Extending Web Service Flow Models to Provide for Adaptability

Dimka Karastoyanova, Alejandro Buchmann

Technical University Darmstadt
Computer Science Department
Hochschulstrasse 10
64289 Darmstadt, Germany
dimka@gkec.tu-darmstadt.de
buchmann@informatik.tu-darmstadt.de

Abstract. Web Service compositions combine characteristics of both the Web service (WS) and workflow technologies. The workflow technology is good at explicitly defining integration logic. Web services bring in standardized service description and communication over the Web, and thus provide for seamless integration across organizational boundaries. The two technologies mutually amplify their benefits in this combination. Nevertheless, WS compositions (WS-flows) inherit also some of the deficiencies both technologies exhibit. Notably, traditional workflow technologies lack sufficient support for workflow adaptability, which is a prerequisite for ensuring business processes flexibility.
In this paper we argue that any standardization activity in the field of WS-flows must gravitate around a unified process meta-model. Additionally, adaptability should be considered as an inseparable part of the WS-flow meta-model constructs. We propose process meta-model extension constructs that accommodate WS-flow adaptation in implementation independent manner. The presented constructs allow for run time modifications of participating WS instances and WS types, and changes in business logic.

1. Introduction

The focus of this paper is on Web service compositions. Composing Web services is an approach relying on the advances in the development of workflow management and the Web service (WS) technologies. We use the terms WS composition and WS-flow (Web Service Flow) interchangeably, to denote composite WSs created using a process-oriented approach, similar to the one used in traditional workflow. However, due to the WS-specific features composing simple tasks into complex processes in a WS environment is somewhat different from traditional workflow. Additionally, WS composition specifications [12, 7] put their standardization focus on specifying unambiguously a process model, rather than imposing restrictions on the implementation of process execution environments.

One of our objectives here is to discuss the need for a *unified WS-flow meta-model* for promoting standardization, portability of WS-flow schemata, implementation independence and technology leverage. We are also convinced that *adaptability*

support should also be presented as an inseparable part of process definitions. In the same time, the standardization focus should be kept and implementation independence maintained. We pursue support for adaptation of WS-flow behavior at run time by proposing several meta-model extension constructs. These constructs can accommodate multiple implementation approaches and give the users and developers the freedom to explicitly direct the way in which a WS-flow has to react to changes in environment factors.

The paper is organized as follows: In section 2 we emphasize the need for a unified WS-flow meta-model and the benefits of having such a model. We argue that this model should also provide constructs that incorporate adaptability in a standardized way. We briefly discuss the way adaptability has been addressed in traditional workflows in section 3. Section 4 dwells on the effect the WS-specific features have on the techniques for addressing process adaptability. Based on the revised range of factors influencing the normal execution of WS-flows we present a classification of possible approaches to WS-flow adaptation in reaction to changed conditions (section 5). Our goal is to group adaptability approaches in a generic way independent of implementation features. In section 6 we introduce the meta-model extension constructs for adaptability. The constructs are meant to present, in implementation independent fashion, run time changes in the WS-flow behavior such as: WS instances swapping, changing service types performing on behalf of a process, and modifications in process logic. The implications and advantages of these extension constructs are briefly discussed. Summary and conclusions are presented in the closing part.

2. On the need for a unified WS-flow meta-model with built-in flexibility

The Workflow Management Coalition (WfMC) [33] standardizes the design of workflow management systems (WfMSs) with the purpose to make them interoperable. The WfMSs vendors created their own workflow models, the corresponding languages and execution environments; usually they cannot interoperate [2, 8].

Still, there are similarities in the languages used to describe workflows. Generally, workflows are described in terms of several aspects or dimensions [22, 26, 8] including tasks, control flow, data flow, participants (humans, resources, application programs), organization hierarchy and others; but there is no consensus on how many dimensions are necessary and sufficient to describe workflows. The existing WS composition languages reuse the concepts of workflow and define XML elements for different activities (tasks), control and data flow, and participants [7, 12]. The advantages of WS technology have been utilized. The WS technology alleviates the difficulties in creating processes because of its standardization focus (see section 4). Therefore no participants other than WSs have to be accounted for in the WS-flow definitions. Organization specific characteristics are also abstracted. Apart from leveraging the standardization focus of WSs, WS-flows promote standardization further. All standardization efforts target *standardization for process interoperability*

as opposed to standardization for interoperability among WfMSs. Even though the existing WS composition specifications define how WS-flow definitions have to be created, in a manner independent of the execution engines, there is still room for improvement. Currently, all specifications focus on the definition language specifics only. We are convinced that all standardization activities related to WS-flows have to gravitate around a *unified*, common *process meta-mode*l. This meta-model should describe the structure of elements to be used to create a WS-flow and their relationships.

