Automating the Development of Web Service Compositions Using Templates

Dimka Karastoyanova, Alejandro Buchmann

Computer Science Department
Technische Universität Darmstadt
Hochschulstrasse 10
64289 Darmstadt
dimka@gkec.tu-darmstadt.de
buchmann@informatik.tu-darmstadt.de

Abstract: The development of Web Service compositions has not yet been automated. Web Service-based process definitions can be created automatically using Web Service compositions templates. Templates are units of code and functionality reuse. We argue that templates can be used to implement coordination protocols roles, design patterns, algorithms and domain specific business processes. We also discuss the implications of this approach. The success of this approach depends mainly on the existence of appropriate tools.

1 Introduction

Web Services (WSs) are still being used in fairly simple applications. The technology lacks important features [Al03] to be considered a full-fledged middleware technology. The principle of design composability has been followed by all WS specifications [Fe03]. But composability in terms of the ability to compose simpler WSs in more complex ones is still marked by incomplete support. Currently, the attention of many researchers is concentrated on the issue of creating complex WSs with more valuable functionality; however, the community still tries to find out the best way to specify composite WSs.

In this paper we approach a slightly different issue related to WS compositions – this is the issue of automating the process of producing WS compositions. We use the terms WS compositions and Web Service Flows (WS-flows) [Ka04] interchangeably. In our view the development and execution of WS-flows should be approached in a standardized way [Ka04]. We are convinced that providing reusable process definitions based on a common process model that can be translated into multiple definition languages can enormously facilitate the development of WS compositions. Such approach will surely boost the acceptance of WSs, in addition to the gradual improvement of the technology characteristics.

We argue that a technology is successful if it can be put into practice with minimal effort; that requires automated application development. Our discussion is based on an example of a very simple business process (section 3).

We focus on the notion of *templates* as units of code and functionality reuse (section 4). We explain how templates for WS-flows can be defined and discuss extensively the implications and the difficulties of that approach. We comment on our future work in section 4. Conclusions are stated in the closing part.

2 Related work

To be able to reveal the importance and the contribution of employing templates for WS compositions here we present a short review of related approaches and earlier works. One of the most important characteristic of the WSs paradigm is the notion of functionality reuse. This principle is followed in all specifications that attempt to describe WS compositions, too [Ar02, Cu03], but it is not yet exploited completely.

In traditional application programming reusability has been a main approach towards automation of application development. The term "template" has been used in the C++ programming language. The C++ templates [CE02] take data types, pointers, functions and templates as parameters. Experience in application design is described by design patterns.

Design patterns are used in the traditional workflow technologies [AA03] and standardize process control and data flows design. Currently, design patterns for WS-based compositions are also being defined [AA03] and modelled in UML [GS04].

Visual programming approach is used in the BPEL editor [AW03] as a plug-in component to the Eclipse [EF] platform.

However, automated creation of WS-based process definitions is still inadequately addressed. One reason is the fact that existing approaches for automation of compositions development (e.g. from the traditional workflow technology and techniques for composing software components) cannot be applied directly when composing WSs. There are certain differences between composing software components, and tasks in workflows on the one hand, and WSs on the other. These differences are mainly explained by those features of WSs that distinguish them from other middleware technologies [RP04].

3 Example

In this section we present an example of a simple business process and show how parts of it can be delimited as units of reusability – *templates*. In our scenario a user sends a request for a cross-currency calculation to a composite WS, called "converter". The converter WS performs a simple sequence of tasks as follows (Figure 1):

1. Checks what kind of conversion is required by the user.
2. Gets a cross-currency exchange rate from a rate generator WS (a bank).
3. Requests a simple calculation (here multiplication) from a calculator WS.
4. Sends the result of the conversion to the client.

The tasks the converter WS has to perform are depicted by the activity diagram in Figure 1. The computerized representation of this activity diagram is the subject *WS composition* deals with. The functionality represented by this diagram can be implemented using a process-based approach, hence the term *composite WS*. There are two alternative paths in the diagram, determined by the choice of conversion method. In this particular process these two alternative paths are the same with respect to the type of activities they employ, their ordering and data manipulation (Figure 1), i.e. they implement the same functionality. The only difference is in the values of the activities attributes. The business process represented by the activity diagram can be implemented in any of the available WS-based process definition languages.

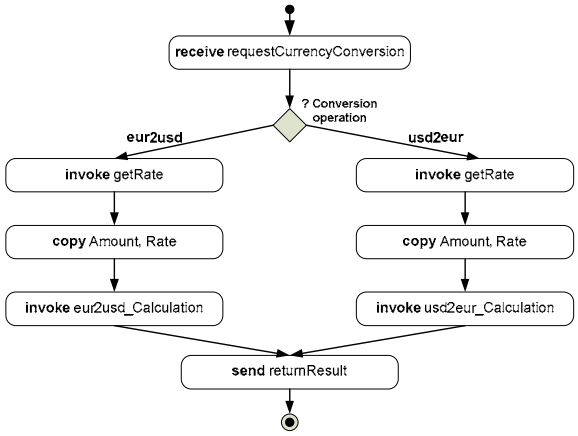

Figure 1: Activity diagram of the converter WS

For space limitations we show only the piece of code that repeats (twice) in a BPEL process definition.

```
<sequence name="conversion_operation">
    <invoke  partner="rateProvider"
             portType="Cross-CurrencyRateService"
             operation="getRate"
             inputVariable="rateRequest"
             outputVariable="rate"/>
    <assign >
             <copy>  <from variable="request" part="amount" />
                     <to variable="Amount_and_Rate" part="amount"/> </copy>
             <copy>  <from variable="rate" part="getRateReturn"/>
                     <to variable=" Amount_and_Rate" part="rate-value"/> </copy>
    </assign>
    <invoke  partner="converter"
             portType="CalculatorService"
```

```
                operation="conversion_operation"
                inputVariable =" Amount_and_Rate"
                outputVariable="result"/>
</sequence>
```

Figure 2: A piece of repeatable BPEL code

If given this piece of code, the developer would have to code only three additional activities to create a process definition for the above scenario (see Figure 1). To create a full-fledged business process exception handling activities, default control flow paths and others would also have to be added. But the time spared as a result of development automation can be used to define the most appropriate exception handling and compensating mechanisms; such mechanisms are an inseparable part of business process logic. Moreover, the generation of code implementing these mechanisms can also be subjected to automation.

4 Applying templates to automate the development of WS-flows

Reusing repeatable parts of a process definition can facilitate the automatic creation of this (Figure 2) and similar WS-based processes. We denote such repeatable pieces of code with the term *template.* Templates are units of *code and functionality reuse* [KB04]; not only code is reused but also the *behaviour* the template implements. The *parameters* a template can take are the values of the attributes of activities. For instance, possible parameters of a template in BPEL [Cu03] are the values of attributes of all activities in a process: portTypes, operations, variables, roles etc. In XML documents, i.e. in WS-flow definitions, all identifiers of elements, attributes and their values are strings (compare with C++ templates [CE02]).

The question of what can be substituted by a parameter in a WS-flow definition helps us to identify what kinds of templates can be created. Basically, any repeatable collection of process activities can be represented by a template with the appropriate parameters. However, a meaningful and reasonable use of templates would be to represent implementations of:

- *Design patterns* - we consider it useful to leverage design experience automatically, by providing templates for design patterns that can be used with any process definition language.
- *Algorithms* - various computing algorithms, whose use directly in the process logic is required (instead of in a separate WS), can be reused in the form of templates.
- Repeatable collections of activities specific for a particular *domain* - there are standards addressing the definition of multi-party business processes for different domains; however, these have not yet been used to support reusability and automation in the field of WS compositions development for specific domains. *Domain specific business processes* standardize not only the desired flow of a business process but also exceptions handling in certain situations, compensating activities and others. These can also be presented in parameterized form.

- *Coordination protocols roles* exploit the inherent relationship between the coordination protocols in a multi-party interaction and the internal implementation of the participating composite WSs [Al03, Ka04]. Such templates can be created and used to generate the skeletons of WS-flows definitions that comply with B2B coordination protocols.

The use of templates can help to automate the production of WS-flows definitions, hide complexity and shorten development time. But this approach faces some problems. For example, how templates are provided for use to developers. Templates can be stored in local registries of organizations, but to be really useful the knowledge and experience they represent must be provided to others, either in a special purpose public registry or library; the use of UDDI [Be02] registry is also a possibility – this requires specification change. Additionally, the problem of templates description and classification is also relevant.

The greatest benefits of using templates would be reaped if templates are created based on a *unified process model* [KB04] independent of any existing process definition language model.

