Supporting User-Friendly Composition of Web Services in the Eclipse Platform

Niels Joncheere, Wim Vanderperren, Mathieu Braem, and Ragnhild Van Der Straeten

System and Software Engineering Lab (SSEL)
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
Phone: +32 2 629 29 64
`{njonchee,wvdperre,mbraem,rvdstrae}@vub.ac.be`

Abstract. State-of-the-art workflow languages for web services are much better suited for web service composition than traditional general-purpose programming languages. Such workflow languages, however, still require a lot of in-depth technical knowledge. In order to facilitate service composition without requiring such in-depth technical knowledge, a higher level of abstraction is required. We therefore propose a visual service composition environment, which allows user-friendly composition of web services using high-level composition templates, and which supports aspect-oriented software development. This paper provides an overview of the Eclipse plug-in that implements this service composition environment.

Keywords. Aspect-oriented software development, Eclipse, web service composition

1 Introduction

Over the last years, *web services* [1] have been gaining a lot of popularity as a means of integrating existing software in new environments. Basic web services can be created by exposing existing applications to the internet using XML front-ends. By composing a number of basic web services, new web services can be created that provide more advanced functionality. These compound web services can then be used by other web services, further improving software reusability.

Originally, the only way to compose web services was by manually writing the necessary glue-code in programming languages such as C and Java. It quickly became clear, however, that a composition of web services is more naturally captured by dedicated *workflow languages* [2] than by general-purpose programming languages. Today, the most popular workflow language with regard to the composition of web services is the *Business Process Execution Language* (WS-BPEL) [3]. WS-BPEL processes are platform- and transport-independent, and are expressed using XML. Recently, a higher-level visual notation for WS-BPEL, called the Business Process Modeling Notation (BPMN) [4], has been proposed.

Meanwhile, *aspect-oriented software development* (AOSD) has been proposed as a means of improving *separation of concerns* [5] in software. AOSD is based on the observation that a number of concerns in software (such as logging [6] and billing [7]) cannot be modularized using object-oriented software development: a program can only be decomposed in one way (i.e. according to the class hierarchy), and concerns that do not align with this decomposition end up scattered across the program and tangled with one another. This problem is dubbed "the tyranny of the dominant decomposition" [8]. AOSD allows expressing such *crosscutting concerns* in well modularized aspects, so that adding, modifying or removing such concerns does not require changes to the main program.

Although initial research on AOSD has concentrated on applying its principles to the object-oriented programming paradigm, Arsanjani *et al.* [9] and others [10–12] have shown that AOSD has a lot of potential in a web services context, too.

Although workflow languages are better suited for web service composition than general-purpose programming languages, they still require a large amount of in-depth technical knowledge. In order to facilitate service composition without requiring such in-depth technical knowledge, a higher level of abstraction is required. We therefore propose a visual *service composition environment* (SCE), which allows user-friendly composition of web services using reusable *composition templates*, and which supports encapsulating crosscutting concerns using AOSD techniques. This environment is implemented as a plug-in for the Eclipse[1] platform. The goal of this paper is to provide an overview of the SCE plug-in.

The outline of the paper is as follows: Section 2 provides an overview of the SCE, Section 3 describes related work, and Section 4 states our conclusions and future work.

2 The Service Composition Environment

2.1 Main Concepts

The SCE introduces three main concepts:

– **Services** are the basic building blocks of the SCE. They correspond to concrete web services. In addition to the usual WSDL [13] API specification, a service is documented using a WS-BPEL process that specifies the external protocol the service adheres to (e.g. first expect `login`, then `request`).
– **Composition templates** are used to compose multiple services. They are abstract descriptions of web service compositions, and may contain one or more placeholders for services. Composition templates are expressed using one or more abstract WS-BPEL processes. Service placeholders are unbound partner links in the WS-BPEL process.
– **Aspects** encapsulate crosscutting concerns and can be deployed to services and composition templates. Aspects are implemented using Padus [14], an

[1] `http://www.eclipse.org/`

aspect-oriented extension to WS-BPEL. Section 2.2 provides a brief introduction to Padus.

Figure 1 illustrates how these three concepts are visualized in the SCE. The SCE retrieves the services, composition templates and aspects that should be available for composition from a library. The library is organized in categories, contains additional keywords and a description for each entity, and allows searching for a specific entity.