Having a *unified meta-model for WS compositions* at hand could bring benefits for several reasons, listed next:

- *automation* of WS-flow development – standardized model constructs can be used by WS-flow development tools to generate process definitions. Furthermore meta-model constructs can be combined into reusable units of code and functionality – the so-called *templates* [25, 29]. Automating WS-flows production could simplify and accelerate development; the main principle is code and functionality reuse.
- *portability* of definitions is promoted by the unified way of describing WS-flows. Having a single format for describing WS-flows allows different execution environments to be used to carry out a WS-flow definition. For this, however, all constructs have to be defined in an implementation independent form in the process language meta-model.
- *translation to existing languages* – a common meta-model would allow for translating the WS-flow definitions in existing workflow and WS-flow languages based on model mappings; we believe this approach is much more precise than using syntax mappings. Technology and workflow engines can thus be reused.
- *dynamic features* of processes and *adaptability* can be incorporated directly into the model using its constructs. All adaptation primitives of the engine and their implementation specific features are kept transparent, i.e. hidden from process developers and users.
- *freedom to choose* implementation paradigm and reuse legacy process *execution environments* would be given to companies, provided all environments comply with the meta-model – this is not possible in contemporary workflow and WS-flow technologies [8, 2].

To make the community agree on a model would require not only providing a unified way of describing WS-flows in terms of an agreed-upon number of perspectives. To provide both the companies and their processes with flexibility imposes the need for incorporating adaptability in the meta-model constructs. To put it into perspective, the adaptability of traditional workflows has been enabled by *primitives implemented by the WfMSs* – the adaptability has been ensured by vendor specific implementations. There is no standard way to approach adaptability in traditional workflow management; to the contrary, there is abundance of WfMSs and multiple implementation approaches.

In this respect our main concern is extending the process model features with the ability to flex in reaction to business and technical factors of the environment that change in random fashion. The meta-model extension for adaptability should comprise constructs capable of hiding implementation specifics while allowing for

process definition modifications, instance level changes etc. (Table 1). This is quite in sync with the strategy of the BPMI.org initiative [10] for creating extension layers to BPEL.

The advantages of having model constructs for adaptability include:

- *hiding the implementation specifics* of execution environment primitives that do the actual adaptations.
- standardized way of *describing and directing how a process and/or its instances have to change*, including the ability to specify QoS criteria at run time. As a result the process definitions remain portable, while being rendered adaptable.
- *freedom to choose engine*, and *implementation paradigm* is preserved. The extension constructs are meant to accommodate multiple approaches to adaptability, even those used in the field of traditional workflows.
- ability to accommodate the results of the progress in the following research areas: *semantic description of WSs*, classification according the WS semantics, description of *policies for service selection*, and *quality of Web services models*.

To the drawbacks counts the increased complexity of coding the processes. It is due to the fact that developers have to code additional activities and moreover to understand their purpose; however, this could be overcome by providing tools for automated WS-flow production. The tools in turn depend on the meta-model.

3. Approaches to adaptability in traditional workflow technology

The workflow technology has evolved to a mature EAI technology in the last decade. Most vendors realized the great benefits from using workflows, and therefore there are a number of vendor-specific workflow models and products. The standardization effort of the WfMC [33] did not have the desired success because of its objective to *standardize for interoperability among workflow management systems* (WfMSs), rather than to provide an unambiguous process model specification [6, 2]. As a result there is no standard workflow model. The numerous vendor-specific products use their own overlapping models and definition language.

Similarly, each product vendor addressed the need for adaptability and flexibility of workflows in different ways. Variety is abundant with respect to products [11, 22, 27] and implementation approaches [16, 13, 11, 18] targeting development and execution of flexible workflows. But the *support for adaptability is still insufficient* and most importantly it is *locked into vendor specific implementations*.

