On Design Principles for Realizing Adaptive Service Flows with BPEL

Manfred Reichert[1] and Stefanie Rinderle[2]

[1]Information Systems Group, University of Twente, The Netherlands
m.u.reichert@utwente.nl
[2]Dept. Databases and Information Systems, Ulm University, Germany
stefanie.rinderle@uni-ulm.de

Abstract: Web service technology offers a promising approach for realizing enterprise-wide and cross-organizational business applications. With the *Business Process Execution Language for Web Services* (BPEL, also known as WS-BPEL or BPEL4WS) a powerful language for the process-oriented composition and orchestration of Web services exists. However, BPEL flow specifications tend to be too complex, and current BPEL engines do not provide the flexibility needed to cover the broad spectrum of business processes we have to deal with in practice. Neither ad-hoc deviations form the pre-specified BPEL schema (e.g., to add or move activities for single flow instances) nor the propagation of BPEL schema changes to a collection of flow instances have been supported by BPEL engines yet. This limits their applicability to only well-structured, rigid service flows. In this paper we address fundamental issues, which arise when enriching BPEL engines with dynamic change capabilities. We focus on the question how such changes can be realized in a correct and consistent manner and which prerequisites must be met in this context. In particular, we restrict BPEL to a reasonable subset of language elements in order to decide on these fundamental questions. By offering flexibility and adaptability in a controlled and reliable way the so promising Web Service technology will broaden its application scope significantly.

1 Introduction

Today there is a high need for active coordination of the various, distributed tasks necessary to perform enterprise-wide business applications. Service-oriented architectures offer a promising approach in this context [WCL$^+$05]. Usually, they provide a framework for specifying, implementing, and registering application services as well as for composing them in a process-oriented manner. The latter enables stateful interactions among the services and provides the basis for reliable process orchestration. In this context, the *Business Process Execution Language for Web Services* (BPEL, also known as WS-BPEL or BPEL4WS) has emerged as de-facto standard for implementing processes based on (Web) services [vDt$^+$05, OAS06]. Systems like Oracle BPEL Process Manager, IBM WebSphere Process Server, and Microsoft BizTalk Server support BPEL, thus illustrating the practical relevance of this process description language. Though intended as a language for connecting web services, it can be expected that in near future a wide variety of process-aware information systems (PAIS) will be realized using BPEL [vDt$^+$05].

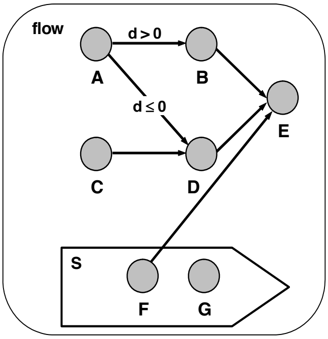
Whilst BPEL is an expressive language, it is by far too complex to realize more advanced process support functions. In particular, the aforementioned BPEL engines do not provide the flexibility and dynamism needed to cover the broad spectrum of business processes we can find in todays organizations. Neither ad-hoc deviations from the pre-planned service flow (e.g., to add, shift or delete activities in order to deal with an exceptional situation) nor the propagation of a business process change to a collection of flow instances have been supported yet. This, in turn, requires from users to bypass the system in exceptional cases or prohibits already running flow instances from being adapted to an optimized business process. In many domains, however, PAIS will not be accepted by users when rigidity comes with them. Creating service-oriented applications without a vision for adaptive process management, therefore, is shortsighted and expensive. Indeed, insufficient flexibility has been a primary reason why workflow technology failed in many process automation projects in the past. Obviously, similar problems will result for BPEL-based process engines if dynamic flow changes are prohibited a priori by a bad language design.

Barriers on the way towards adaptive BPEL flows are the complexity of BPEL and the missing formal semantics of this language. Though several approaches exist in this context (e.g., [Mar05, OvB$^+$05, RRD04a]), current formalizations and verification methods are incomplete (i.e., focussing on selected BPEL language elements only) and are not standardized. In particular, one will not be able to reason about the correctness of dynamic flow changes if there exist no formal basis and no mechanism for reasoning about correctness issues. In this paper we sketch how existing concepts from the area of adaptive processes (see [RRD04b] for an overview) can be applied to restrict and configure the BPEL language in a way such that it becomes a candidate for realizing adaptive flows. Due to lack of space we mainly focus on design principles and omit formal issues.

Section 2 summarizes background information related to the modeling of BPEL flows. Based on this, Section 3 introduces guidelines for a corresponding flow execution model. Using this model, Section 4 deals with the question how to check whether a given BPEL flow instance is compliant with a modified flow schema or not. Section 5 copes with the state adaptations becoming necessary in this context. Finally, Section 6 discusses related work and Section 7 concludes with a summary and an outlook.

