An Approach to Parameterizing Web Service Flows

Dimka Karastoyanova [1], Frank Leymann [1], and Alejandro Buchmann[2]

[1] IAAS, Universität Stuttgart, Germany
{karastoyanova, leymann}@informatik.uni-stuttgart.de
[2] Computer Science Department, Technische Universität Darmstadt, Germany
buchmann@informatik.tu-darmstadt.de

Abstract. The flexibility and reusability of Web Service flows (WS-flows) are limited especially by the fact that portType and operation names are hard-coded in the process definition. In this paper we argue that through parameterization and substitution WS-flows flexibility can be improved, while reusability is enhanced. We introduce a meta-model extension to enable run time evaluation of parameter values and thus discard the need to predict any possible partner service types during process modeling. The extension enables also run time changes in portType values. We show how the approach can be mapped to BPEL. We discuss prototypical implementation for the extended functionality and present conclusions and ideas for future work.

1 Introduction

The advances of the Web Service (WS) technology facilitate platform and programming language independent application integration. The technology has matured in the last years at a great pace. In this work we concentrate on making compositions of WSs (also called Web Service Flows or WS-flows) more flexible and also more reusable. We present an approach for creating flexible WS-flows by introducing additional degree of freedom with respect to the partners' portTypes and operations, in particular to their names. It not only boosts the reusability of process models but also decreases model complexity, increases their flexibility and minimizes the process maintenance effort. The approach boils down to the concept of *parameterized processes* – essentially presenting portTypes and operations using parameters and defining run time parameter substitution policies. Parameterized processes (section 2) are flexible because process definition independence of concrete portTypes and operations is achieved. To be able to execute such process models parameters' values must be resolved at run time using an evaluation strategy that specifies what mechanism has to be executed to return parameter values (section 2). We also show how the concepts can be mapped to BPEL [2] (section 3).

Despite being viewed as very flexible, WS-flows definitions still hard code participants' types in terms of portTypes and operations (names). Hard-coding presumes precise knowledge of the naming of portTypes and operations. In fact, while it is possible in practice to agree and standardize messages sent and received, it turns out to be very difficult or impossible to agree on grouping such operations in portTypes and their naming. This implies several deficiencies of existing WS-flows.

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 533 – 538, 2005.

Process models need to accommodate the fact that types of participants in a process are identified by their portType/operation names and that equivalent functionalities can be potentially exposed under different names. Modeling alternative control flow paths reflecting alternative portTypes/participants [6], [4] is one way to tackle his issue. This *increases* the *complexity* of process models significantly. On the other hand, one needs to *make a decision* as of which concrete providers would exist at the time of process execution, which in a long-running setting is impractical and impossible. This decision must be postponed till execution time of every process instance, so that the process would be able to appreciate for unknown services.

Having a model (Fig. 1) with portType names of two suppliers of hard disk drives (HDD) fixed means the process owner has decided on the set of providers to be invoked alternatively in any of the instances of the process model; the instances of this process model invoke only instances of either of those WS types. Such decision neglects all providers that could get exposed as WSs at a later point in time (e.g. portType called WS_pT3). To involve any other WS types requires modification of the process model and redeployment, which is deemed *inflexible* for long-running processes, unless the decision on the type of participants is postponed till run time.

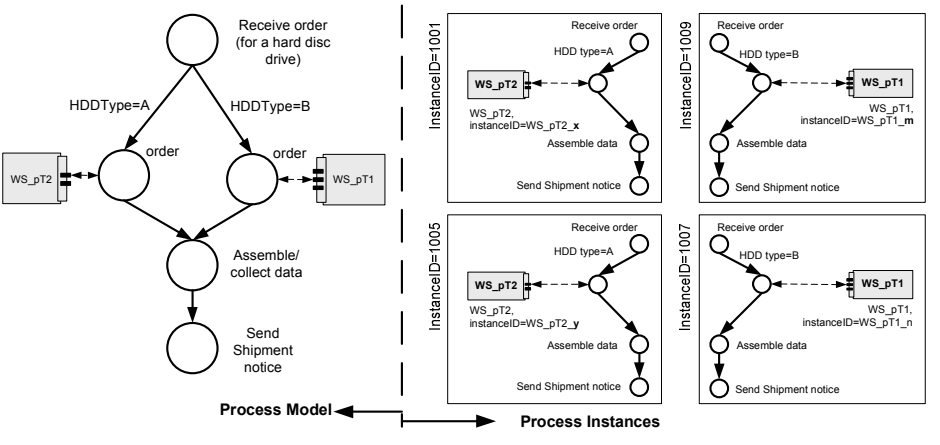

Fig. 1. State-of-the-art WS-flow

Existing WS-flows exhibit *limited reusability* because of insufficient support for loose-coupling, and hard-coding of partner WSs. Indeed there is complex, sometimes industry-specific, functionality carried out in a similar way but companies cannot always directly reuse a process model with the service providers coded, because they might need or wish to interact with providers discovered at run time.