Together with this there have been attempts to both classify the kinds of changes that might occur in the environment and thus influence the workflows, and to produce a list of corresponding approaches to adapting to those changes [17, 3, 18].

In [17] and [3] the authors classify the levels/perspectives of a workflow related to change. [17] states that workflows have to be able to adapt to changes on several levels: domain level (changes in business context), process level (process schema and tasks changes), resource level (software, organization and data model changes) and infrastructure level (software and hardware infrastructure). In [3] the authors provide a similar classification except that there is an additional task perspective, which is a part of the process level in [17]; and the domain level is not included.

So far in those and similar classifications the fact that each process is created using the constructs of a *meta-model* has not been considered. Domain-specific models reside on the same level as the meta-models; they are actually to be understood as domain-specific extensions of the meta-model.

The process perspective is the one that prescribes the schema of a workflow. Most of the changes in the environment lead to changes in the schema of the processes (e.g. tasks are added or deleted, control flow paths are added etc.). Dealing with process schema adaptation has been paid the greatest attention so far. Generally, the types of adaptations to be performed on the process level of a workflow in reaction to environment change are grouped according to whether they are build-time or run-time changes, process schema or process instance change, single instance change or multiple instance changes; and combinations of these criteria [3, 17, 18].

4. Adaptability in WS-flows

This section provides a short overview of the basic WS characteristics and how they affect the approaches to WS-flows adaptability. Based on this and on the discussion from the previous section we identify the perspectives of a WS-flow definition relevant to adaptation. Subsequently, we classify the approaches to WS-flows adaptability (section 5).

4.1 Effect of the WS-specific characteristics on the approaches to WS-flows adaptability

WS compositions inherit the intrinsic characteristics of WSs. WSs expose *stable, unified interfaces* described in WSDL [32] (the de facto standard for WS description). A WS interface describes the service functionality in terms of the messages it can consume and produce, their parameters, and the types of the exchanged data; this is because the WS paradigm assumes communication by means of messaging to ensure *loose-coupling* among services. Thus the functionality of a WS is presented in a platform- and language-independent manner. Hence the WS interface hides the implementation specifics of the functionality it exposes.

In a WS-flow tasks are performed only by WSs. Since all WSs are described in a unified manner, WS-flows do not distinguish explicitly between human participants, resources or applications. Therefore changes in the types of participants and their infrastructure can be ignored in the definitions of WS-flows. So changing the actual resource or application behind a WS interface will not affect the WS-flow definition. As long as a participant provides the functionality defined by the exposed WS interface, the results of any reconfiguration of its infrastructure remains hidden for the process, too. This means that any changes in the participants' infrastructure and application specifics will not affect the WS-flow schema and its instances.

In traditional workflows all participants are assigned a role related to an organizational model. However, the organizational perspective in a WS-flow becomes irrelevant. The participating WSs are not assigned a role with respect to any organizational model, because WSs interfaces hide these specifics, too. Therefore

WS-flows definitions do not account for any organizational model. And if the definition ignores the organizational specifics then changes in the organizational models can affect neither the WS-flow definition nor its instances.

In conclusion we state that due to the characteristics inherent to WSs WS-flows do not have to be capable of adapting to changes in the types of the participants, their role in an organization and the infrastructure.

4.2. WS-flows dimensions relevant to WS-flow adaptability

In the previous sub-section we claimed that we can ignore changes in the infrastructure, software components, data and organizational models as possible reasons for WS-flows adaptation. Based on this claim we restrict the number of groups of changes, which a WS-flow must react to, to the ones shown in Figure 1. This is an adapted version of the classification of workflow adaptation layers shown in [17].