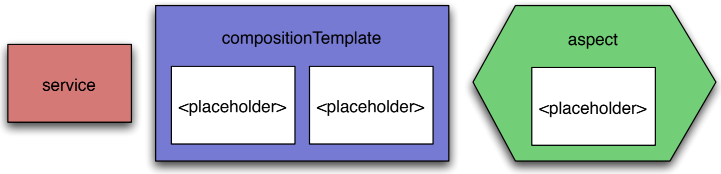


Fig. 1. SCE visual elements

2.2 Padus: An Aspect-Oriented Extension to WS-BPEL

A detailed explanation of Padus is outside the scope of this paper; the interested reader can find more information on Padus in the following paper: [14]. Instead, this section aims to introduce the features of Padus that are most relevant to the SCE.

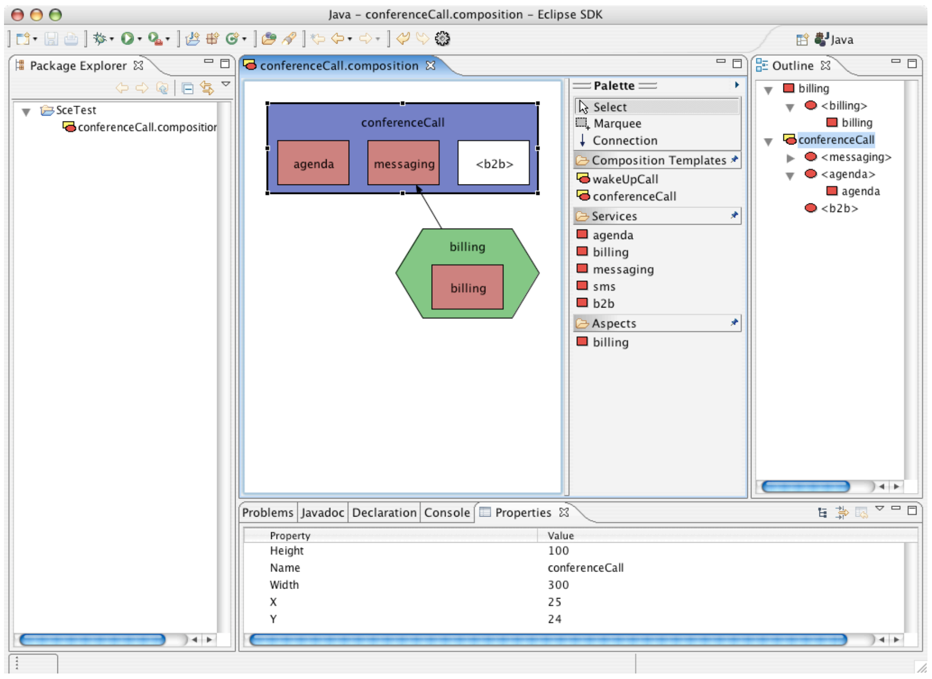
Padus is an XML-based language, and introduces two main concepts: *aspects* and *aspect deployments*. An aspect is a reusable description of a crosscutting concern, and contains one or more pointcuts and advice. A pointcut selects interesting points in the execution of the target WS-BPEL process (called joinpoints), and exposes target objects to the advice. The pointcut language of Padus is a logic language based on Prolog, and is thus very expressive [15]. The complete target WS-BPEL process is reified as a collection of facts that can be queried by the pointcut. The advice language is WS-BPEL, extended with some AOSD-specific constructs. Another advantage of the Padus language compared to other aspect-oriented extensions to WS-BPEL (e.g. AO4BPEL [10]) is the introduction of an explicit deployment construct. This deployment applies one or more aspects to one or more target WS-BPEL processes. Furthermore, it allows to clearly specify the composition (e.g. precedence) of the aspects.

The Padus technology is based on a traditional static weaver that processes the target WS-BPEL processes and generates new WS-BPEL processes containing the advice code where required. The main advantage of this approach is the compatibility with existing infrastructure, as the output can be deployed on any WS-BPEL-compatible engine.

2.3 SCE GUI

The SCE is implemented using the Eclipse Graphical Editing Framework (GEF)[2], which facilitates creating a graphical editor based on a model-view-controller architecture.

When the SCE plug-in is loaded in Eclipse, new compositions can be created and existing compositions can be opened. When a composition is opened, three views are of importance: the editor view, the outline view, and the properties view. Figure 2 provides an overview of the SCE plug-in's interface.



Property	Value
Height	100
Name	conferenceCall
Width	300
X	25
Y	24

Fig. 2. SCE overview

The editor view (in the middle of the screen) is used to edit compositions, and consists of two main parts: a large drawing canvas, and a smaller palette. The palette contains some selection and connection tools, and shows the available services, composition templates and aspects as they are loaded from the library. By double-clicking on an entity, the configured editor for that entity is launched (e.g. a BPMN editor for a composition template).

The outline view (at the right of the screen) shows a tree-based overview of the state of the composition, and the properties view (at the bottom of the

[2] `http://www.eclipse.org/gef/`

screen) shows the properties of the element that is currently selected in the editor view or in the outline view.