2 Background Information

BPEL uses an expressive meta model for creating XML-based descriptions of business processes based on the interactions between the process and its partners. In the following we omit XML specifications and use graphical illustrations instead. In a BPEL flow, the interaction with each partner occurs through web service interfaces. Process activities can invoke service operations of partners synchronously, or they may receive messages from service invocations of partners and reply to them asynchronously later. BPEL provides a variety of possibilities to describe the desired flow logic. For example, Fig. 1 shows a



```
...
<flow>
  <links>
     <linkname = “L_AB”/>
     <linkname = “L_AD”/>
     <linkname = “L_FE”/>
     …
  </links>
  activity A
     <source linkName = “L_AB” transitionCondition=”d>0”/>
     <source linkName = “L_AD” transitionCondition=”d≤0”/>
  activity B
     <target linkName = “L_AB”/>
     <source linkName = “L_BE”/>
  …
  activity E
     joinCondition = “(L_BE OR L_DE) AND L_FE”
     <target linkName = “L_BE”/>
     <target linkName = “L_DE”/>
     <target linkName = “L_FE”/>
  <sequence name = “S”>
      activity F
         <source linkName = “L_FE”/>
      activity G
  </sequence>
</flow>
```

Figure 1: Example of a BPEL flow schema (modeled in a graph-like fashion)

BPEL flow schema which has been modeled in a graph-like fashion, i.e., the flow logic is described as a network of activity nodes and (control) links. For such a graph-based BPEL flow, a variety of configuration facilities exist, like the assignment of transition conditions to (control) links (i.e., predicates on flow variables), and the definition of activity join conditions. However, the flow from Fig. 1 can be also modeled in a more block-oriented fashion as depicted in Fig. 2. This flow schema is based on structured activities (sequence, flow, switch) and two links which synchronize activities from parallel branches. Generally, structured activities can be nested. In our example, the switch activity is contained in a sequence activity, which itself is surrounded by a flow activity (representing parallelism).

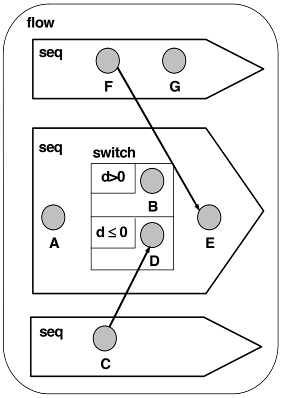


Figure 2: BPEL flow from Fig. 1 modeled in a block-like fashion

Generally, it is possible to mix both modeling styles by having links crossing the boundaries of structured activities. On the one hand BPEL provides high expressiveness and many ways to express the same thing; on the other hand this aggravates flow modeling, analysis and verification (see [RRD04a] for details). In order to reduce this complexity and to be able to reason about correctness properties of BPEL flow specifications, in [RRD04a] we have introduced a classification for BPEL flow schemes. In this work, we have also shown that block-structured modeling, together with the controlled use of links, offers advantages when compared to the graph-like modeling style. Note that such considerations become necessary when reasoning about the correctness of flow schema changes. However, for the rest of this paper we abstract from issues related to schema correctness, and assume that changes of BPEL flow schemes are handled in a correct manner. Instead we focus on the propagation of such flow schema changes to running flow instances.

3 Designing an Execution Model for BPEL Flows

A prerequisite for dynamically adapting the structure and state of BPEL flow instances is the provision of a formal semantics. It constitutes the basis for both the correct execution of BPEL flow instances and the correct adaptation of their states when applying dynamic changes. Regarding flow instance changes it must be also possible to quickly decide whether a given flow instance or a collection of flow instances is compliant with the modified flow schema, and may therefore be migrated to it. In the following we discuss guidelines for the design of such an (adaptive) flow execution model for BPEL. Our considerations are based on a previous survey on formal models for adaptive process management (cf. [RRD04b]). Due to lack of space we focus on fundamental issues and exclude more advanced BPEL process patterns (e.g., scopes and exception handlers).

We distinguish between different states a single BPEL activity (i.e., a single process step) may run through. Initially, an activity has state *NotActivated*. It will change to *Activated* (i.e., the activity becomes enabled) if all preconditions for executing the activity are met. When starting it (e.g., by invoking a Web Service operation or by receiving a message from a partner) we obtain activity status *Running*. At successful termination activity status passes to *Completed* whereas processing failures result in status *Failed*. Finally, status *Disabled* represents activities of non-selected paths (e.g., activities whose execution has been skipped due to a dead path elimination or which belong to a non-selected branch of a switch block). In addition, for each instance I an execution history $H_I =< e_0, ..., e_k >$ is kept. It logs events related to the start and termination of flow activities.