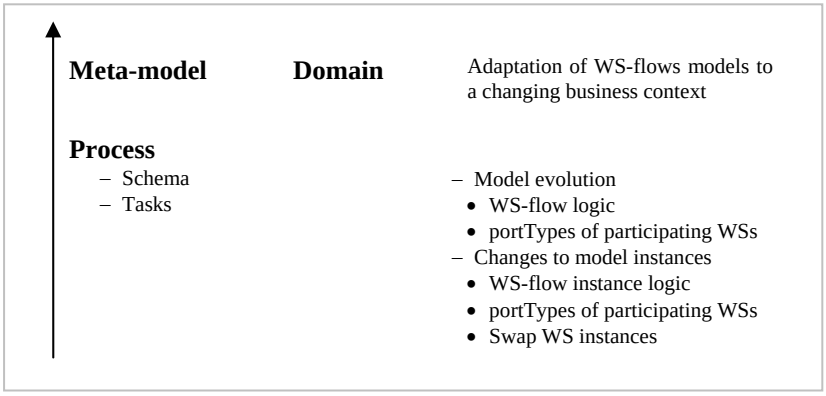

Figure 1. Layers of WS-flows adaptation.

We reduce the number of adaptability relevant perspectives to the process perspective and meta-model perspective. We are convinced that a WS-flow must be able to react to the need for *changing its process definition* (process layer). This means that the WS-flow has to be able to modify the tasks it has to perform and their execution order, and the types of the participating WSs. These changes are possible on schema and instance level. Additionally, *changing meta-models* and their extended domain-specific versions might also be required due to altered business and legal requirements.

Further in this paper we shall focus on the process layer only.

In summary, the classification of workflow adaptation layers from [17] can be simplified significantly in the context of WS-flows. In other words, WS-flows can react to changes in the environment with the same degree of adaptability, while using simpler approaches than the ones used in traditional workflows.

5. Classification of approaches to WS-flows adaptability

In section 4 we concluded that WS-flow can be rendered equally adaptable to changes in the environment even when less complicated techniques are applied. Next, we elaborate on the approaches that are sufficient to render a WS-flow flexible in reacting to modified conditions. We group these approaches as shown in Table 1. This classification shows adaptation approaches that are to be used to modify WS composition models and instances, i.e. the *process layer* (Figure 1).

A WS-flow can react (or foresee how to launch a reaction) to the factors presented in Figure 1 either during the time it is being modeled and created – *build time*, or during its execution – *run time*. In each group changes can be performed either on the model or instance level. By changing a process definition at build time developers can react to any kind of changes in the business environment. But the business environment evolves constantly while most processes are being executed. A higher degree of adaptability would be achieved if WS-flows could flex to changes appropriately while being at their execution phase. The adaptations that must be enabled at run time should avoid any termination of the process instances; terminating processes might result in losing WS-flows history, sensitive data, time, and eventually customers.

Table 1. Classification of approaches to WS-flows adaptability.

		Evolution level	**Approach**
WS-flow life cycle phase	*Build time*	Model evolution	Any kind of change possible
		Instance evolution	Flexibility by selection [18]
	Run time	Model (schema) changes	Change in: – portType – control flow – data flow
		Instance changes	Change in: – WS instances – Process logic: • portType • control flow • data flow

On the schema level at run time changes can be enforced on the business logic of a composition and on the types of WSs it invokes.

The instance changes at run time are the ones specifying how the execution flow of one or more WS-flow instances has to be diverted. This may be done by altering the instances of WS abstract descriptions performing on behalf of a process, or by modifying the process logic – this includes changing portTypes, control flow and data flow. Runtime changes are also associated with the term flexibility by adaptation [18]. In [18] the authors use the term to denote the approach of creating a process schema version, which includes the desired changes.

The classification on Table 1 puts the framework for the discussion in the rest of the paper. This classification is the basis for specifying generic meta-model constructs

that would be used for creating WS-flows with built-in adaptability in a standardized manner.

6. Model extensions for adaptability

The approaches we would like to enable with the meta-model constructs introduced in this section are summarized in Table 1. Here we pay attention to the run time adaptations of WS-flows, since they are the more challenging problem; run time changes are not sufficiently addressed even in traditional workflows. We use code listings in BPEL [12] to demonstrate the model constructs. One reason for using BPEL syntax is that it is the specification that has gained considerable acceptance in the WS community. Moreover, it is the only WS-based process definition language for which execution environments already exist, either incorporated in workflow management systems that additionally support other models and languages [20, 21], or as stand-alone environments supporting exclusively BPEL [19, 28, 4, 31]. Besides, in keeping with the principles of the WS paradigm we prefer to go for reusing and extending existing meta-models and syntax (and thus leverage technology and implementations), instead of creating new model and yet another language from scratch.

