Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz
Kommunikation in Verteilten Systemen 2011
(WowKiVS 2011)

Concurrent Workflow Evolution

Mirko Sonntag and Dimka Karastoyanova

12 Pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Concurrent Workflow Evolution

Mirko Sonntag and Dimka Karastoyanova

Institute of Architecture of Application Systems
University of Stuttgart
Universitaetsstrasse 38
70569 Stuttgart, Germany
{sonntag, karastoyanova}@iaas.uni-stuttgart.de

Abstract: Workflow evolution is a collective term for concepts that deal with changes of workflow models. Adapted workflow models are deployed on a workflow engine as new model version. That means two versions of the same model are deployed on the engine. Typically, this results in conflicts between the workflow models. For example, how does a client find and choose the desired workflow version to instantiate? Typically, these problems are solved by deactivating the old model. New instances can only be created for the new model. In our work on scientific workflows we recognized that there are cases where it is desired to keep the old model activated. In this paper we investigate what it means to have several model versions active. We develop a general concept for this "concurrent workflow evolution" that solves emerging problems. Moreover, we show how this concept can be realized with BPEL.

Keywords: Workflow evolution, scientific workflows, Model-as-you-go, BPEL.

1 Introduction

Most existing tools in the area of scientific workflow management do not rely on workflow concepts as known from the conventional workflow technology [1], e.g. distinction of process models and instances, workflow auditing and monitoring, or flexibility. Instead many scientific workflow systems are developed from scratch based on proprietary workflow languages. One of the reasons for this fact is that scientists and scientific applications impose requirements on workflow systems for scientific simulations and computations that are not satisfied by conventional workflow management systems, e.g. data orientation or steering workflow execution from within the workflow modeling tool [2].

We argue that the workflow technology can be of high value for scientists because of the aforementioned features. Especially flexibility and the adaptation of running workflows are interesting for scientists because it can support them in creating experiments in a trial-and-error manner. In our former work we proposed the model-as-you-go approach that facilitates modeling, execution, adaptation, and monitoring of scientific workflows based on the conventional workflow technology [3]. An important aspect of model-as-you-go is the modification of running workflow instances. Scientists may adapt computations to enforce convergence of results, to repair failures, or to modify or complete the logic of an experiment. That means model-as-you-go requires workflow evolution and instance migration techniques.

Workflow evolution and instance migration in the area of business workflows are extensively discussed in existing work [4, 5, 6]. In a nutshell, a workflow model is modified during workflow evolution. The old and the new workflow models are then available to clients.

Usually, the new model is active (i.e. can be instantiated), while the old model works as schema for running instances. Selected workflow instances can be transferred (i.e. migrated) to the new model if compliance rules are satisfied [7].

In our work with scientific workflows we recognized that deactivation of the old model introduces a restriction, which hampers the fulfillment of an actual requirement in scientific workflows. There are cases where it is required to keep both the old and new model active, e.g. to be able to re-execute a former configuration of an experiment. A few existing business workflow systems already provide this kind of feature [8, 9], which shows that it is also a relevant requirement in business scenarios. However, a general and implementation-independent concept for workflow evolution with more than one active model versions in combination with instance migration techniques is missing so far.

In this paper we contribute a concept for workflow evolution with two or more active process models. We denote this approach *concurrent workflow evolution*. We show which problems arise when having several active versions of a model and how they can be addressed. Where possible we adopt existing concepts for instance migration, change operations and compliance checks. The devised concept is independent of concrete workflow languages. As an example we show its implementation with BPEL.

The remainder of the paper is structured as follows. Section 2 outlines related work in the area of workflow evolution and adaptation as well as a classification for workflow changes derived from existing work. Section 3 shows a scenario for scientific workflows that implement a simulation for the mutation of bacteria. Section 4 presents the concept of concurrent workflow evolution; Section 5 its implementation. The paper concludes in Section 6.

