ReFFlow: A MODEL AND GENERIC APPROACH TO FLEXIBILITY OF WEB SERVICE COMPOSITIONS

Dimka Karastoyanova and Alejandro Buchmann

Darmstadt University of Technology, Hochschulstrasse 10, 64289 Darmstadt, Germany
dimka@gkec.tu-darmstadt.de
buchmann@informatik.tu-darmstadt.de

Abstract:
This paper addresses two important aspects of Web service compositions - on the one hand, flexibility of Web service (WS) compositions, and on the other the automatic development of such compositions. The existing WS composition models and languages do not support process adaptability at run time. In this respect we propose to extend the existing models with generic constructs to accommodate dynamic features of processes that ensure adaptability. The proposed constructs enable swapping instances of participating Web service and changing process schema at run time. The ultimate goal is to define a unified WS composition model that enables the development of portable process definitions with built-in flexibility. We focus on such a process model that does not impose any specific implementation approach. The second major issue we address is the automated development of WS compositions. A prerequisite for the automated development is the availability of a process model. Additionally, WS-flows templates can be used to promote reusability and automation further, and can be classified for specific business domains.

1. Introduction

Web Service (WS) composition is a term denoting both the process of composing complex WSs from simple ones and the composite WSs themselves. WS compositions exhibit characteristics of both workflow and WS technologies. We use the terms WS-flow (Web Service Flow) [17] and WS composition interchangeably throughout this paper.

Traditional workflow technologies are capable of defining explicitly how information is dispatched among disparate applications and are also good at defining business logic for the integration of distributed, heterogeneous systems. Workflows are an EAI technology that complements the message brokers providing for interoperability, with task routing, i.e. workflows automate the control and data flow among different applications. Just like any workflow a WS composition is described in terms of tasks, ordered in a meaningful way, and subject to constraints and business rules to achieve a business goal. We recognize the need for WS compositions adaptable to changes in the environment – legal, business and other constraints and rules vary constantly. That is why the focus of this paper is on adaptable WS compositions. In addition we pay attention on automating the WS-flow production.

WSs, on the other hand, provide stable unified interfaces and thus allow services running on different systems to communicate in a standardized way, independent of platform characteristics.
The participants in WS-flows are only WSs, which means that implementation complexity of process tasks is hidden. This means two things. First, WS-flow developers do not have to care about different types of participants (as it is in the traditional workflows). Second, they cannot simply deduce implementation details and decomposition of the WSs they use, i.e. developers can only be certain about the type of the messages a participating WS consumes and produces, which is the data one can get from the interface description [22] of WSs. The inherent characteristics of WSs are already put in use by the existing WS composition models and specifications.

As a result of the great attention paid to WS-flows there are numerous overlapping specifications approaching this issue. Good classification and analysis of these approaches can be found in [19], [17] and [1]. But the existing WS composition languages and approaches of traditional workflow management provide insufficient support for flexibility and adaptability of processes to the continuously changing business environment. Traditional workflow technologies struggled with the problem of adapting to unforeseen exceptional circumstances. One reason for this is that they aim at standardizing for interoperability among the workflow management systems, which results in a number of industry driven specifications and their corresponding implementations that in general cannot interoperate [1], [5]. The Workflow Management Coalition (WfMC) defines the so-called Reference model for the architecture of the workflow management systems (WfMSs) and requires support of several APIs [14] to ensure compliance and interoperability. WfMC did not attempt to standardize the process model and most process definition languages are not precise enough to avoid interpretation [1]. Besides, support for flexibility has been incorporated in the implementations of the WfMSs, rather than in a standard process model. Any implemented features for flexibility remain vendor-specific and bound to a specific process language and execution environment. To date, there are no WS-flows specifications and no process management systems that can define, execute and manage flexible WS-based processes.