2.4 Composition

In order to create a composition in the SCE, it suffices to drag a composition template on the composition canvas and fill the placeholders with concrete services. Aspects can be connected to services, meaning that they will only be applied to these concrete services, or to a complete composition template, meaning that they will be applied to all the services that take part in this composition.

The composition shown in Figure 2 contains a composition template called "conferenceCall" with three placeholders. Two services called "agenda" and "messaging" have been added to the composition template's placeholders, while one placeholder is still empty. This placeholder could for instance be filled with the "b2b" service available in the library. The composition also contains an aspect called "billing", which is connected to the "messaging" service. A service called "billing" has been added to the aspect's only placeholder. The result of this composition is that the conference call application will work using the selected services, and that a billing aspect, which invokes the billing service, is deployed to the messaging service to bill the messaging actions selected by the aspect's pointcut.

2.5 Verification

An important requirement of the SCE is that it guides users in creating valid compositions without requiring in-depth technical knowledge. The SCE accomplishes this by verifying whether compositions are valid while they are created: when a service is dragged onto a placeholder, the SCE checks whether the service's protocol is compatible with the composition template's protocol. In case the service turns out to be incompatible, a report is generated that provides mismatch feedback to the user. Compatibility checking based on protocols rather than plain APIs is possible because every service is explicitly documented with a protocol specification expressed in WS-BPEL.

In literature, a wealth of research exists on the topic of protocol verification [16–20]. Our verification engine is based on the PacoSuite approach [21], which introduces algorithms based on automata theory to perform protocol verification. In order to provide protocol verification in the SCE, the WS-BPEL specifications of each service, aspect and composition template are translated into deterministic finite automata (DFA). By applying the algorithms introduced by the PacoSuite approach, the SCE can decide whether the service's protocol is compatible with the composition template's protocol.

2.6 Code Generation and Deployment

When the composition is complete and validated, the user may choose to generate the resulting composition and deploy it on a WS-BPEL engine. This will start the code generation process, which will bind the unbound partner links in the composition templates. An aspect deployment is automatically generated for the aspects contained in the composition. The Padus weaver is then employed to weave the aspects into the resulting WS-BPEL processes based on the aspect deployment specification.

A resulting composition can also be imported back into the library as a new service. The generated WS-BPEL process then serves as documentation for the new service. Apart from specifying a name and some other properties, this process is also automated.

The SCE also includes a built-in WS-BPEL engine that can be used to immediately execute a resulting composition. This feature is meant to be able to quickly assess the result rather than to be the real deployment target. We are currently working on improving the integration of this engine, so that it can be used as a debugger for compositions by providing feedback directly to the SCE.

3 Related Work

Documenting components with protocol documentation is already well investigated in literature. Campbell and Habermann [16] introduced the idea of augmenting interface descriptions with sequence constraints already in 1974. More recent work includes the Rapide system [17] or the PROCOL system [18]. In the research area of component based software development, several component composition environments are available that lift the abstraction level for component composition. Yellin and Strom [20], Reussner's CoCoNut project [19] and PacoSuite [21] for example also employ automata to document components. PacoSuite is one of the most advanced component composition environments and supports higher-level component composition based on sequence charts. The main advantage with respect to the other work on protocol verification is that PacoSuite supports multi-party connectors, whereas other approaches typically only support binary connectors. The PacoSuite approach is, however, domain dependent, and is only targeted at the simple JavaBeans component model.

BPMN is a graphical notation for specifying workflows, and aims to become the de facto graphical standard similar to WS-BPEL for workflow languages. BPMN allows for a higher-level graphical notation for processes in comparison to WS-BPEL, and is in fact complementary to our approach. A BPMN-based editor that is able to import/export WS-BPEL can for instance be used to edit the specification of a composition template. As soon as there is a standardized file format for BPMN, the SCE can also directly support BPMN for the documentation of services and composition templates, instead of or next to WS-BPEL.

4 Conclusions and Future Work

In this paper, we introduce a visual service composition environment that allows to easily compose new services. Composition templates specify the interaction of several abstract placeholders in a reusable manner. A service composition is created by visually binding the placeholders with concrete services. The SCE also supports crosscutting concerns encapsulated using Padus aspects: they can be visually deployed onto services or composition templates. The SCE allows to automatically verify a composition based on the API and protocol specification of the services, aspects and composition templates. Code that realizes the composition can be automatically generated. The result is again a service expressed using WS-BPEL that can be deployed on any WS-BPEL-compatible execution engine.

Currently, the library of available composition templates, services and aspects is a custom solution and limited to local files. In the future, we plan to investigate support for the industrial standard for discovery of web services called UDDI[3].

In related research [22], we define a concern-specific language (CSL) for the billing concern. We are exploring the possibility of integrating this CSL or its graphical representation in the SCE.

Acknowledgements

This research is partly funded by Alcatel Belgium and the Institute for the Promotion of Innovation Through Science and Technology in Flanders (IWT-Vlaanderen) through the WIT-CASE project.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V., eds.: Web Services: Concepts, Architectures and Applications. Springer-Verlag, Heidelberg, Germany (2004)
2. Du, W., Elmagarmid, A.: Workflow management: State of the art vs. state of the products. Technical Report HPL-97-90, Hewlett-Packard Labs, Palo Alto, CA, USA (1997)
3. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, version 1.1 (2003) `http://www.ibm.com/developerworks/library/ws-bpel/`.
4. White, S.A.: Business Process Modeling Notation (BPMN), version 1.0 (2004) `http://www.bpmn.org/`.
5. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Comm. ACM **15**(12) (1972) 1053–1058
6. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of AspectJ. In Knudsen, J.L., ed.: Proc. ECOOP 2001, LNCS 2072, Berlin, Springer-Verlag (2001) 327–353
7. D'Hondt, M., Jonckers, V.: Hybrid aspects for weaving object-oriented functionality and rule-based knowledge. In Lieberherr, K., ed.: Proc. 3rd Int'l Conf. on Aspect-Oriented Software Development (AOSD 2004), ACM Press (2004) 132–140
8. Ossher, H., Tarr, P.: Using subject-oriented programming to overcome common problems in object-oriented software development/evolution. In: Proc. 21st Int'l Conf. on Software Engineering, IEEE Computer Society Press (1999) 687–688

[3] `http://www.uddi.org/`

9. Arsanjani, A., Hailpern, B., Martin, J., Tarr, P.: Web services: Promises and compromises. Queue **1**(1) (2003) 48–58
10. Charfi, A., Mezini, M.: Aspect-oriented web service composition with AO4BPEL. In Zhang, L.J., ed.: Proceedings of the 2nd European Conference on Web Services (ECOWS 2004), Erfurt, Germany, Springer-Verlag (2004) 168–182
11. Cottenier, T., Elrad, T.: Dynamic and decentralized service composition with Contextual Aspect-Sensitive Services. In: Proceedings of the 1st International Conference on Web Information Systems and Technologies (WEBIST 2005), Miami, FL, USA (2005)
12. Verheecke, B., Vanderperren, W., Jonckers, V.: Unraveling crosscutting concerns in web services middleware. IEEE Software **23**(1) (2006) 42–50
13. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL), version 1.1. W3C Note 15 March 2001, World Wide Web Consortium (2001) `http://www.w3.org/TR/wsdl`.
14. Braem, M., Verlaenen, K., Joncheere, N., Vanderperren, W., Van Der Straeten, R., Truyen, E., Joosen, W., Jonckers, V.: Isolating process-level concerns using Padus. In: Proceedings of the 4th International Conference on Business Process Management (BPM 2006), Vienna, Austria, Springer-Verlag (2006) (to appear).
15. Gybels, K., Brichau, J.: Arranging language features for pattern-based crosscuts. In Akşit, M., ed.: Proc. 2nd Int'l Conf. on Aspect-Oriented Software Development (AOSD 2003), ACM Press (2003) 60–69
16. Campbell, R., Habermann, A.: The specification of process synchronisation by path expressions. In: Proceedings of an International Symposium on Operating Systems. (1974)
17. Luckham, D., Kenney, J., Augustin, L., Vera, D., Bryan, D., Mann, W.: Specification and analysis of system architecture using Rapide. IEEE Transactions on Software Engineering **21** (1995)
18. van den Bos, J., Laffra, C.: PROCOL: A concurrent object-oriented language with protocols delegation and constraints. Acta Informatica **28** (1991) 511–538
19. Reussner, R.H.: Automatic component protocol adaptation with the CoCoNut tool suite. Future Generation Computer Systems **19**(5) (2003) 627–639
20. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Transactions on Programming Languages and Systems **19**(2) (1997) 292–333
21. Wydaeghe, B.: PacoSuite: Component Composition Based on Composition Patterns and Usage Scenarios. PhD thesis, System & Software Engineering Lab, Vrije Universiteit Brussel, Brussels, Belgium (2001)
22. Braem, M., Joncheere, N., Vanderperren, W., Van Der Straeten, R., Jonckers, V.: Concern-specific languages in a service creation environment. In: Proceedings of the 2nd International Workshop on Aspect-Based and Model-Based Separation of Concerns in Software Systems (ABMB 2006), Bilbao, Spain, Elsevier (2006) (to appear).