 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 becomes particularly important in the context of 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 should 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 become necessary for more complex evolution scenarios.

REFERENCES

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

Bille, P. (2005) . A survey on tree edit distance and related problems. *Theoretical Computer Science,* 337(1-3), 217-239.

Bodenstaff, L., Wombacher, A., Reichert, M. & Jaeger, M. (2009) Analyzing impact factors on composite services. In: IEEE, 6th Int'l Conf. on Services Computing (SCC'09), 218-226.

Bodenstaff, L., Wombacher, A., Reichert, M. & Jaeger, M. (2008). Monitoring dependencies for SLAs: the MoDe4SLA approach. In: IEEE 5th Int'l Conference on Services Computing (SCC 2008), 21-29.

Casati, F., Ceri, S., Pernici, B. & Pozzi, G (1998). Workflow evolution. Data and Knowledge Engineering, 24(3), 211–238.

Cormen, T.H., Leiserson, C.E., Rivest, R.L. & C. Stein: Introduction to algorithms, 2nd edition, MIT Press.

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

Dadam, P. & Reichert, M. (2009) The ADEPT project: a decade of research and development for robust and flexible process support - challenges and achievements. Computer Science - Research and Development, 23(2), 81-97.

De Medeiros, A., van der Aalst, W. & Pedrinaci, C. (2008). Semantic process mining tools: Core building blocks. In: 16th European Conference on Information Systems.

De Medeiros, A., van Dongen, B.F. , van der Aalst, W.M.P. & Weijters, A.J.M.M (2004) Process mining: extending the alpha-algorithm to mine short loops. BETA Working Paper Series, WP 113, Eindhoven University of Technology, The Netherlands

Field, S. & Hoffner, Y. (2003) Web services and matchmaking. Intl. Journal of Networking and Virtual Organisations 1(3), 16-32

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

Hidders, J., Dumas, M., van der Aalst, W., ter Hofstede, A. & Verelst, J. (2005). When are two workflows the same? In: *CATS '05,*, 3-11

Kloppmann, M., König, D., Leymann, F., Pfau, G. & Roller, D. (2004) Business process choreography in WebSphere: Combining the power of BPEL and J2EE. IBM Systems Journal 43(2), 270-296.

Koehler, J. & Vanhatalo, J. (2007) Process anti-patterns: How to avoid the common traps of business process modeling, Part 1. IBM WebSphere Developer Technical Journal.

Labio, W. & Garcia-Molina, H. (1996). Efficient snapshot differential algorithms for data warehousing. In: Proc. VLDB '96, San Francisco, USA, 63-74

Li, C., Reichert, M. & Wombacher, A. (2009a) What are the problem makers: ranking activities according to their relevance for process changes. In: IEEE 7th Int'l Conference on Web Services (ICWS'09), 51-58

Li, C., Reichert, M. & Wombacher, A. (2009b) Discovering reference models by mining process variants using a heuristic approach. In: 7th Int'l Conference on Business Process Management (BPM'09), LNCS 5701, 344-362

Li, C., Reichert, M. & Wombacher, A. (2008a) Discovering reference process models by mining process variants. In: 6th Int'l Conference on Web Services (ICWS'08), 45–53.

Li, C., Reichert, M. & Wombacher, A. (2008b) On measuring process model similarity based on high-level change operations. In: 27th Int'l Conference on Conceptual Modeling (ER'08), LNCS 5231, 248–264.

Ly, L.T., Rinderle, S. & Dadam, P. (2008) Integration and verification of semantic constraints in adaptive process management systems. Data and Knowledge Engineering, 64(1), 3–23.

Müller, D., Reichert, M. & Herbst, J. (2007) Data-driven modeling and coordination of large process structures. In: 15th Int'l Conf. on Cooperative Information Systems (CoopIS'07), Springer, LNCS 4803, 131-149.

Mutschler, B., Reichert, M. & Bumiller, J. (2008) 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.

OMG. Business Process Modeling Notation (2008), V1.1, OMG document number: formal/2008-01-17 edition.

Ouyang, C., van der Aalst, W., Dumas, M. & ter Hofstede, A. (2006) Translating BPMN to BPEL. Technical Report BPM-06-02, BPM Center.

OWL-S (2004). http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

Peltz, C. (2003) Web services orchestration and choreography. Computer, 36(10), 46–52.

Pinter, S.S. & Golani, M. (2004) Discovering workflow models from activities' lifespans. *Comput. Industry*, 53(3), 283-296.

ProM (2009). http://prom.win.tue.nl/tools/prom/

Rahm, E. & Bernstein, P. (2001). A survey of approaches to automatic schema matching. VLDB Journal, 10(4), 334–350.

Reichert, M., Rinderle, S. & Dadam, P (2004). On the modeling of correct service flows with BPEL4WS. In: Workshop on Enterprise Modeling and Information Systems Architecture (EMISA'04), LNI P-56, 117–128.

Rinderle, S. (2004) Schema evolution in process management systems. PhD thesis, Ulm University.

Rinderle-Ma, S. & Reichert, M. (2008) Managing the life cycle of access rules in CEOSIS. In: 12th IEEE Int'l Enterprise Computer Conference (EDOC'08), 257-266.

Rinderle, S. & Reichert, M. (2006) Data-driven process control and exception handling in process management systems. In: 18th Int'l Conference on Advanced Information Systems Engineering (CAiSE'06), LNCS 4001, 273–287.

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

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

Rinderle, S., Reichert, M. & Dadam, P. (2004b) Disjoint and overlapping process changes: challenges, solutions, applications. In: 11th Int'l Conference on Cooperative Information Systems (CoopIS'04), LNCS 3290, 101–120.

Rinderle-Ma, S., Reichert, M. & Jurisch, M. (2009) Equivalence of web services in process-aware service compositions. In: IEEE 7th Int'l Conference on Web Services (ICWS'09), July 2009, 501-508.

Rinderle-Ma, S., Reichert, M. & Weber, B. (2008) Relaxed compliance notions in adaptive process management systems. In: 27th Int'l Conference on Conceptual Modeling (ER'08), LNCS 5231, 232–247.

Rinderle, S., Wombacher, A. & Reichert, M. (2006) Evolution of process choreographies in DYCHOR. In: 14th Int'l Conference on Cooperative Information Systems, LNCS 4275, 273–290.

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

Tan, P.N., Steinbach, M. & Kumar, V. (2005) Introduction to data mining. Addison-Wesley.

Terai, K., Izumi, N. & Yamaguchi, T. (2003) Coordinating web services based on business models. In: Int'l Conference on Electronic Commerce, 473–478.

van der Aalst, W. (2003) Don't go with the flow: Web services composition standards exposed. IEEE Intelligent Systems.

van der Aalst, W. & Basten, T (2002). Inheritance of workflow: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125-203.

van der Aalst, W., de Medeiros, A. & Weijters, A. (2006) Process equivalence: comparing two process models based on observed behavior. In: 4th Int'l Conference on Business Process Management (BPM'06), LNCS 4102, 129–144.

van der Aalst, W., de Beer, H.T. & van Dongen, B.F. (2005) Process mining and verification of properties: an approach based on temporal logic. In: 13th Int'l Conference on Cooperative Information Systems (CoopIS'05), LNCS 3760, 130-147.

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

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

van der Aalst, W. & Weijters, A. (2004) Process mining: a research agenda. Computers in Industry, 53(3), 231-244.

van Glabbeek, R.J. & Weijland, W.P (1996). Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3), 555-600.

Wang, H.H., Gibbins, N., Payne, T.R. Saleh, A. & Sun, J. (2007) A formal semantic model of the semantic web service ontology (WSMO). In: ICECCS'07, 74-86

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

Weber, B., Reichert, M., Wild, W. & Rinderle-Ma, S. (2009) Providing integrated life cycle support in process-aware information systems. Int'l Journal of Cooperative Informtion Systems, 18(1), 115-165.

Wombacher, A., Mahleko, B. & Neuhold, E. (2004) IPSI-PF: A business process matchmaking engine. In: Int'l Conference. on Electronic Commerce, 137-145.

Wombacher, A. & Martens, A. (2007) Soundness and equivalence of Petri Nets and annotated finite state automate. In: IEEE Int'l Conference Digital Ecosystems and Technologies.

Wombacher, A. & Rozie, M. (2006) Evaluation of workflow similarity measures in service discovery. Workshop Service Oriented Electronic Commerce, LNI P-80, 51-71.

Yang, J. & Papazoglou, M. (2004) Service components for managing the life-cycle of service compositions. Information Systems, 29(2),97–125

International Journal of Web Services Research, 8(1):41-67, 2011

ABOUT THE AUTHORS

Stefanie Rinderle-Ma obtained her Ph.D. and her Habilitation in Computer Science at the University of Ulm, Germany. Since January 2010 she has been appointed as full professor at the University of Vienna (Austria) where she is head of the Workflow Systems and Technology Group. During her postdoc period, Stefanie stayed at the Universities of Twente (The Netherlands), Ottawa (Canada), and Eindhoven (The Netherlands) where she worked on several projects on process visualization and modeling as well as on process mining. Her research interests include adaptive processes and services, integration of semantical constraints with process executions, business process compliance, and the controlled evolution of organizational constraints. Stefanie co-organized several international workshops and was General Workshop Chair of the BPM'09 workshops in Ulm.

Manfred Reichert holds a PhD in Computer Science and a Diploma in Mathematics. Since January 2008 he has been appointed as full professor at the University of Ulm, Germany, where he is co-director of the Institute of Databases and Information Systems. Before he was working as Associate Professor at the University of Twente (UT), where he was leader of the strategic research orientations on e-health and on service-oriented architectures. His major research interests are next generation process and service management technology (e.g., adaptive services, service lifecycle management, process visualization, data-driven services), service-oriented architectures (e.g., service interoperability, service evolution), and advanced applications for ICT solutions (e.g., e-health, automotive engineering). Together with Peter Dadam he pioneered the work on the ADEPT process management system. Manfred has been participating in numerous research projects in the BPM area and contributed numerous papers. Further, he has co-organized international and national conferences and workshops. In particular, Manfred was PC-Co-Chair of the BPM'08 conference in Milan and is General Co-Chair of the BPM'09 conference in Ulm. Manfred is co-founder of the AristaFlow Corp.

Martin Jurisch holds a Diploma in Computer Science. Since 2008 he has been Executive Director of the AristaFlow Corp. Within this company the ADEPT2 research prototype of a next generation process management technology is transferred into an industrial-strength product version called AristaFlow BPM Suite. This software covers the complete process and service life cycle, and provides advanced features like service composition in a plug & play like fashion, flexible service orchestration, dynamic changes of in-progress services, and composition schema evolution. His major research interests include adaptive processes, service-oriented architectures and internals of process management technology (e.g., how to represent dynamically changed process instances within a process management system).