In the ReFFlow project [20] we combine the experience and advances of several fields in computer science to create flexible WS compositions: workflow technologies, software engineering, and the Web service technology. A short overview of this project is given in section 2. The ReFFlow project defines a methodology for development and execution of reusable flexible WS compositions and advocates the creation of a common WS-flow meta-model. In section 3 we introduce extensions to the existing WS composition models to enable adaptability in generic way and thus overcome those models’ deficiencies with respect to flexibility. We propose model extensions to provide for WS instance swapping and service type changes during the execution of WS-flows. We elaborate on the influence of the model extensions on the engine implementation and the needed infrastructure. In the same section we point out the benefits of using the extended model. Section 4 tackles the issue of automating the creation of WS compositions and points out open issues. We present conclusions and an overview of our future work in the closing part.

2. ReFFlow – project and methodology

In the ReFFlows project [20] we define a methodology for development and execution of WS-flows [17]. It reflects our conviction that development and execution of WS compositions should be approached in a standardized way similar to the approaches known from applications development. The ReFFlow methodology is based on a refined process life cycle [16]. It has several phases. Each phase prescribes specific approaches to address different problems encountered when modeling and
producing WS compositions. The project has two main, related objectives influencing each other. On the one hand, the methodology builds on the need for a common model for defining WS-flows. We believe the constructs of such a model and their relationships can account for dynamic features of processes at run time in generic way and thus facilitate their adaptability, regardless of the WS-flow engine implementation. On the other hand, apart from targeting flexibility of WS compositions, the project addresses the automation of the creation of WS-flow definitions with built-in flexibility. The main principle we follow in order to automatically produce WS-flows is reuse of process models and definitions. While coding WS-flows in an automated manner is promoting faster and easier development of the WS-flows, and is a good start towards broader acceptance of the WS paradigm as a whole, it does not suffice when flexibility of business processes is the ultimate goal. We are convinced that a combination of both automated coding and a common process model able to incorporate process flexibility in its constructs are the best approach to start with.

3. Extending the existing WS composition models

In keeping with the WS paradigm and its principle of leveraging technology and specifications, we find it reasonable to reuse and extend existing models in order to create a common WS-flow metamodel, instead of creating a new, overlapping model from scratch. Traditional workflow technologies [23], [7], [15] and WS-based process definition languages [4], [8] provide constructs for representing business logic, tasks routing, data flow, control flow, exception handling and task compensation, etc. Nevertheless, all existing WS-based process definition languages are considered incomplete with respect to their ability to define all necessary control and data flow patterns [1]. This fact alone calls for enriching existing models. Additionally, the existing WS-flows models and languages make the assumption that all participants in a process, WS instances in particular, are known at modeling time. We are convinced that certain degree of flexibility can be accommodated by the model constructs and thus provide developers and users with the freedom to choose how the process is to flex and to specify criteria to determine this choice.

Providing flexibility to systems is a problem pertaining not only to business processes and their computerized representations – the workflows, but to applications development as well. There are lots of different approaches addressing flexibility in both areas [9], [7], [11], [13]. We combine the advantages of those approaches in our work on the ReFFlow project. Next we describe the ways in which a WS-based business process can be provided with flexible behavior. Classifying these approaches will simplify the following discussion on model extensions. The classification reveals how existing models can be improved. Then, in section 3.2, we explain the model constructs we use to extend the existing WS compositions models. The benefits of using a common model extended with constructs accommodating flexibility are brought forward in section 3.3.

3.1. Classification of approaches to WS-flows adaptability

Table 1 shows a classification of approaches to WS-flows adaptability. As a starting point for this classification we use previous work in traditional workflow [12], [13]. There are some characteristics of WS-flows, especially those inherited from the WS technology, that make it
simpler to achieve flexibility, as compared to the ones used in traditional workflows (see [12]). In our view, the fact that WS-flows have to handle only a single type of participants, i.e. WSs is a significant simplification. This discards the necessity of tackling resource and organization management for multiple participant types, since such variety does not exist. Generally, in a WS-flow schema, only the portType of WSs that should perform a certain task is prescribed and the service’s role (e.g. BPEL [8]). Moreover, the fact that WS are XML-based technology maintains simplicity by the fact that all identifiers of activities, their attributes and values are actually strings.