A flow execution model for BPEL must provide rules for initializing activity states when creating new flow instances and for adapting these states when activities are started or completed. More precisely, these rules must enable the runtime system to decide in which flow states a certain activity can be activated, and to determine the correct follow-up state of the flow instance when activities are completed. To achieve this, for basic as well as structured activities a well-defined operational semantics is required. For example, an adequate representation of both activity and link states – also denoted as activity / link markings in the following – is needed. Generally, different options exist in this context. The first one uses only one type of control token passing through each flow instance (True-Tokens). An alternative is to use two types of tokens (True- and False-Tokens). Simplistically, True-Tokens trigger activities that are to be executed next and False-Tokens describe disabled activities. Flow description formalisms which solely use True-Tokens include Petri-Nets [RRD04b].

Approaches which additionally use False-Tokens usually preserve the markings of already passed regions (except for loop backs) and explicitly mark disabled activities. These approaches can be further divided according to the way they represent the tokens. One option is to gain them from execution histories (which log events like activity start / completion. Alternatively, (model-inherent) activity markings, representing a consolidated view on execution logs, can be used [RRD04b].

As discussed in [RRD04b] the use of a flow execution model with True-/False-Tokens offers the best perspectives with respect to dynamic process changes. A corresponding example is depicted in Fig. 3. As can be easily seen from this figure, activity markings do not only indicate whether an activity is currently activated or disabled, but also provide a consolidated view on previous instance execution; i.e., on the execution history of the respective instance. As we show in the following this is exactly what we need to efficiently check compliance of a flow instance with a modified BPEL flow schema and to automatically adapt markings of this instance when migrating it to the new schema version.

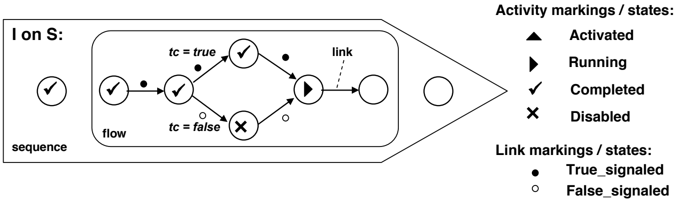


Figure 3: BPEL flow instance with activity markings (True-/False semantics)

In principle, BPEL is coherent with this way of representing activity states. However, in our context we need a more formal definition of the underlying operational semantics as currently provided by the BPEL specification. In any case, BPEL shows a good potential for satisfying the following basic properties needed for the support of adaptive flows:

1. The flow execution model must offer different kinds of activity markings to distinguish between executed and disabled flow paths (True-/False-Semantics).

2. Internal activity states must be reflected by the flow execution model. For example, it must be possible to distinguish between enabled activities and already started ones. This is crucial for checking compliance of a flow instance with a modified schema.

3. The markings of already passed regions should be (logically) preserved (except loop backs); they provide a consolidated view on the previous instance execution.

4. The flow execution model must provide precise execution and marking rules, which state under which marking an activity can be executed and which new marking results when completing it. Such a flow execution model should be based on a graph-based representation of BPEL schemes.

Fig. 3 shows a simple BPEL flow instance – a sequence block with an embedded flow activity as well as embedded basic activities – together with its current activity and link markings. Note that this representation already satisfies the execution properties 1–3 mentioned above. At this point it is important to mention that activity/link markings only represent a logical view on the respective flow instance. How these markings are internally represented, how they are physically stored, and which optimizations are conceivable in this context depends on the concrete BPEL implementation and is outside the scope of this paper. The same applies to the concrete set of marking and execution rules.

4 A Correctness Criterion for Changing BPEL Flow Instances

We now deal with the challenge to propagate changes of a BPEL flow schema to corresponding flow instances. The ambition must be to preserve correctness of these instances when dynamic changes are carried out. We need adequate criteria to decide whether an instance I is compliant with a changed BPEL schema; i.e., whether the change can be correctly propagated to I without causing errors (like deadlocks or improperly invoked services). For this we need an adequate correctness criterion which does not needlessly exclude a flow instance from being migrated to a new schema version. Furthermore, it must be ensured that correctness can be efficiently checked by the BPEL engine.

4.1 Preliminaries

We assume that BPEL schema changes are correctly accomplished. However, this is not sufficient to guarantee a consistent execution behavior of flow instances when migrating them to the new BPEL schema version. Additionally, we must consider instance states. To illustrate the problems resulting from an uncontrolled migration consider the example from Fig. 4. The BPEL schema S from Fig. 4a) represents a *sequence* consisting of activities A, B and C. Assume that S is correctly transformed into S' by adding activities X and Y as well as a data dependency between them – Activity Y reads variable d which is written by the preceding activity X (cf. Fig. 4a). Assume further that these changes are propagated in an uncontrolled way to the flow instances from Fig. 4c) (currently running on S). Concerning instance I_1 no problem would occur since its execution has not yet entered the region affected by the change (activity A is still running). Uncontrolled migration of instance I_2, however, would cause malfunctions. First, an inconsistent state would result – B is already running though its (new) predecessor X (on S) has not been completed. Second, the newly inserted activity Y might be invoked though variable d has not been written (by X). For flow instance I_3 migration would be possible. When doing so, however, activation of B has to be undone and the added activity X be activated instead. – In the following we do not distinguish between changes of single flow instances (e.g., to deal with exceptions) and adaptations of a collection of flow instances (e.g., to deal with a process type change). Instead we focus on fundamental issues related to dynamic changes of BPEL flow instances and sketch how they can be addressed.