2 Related Work and Workflow Change Classification

Much work has already been done in the area of ad hoc changes in workflows and workflow evolution. Casati et al. (1996) [4] invented a workflow modification language to adapt workflow schemata while keeping structural and behavioral consistency. The language foresees a set of primitives/change operations, e.g. for adding activities. Additionally, the authors present a taxonomy of workflow evolution policies. Van der Aalst et al. (2000) [5] discuss dynamic changes of workflows based on workflow nets. Techniques for dealing with running instances are shown (restart, proceed, transfer) and the importance of syntactic and semantic correctness as well as management of modified workflows is emphasized. Reichert and Rinderle (2006) [10] developed a formal semantics for BPEL that enables the modification of running BPEL process instances. BPEL schemas are adapted with the help of change operations and these changes are propagated to running instances. Prior to migration of an instance its compliance to the new schema is checked based on the instance's execution history and activity/link markings. Weber et al. (2007) [11] present process change patterns for the modification of workflows. Two types of patterns are described: adaptation patterns (for structural changes of process models) and patterns for predefined changes (for changes in particular regions of processes). Weber et al. (2008) [12] discuss change support features like schema evolution, version control, or instance migration. Schonenberg et al. (2008) [6] propose a taxonomy for flexibility strategies. The strategy "flexibility by change" with evolutionary changes is similar to workflow evolution: New process instances follow the new model; the old model cannot be instantiated.

In order to clarify the relation of our approach to the state-of-the-art we outline the classification of workflow changes based on a subset of existing work (see Table 1). In our previous work we presented a classification of possible adaptation types for service compositions in the context of the workflow life cycle and workflow dimensions [13]. The focus in this work is on the comparison of which of the models (old, new, ad-hoc) is active, i.e. can be instantiated, after a change has been performed. *Ad-hoc changes* are applied to single workflow instances. In order to change a single workflow instance an adapted ad-hoc workflow model is needed that is used as the schema exclusively for this instance [5, 6] (note that ad-hoc models do not have to be represented in terms of an explicit artifact). New instances are created according to the old, unchanged workflow model. *Workflow evolution* denotes changes of an existing workflow model [4, 5, 6] during the modeling phase of the process model, while there are running instances of the process. Usually, a new version of the model is deployed. Different policies can be used to manage the running instances [4]. All running instances can be aborted (1) or the system waits until they terminate (2). After that instances of the new model can be created. The progressive policy (3) is the most common approach for workflow evolution. Running instances may proceed according to the old model (*concurrent to completion* policy), may be migrated to the new model (*migration to final workflow* policy), may be aborted (*abort* policy), or may be migrated to ad-hoc models (*migration to ad-hoc workflow* policy). In the latter case, the ad-hoc model is active only for one or more selected instances. In all cases of the progressive policy the new model is active immediately and is therefore the schema for all newly created instances. The existing ad-hoc changes and workflow evolution approaches have in common that they keep only one version of a process model active at a particular point in time. Existing workflow evolution techniques discard the old model once all its instances are completed. The classification reveals that an additional type of workflow evolution is missing where the old model remains active (shaded row at the bottom of Table 1). This missing approach is subject of this paper.

Table 1: Classification of existing approaches for workflow changes. An approach for a workflow evolution mechanism where the old model remains active is missing. We therefore introduce the concurrent workflow evolution (the shaded row at the bottom) which is currently not accounted for by existing concepts.

Strategy		**Applies to**	**Model**		
			Old	**New**	**Ad-hoc**
Ad-hoc [5], momentary change [6]		Single instance	Active	n/a	Active for selected instances
Workflow evolution [4], evolutionary change [6], structural change [5]	(1) Abort [4]	Model	Inactive	Active (after abort of running instances)	n/a
	(2) Flush [4]	Model	Inactive	Active (after termination of running instances)	n/a
	(3) Progressive [4], on-the-fly [6]	Model	Inactive	Active (immediately)	Active for 1 instance (migration to ad-hoc workflow [4])
	(4) Concurrent	Model	Active	Active	n/a

3 Scenario

As a sample scenario we take the computer simulation of the reproduction of bacteria in a biological system with certain environment parameters. Imagine a scientist named Jim that models the behavior and metabolism of bacteria with the help of a workflow. Each workflow instance then represents a single bacterium. The simplified bacteria model is shown in Figure 1a (as BPEL process). A `while` loop contains the recurring behavior of the bacteria type such as ingesting minerals or synthesizing enzymes. An `onAlarm` event implements the reproduction of bacteria after 40 minutes. Reproduction is in fact the creation of a new instance of the bacteria process model. An external (`onMessage)` event represents the case when the bacterium gets in contact with an antibiotic. The process instance is then terminated which models the dead of the bacterium. Environment conditions (e.g. concentration of minerals) and other domain specific knowledge (e.g. synthesizing and behavior of enzymes) are taken into account by the respective executables that are invoked. In case of BPEL these would be (stateful) Web services (WSs) that themselves can be realized using BPEL.