6.1. Enabling WS instance swapping

To enable *WS instances swapping* we introduce the find_bind construct. This construct has to be used to address WS-flows *instance level modifications* at *run time* (Table 1). It is mapped to an activity of the same name in the process language syntax; it can be easily appended to the extensible BPEL syntax. The <find_bind> activity is meant to be executed before each WS invocation in a WS-flow; in BPEL these are the activities <invoke>, <receive> and <reply>.

The “find and bind” mechanism involves [6, 23]:

- a look up of WSs instances complying with the portType specified in the subsequent invocation activity,
- a selection of a single WS instance and
- binding to it.

The search for compliant WS instances is performed in a UDDI registry [9]. The choice of a single WS instance must be directed by the users by means of criteria such as service availability, price, and others related to the quality of the service (QoS). This activity is meant to and is capable of accommodating selection according to semantic description of the service [15], too. It is necessary to introduce such an activity because currently the WS-flows definitions include references to specific WS instances. Respectively, the existing execution environments (e.g. [4, 19]) require references to specific WS instances during WS-flows deployment. We find this approach inflexible; therefore, introducing a policy-based selection of and dynamic binding to WS instances at run time has to overcome the problem of hard-coding

service instances. An example of the <find_bind> activity is presented in the next code listing.

```
<process name="ConvertCurrencyBP">
<!-- details -->
  <find_bind>
   <find_businessService()/>
   <get_bindingTemplate()/>
   <apply_policy()/>
  </find_bind>
  <invoke name="ConversionRequest"
    partnerLink="converter"
    portType="CurrencyConvService"
    operation="usd2eur"
    inputVariable="C_and_Rate"
    outputVariable="result"/>
<!-- details -->
</process>
```

Listing 1. Find and bind mechanism - representation in BPEL.

There are several implications of using this activity. A QoS model for WSs and a corresponding notation for QoS classification are needed to support meaningful WS instance selection. A means to describing choice policies, based on various criteria is also a must. To date, these are unavailable. Currently, only simple policy-based selection can be performed according to service availability, price, and service location. Storing binding information for the selected WS instance is also an important issue. The service instance URL can either be stored explicitly in a process variable or taken care of transparently by the execution environment. To the flaws of the "find and bind" approach counts the additional HTTP call to the UDDI registry for each service invocation. In this respect optimization of the mechanism is necessary to maintain performance; for instance, performing the finding of services and binding to them whenever a criterion falls under a threshold value; optimization is related to the implementation of the mechanism though. For more details on this construct refer to [23, 29].

6.2. Enabling process schema changes at run time – <evaluate> activity

In Table 1 we introduced two types of possible changes in WS-flow schema: portType changes and process logic modifications (control and data flow).

Changes in the process schema and port types may be performed for only some of the instances of the WS-flow or for all of them. Therefore we believe that changes on both model and instance levels can be made possible using one and the same model construct. Therefore, in the remaining part of this section we introduce the `<evaluate>` activity. Currently it has two different versions. Our future research targets the specification of a single activity definition that would serve both types of modifications.

6.2.1. Changing service types at run time

Sometimes a change in the service type might be required for a WS-flow to be able to reconfigure in reaction to changed conditions; therefore mere swapping of WS instances complying with portTypes would not suffice. Here we introduce a meta-model construct – the <evaluate> activity – that allows for changing the types of services performing on behalf of a WS-flow. The activity encloses an invocation activity. It can be appended to the BPEL syntax. Basically, this construct has to allow the users to specify an alternative portType for the execution of a WS invocation activity. The idea is similar to the one presented in [30] where the eval() function is defined as a way to execute a function with new parameters values unknown prior to run time. The example representation of the <evaluate> activity is given next; irrelevant details are omitted.

```
<process name="ConvertCurrencyBP">
<!-- additional details -->
<!-- evaluate -->
  <evaluate
          name=”ConversionRequest”
          activated=”true”
          portType-new=”nsws2:ConvertCurrencyService”
          operation-new=”usd2eur”>
<!-- invoke Converter -->
          <invoke name="ConversionRequest"
             partnerLink ="converter"
             portType="nsws1:CurrencyService"
             operation="usd2DM"
             inputVariable="Currency_and_Rate"
             outputVariable="result"/>
  </evaluate>
<!-- additional details -->
</process>
```

Listing 2. Example of the <evaluate> activity in BPEL syntax.