If templates are created with the constructs a *common unified process model* then they could be composed to create process definitions independent of a particular process definition language. These process definitions are in fact combinations of templates, which we call *parameterized WS-based processes*. Such process definitions can be converted into multiple languages [Ka04] using meta-programming techniques [CE02] or XML transformations.

The concept of *sub-processes* is not explicitly represented by any of the existing model constructs for WS compositions. This owes merely to the fact that WS-based processes are WSs themselves, and therefore can be invoked in the same way any other WS is invoked to perform a task for the process. Related to this, we claim that templates can be similarly applied in producing both parameterized processes and sub-process.

The most meaningful application of process templates is for automatic development of domain specific processes and coordination protocols. It is not reasonable to cover the great variety of processes with a single collection of generic process templates. Our future work is focused on defining a collection of specific business domain templates, and creating tools for automated process definition generation [RP04]. These tools will be supported by a repository that will store the WS-flow meta-model, its domain specific extensions and the templates. Repositories are such systems that manage meta-models, models and their instances and keep all the artefacts they store consistent and correct.

Apart from being extremely helpful for automating WS-flows production, templates play also a significant role in providing *adaptability* to WS-flows. Some traditional approaches to workflow adaptability use code generation at run time for representing a new, alternative execution flow [Ca00]. Code generation is equally important for providing WS compositions with adaptability. In this sense, templates can be used to generate code for the alternative execution flows faster; again, only the templates' parameters would have to be submitted by the users [RP04].

5 Conclusions

Reuse of code, functionality, and design principles is common approach to automating applications development. WS-flows development automation is not yet sufficiently addressed. We propose the use of units of code and functionality reuse – the so-called *templates*. We argue that templates can be created to reuse design patterns code representations, coordination protocols roles, algorithms, and domain or company specific collections of business process activities. If the templates are created using the constructs of a unified WS-based process meta-model, they can be composed into *parameterized WS-based process templates*. Such process templates could serve as basis for developing process definitions in multiple WS-based process definition languages. We are convinced that following the principles of reusability and interchangeability of collections of activities, i.e. templates, is the most appropriate way to automate WS compositions development. Moreover, templates could extremely facilitate the approaches to providing adaptability to WS-flows.

References

[AA03] v.d.Aalst, W., Almering, V.: Workflow Patterns. 2003. available on the Internet, http://tmitwww.tm.tue.nl/research/patterns/

[Al03] Alonso, G., et al.: Web Services. Concepts, Architectures and Applications. Springer-Verlag. Berlin Heidelberg New York, 2003.

[Ar02] Arkin, A. et al.: Business Process Modeling Language. BPMI.org, 2002.

[AW03] IBM AlphaWokrs: IBM BPWS4J Editor. 2003.

[Be02] Bellwood, T., et al.: UDDI Version 3.0. IBM, HP, Intel, Microsoft, Oracle, SAP. 2002.

[Ca00] Casati, F., et al.: Adaptive and Dynamic Service Composition in eFlow. In Proceedings of CAiSE 2000, LNCS 1789, Springer Verlag, 2000.

[Cu03] Curbera, F., et al.: Business Process Execution Language for Web Services (BPEL4WS) 1.1. May 2003. http://www.ibm.com/developerworks/library/ws-bpel

[CE02] Czarnecki, K., Eisenecker, U.: Generative Programming: methods, tools, and applications. Addison-Wesley, 2nd edition, 2002.

[EF] Eclipse Foundation, 2004. http://www.eclipse.org/

[Fe03] Ferguson, D. et al.: Secure, Reliable, Transacted Web Services: Architecture and Composition. September 2003. http://www-306.ibm.com/software/solutions/webservices

[GS04] Grønmo, R., Solheim, I.: Towards Modelling Web Service Composition in UML. In Proceedings of WSMAI-2004, INSTICC Press 2004, pp. 72-86.

[Ka04] Karastoyanova, D.: A Methodology for Development of Web Service-based Business Processes. In Proceedings of AWESOS 2004, Monash University 2004.

[KB04] Karastoyanova, D., Buchmann, A.: A Procedure for Development and Execution of Process-based Composite Web Services. Poster presentation, In Proceedings of ICWE 2004.

[RP04] ReFFlow Project, 2004. http://www.informatik.tu-darmstadt.de/GK/research.html