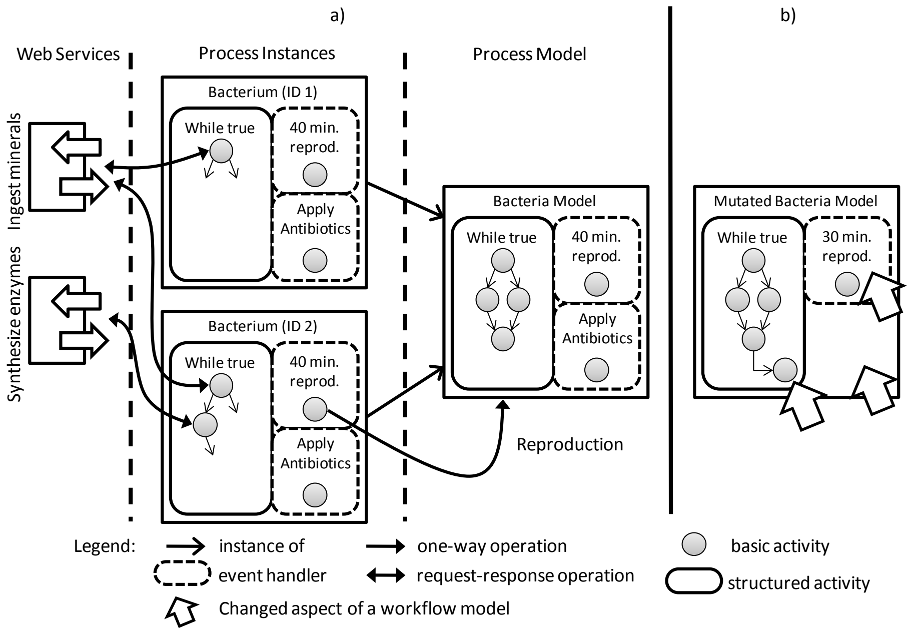

Figure 1: Workflow-based simulation of bacteria reproduction in a biological system (a). The original workflow model is adapted to simulate a mutation of a bacterium (b).

Now imagine Jim runs and monitors the simulation. He then wants to simulate a mutation of a single bacterium. Mutation of bacteria is a quite normal process and can usually be initiated by a faulted reproduction of bacteria or due to external influences such as UV radiation. While the *reason* for a mutation could be modeled in a computer system with occurrence probabilities, the *effects* of a bacteria mutation in a computer simulation have to be introduced by human intelligence. An intervention of Jim is needed to model the changed behavior of the mutated bacterium. He has to create a new version of the bacteria workflow model and migrate a single workflow instance (the mutated bacterium) to the new model. Note that the instance migration is needed because Jim deals with a mutation due to external influences (e.g. UV radiation) where a living bacterium mutates. Now two bacteria types exist in the simulated system.

The modified bacteria model (i.e. the mutation) is illustrated in Figure 1b. It differs from the old model in three main points (marked with numbered arrows): (1) Reproduction happens after 30 minutes (modified `onAlarm`); (2) the new bacteria type is resistant against the

antibiotics (deleted `onMessage` for antibiotics event); and (3) O_2 (oxygen) is now produced as result of the bacterium's metabolism (insertion of a new activity).
Note that the example of bacteria simulation would probably not perform well for a realistic amount of bacteria. This example intends to present a real world scenario that covers all aspects of the concept. Performance and scalability issues are not considered and are not important to understand the conceptual problems this paper deals with.

4 Concurrent Workflow Evolution Policy

Applying the existing approaches of the workflow change classification to our scenario has the following consequences: (1) Ad-hoc: a single bacterium can incur a mutation (e.g. because of UV radiation) but this bacterium cannot conduct cell division/reproduction (its ad-hoc workflow model cannot be instantiated); (2) Evolution (abort): all existing bacteria have to be killed immediately before the mutated bacterium strain can emerge; (3) Evolution (flush): the system has to wait until the existing bacteria strain has died out before the mutated bacterium can emerge; (4) Evolution (progressive): existing and mutated bacteria can co-exist but only mutated bacteria can reproduce. None of these approaches realizes the expected behavior of the described system completely.
An approach is needed that allows modifying a workflow model such that new instances can be created according to the old and the new model. The approach will support the requirements of the example of bacteria modeling as shown in Figure 2. Note that details of the running instances are not important here and are therefore omitted. The bacteria model is modified towards a mutated bacteria model. The workflow instance with ID=2 follows the old model. The workflow instance with ID=1 represents the bacterium that is mutated due to external influences. It therefore needs to be migrated to the new model. Both model versions are active: new instances of both models can be created, i.e. bacteria from both strains can reproduce.