2 Parameterized WS-Flows

We observe similarity in processes such as credit approval, payment, order placement, and so on and in some parts they differ only in the identifiers of the performing

services. These processes also include alternative paths that are only there because of the differently named providers of same functionality, e.g. the scenario in Fig. 1.

To benefit from loose coupling and enable process model reusability we need to ensure that portTypes and/or operations are interchangeable from the process viewpoint and can be exchanged. Therefore we use *parameters* to substitute only *portType names and operation names* of WSs a process interacts with. It is also possible to represent other process model elements in parameterized form [3] (e.g. transition conditions, message types and parts, activity types, data manipulation activities); it is out of the scope of this work. Our approach here takes upon parameterizing portType and operation names in process activities standing for an interaction with partners (called interaction activities [7]).

Parameterized processes are defined as WS-flows having one or more interaction activities' portType and/or operation names substituted by parameters.

A parameterized WS-flow is presented in Fig. 2. Unlike the example scenario in Fig. 1 the portType and operation names of the two alternative HDD suppliers (WS_pT1 and WS_pT2) have been substituted by a parameter (WS_pT=X). We observe that when executed the parameterized process instances are not only able to involve the service types used in the initial example but rather there are also instances that could interact with other types of services (e.g. WS_pT-N) even if they were unknown at the time of process modeling. This imposes the need to compute the parameter values at run time.

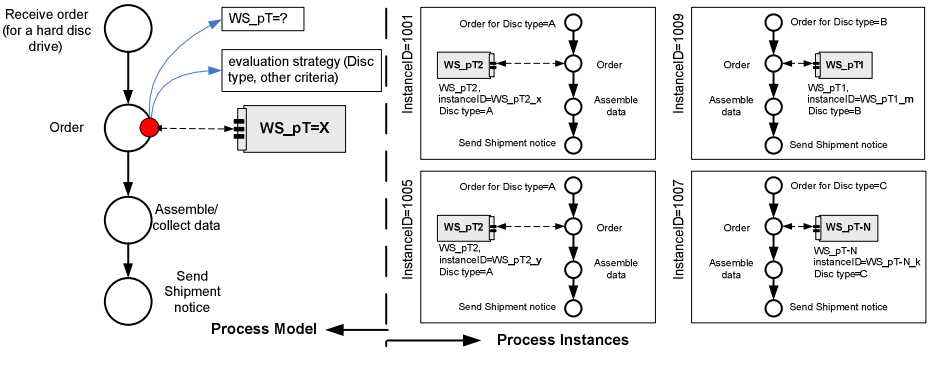

Fig. 2. Parameterized process and its instances

Run time evaluation of parameter values is performed on per process instance basis and is in general an algorithm that selects a WS type out of a set of WS types that meet a set of imposed requirements. The set of compliant service types varies from one process instance to another, because of the potentially different initial data. Once a parameter value is calculated it is substituted in the activity which then initiates an interaction with the WS. In our current work we assume that the messages constrain the choice of appropriate portType/operation values, i.e. messages are part of the search criteria.

Each parameterized activity must have the computing capability to resolve the values of the parameters; otherwise the process instances will be blocked waiting

because of unknown values or result in engine faults to be repaired by an administrator.

The four major alternatives (or strategies) we specify for obtaining the value of a portType/operation parameter in an interaction activity are: (i) static provision of portTypes and operations, (ii) prompt (the user) strategy, (iii) query and (iv) from variable. These alternatives define in a declarative manner how a service is to be discovered and what is required from the service, and neglect any reference to name of the portType (and operation). Selection of a compliant service type must be followed by a step of binding to a concrete port [4], [5].

The *static* strategy specifies concrete parameters values, which can be supplied during process modeling or upon process deployment (for completeness only).

The "*prompt (the user)*" allows users to provide a process instance with parameter values. This and the rest of the strategies involve instantiation of parameterized WS-flows. Prompts can be issued to users at the time of process instantiation for all parameterized activities, or may be signaled to the user every time the execution of a process instance reaches an activity with unknown service type.

The "*query*" strategy uses in-lined or referenced query definitions which are to be executed against a WS discovery component [7]. Queries contain service type selection criteria including the messages it must accept and return, semantics, and QoS. Since it is unrealistic to obtain a single compliant portType/operation pair the result of a query is a ranked, non-empty list of compliant portTypes/operations. One of the service types returned is used in the concrete WS-flow instance.

The "*from variable*" strategy postulates that the value for a parameter is to be copied from a variable defined in the process. The values stored in this variable may be obtained from a partner in a previous message exchange.

Strategies can be combined to ensure parameter values are resolved. This can be implemented either on engine/infrastructure level or on process model level (using fault handlers).

3 Parameterized Processes in BPEL

In this section we show how parameterized processes can be defined in BPEL. The current BPEL specification [2] involves neither portType nor operation parameters. All portTypes and operations of participating WSs are coded in the activities defining interaction with WSs. In BPEL these activities are the <*invoke*>, <*receive*> and <*reply*> activities; portType and operation names are specified by activity attribute values. Since BPEL definitions are representation of a WS-flow model in text and all portTypes and operations are also strings, it is only natural to be able to substitute any attribute value with any string. Such processes that include parameterized attributes are not executable for the simple reason that there is no way to provide the process instances with these missing values. Therefore we introduce an extension to the BPEL meta-model that accommodates the needed parameter values evaluation – the one described in the previous section. For this reason we define an extension to the standard elements section of the <*invoke*> activity. The <*evaluate*> extension is the meta-model element that corresponds to the mechanism to compute parameter values. The code in Listing 1 presents an example.

The attributes of the <*evaluate*> element are as follows:

- *activated* - has the values of *"yes"* or *"no"* and states whether the evaluation is enabled; *"yes"* means that the calculation has to be performed.
- *changeType* – specifies the evaluation strategy.
- *substitute* – used for passing input/output parameter values for strategy computation mechanism.

```
<process name="Process_name"> …
<invoke name="activity" partnerLink="partnerLink" portType="portType"
operation="operation" inputVariable="…" outputVariable="…">
      <evaluate activated="yes|no" changeType = "static |
      portType/operation | query | fromVariable" substitute="value"/>
</invoke> …
</process>
```

Listing 1. An example representation of the <*evaluate*> extension in BPEL

A summary the mapping of the evaluation strategies on BPEL is shown in Table 1. We do not recommend the use of a static strategy directly mapped to the *changeType* attribute in BPEL. Having the evaluate element though, allows us to enforce another type of strategy at run time [4]; appropriate tooling (for process instance monitoring) is needed.

Table 1. Mapping the evaluation strategies to BPEL constructs – an overview

Strategy	***activated***	***changeType***	***substitute***
Static	*"no"*	Any alternative	substitute value for another alternative
	"yes"	*"static"*	Concrete portType and operation
Prompt	*"yes"*	*"portType/ operation"*	portType, operation names provided by user
Query	*"yes"*	*"query"*	Query identifier / In-line query (string)
From variable	*"yes"*	*"fromVariable"*	Expression pointing to variable containing parameter values

We have extended the open-source engine ActiveBPEL [1] to implement the <*evaluate*> extension. For each strategy type an on-purpose mechanism has been implemented. Additional data structures have been defined for each parameterized activity and get populated with parameter values at run time after a strategy has been executed. A special-purpose invocation handler has been implemented; it generates dynamically a call to one of the ports implementing the discovered portType/operation [5]. Other implementation additions are: an extended parser for BPEL processes containing the <*evaluate*> element and the strategies, data structures storing data related to each process instance and its parameterized activities, portTypes and operations names, port locations, strategy. A monitoring tool has been implemented to track the execution of all process instances. It is instrumental especially for the *"prompt"* strategy; the tool is used to prompt users to supply the required parameter values. Currently our implementation relies on a simple

component for executing the query strategy because of missing standardized approach to describing WS semantics and QoS.

4 Conclusions and Future Work

Parameterized processes *aim at standardization* of process models and improve process flexibility and reusability. Flexibility by adaptation and flexibility by avoiding change [4] are supported by the proposed approach.

Parameterization simplifies WS-flow models. The simplified *control flow* (reduced number of activities and eliminated need for Dead Path Elimination [6]), yields performance improvement. *Fault handling* and *compensation* in parameterized processes require special care. *Fine tuning* of WS-flows is possible by adjusting parameter values' evaluation criteria. Missing semantic description standard and incomplete QoS models affect the search and discovery of WS types and impair a full-fledged application of the approach. It is possible to model both *synchronous* and *asynchronous communication* modes with parameterized activities. The asynchronous communication mode faces difficulties due to the insufficient capabilities to express guaranteed delivery of functionality on behalf of the partner as a combination of two one-way operations. The partner WS can return a result using the *ReplyTo* field [7] of the messages sent by the process, the operation name exposed by the process for the return call, and other correlation data.

Our future work includes experimenting with parameterizing other activity types, variables, transition conditions, and the corresponding infrastructure implementation.

References

1. Active BPEL. August 2004. http://www.activebpel.org/
2. Curbera, F. et al.: BPEL4WS Specification Version 1.1. May 2003.
3. Karastoyanova, D., Buchmann, A.: Automating the development of Web Service compositions using templates. In Proc. of GPA Workshop, Informatik2004, 2004.
4. Karastoyanova, D., Buchmann, A.: Extending Web Service Flow Models to Provide for Adaptability. In Proc. BPMSOA Workshop, OOPSLA '04, October 2004.
5. Karastoyanova, D., Leymann, F., Buchmann, A.: Extending BPEL for Run Time Adaptability. In Proc. of EDOC'05, 2005.
6. Leymann, F., Roller, D.: Production Workflow. Concepts and Techniques. Prentice Hall Inc., 2000.
7. Weerawarana, S. et al.: Web Services Platform Architecture. Prentice Hall 2005.