Our previous example has demonstrated that the applicability of a dynamic flow change

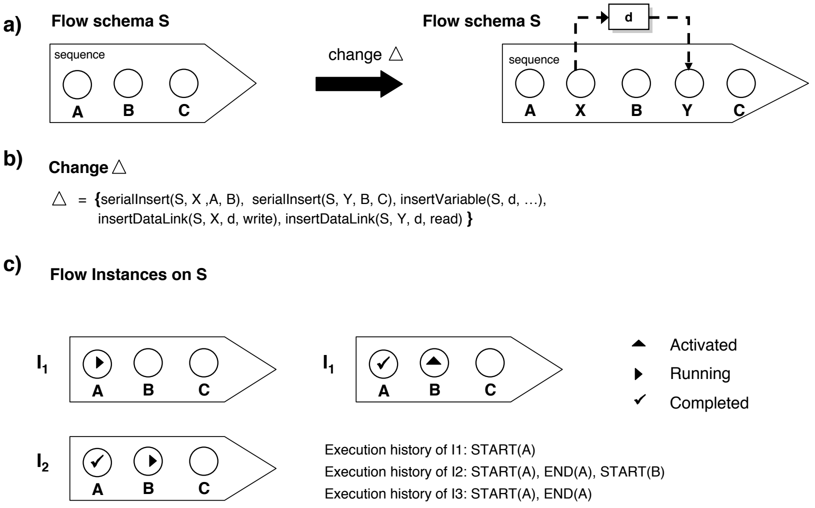

a)
Flow schema S
sequence
A
B
C
change △
Flow schema S
d
sequence
A
X
B
Y
C
b)
Change △
△ = {serialInsert(S, X ,A, B), serialInsert(S, Y, B, C), insertVariable(S, d, …), insertDataLink(S, X, d, write), insertDataLink(S, Y, d, read) }
c)
Flow Instances on S
I1
A
B
C
I1
A
B
C
Activated
Running
Completed
I2
A
B
C
Execution history of I1: START(A)
Execution history of I2: START(A), END(A), START(B)
Execution history of I3: START(A), END(A)

Figure 4: BPEL flow schema and related flow instances

depends on the current flow state as well as on the kind of changes being applied. Let I be a flow instance with BPEL schema S; assume that S is transformed into another "correct" BPEL schema S' by applying change Δ. Then two fundamental issues arise:

1. Can Δ be correctly propagated to flow instance I, i.e., without causing errors and inconsistencies? If yes, we denote I as compliant with respect to S'.
2. Assuming I is compliant with S', how can we smoothly migrate it to S' such that its further execution can be based on this new BPEL schema? Which state adaptations (e.g., enabling/disabling of activities) become necessary in this context?

While the first issue concerns pre-conditions on the current state of I, the second one is related to post-conditions that must be satisfied after propagating the change. In any case we have to provide an efficient solution which enables automatic and correct compliance checks as well as automated instance migrations.

4.2 A General Correctness Criterion

In [RRD04c] we have introduced a general correctness criterion, which is independent of the used process description formalism and which allows us to argue about the correctness of dynamic flow changes. In more detail, this criterion enables us to decide whether a given flow instance can be migrated to the modified schema or not (so-called *compliance*). In addition, we provide automated procedures to determine the correct follow-up state of compliant flow instance resulting after their migration.

Simplistically, flow instance I would be compliant with a (modified) flow schema S' if I could have been executed according to S' as well, and would have produced the same effects on flow state and variables. Trivially, this will be always the case if I has not yet entered the region affected by the flow schema change. Generally, we need information about previous flow execution to decide on this property and to determine correct follow-up states for the (compliant) flow instances to be migrated. For this purpose, we can make use of the execution history which is kept for each flow instance (cf. Fig. 4c) independent of the applied flow description language. Usually, this history logs events related to the start / completion of flow activities. Obviously, flow instance I with history H_I will be *compliant* with S' (and therefore be able to *migrate* to this schema) if H_I can be produced on S' as well. Consequently we then obtain a correct new state of flow instance I on schema S' by replaying all events from H_I on S' in chronological order. Taking our example from Fig. 4 this holds for I_1 and I_3, but does not apply to I_2. Furthermore, when replaying H_I on S', we obtain the new flow states as sketched above. Note that the aforementioned criteria are independent of the underlying flow description language.