In Listing 2 the <evaluate> activity encloses an <invoke> activity, which performs an operation (usd2DM) on a WS described by a particular portType (nsws1:CurrencyService). To be able to change the originally specified portType attribute value in the <invoke> activity, the users have to use a tool that allows access to the WS-flow instance under consideration. The tool has to allow the users to specify new values for the portType and operation, which are then to be used in the enclosed invocation activity's execution. The user is given the opportunity to change the type of service used by a process. In our example, the user could specify new values for portType (nsws2:ConvertCurrencyService) and operation (usd2eur) and thus change the example currency conversion service so that it calculates conversion from US dollar to Euro, instead of US dollar to German mark (operation="usd2DM"). Note that Listing 2 presents the new values of the parameters provided by the user at run time. However, the attributes of the <evaluate> activity should be initially populated by the developer with default value for the alternative portType (i.e. at build time). The values of all attributes of the <evaluate> activity substitute the values of attributes in the enclosed

invocation activity; the attributes of the enclosed activity that do not have an alternative value specified in the `<evaluate>` activity keep their original values.

There are several ways in which the change of portTypes (i.e. in the process definition) can be implemented. Some existing approaches from the field of workflow management use code generation for the new alternative and migration of (one or more) process instances from the old definition to the newly generated one [11, 18]. A procedure, similar to the one used in Active XML [5] can also be involved for dynamic code generation. Reflection has been applied in workflow technologies for the same purpose [16]; process tasks (activities) can be carried out with parameters different from the originally specified. Aspect-oriented programming (AOP) is another possibility [13]. Existing WS-flows execution environments can be extended with aspects implementing the semantics of the `<evaluate>` activity. A simple scenario in this respect would be to execute an aspect every time the schema interpreter meets an activated `<evaluate>` activity and thus perform the changes in process instances according to the parameters given by the user.

Changing a portType in a process definition requires also binding to WS instances not known at build time. This leads to using the find and bind mechanism, expressed by the `<find_bind>` activity.

We are aware that producing a WS-flow definition becomes a bit more complex because the `<evaluate>` activity must be coded and moreover understood. But coding can be automated; this is one of the goals of the ReFFlow project [29] in its part related to automating WS-flows production. Besides, we are certain that to include a single activity to the existing models that can accommodate different implementation approaches to adaptation is a much more appropriate and feasible solution than extending the existing models with multiple activities for every possible implementation approach. For instance, special-purpose extension activities can be defined to map the existing aspect-oriented programming features [13]; approaches related to semantic description of WSs can also be represented as separate meta-model constructs. We consider these and similar approaches related to the implementation of the engine. Therefore execution engines that do not implement one of these adaptability approaches would not be able to interpret such WS-flow definitions; this in turn hampers technology reuse.

Even though some changes in the process definition can be performed using existing traditional workflow approaches (i.e. no need to reinvent existing implementation approaches), including the `<evaluate>` activity is both appropriate and meaningful. We gain considerable flexibility by including such a construct into the meta-model. It can accommodate the existing approaches to adaptability and hide their complexity and implementation specifics from the WS-flow developers and users. The basic idea here is to find the best possible way of representing portTypes and process logic changes in a general way. Such a general representation can enable the use of various implementation paradigms in different execution engines, and in the same time keep the process descriptions standard/unified, and thus reusable and portable on the different execution environments. Moreover, the user gains additional freedom by being allowed to specify attribute values as needed or desired in a standard way and all this at run time, regardless of the engine and the implementation approach used.

6.2.2. Changing process logic at run time

We plan to extend the `<evaluate>` activity with additional attributes in the near future. Our intent is to provide for such changes in the definition of WS-flows that do not affect portTypes and operations only (see section 6.2.1.), but also allows for replacing a collection of activities with another one. When a portion of the process definition is substituted with another one the process logic (both control and data flow) inevitably changes. Doing so could provide an alternative path in the process execution, which has not been foreseen and specified in the original process schema. An example of a possible extension to the `<evaluate>` activity is shown next (Listing 3).

```
<process name="ConvertCurrencyBP">
<!-- additional details -->
<!-- evaluate -->
   <evaluate name="Conversion"
         activated="true"
         substitute="T1">
         <!-- Conversion sequence -->
         <sequence name="Conversion">
            <invoke partnerLink="parter1"
                    portType=" " … />
            <assign> <copy> …
                     </copy></assign>
            <invoke partnerLink="parter2"
                    portType=" " … />
         </sequence>
   </evaluate>
<!-- additional details -->
</process>
```

Listing 3. Substituting process logic.

The `substitute` attribute of `<evaluate>` activity takes as a value an identifier of the new piece of logic. The example from above expresses the following meaning: the set of activities enclosed in the `<evaluate>` activity has to be substituted by the collection of activities with the identifier T1. This implies generating a bigger piece of process code at run time. The users have to be able to generate the missing piece of code (here T1) fast enough in order to avoid any delays in the execution of the process instance. The substitute business logic could be stored in an abstract form in a domain specific registry, e.g. as a template, and then used whenever needed. Templates are reusable pieces of code and functionality [25], which can implement design patterns [1], domain-specific collections of activities and others in a generic way. The parameters of the templates have to be substituted at run time by the users to get a concrete version of the substitute code.

Again, the implementation of the WS-flow engine has to take care about generating code from a template. The engine implementation can be based on any paradigm; it only needs to implement the meta-model. Additionally, the management system must ensure that only legal substitution changes are enforced.

We still have not tackled the case in which *changing a portType* would *result in changing the process logic*. Because of the fact that a portType can be used to identify

a complex WS-flow the above two versions of the `<evaluate>` activity would probably have to converge in a single version. This is a topic of major interest for our research in the immediate future [29].

A very important issue here is to what extent these approaches can be enacted without the necessity of user intervention. Automating the WS instance swapping and portType changes at run time can be greatly facilitated by semantic description of WSs [15]. This is a research area currently progressing at good pace. We are certain that the results of the research in this area can be accommodated by the presented meta-model constructs. In this relation, of great importance to the success of our work would be the ability to classify reusable templates semantically. These and other issues, as well as finalizing the implementation the supporting tools are in the focus of our current and future work.

7. Conclusions

WS compositions are the natural consequence of the convergence of workflows and WSs. The discussion in this paper is based on the fact that WS-flows combine features from both workflow and WSs and their benefits mutually amplify. We pointed out the fact that there is no standard way for modeling and describing workflows, let alone for WS-flows. We are convinced that any standardization efforts should gravitate around a WS-flow meta-model. Additionally, we argue that execution environments compliance criteria should also be based on a unified WS composition meta-model. We also presented the benefits of using such a model. One of the advantages of a unified WS-flow meta-model is that it provides a common framework for enabling features in a standardized way by means of standardized meta-model extensions. To these features counts the WS-flow adaptability. In this relation we provided a classification of possible changes in WS-flows definitions and their instances in reaction to changes in the business rules, software infrastructure, law and other factors of any business domain. We presented meta-model extension constructs for built-in adaptability; in particular the `<find_bind>` construct intended to enable dynamic swapping of WS instances, and the `<evaluate>` construct – to be used to allow for run time changes in WS portTypes and in process logic. We are certain that these constructs are a step towards fully adaptable WS-flows without the need of human intervention to guide the adaptability. Therefore we created the proposed meta-model constructs so that they are able to accommodate advances of the semantic WSs area.

References

1. van der Aalst, W.M.P., Almering, V.: Workflow Patterns, available on the Internet. 2003. http://tmitwww.tm.tue.nl/research/patterns/
2. van der Aalst, W.M.P.: Don't Go with the flow: Web service composition standards exposed. IEEE Intelligent Systems, "Trends and Controversies" Issue Jan/Feb 2003.