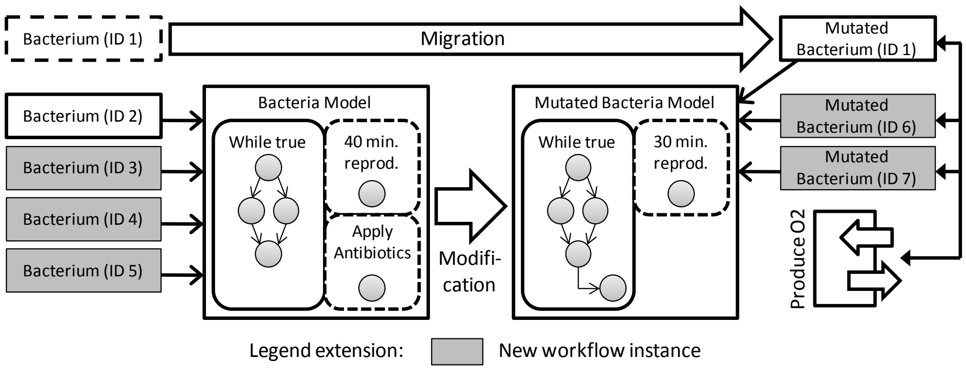

Figure 2: Workflows model of bacterium with the required behavior and required adaptation mechanisms.

We call the needed approach *concurrent workflow evolution.* It is a mix of ad-hoc changes and progressive workflow evolution: A process model is deployed; several instances of that model are running. A new version of that model is created and deployed on the workflow engine that also contains the old model. One or more selected instances can be migrated to the new model version. Not selected instances follow the old model. Both model versions can be instantiated in the following, i.e. are active. Beyond the bacteria scenario, the approach enables scientists to develop workflows iteratively. The current state of an experiment can be preserved by migrating a particular workflow instance to the new model version. Former versions/configurations of experiments can be executed anytime (because they remain active), which may be needed to compare results, throughput, run-time, or utilization of resources.

A workflow engine has to provide a specific operation in order to offer concurrent workflow evolution to its users. The actual implementation of the operation is affected by the workflow language used and the adaptation mechanisms that can be realized by that language. We define the interface for this operation in Listing 1. It is invoked with the modified process model (`modifiedPM`), which is in fact a bundle of several files. The concrete content of the bundle is dependent on the chosen workflow language and engine. The redeployment first suspends all running instances whose identifiers are handed over (`instanceIDs`). Note that ID is used here as generic term that helps to uniquely identify a running process instance. Every instance is checked for compliance to the target model. Concepts for compliance checks can be borrowed from previous work (e.g. [10] and [14]). All selected and compliant instances are then migrated from the old model to the new one. After resuming the instances their jobs are scheduled according to the new model version. The main challenges here are the version management, the migration of running instances to the new model version and the identification of process model versions by workflow clients for subsequent workflow instantiation. The latter two are considered in more detail in the following sections.

In theory, concurrent workflow evolution could be emulated if a new workflow model version is deployed as separated workflow model. But this workaround would not automatically solve the problems of instance migration and addressing the desired workflow model. In fact, it would introduce a new problem: scientists would have to manually deal with the version management by renaming of process models.

```
public boolean deployNewVersion( PMBundle modifiedPM,
                                 List changeOperations,
                                 Identifier[] instanceIDs);
```

Listing 1: Operation to conduct concurrent workflow evolution (in Java-like notation)

Linking Instances to the New Model Version

Migration of workflow instances from one model to another is extensively addressed in existing work. However, practical issues are still open for discussion. Most existing approaches utilize change operations for instance migration [4, 5, 10], but it is not clear *how* these operations are realized. E.g. does an operation `addActivity` mean that a new model version is created? Or is the old model replaced? We need to clarify how exactly these operations are applied and if they are affected by the workflow language used.

As mentioned in the previous section, the `deployNewVersion` operation installs a new process model version and one or more instances of former versions are linked to it, while the original model version is still active. The difficulty when migrating process instances is to handle existing and newly added activities/links in the wave-front correctly. The *wave-front* consists of all workflow elements (e.g. activities, links) that are scheduled for execution or that are currently executed. For already completed elements (i.e. the past) and not scheduled ones (i.e. the future) no problem is imposed because completed activities/links cannot be modified and inactive ones have not been instantiated yet (i.e. every modification is allowed). The actual migration of a process instance from one model to a new one consists of three main steps:

1. Release elements in the wave-front from their respective original model elements
2. Connect elements in the wave-front to the new model elements
3. Conduct change operations for existing or new wave-front elements

Step 1 is straight-forward. It can be done by iterating all wave-front elements and setting the model reference to `null`. In step 2, the difficulty is how to find the correct model element in the new process model. It is not feasible to rely on model element names because they may not be unique (e.g. in BPEL) or the names may be changed in the new model. We therefore propose to store an expression that uniquely identifies the old model element before releasing the connection. This expression can be used to find the correct model element in the new model version. For XML-based workflow languages XPath can be used as expression language. The expression `process[1]/sequence[1]/receive[2]`, e.g., would reference the second `receive` activity in the first `sequence` activity of a BPEL process. For non-XML languages other identification mechanisms have to be utilized. The connection to the new element may be done manually by a human user, which would not require finding the new element to connect to; however this can be considered a special case of the approach above.
After re-connecting the instance elements of the wave-front to the new model version, the migration process is not finished yet. Change operations are needed for two purposes (step 3). First, instance elements of the wave-front may need to be reconciled with the respective model element if it was changed, e.g. an attribute was modified. Second, change operations are needed to create new instance elements in the wave-front if the new model version contains new elements that belong to the wave-front of the considered workflow instance. For example, an outgoing link was added to an already completed activity. The link's target has to be an activity in the instance's future (otherwise the link would belong to the instance's past). Hence the link belongs to the wave-front. In this case the appropriate change operation would explicitly trigger the execution of the link. This case was already recognized in [10]. The authors do not discuss the problem that arises when evaluating the transition condition of the link. It may not be semantically correct to evaluate the link based on the workflow instance's current data. This imposes the requirements on the execution infrastructure to keep data needed for link evaluation. Typically, this can be realized with the help of the audit trail [1]. The auditing component may have to be extended to explicitly store the needed information.

Table 2: Design of the change list with two examples for BPEL

Change Operation	Subject Type	Model Element
Add	Activity	`process[1]/sequence[1]/invoke[3]`
Modify	Link	`process[1]/flow[1]/links[1]/link[2]`

An important issue we need to address is related to the question how the engine knows which change operations on a process instance to execute automatically (manual instance migration is not considered here because scientists should not notice such technical details at all). The engine just has the old and new process models at hand and needs to derive the needed change operations. Both process models could be intersected to get the difference between them. After that it can be derived which activities/links were added/deleted/modified. From these differences and the current wave-front of the instance the needed change operations can be inferred. The problem here is that the semantics of changes could get lost. For example, if activity A was renamed to B, an intersection would state that activity A was deleted and B was added. That would enormously influence the compliance of workflow instances to a changed model (renaming of an activity in the wave-front is possible; deletion is not possible). We therefore foresee an additional parameter for the `deployNewVersion` operation that is a list of all changes that are conducted to transform the old to the new model. If a modeling tool is used to adapt the process model, such a list can be easily created. The modeling tool then needs to be extended by a component that tracks the conducted changes [3]. The change list can also be part of the provenance record [15] that is very important in the field of scientific workflows to show the origin of data to scientists.

Table 2 shows an example for a change list. Note that the list contains elements with dependencies and therefore some of the elements impose an ordering on the respective change operations. *Change operations* can be `add`, `delete`, and `modify`. Note that the `move` operation can be represented by the two operations `delete` and `add`. *Subject types* are dependent on the workflow language. For graph-based languages at least activities and links must be supported. The *model element* uniquely identifies the element the change operation applies to. In XML-based workflow languages this could be realized with XPath expressions.