Table 1. Classification of approaches to WS-flows adaptability

<table>
<thead>
<tr>
<th>Life cycle phase</th>
<th>Approach</th>
</tr>
</thead>
</table>
| Build time: (static) | • Change process schema  
                       | • Flexibility by selection       |
| Run time: (dynamic) | • Swap WS instances                
                       | • Change process schema           
                       | • Change portTypes               
                       | • Change process logic            |

The two major phases in the life cycle of a WS-flow require different approaches towards ensuring adaptability. Process evolution can be provided at build time, which is considered static and therefore inflexible approach (see Table 1). At build time any kind of changes can be made – one can change both participating Web services types and WS instances; process logic can also be changed as found appropriate. At build time a minimal degree of run time adaptability can be provided for by the so-called flexibility by selection [13] approach. To ensure the degree of flexibility this approach promotes, existing WS composition languages define constructs for branching, choice, repetitions etc. [8], [4], [23], [15]. Thus, it is possible to prescribe alternative control and data flow branches in the execution of WS-flows. Developers draw on their experience in the particular business domain, and foresee and model these alternative execution paths. However, it is impossible and impractical to make provisions for all alternative paths a process might take. Should an unforeseen, exceptional or erroneous situation occur the process definitions must be rewritten; for this the process instances must be stopped, and after the change started again. Hard-coding alternative execution paths of a process is a static approach (Table 1) ensuring minimal flexibility. In the world of business interactions terminating a process instance might have the consequence of loosing sensitive data, process history, time, and even customers.

A more appropriate approach towards adaptable behavior is the one known as workflow evolution. Workflow evolution was put into play in traditional workflows and is associated with changing process schema and instances at run time [7], [11], [12]. The WfMSs provide special-purpose primitives to enable workflow evolution by incremental modifications of workflow definitions; examples include inserting and deleting process tasks, meta-programming techniques [9] such as code generation and instance migration [7], reflection [11] and meta-objects manipulation, and others. However, all these features are not accounted for in the process models, but rather are implemented by the workflow engines. This is in the line of standardizing for process management systems interoperability, and not for enabling process definition standardization and portability. The feasible and meaningful run time changes of WS-flows that provide for adaptable behavior can be organized in two major groups. The first group is related to the ability to swap WS instances complying with the same abstract description (portType) [22] at run time. It allows for changing the WS instances acting on behalf of a process at run time. However, in certain situations this might not suffice. For example, if all WS instances complying with the portType specified in the process
definition fail and are not available during the execution of the process, or if a service type is needed that provides almost the same functionality with some (even slight) difference in the abstract description, or even if due to changes in business or legal constraints it is necessary to use a totally different service type. These scenarios require ability to substitute types of services that perform tasks for the process with other service types, i.e. other portTypes. To do this it is necessary to perform a change in the process schema at run time. A WS-flow schema change at run time can also be used to alter task sequencing (control and data flow) in (parts of) the process. This means that an alternative path with different logic could be defined dynamically and executed at run time.

The existing technologies do not focus on providing a process model to accommodate those approaches. In the next section we introduce model constructs that aim at providing instance swapping and changing portTypes at run time.

3.2 Model Extensions

In the previous section we stated a number of alternatives way of rendering WS compositions flexible. Here we introduce two model constructs and use BPEL4WS syntax examples to show how they target adaptability; these constructs reflect the classification on Table 1. Incorporating model extensions has implications on the implementation of the process management systems. Besides, all possible implementation approaches influence the model constructs, since the attributed of the model constructs have to be mapped to parameters, data types, operations and other data, necessary to execute the processes. These issues are also shortly discussed.

3.2.1. Enabling WS instance swapping

To enable WS instances swapping at run time we define a find_bind construct, mapped to an activity of the same name in the process language syntax. The <find_bind> activity is to be executed before each WS invocation in a WS composition, e.g. before <invoke>, <receive> and <reply> activities in BPEL. It involves: a look up of WSs instances defined by the portType specified in the subsequent invocation activity, a choice of a single WS instance and binding to it [3], [16]. The search is performed in a UDDI registry [6]. 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 [10], too. For more details and a code example on this construct refer to [16].