The described criterion is still too restrictive to serve as general correctness principle. Particularly with respect to loops[1] it may needlessly exclude flow instances from being migrated to a modified schema. As an example take a flow instance I for which a loop block is in its 2nd iteration. Assume that the flow schema shall be modified by adding activities to a non-entered region from this loop block. Taking the above criterion this change will be not allowed if the first loop iteration of I is not compliant with the resulting flow schema; i.e., H_I could not be completely produced on the modified schema. However, excluding such flow instances from migrations is not in accordance with practice. To cope with this we relax the above criterion by (logically) discarding those history entries from the execution history produced within another loop iteration than the last (completed loops) or current one (running loops). We denote this reduced view on execution history H_I as *reduced execution history* RH_I. Based on this we can apply the following fundamental correctness principle for dynamic changes of BPEL flow instances:

Axiom 1 (Dynamic Change Correctness) *Let I be a BPEL flow instance on schema S with execution history H_I and reduced execution history RH_I. Assume further that S is transformed into a correct schema S' by applying change Δ. Then:*

1. *Δ can be correctly propagated to BPEL flow instance I iff RH_I can be produced on S' as well – for this case I is said to be compliant with S'.*
2. *When propagating Δ to a flow instance I, which is compliant with S', the correct state of I on S' (e.g., activity markings) can be obtained by replaying RH_I on S'.*

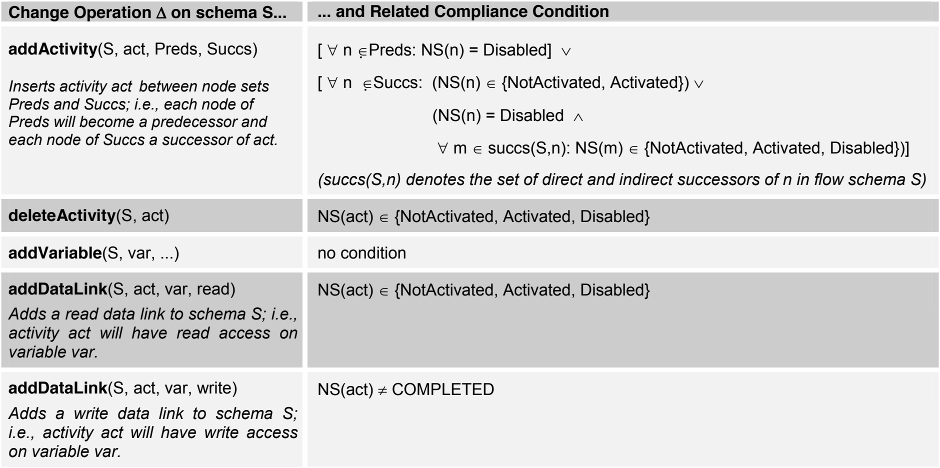
4.3 Rules for Automatically and Efficiently Checking Compliance

The challenge is how to efficiently guarantee compliance and how to determine the correct new state of compliant flow instances (according to Axiom 1) in BPEL. Certainly, it

[1] Loops can be modeled in BPEL by means of a specific block element.

would be no good idea to access the whole (reduced) execution history and to try to replay it on the modified BPEL flow schema. This may cause a performance penalty due to the large number of instances to be treated. In the following we present optimized rules and procedures for BPEL flows in order to ensure dynamic change correctness according to Axiom 1. For selected change operations we examplarily sketch compliance conditions which can be efficiently checked by an adaptive BPEL flow engine. We apply the following design principles: (1) We consider change semantics and change context in order to precisely specify which state information is needed for checking compliance. (2) We make use of the dynamic model properties sketched in Section 3. In particular, the derivation of compliance rules benefits from model-inherent activity markings which already provide a consolidated view on the (reduced) history of a flow instance.

In the following let S be an executable BPEL flow schema and let I be a flow instance on S with marking MS = (NS, ES)[2]. Assume that S is correctly transformed into another BPEL flow schema S' by applying change Δ[3]. For selected changes Δ, Fig. 5 shows the formal conditions under which we can ensure compliance of I with the modified BPEL flow schema S'. More precisely, one can formally prove that flow instance I is compliant with flow schema $S' = S + \Delta$ (according to Axiom 1) if the compliance rule related to Δ is satisfied. Note that none of the change operations depicted in Fig. 5 requires access to the (complete) execution history. Generally, this cannot always be achieved. Concerning the insertion of a new link, for example, in certain cases compliance can be guaranteed even if the target activity of this link has been started, completed, or disabled. To decide on this, however, additional information from execution logs is needed.