3. van der Aalst, W.M.P., et al.: Adaptive workflow. On the interplay between flexibility and support. In Proceedings of ICEIS 1999, 1999.
4. Active BPEL. August 2004. http://www.activebpel.org/
5. Active XML: http://www-rocq.inria.fr/gemo/Gemo/Projects/axml/
6. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Concepts, Architectures and Applications, Springer-Verlag. Berlin Heidelberg New York, 2003.
7. Arkin, A. et al.: Business Process Modeling Language. BPMI.org, 2002.
8. Baeyens, T.: The State of Workflow. TheServerSide.com, May 2004. http://www.theserverside.com/articles/content/Workflow/article.html
9. Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y.L., Januszewski, K., Lee, S., McKee, B., Munter, J., von Riegen, C.: "UDDI Version 3.0", IBM, HP, Intel, Microsoft, Oracle, SAP. 2002. http://uddi.org/pubs/uddi_v3.htm
10. BPMI.org: BPMI.org Phase 2. Insight, Innovation, Interoperability. 2004
11. Casati, F., et al.: Adaptive and Dynamic Service Composition in eFlow. In Proceedings of CAiSE 2000, LNCS 1789, Springer Verlag, 2000.
12. Curbera, F., Goland, Y., Klein, J., Leyman, F., Roller, D., Thatte, S., Weerawarana, S.: Business Process Execution Language for Web Services (BPEL4WS) 1.1. May 2003. http://www.ibm.com/developerworks/library/ws-bpel
13. Courbis, C., Finkelstein, A.: Towards an Aspect Weaving BPEL Engine. In Proceedings of AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS), Lancaster, UK, March 2004.
14. Czarnecki, K., Eisenecker, U.: Generative Programming: methods, tools, and applications, Addison-Wesley. 2nd edition, 2002.
15. DAML Services: available on the Internet: http://www.daml.org/services/owl-s/
16. Edmond, D., ter Hofstede, A. H. M.: Achieving Workflow Adaptability by Means of Reflection. ACM SIGGROUP Bulletin, Volume 20, Issue 3, December 1999.
17. Han, Y., Sheth, A., Bussler, Chr.: A Taxonomy of Adaptive Workflow Management. In Proceedings of the "Towards Adaptive Workflow Systems" Workshop at the 1998 ACM Conference on Computer-Supported Cooperative Work (CSCW98), Seattle, 1998.
18. Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K., Teschke, M.: A Comprehensive Approach to Flexibility in Workflow Management Systems. In Proceedings of WACC'99, 1999.
19. IBM AlphaWorks, "IBM Business Process Execution Language for Web Services Java™ Run Time (BPWS4j)", IBM, 2002, http://www.alphaworks.ibm.com/tech/bpws4j
20. IBM developerWorks: WebSphere Business Integration Server Foundation Process Choreographer. http://www-106.ibm.com/developerworks/websphere/zones/was/wpc.html
21. Intalio, Inc.: Intaglio|n^3 BPMS, 2004. http://www.intalio.com/products/index.xpg
22. Jablonski, S., Bussler, C.: Workflow Management. Modelling Concepts, Architecture and Implementation. International Thomson Computer Press, London, 1996.
23. Karastoyanova, D., Buchmann, A.: Development Life Cycle of Web Service-based Business Processes. Enabling dynamic invocation of Web services at run time. In Proceedings of The 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure 2004 (WSMAI-2004), April 2004.
24. Karastoyanova, D.: A Methodology for Development of Web Service-based Business Processes. In Proceedings of AWESOS 2004, Monash University 2004.
25. Karastoyanova, D., Buchmann, A.: Automating the development of Web Service compositions using templates, to appear In Proceedings of Geschäftsprozessorientierte Architekturen Workshop, at Informatik2004, Ulm 2004.
26. Leymann, F., Roller, D.: Production Workflow. Concepts and Techniques. Prentice Hall Inc., 2000.

27. zur Muehlen, M.: Process Management Standards Overview. Stevens Institute of technology, Hoboken, NJ, 2003. available on the Internet: http://www.wfmc.org/standards/docs/Process_Management_Standards_files/frame.htm
28. Oracle Corporation: Oracle BPEL Process Manager 2.0. 2004. http://www.oracle.com/technology/products/ias/bpel/index.html
29. ReFFlow Project. http://www.dvs1.informatik.tu-darmstadt.de/research/refflow/index.html
30. Templ, J., "Metaprogramming in Oberon" PhD Dissertation. ETH Zürich, 1995. http://citeseer.ist.psu.edu/templ94metaprogramming.html
31. Twister, 2004. http://www.smartcomps.org/twister/
32. W3C: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Working Draft, 2003. http://www.w3.org/TR/wsdl20
33. Workflow Management Coalition: http://www.wfmc.org