There are several implications of using this activity. A QoS model for WSs and a corresponding notation for QoS description are needed to support the choice of WS instance. Furthermore, a means to describing choice policies, based on various criteria is also a must. To date, these are unavailable. So far, 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 one additional HTTP call for each service invocation used for the search.
in a UDDI registry. As it results in deteriorated performance optimization is necessary, e.g. by performing the finding of services and binding to them only when it is necessary; for instance, whenever a criterion falls under a threshold value.

3.2.2. Enabling portType changes

In reaction to changing process rules and environment the service types of the involved WSs might need to be modified (see Table 1). We introduce a meta-model construct – the <evaluate> activity – to address the change of types of services performing for a WS-flow; it can easily be appended to the BPEL syntax. The activity encloses an invocation activity. 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 [21]; there the eval() function is introduced as a way to execute a given function with new values of its parameters known only at run time. Listing 1 presents an example of this activity; irrelevant details are omitted.

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

Listing 1. The <evaluate> activity example in BPEL syntax.

In the example above, 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 be provided with a tool to allow access to the WS-flow instance under consideration. Such a tool has to allow the users to specify the new values of the portType and operation. These are to be used in the enclosed invocation activity as new attribute values. Thus the user is given the opportunity to change the type of service used by a process. In the example, the user could specify new values for portType (nsws2:ConvertCurrencyService) and operation (usd2eur) and thus change the currency conversion service so that it calculates conversion from US dollar to Euro (operation="usd2eur"), instead of US dollar to German mark (operation="usd2DM"). Note that Listing 1 presents the new values of the parameters, which the user could have provided at run time (i.e. this is how the process definition looks like after the user has induced portType changes). However, the attributes of the <evaluate> activity should be initially populated by the developer with default values 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 retain their original values.
It is true that the creation of a process definition becomes a little bit more complex due to the requirement to the developers to code the `<evaluate>` activity and mainly understand it. However, we are certain that coding can be automated (see next section, [17], [20]). Moreover, including a single activity to the existing models that is capable of accommodating different implementation approaches is much better and feasible solution than extending the existing models with multiple activities for every particular implementation approach.

There are several ways in which the `<evaluate>` activity can be implemented. Some existing approaches use code generation for the new alternative and migration of (one or more) process instances from the old definition to the newly generated one. A procedure, similar to the one used in Active XML [2] can also be involved for dynamic code generation. Reflection has been applied in workflow technologies for the same purpose [11]; process tasks (activities) can be performed with parameters different from the originally specified ones using this approach. Aspect-oriented programming (AOP) [9] is another possible implementation of the `<evaluate>` activity.

To the implications of changing a service types at run time belongs the requirement to find a WS instance of the newly assigned portType and in turn bind to it. This results in using the find and bind mechanism, expressed by the `<find_bind>` activity.

Our future work in this context aims at extending the `<evaluate>` activity with additional attributes such that the change in the definition does not affect portTypes and operations only, but allows for replacing a more complex collection of activities with a different one. We are confident that replacing a part of the process definition with another one means altering process logic (both control and data flow). Doing so aims at providing an alternative path in process execution, which has been unknown prior to run time. The semantics we would like to enforce should allow for substituting a collection of activities enclosed in the `<evaluate>` activity with another one. This will inflict generating a bigger piece of process code at run time. The users have to be able to generate the missing piece of code fast enough in order to avoid deteriorated process instance performance. The business logic to be used to substitute the original one can be stored in the form of a template (see section 4, [17], [18]) in a domain specific registry.

Again, the implementation of the WS-flow management system has to take care about generating code from a template by providing a tool for this purpose. The engine implementation can be based on any paradigm. Additionally, the WS-flow management system must ensure that only legal substitution changes are enforced.

A very important issue here is to what extent these approaches can be enacted without the necessity of users’ intervention during run time. Automating the WS instance swapping and portType changes at run time can be greatly facilitated by semantic description of WSs. This is a research area undergoing development. We are certain that the result of the development in this area can be accommodated by the model constructs presented above. In this respect, 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 work in the future.
3.3. Benefits of a unified WS composition meta-model

We can profit from the existence of a common WS-flows model with built-in adaptability features for several reasons. Such a model is instrumental for providing process definition portability because it could be standardized. Moreover, the model can enable transformations to other languages based on model constructs and not on language-to-language mappings. Such transformations are especially necessary in the case users and organizations would not like to commit themselves to a standardized syntax and prefer to leverage a language or technology they already use.