Change Operation Δ on schema S...	**... and Related Compliance Condition**
addActivity(S, act, Preds, Succs) *Inserts activity act between node sets Preds and Succs; i.e., each node of Preds will become a predecessor and each node of Succs a successor of act.*	[∀ n ∈Preds: NS(n) = Disabled] ∨ [∀ n ∈Succs: (NS(n) ∈ {NotActivated, Activated}) ∨ (NS(n) = Disabled ∧ ∀ m ∈ succs(S,n): NS(m) ∈ {NotActivated, Activated, Disabled})] *(succs(S,n) denotes the set of direct and indirect successors of n in flow schema S)*
deleteActivity(S, act)	NS(act) ∈ {NotActivated, Activated, Disabled}
addVariable(S, var, ...)	no condition
addDataLink(S, act, var, read) *Adds a read data link to schema S; i.e., activity act will have read access on variable var.*	NS(act) ∈ {NotActivated, Activated, Disabled}
addDataLink(S, act, var, write) *Adds a write data link to schema S; i.e., activity act will have write access on variable var.*	NS(act) ≠ COMPLETED

Figure 5: Examples of compliance rules for dynamic changes of BPEL flows

We exemplarily describe the compliance condition of change operation addActivity. Re-

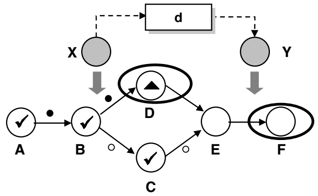
[2] To each activity of the respective flow instance NS assigns one of the markings *NotActivated*, *Activated*, *Completed*, *Disabled* or *Failed*, and to each link of the respective flow instance ES assigns one of the states *NotSignaled*, *TrueSignaled* or *FalseSignaled*

[3] What BPEL schema correctness means in our context has been discussed in [RRD04a]

garding the insertion of an activity act between two node sets $Preds$ and $Succs$ compliance can be guaranteed if all activities from $Succs$ possess one of the markings *Activated* or *NotActivated*. In this case, none of the successors of act has yet written an entry into the execution history H_I. A simple example is depicted in Fig. 6 where activities X and Y as well as a data dependency between them are added. Furthermore, compliance can be also ensured if all activities from $Preds$ are marked as *Disabled*. Then the added activity will be disabled as well, i.e., its insertion will have no effect on compliance. Finally, activity act may be added as predecessor of a disabled activity provided that none of the successors of this activity has a marking other than *NotActivated*, *Activated*, or *Disabled*.

change operation △ ...	**... and related compliance condition**
addActivity(S, act, Preds, Sucss)	[∀ n ∈ Preds: NS(n) = Disabled] ∨ [∀ n ∈ Succs: (NS(n) ∈ {NotActivated, Activated}) ∨ (NS(n) = Disabled ∧ ∀ m ∈ succ(S,n): NS(m) ∈ {NotActivated, Activated, Disabled})]

flow instance I on S:



Change △ = { addActivity(S, X ,B, D),
addActivity(S, Y, E, F), ... **}**

NS(D) = Activated, NS(F) = NotActivated
(NS: Node/Activity State)

⇨ **I compliant with modified flow schema**

Figure 6: Checking compliance when inserting new activities

Fig. 5 shows compliance conditions for several other change operations (e.g., deletion of activities, insertion of data links). In a similar way, compliance conditions for other change primitives can be derived (e.g., insertion/deletion of structured activities, update of a transition/join condition, etc.). Altogether we can state that compliance (cf. Axiom 1) very often can be decided on basis of current activity markings; i.e., the engine must not explicitly check the producibility of execution histories on the modified schema. In order to come to a complete solution, corresponding compliance conditions must be derived for all basic change operations applicable on a BPEL schema. Based on them, one can develop high-level change operations and related compliance rules. Since the latter can be derived by merging compliance conditions of the applied change operations and by discarding unnecessary expressions (e.g., conditions on nodes not present in the original flow schema) we do not consider them in this paper. As mentioned, Fig. 5 only presents the compliance conditions for selected change primitives. One remaining task is to provide a complete set of change operations on BPEL schemes and to derive respective compliance conditions for them. One challenge in this context is the high expressiveness of the BPEL4WS language.

5 Automatically Adapting BPEL Flows After Changes

We exemplarily show how compliant flow instances can be migrated to a changed flow schema. One problem to be solved is the correct and efficient adaptation of activity and link markings. How extensive marking adaptations turn out for an instance I to be migrated depends on the kind and scope of the change. Except initialization of newly added activities and links, for example, no adaptations will become necessary if execution of I has not yet entered the change region. In other cases extensive marking adaptations may be required. An activity X, which is enclosed by a flow activity, may have to be deactivated if new links are inserted with X as target activity. Conversely, an added activity Y will have to be immediately activated or disabled if all predecessors of Y possess a final marking (*Completed, Disabled*). As shown in Fig. 7, it may even become necessary to reset disabled activities in their state when adding an activity.

According to Axiom 1,(2) the marking of a migrated flow instance must be the same as it could be obtained when replaying the respective (reduced) history on the modified flow schema. Following the sketched approach, however, it does not become necessary to access execution logs. Instead the necessary adaptations of activity/link markings can be automatically and efficiently done by (re-)evaluating the markings of those activities/links, which constitute the context of a change region. Regarding activity insertion, for example, one must re-evaluate the marking of the added activity itself as well as of its successors. With respect to insertion of a new link, the marking of this link and its target activity must be evaluated. We omit a presentation of algorithmic details and discuss the basic principles along two simple examples instead.

Example 1 (Adding a new activity): A first example is depicted in Fig. 7. Control flow is defined by a network of links. Activities A and D have been completed, whereas activities B and C have been disabled due to a dead path elimination. Let us assume that activity X shall be inserted as successor of A and as predecessor of C. Though C is disabled, this change is allowed since there is no successor of C which has been started yet. Thus the given flow instance is compliant with the modified BPEL schema. When migrating it to this schema, several marking adaptations become necessary to correctly proceed with the flow of control. At first, the incoming link of activity X is evaluated to True_Signaled (since A has marking Completed and the inserted link has no transition condition). This, in turn, leads to activation of X. Since the outgoing link of X cannot be signaled at the moment, the marking of its target activity C (and of C's successor E) is reset to NotActivated. – This simple example illustrates that changes cannot only be applied to the not yet entered regions of a BPEL flow schema. Very helpful for efficiently deciding on the compliance of a flow instance with a modified BPEL schema and for adapting markings is the distinction between completed and disabled activities (True-/False-Semantics) as well as the preservation of markings of already passed regions.

Example 2 (Changing Activity Orders): Consider Fig. 8. Let us assume that, at a certain point in time, a BPEL flow instance looks like the one depicted in Fig. 8a): Activities A and B have been completed and activity C is currently activated (i.e., C is read to start). Normally, A, B, C, D, and E have to be executed in sequence. Assume that an exception occurs which makes it necessary to immediately perform D (which is currently in state

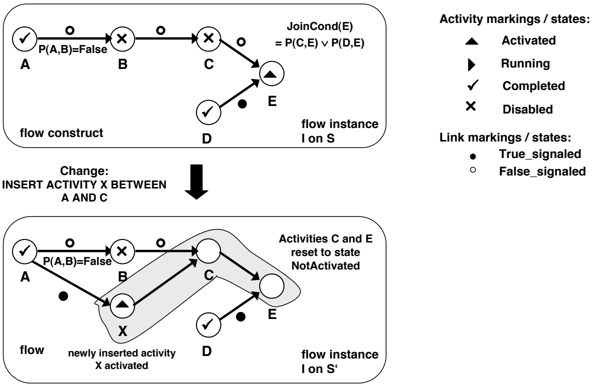


Figure 7: Insertion before a disabled activity

NotActivated) while maintaining the order of all other activities. At first, data dependencies for D are checked; D is immediately executable, in principle, because it reads flow variable d which has been already written by preceding activity B. In order to make it possible that D can be immediately started, it must no longer be a successor of C (because it would have to wait for completion of C). Instead it must be arranged in parallel to C. This can be achieved, for example, by inserting a flow activity (after B) and by placing C and D within it. Finally, the marking of the shifted activity D is re-evaluated leading to the activation of D (the predecessor B of the respective flow activity is completed). Fig. 8c) illustrates how the flow instance looks like after this change.

As a second order changing example, once again take Fig. 8a) and assume that the execution order of activities D and C shall be swapped. This change is allowed since activity C has not been started yet, i.e., the flow instance is compliant with the resulting BPEL schema. (If activity C had been already started or completed, the flow instance would be not compliant with the intended schema (completion of D before starting C); i.e., the change would be not possible.) Finally, markings of activities C and D, which constitute the change region, are re-evaluated leading to activation of D and deactivation of C.

6 Related Work

The introduction of service oriented architectures has increased the flexibility of process-centered software architectures. For example, late binding of service implementations to service specifications enables late modeling of sub-processes as described in [Han97, HJH96]. However, this kind of flexibility is orthogonal to structural flow adaptations.

There are only few approaches which systematically deal with modeling and verification issues related to BPEL control flow specifications [BK03]. As shown in [KMW03, WvDt02], flaws such as service flows which run into deadlocks or service operations which are invoked with missing input data can be avoided by using BPEL as flow de-

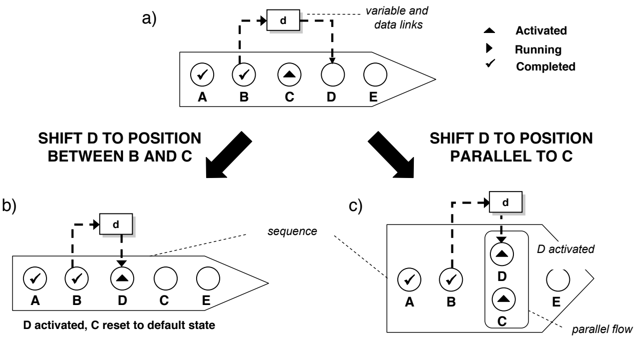

a)
d
variable and data links
Activated
Running
Completed
A
B
C
D
E
SHIFT D TO POSITION BETWEEN B AND C
SHIFT D TO POSITION PARALLEL TO C
b)
d
sequence
A
B
D
C
E
D activated, C reset to default state
c)
d
D activated
A
B
D
C
E
parallel flow

Figure 8: Changing activity order of an in-progress flow instance

scription language. Reason is that it allows to model the flow logic of business processes separately and independently from the implementation of the used Web Services.

In addition to static correctness issues lots of research has been spent on the dynamic adaptation of business processes in general ranging from ad-hoc modifications of single process instances [RD98] to process schema evolution [vB02, CCPP98, RRD04b, Wes01]. However, so far, it has not been investigated whether or not – and if yes how – these results can be applied to BPEL flows as well. In this paper we have not developed new concepts for adaptive processes, but have shown that existing approaches can be applied to BPEL as well. However, this necessitates a more formal and profound basis for this language.

There is ongoing work on the evolution of process choreographies (in addition to process orchestrations such as in this paper) [RWR06]. Focus is put on the correctness of a process choreography after changing the private process of one of the partners involved. For this the authors provide criteria based on which the correct propagation of private process changes to public and private processes of the other partners can be ensured.

7 Summary

We have proposed an approach for adapting BPEL flows based on guidelines for modeling and execution. Correctness requirements are met by a general correctness criterion based on execution logs. In order to verify correctness in an efficient manner (e.g., at the presence of thousands of running BPEL flows) we have elaborated state conditions which are easily to check. Furthermore, we have shown how to adapt BPEL flow states after applying a change. Due to lack of space we have omitted formal details and algorithms, and have presented examples instead. This work has focused on process orchestrations. However, in future work, we aim at tackling change and evolution of process choreographies as well. Challenges in this context comprise the propagation of private flow changes to the public and private flows of partners in order to preserve correctness of the choreography.

References

[BK03] F. Breugel and M. Koshkina. Verification of business processes for web services. Technical Report CS-2003-11, York University, Ontario, CA, 2003.

[CCPP98] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data and Knowledge Engineering*, 24(3):211–238, 1998.

[Han97] Y. Han. *Software Infrastructure for Configurable Workflow Systems.* Dissertation, Fachbereich Informatik, TU Berlin, 1997.

[HJH96] T. Herrmann and K. Just-Hahn. Organizational Learning with Flexible Workflow Management Systems. *SIGOIS Bulletin*, 17(3):54–57, 1996.

[KMW03] R. Khalaf, N. Mukhi, and S. Weerawarana. Service-oriented composition in BPEL4WS. In *Proc. WWW'03*, Budapest, 2003.

[Mar05] A. Martens. Analyzing Web Service based Business Processes. In *Proc. Int'l Conf. on Fundamental Appr. to Software Eng. (FASE'05)*, LNCS 3442, Edinburgh, April 2005.

[OAS06] OASIS. *Web Services Business Process Execution Language Version 2.0 - Committee Draft*, 2006. http://www.ibm.com/developerworks/library/ws-bpel/.

[OvB^{+}05] C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, and H.M.W. H.M.W. Verbeek. Formal semantics and analysis of control flow in WS-BPEL. Technical Report Ext. rep. 05-13, BPM Center Report, Eindhoven, 2005.

[RD98] M. Reichert and P. Dadam. ADEPT$_{flex}$ - Supporting Dynamic Changes of Workflows Without Losing Control. *JIIS*, 10(2):93–129, 1998.

[RRD04a] 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.

[RRD04b] S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey. *Data and Knowledge Engineering, Special Issue on Advances in Business Process Management*, 50(1):9–34, 2004.

[RRD04c] 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.

[RWR06] S. Rinderle, A. Wombacher, and M. Reichert. On the Controlled Evolution of Process Choreographies. In *Proc. 22nd Int. Conf. on Data Engineering*, Atlanta, 2006.

[vB02] W.M.P v.d. Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoret. Comp. Science*, 270(1-2):125–203, 2002.

[vDt^{+}05] W.M.P. v.d. Aalst, M. Dumas, A.H.M. ter Hofstede, E. Verbeek, and P. Wohed. Life after BPEL. In *Proc. WS-FM 2005*, LNCS 3670, pages 35–50, 2005.

[WCL^{+}05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Toney Storey, and Donald Ferguson. *Web Services Platform Architecture.* Prentice Hall, 2005.

[Wes01] M. Weske. Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In *Proc. HICSS-34*, 2001.

[WvDt02] P. Wohed, W.M.P. v.d.Aalst, M. Dumas, and A. ter Hofstede. Pattern Based Analysis of BPEL4WS. Technical Report FIT-TR-2002-04, QUT, Australia, 2002.