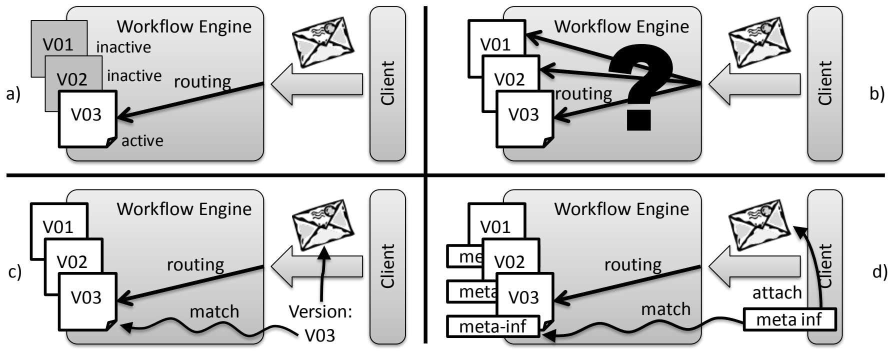

Figure 3: Routing requests to workflow model versions

Addressing Process Model Versions

The second problem, unique identification of different process model versions, is typically circumvented by unique, engine-dependent version numbers that are attached to the process model name. However, existing engines use these proprietary version numbers for *engine-internal purposes.* They are not used to route messages to different process model versions. This simple solution works as long as only one of the model versions is active. The number of the model version is not needed to create a new instance of a workflow model version because only one version can be instantiated (see Figure 3a). If several versions are active at the same point in time, the engine cannot route an instantiation request to the correct model version (Figure 3b). The client could rely on the version number to address the desired process model version (Figure 3c). This would increase the coupling between client and engine because different engines could make use of different formats for versions (e.g. integer values, strings, complex types). Another solution is based on meta-information to find the correct process model (Figure 3d). Meta-information has to be specified by the workflow modeller (or can be generated automatically) and is then part of a process model bundle. Key-value pairs are an appropriate implementation for meta-data. In our example of bacteria, meta-data could be

`bacterium: E. coli; resistence: bla`. That means the process model describes an Escherichia coli bacterium with a resistance against beta-lactam antibiotic (bla). Such functionality is also applicable to business workflows, such as a loan approval process for private customers: `product: loan; customer-type: private`. The engine has to guarantee uniqueness of meta-information for every workflow model. Workflow clients can use this meta-data to address the desired workflow model. It is required that clients are enabled to query and—with administrator rights—change meta-information of process models. Appropriate operations have to be provided by the engine.
That means clients need to specify an additional parameter (the version number) or meta-information to create a new workflow instance. The specification of this parameter should be optional. New policies for workflow invocation can be invented (e.g. newest/default version wins, specified version wins). Note that subsequent communication between a client and a workflow instance can be carried out without having to specify the version number/meta-data because common correlation mechanisms can be used for message routing.

Application to Business Scenarios

The concept of concurrent workflow evolution is not restricted to scenarios in scientific workflow management. An application in business workflow management also makes sense. Imagine a company that produces cars. There are different types of cars—each is built according to a process model. It may happen that a customer has special requirements that are not reflected in the original process model of that car, e.g. tinted window panes or an additional cup holder. These needs have to be accounted for during car production, i.e. at runtime of the process model. Maybe the customer's request is so useful that the company decides to provide it to other customers, too. A new product variation is born. The concept of concurrent workflow evolution enables both to change the customer's car on-the-fly and to produce more of these changed cars in future while production of the standard car is continued.

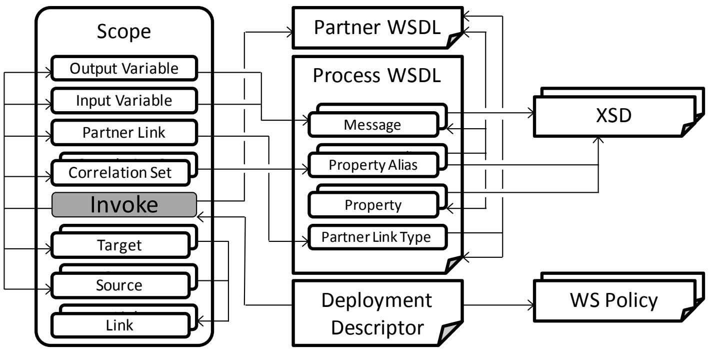

Figure 4: Dependencies of an `invoke` activity in the deployment bundle

5 Implementation with BPEL

Concurrent workflow evolution is an abstract concept that can be implemented with different concrete workflow languages. In this section we discuss a realization with BPEL as it is broadly accepted in industry and research. When modifying BPEL processes, it is important to have in mind that change operations on nodes and edges (e.g. insert edge, delete node) are not sufficient to satisfy the complexity of the language. Many activities (e.g. `receive`) have

dependencies to partner links, variables, correlation sets, or message exchanges. Moreover, interaction activities (message sending/receiving activities) can have semantic dependencies (e.g. invoke-receive pairs). Finally, there are several kinds of handlers that are triggered by different kinds of events (e.g. exceptions).
Especially, interaction activities (indirectly) depend on several documents of the deployment bundle. Figure 4 illustrates how an `invoke` activity can be linked in the bundle. Due to these complex dependencies the change operations needed to change running BPEL process instances are more complex than in simple graphs with nodes and edges. Especially, it is needed to provide different change operations for different activity types. E.g. inserting a `pick` seriously distinguishes from inserting an `empty` activity.

Change Operations for BPEL Processes

Concurrent workflow evolution foresees the deployment of a new model version. Particular workflow instances are bent to the new model version (usually, exchanging IDs in the workflow engine's DB is sufficient, depending on the engine). The navigator then schedules the instance events according to the new version. Change operations are needed to modify activities in the wave-front. In the following we present the most important change operations.
Scope activities are initialized when they are activated. This initialization encompasses initializing partner links, variables and correlation sets as well as the installation of fault handlers (FH), event handlers (EH) and termination handlers (TH). If the mentioned elements are changed in a currently running scope, the scope has to be re-initialized.
Adding `onAlarm`s in installed EHs or running `pick` activities needs special attention. New timers have to be installed that observe whether an `onAlarm` has to be triggered. There is a timing problem when adding `onAlarm`s with duration expression the timer is started at workflow instance resume time and not when entering the `pick`/EH. In order to balance the time difference the activation time of picks/EHs has to be stored so that the remaining duration of new `onAlarm`s can be inferred. When deleting an `onAlarm` its timer has to be deregistered.
Adding `onMessage`s to an activated `pick` or `onEvent`s to an installed EH means that the workflow instance accepts additional messages when it is resumed after migration. Change operations are needed to register these message receivers with the engine. This is again engine-specific (e.g. in the open source engine Apache ODE [16] new message routes have to be registered). Deletion of an `onMessage`/`onEvent` requires deregistering its message receiver.
When the expression of a running `wait` activity is modified, the timer has to be updated by the change operation. In case of a duration expression the activation time of the `wait` is needed to derive the correct residual duration (similar to a changed `onAlarm` with duration).
BPEL implements WSDL 1.1 request-response operations with activity pairs consisting of an incoming message activity (IMA) and a `reply` activity. If an IMA is already executed in a process instance, deleting the corresponding `reply` activity has implications on the WSDL interface of the process and eventually on the workflow client.

Activation of Several Process Model Versions

In BPEL, process models are exposed/made available as WS. Hence sending a message to a BPEL process is in fact the invocation of a WS. Existing workflow engines make all process model versions available via the same WS endpoint, i.e. there are two or more WS

implementation behind the same WS endpoint, which works fine if only one model version is active. If several versions are active, it is unclear which of the models processes a message. The problem can be solved by applying the two concepts for addressing model versions.
Note that in [17] a versioning extension for BPEL was already proposed that deals with version management. As opposed to this, we focus on how an engine can route a message to a particular BPEL process model version. A BPEL extension is not needed for this purpose.

Routing by version number. The message types of the operations provided by the process model could be extended by an additional field for the version number. Clients can then specify the desired version. Another possibility is to provide the process version under a unique endpoint address by attaching the engine-internal version number, e.g. `http://.../bacteria/V02`.

Routing by meta information. This can be realized with the help of WS-Addressing [18]. The reference parameter section of an EPR can be used as container for meta information that help routing a message to a particular process model version. As described above, the meta information are key-value-pairs. We realize this with a `cwe:property` element with attributes `key` and `value`. Since a process model version can have several of these key-value-pairs, we foresee a `cwe:modelVersionRef` wrapper that aggregates all of these pairs. Listing 2 shows how this meta information is inserted as header into a SOAP message. The process model version is identified by the two properties bacterium type and antibiotics resistance. The BPEL engine must be able to process this header and route the message to the correct process model.

```
<soap:Header xmlns:cwe=”http://concurrent/workflow/evolution”>
  <wsa:To>http://.../bacteria</wsa:To>
  <cwe:modelVersionRef wsa:isReferenceParameter=”true”>
    <cwe:property key=”bacterium” value=”E. coli”>
    <cwe:property key=”resistance” value=”bla”>
  </cwe:modelVersionRef>
</soap:Header>
```

Listing 2: WS-Addressing reference parameter header in SOAP messages

6 Conclusions and Outlook

In this paper we presented the concept of concurrent workflow evolution. It is a new flexibility mechanism that can be seen as a mixture of ad-hoc changes and progressive workflow evolution. A new, adapted version of a workflow model is deployed on a workflow engine. One or more running instances of the old model are migrated to the new one. Unlike existing approaches, both the old and the new model can be instantiated. To the best of our knowledge, none of the existing approaches allows two or more model versions to be active concurrently (even after all instances that are running at the time of the change have been completed). A scientific and a business scenario point up the usability and need of the concept in practice. We have proposed solutions for the two major problems that emerge. The first problem, how to address the correct workflow model version for instantiation, can be solved by a unique name or by meta-information that must be provided by the client. The second problem, how to know the concrete change operations needed for migrating running instances, can be addressed with a list of changes that were conducted to create the new model version from the old one. The

needed change operations can be derived from that list. This list can be easily created in an appropriate modeling tool. The concept can be implemented with arbitrary workflow languages. As an example we showed a realization based on BPEL. We are currently working on a prototype for concurrent workflow evolution of BPEL processes based on the open source workflow machine Apache Orchestration Director Engine (ODE).
Our future work will focus on the concrete compliance rules that have to hold when migrating the workflow instances from one model version to another. We want to allow instance migration for as many instances as possible. Additionally, we have to think about concepts for workflow model version management, especially in the used workflow modeling tool.

Acknowledgements. The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

References

[1] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques*, Prentice Hall, 2000.
[2] M. Sonntag et al., The missing features of workflow systems for scientific computations, in: Proc. of the 21st IASTED Int. Conf. on Modelling and Simulation, 2010.
[3] M. Sonntag and D. Karastoyanova, Next generation interactive scientific experimenting based on the workflow technology, in: Proc. of the 3rd Grid Workflow Workshop (GWW), Software Engineering Conference, GI-Edition LNI, P-160, 2010.
[4] F. Casati et al., Workflow evolution, in: Proc. of Int. Conf. on Conceptual Modeling, pp. 438-455, 1996.
[5] W. v. d. Aalst et al., Adaptive Workflow – On the interplay between flexibility and support, in: J. Filipe, editor, Enterprise Information Systems, pp. 63-70, Kluwer Academic Publishers, 2000.
[6] H. Schonenberg et al., Process flexibility: a survey of contemporary approaches, in: Proc. of 4th Int. Workshop CIAO! and 4th Int. Workshop EOMAS, Springer, 2008.
[7] S. Rinderle et al., Correctness criteria for dynamic changes in workflow systems – a survey, in: *Data and Knowledge Engineering*, 50, 9-34, 2004.
[8] Oracle BPEL Process Manager, `http://www.oracle.com/us/products/middleware/application-server/bpel-home-066588.html`
[9] Bonita Open Solution `http://www.bonitasoft.com/products/BPM__workflow_overview.php`
[10] M. Reichert and S. Rinderle, On design principles for realizing adaptive service flows with BPEL. Proc. of EMISA 2006, GI Lecture Nots in Informatics, LNI P-95, 2006.
[11] Weber et al., Change patterns and change support features in process-aware information systems, in: Proc. 19th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2007), 2007.
[12] Weber et al., Aktuelles Schlagwort: Process Change Patterns, EMISA, 27 (2). pp. 45-51. 2008.
[13] D. Karastoyanova et al., Parameterized BPEL processes: concepts and implementation, in: Proc. of the Int. Conf. on Business Process Management, 2006.
[14] Rinderle et al., Flexible support of team processes by adaptive workflow systems, Springer, Distributed and Parallel Databases , Vol. 16, No. 1, pp. 91-116, 2004.
[15] L. Moreau et al., The open provenance model core specification (v1.1), *Future Generation Computer Systems*. (In press)
[16] Apache Orchestration Director Engine (ODE), `http://ode.apache.org`
[17] M. Juric et al., WS-BPEL Extensions for Versioning, in: Information and Software Technology 51 (2009), pp. 1261-1274.
[18] W3C, Web services addressing 1.0 – core, W3C Recommendation, 2006, `http://www.w3.org/TR/ws-addr-core/`