Automation of process definition creation is promoted, too. A meta-model is a prerequisite for supporting reusability in applications development. In this context a meta-model would greatly facilitate the creation of parameterized WS-flows [18] for special business domains. We proposed some constructs incorporating adaptability of WS-flows. Even though most changes in process definitions can be performed using existing traditional workflow approaches, the flexibility gained by including the proposed activities to the model is much greater. Organizations are given the freedom to use multiple implementation paradigms for their process engines or pick up the engine they prefer and stay interoperable with other companies’ systems. More to that, the users and developers gain additional freedom by being allowed to specify attribute values as needed or desired in a standard way and all this at run time, and without being knowledgeable of the execution engine implementation.

4. Producing WS-flows in an automated manner

Apart from targeting WS-flows flexibility the ReFFlow project promotes automatic development of WS-flows. The project is based on a refined process life cycle [16] and builds on the need for a unified model for creating WS-flows. Coding WS-flows in an automated manner promotes faster and easier development. We address the automated development of WS compositions in the first two life-cycle phases (Figure 1), where we target process definition modeling and generation [17]. The main principle we follow is the reuse of process models and definitions. If process definitions comply with the common model they could be easily transformed into multiple languages. Additional advantage is ensured by the fact that model constructs can accommodate flexibility features to WS-flows. Based on the common, unified process model developers can create reusable process definitions portable across multiple process engines. However, we believe that automatic development of process definitions can further be boosted by delimiting bigger repeatable parts of WS-flows definitions as reusability units we call templates [17], [18]. Templates are units of code and functionality reuse; not only code is reused but also the behaviour the template implements.
Basically, any repeatable collection of activities can be represented by a template with the appropriate parameters. The question of what can be represented by a parameter in a WS-flow definition hints what kinds of templates can be created. A meaningful and reasonable use of templates would be to represent implementations of design patterns, algorithms, domain specific collections of activities, whole domain specific business processes, coordination protocols roles [18]. The use of templates can help automate the development of WS-flows definitions, hide complexity and shorten development time.

Due to the immaturity of the WS technology, there are some problems faced by the idea of using templates. Templates can be stored in local registries of organizations, but to be really useful the knowledge and experience they stand for must be provided to others in a suitable form. Special purpose public registries or libraries can serve for storage; using a UDDI [6] registry is also possible but requires specification change to allow for template storage, description and classification. In this respect templates description and classification is a relevant issue.

Certainly, to be able to compose WS-flows using templates, tools have to be available to allow for using those templates; those tools have to provide suitable means for substituting templates’ parameters. The development of such tools is in the scope of our current work on the ReFFlow project.

Still, in spite of the difficulties, greatest benefits of using templates could be reaped if templates are created based on a common process model independent on any existing process definition language. The ultimate goal in automating WS composition development is to define parameterized WS-based processes, based on templates [18]. Such processes find reasonable application in domain-driven applications that have to comply with B2B coordination and collaboration protocols.

5. Conclusions

Composing WSs is an area still undergoing development. In this paper we motivated the need of standardizing a model for WS compositions instead of standardizing the implementation of the execution environment. We are convinced that a common WS-flows model is the best way to standardize process definition and provide for their flexibility and adaptability. The constructs of a
model help abstract from the implementation approach used and therefore maintain reusability and portability. We introduced model constructs to accommodate adaptability in generic way and provide the users with ability to specify required changes in a unified manner, regardless of complexity and the paradigm used to develop the engine. Our ultimate goal is to accomplish a process model with constructs enabling process adaptability and fostering standardization. We presented two constructs that are intended to enrich the existing composition models with capabilities to swap WS instances and/or WS types at run time. We are convinced that a process model fosters not only process standardization and flexibility, but rather it promotes automating the production of WS-flow definitions – such a model is the basis for providing tools for automation. In the ReFFlow project we currently focus on development of tools for automatic development of WS-flows using templates. Another aspect of our work is the implementation of a WS-flow engine supporting the constructs we introduced in this paper.

6. References: