The service creation environment: A telecom case study

Niels Joncheere
System and Software Engineering Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
njonchee@vub.ac.be

ABSTRACT

Composing web services using current web service composition languages (such as BPEL) requires a large amount of in-depth knowledge. This paper introduces a service creation environment (SCE), which aims to facilitate web service composition by providing a higher level of abstraction and guiding developers in creating valid compositions. The paper presents a case study that investigates how the SCE can be used in a telecom setting, and illustrates the SCE's impact on two important software engineering properties.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Design

Keywords

Web service composition, aspect-oriented programming, concern-specific languages, quality-of-service

1. INTRODUCTION

Over the past years, web services [2] have gained much acceptance, both in academics and in industry, as a means of integrating existing software in new environments. Basic web services can be created by exposing existing applications to a network using XML-based front-ends, which use SOAP [6] for messaging and WSDL [13] for interface description. All of these applications can then be used in the same way, no matter how heterogeneous their underlying implementations may be. Furthermore, new web services can be created by composing a number of basic web services, thus providing more advanced functionality. Naturally, these compound web services can be reused by other web services as well.

Originally, the only way to compose web services was by invoking these web services and processing their results using general-purpose programming languages such as C or Java. It quickly became clear, however, that a composition of web services is more naturally captured by dedicated composition languages. Today, the most well-known web service composition language is the *Business Process Execution Language* (BPEL) [3]. BPEL processes are platform- and transport-independent, and are expressed using XML.

Although dedicated web service composition languages such as BPEL are much better suited for web service composition than general-purpose programming languages, they still require a large amount of in-depth technical knowledge regarding the different language constructs that are available, the interfaces of the concrete web services that can be used, etc. Our approach aims to provide a higher-level, visual *service creation environment* (SCE), which abstracts above this kind of low-level technical knowledge.

Another disadvantage of current web service composition languages is a lack of support for modularization of crosscutting concerns [4]. Therefore, the SCE supports *aspect-oriented software development* (AOSD) [16], a programming paradigm which allows expressing crosscutting concerns in separate aspects. The SCE also supports *concern-specific languages* (CSLs), which are languages specifically designed to express a single concern.

An initial version of the SCE has already been discussed in previous work [8]. The goal of this paper is to present the current state of the SCE, as well as the results of an extensive case study, which was performed in collaboration with our industrial partner. The case study is aimed at evaluating the advantages and disadvantages that may arise when developing a web service composition in a telecom context.

The outline of the paper is as follows: Section 2 gives an overview of the SCE by introducing its general architecture, basic entities, and user interface. Section 3 introduces the use case on which we based our case study, and describes how this use case was designed, implemented, and deployed using the SCE. Section 4 discusses the impact of the SCE on two important software engineering properties. Section 5 provides an overview of related work, and Section 6 states our conclusions and future work.

2. THE SERVICE CREATION ENVIRONMENT

The goal of the service creation environment (SCE) is to provide a higher level of abstraction to web service composition than is currently provided by web service composition

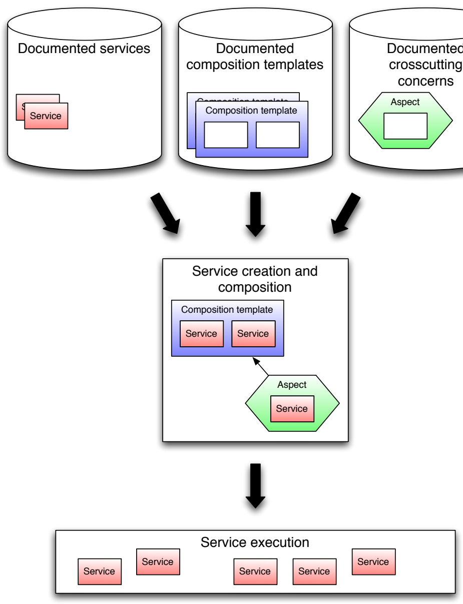


Figure 1: The SCE architecture

languages. Figure 1 provides an overview of the SCE architecture. At the heart of the SCE, there are three repositories, which respectively contain a number of documented services, documented composition templates, and documented aspects. Each of these entities can then be dragged onto a canvas in order to visually create a new composition. During this phase, the user is supported in creating a valid composition: among others, the SCE performs interface and protocol compatibility checking, and computes basic quality-of-service properties of the resulting composition based on the properties of its composing elements. In the final phase of the composition process, the visual composition specification is translated to executable BPEL code, which can be executed on a standard BPEL engine.

2.1 Basic SCE entities

Services are the basic building blocks of the SCE. They correspond to concrete web services that are accessible over the network. The SCE does not impose any requirements on these web services, except that their interfaces should be available as a WSDL documents. WSDL (Web Services Description Language) [13] is an extensible XML-based standard that allows describing the interface of a web service, i.e. the available operations and their parameters (including type information). We have specified an extension to the WSDL specification that allows describing basic quality-of-service properties (e.g. mean response time) of services' operations.

A WSDL document only specifies a service's interface: it does not specify the service's protocol (i.e. the order in which these operations need to be invoked in order to achieve the service's expected behaviour). Therefore, the SCE allows specifying the protocol of a service in a separate document, as an abstract BPEL process.

Composition templates are used to compose services. They correspond to previously implemented web service compositions that have not yet been bound to concrete services. They contain a number of *placeholders* that represent abstract services. By dragging a concrete service onto a placeholder, the abstract service is bound to the concrete service. A composition template is described using an executable BPEL process; the composition template's placeholders correspond to the BPEL process's partner links. Optionally, composition templates can be augmented with quality-of-service constraints.

Aspects are used to encapsulate crosscutting concerns. They are expressed using our aspect-oriented programming language for BPEL, which is called Padus [9]. Similar to composition templates, aspects may contain abstract services.

Padus is an XML-based language, which allows adding advice before, after, or around any activity in a BPEL process. The pointcut language is based on Prolog, and defines a number of built-in predicates that allow selecting any activity or set of activities of a BPEL process. Listing 1 provides an example for a basic logging aspect in Padus, which logs a message before and after any invocation. The `using` element allows adding namespaces, partner links and variables to a BPEL process. These can then be used in the aspect's advice. The `pointcut` element allows defining new pointcuts. The built-in `invoking` predicate selects invocations; its parameters allow specifying the service, port type and operation names of the invocations to be selected. Because none of these parameters are bound to a value in our example, all invocations in the base process will be selected. The `before` and `after` elements define a before and after advice by referring to the generic `log_message` advice, which is parametrized with the message that should be logged.

The SCE defines a basic framework for implementing concern-specific languages on top of Padus. As a proof of concept, we implemented a concern-specific language for billing [7], which uses Padus pointcuts, but introduces advice specifically tailored for billing.

2.2 The SCE graphical user interface

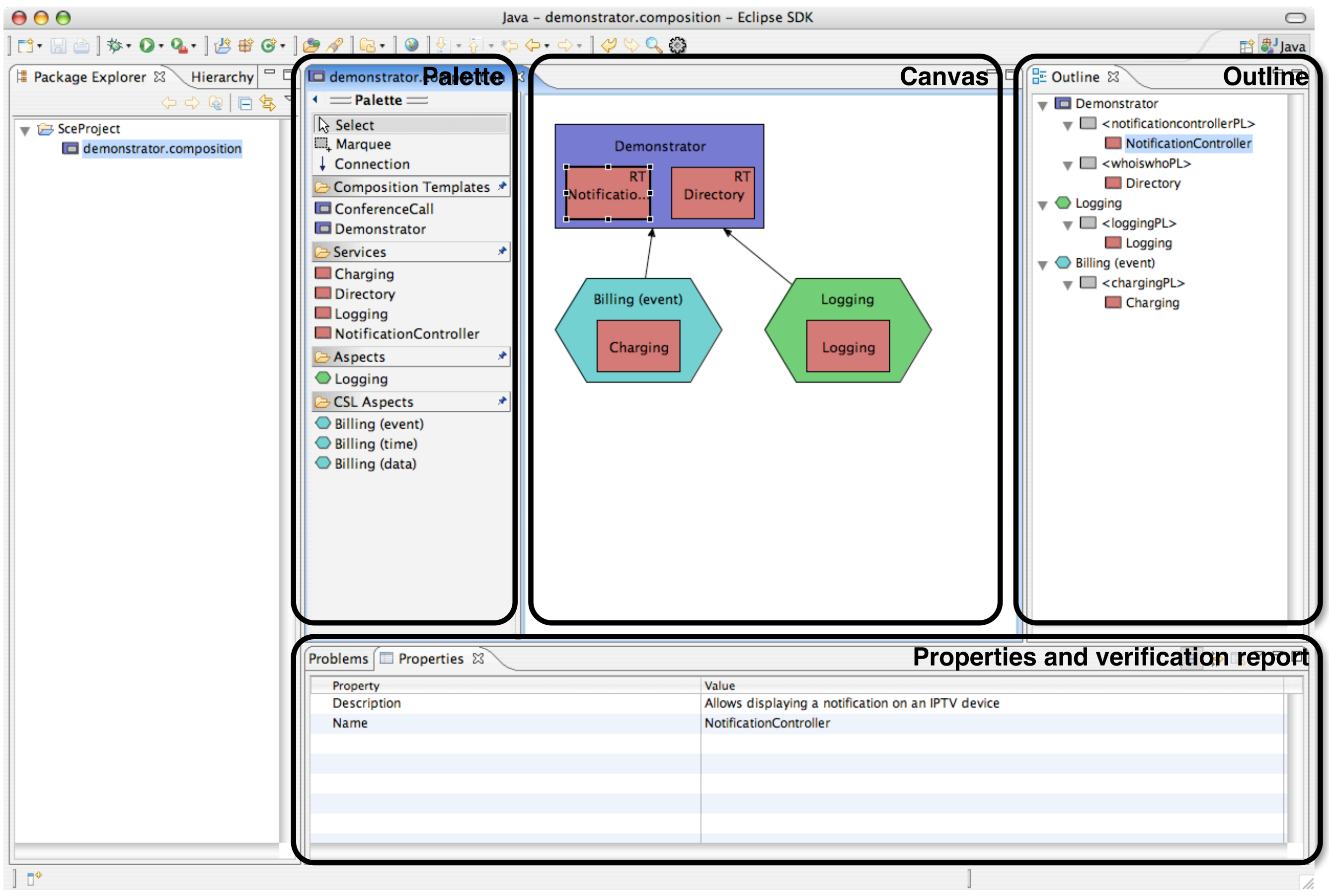
The SCE is implemented as a plug-in for the Eclipse platform.[1] Figure 2 provides an overview of its user interface. The core of the plug-in is the composition editor, which consists of a canvas and a palette. The palette shows all available composition templates, services, and aspects; by dragging these entities on the canvas and onto each other, one can visually create and modify web service compositions. At the right of the composition editor, there is an outline view which shows a hierarchical overview of the composition that is currently being edited. At the bottom of the screen, the properties view shows the properties of the entity that is currently selected in the editor. If the selected entity is an aspect, the properties view also allows modifying its pointcuts (by allowing the developer to textually modify them).

While he/she is editing a composition, the SCE guides the user in creating valid compositions by automatically verifying the compatibility of services with the composition templates and/or aspects to which they are added. If a service is found to be incompatible, an error dialog is displayed. The SCE also allows executing *guidelines*, which are objects that verify certain properties of a composition. The SCE contains a number of built-in guidelines (e.g. with regard to real-time behavior, parallelism, and quality-of-service), but new guidelines can be implemented straightforwardly if the

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

```
<padus:aspect name="Logging"
              xmlns:log="http://ssel.vub.ac.be/logging"
              xmlns:padus="http://ssel.vub.ac.be/padus">

    <padus:using>
        <namespace name="log" uri="http://ssel.vub.ac.be/logging" />
        <partnerLink name="loggingPL" partnerRole="logging" partnerLinkType="tns:loggingPLT" />
        <variable name="loggingRequest" messageType="log:Logging_logMessage_Request_Soap" />
        <variable name="loggingResponse" messageType="log:Logging_logMessage_Response_Soap" />
    </padus:using>

    <padus:pointcut name="all_invocations(Jp, Operation)"
                    pointcut="invoking(Jp, Service, PortType, Operation)" />

    <padus:advice name="log_message(Message)">
        <sequence>
            <assign>
                <copy>
                    <from expression="string('$Message')" />
                    <to variable="loggingRequest" part="parameters" query="/log:logMessage/log:p0" />
                </copy>
            </assign>
            <invoke partnerLink="loggingPL"
                    portType="log:loggingPT"
                    operation="logMessage"
                    inputVariable="loggingRequest"
                    outputVariable="loggingResponse" />
        </sequence>
    </padus:advice>

    <padus:before joinpoint="Jp" pointcut="all_invocations(Jp, Operation)">
        <padus:advice name="log_message('Invoking $Operation...')" />
    </padus:before>

    <padus:after joinpoint="Jp" pointcut="all_invocations(Jp, Operation)">
        <padus:advice name="log_message('Invoked $Operation.')" />
    </padus:after>
</padus:aspect>
```

Listing 1: A basic logging aspect in Padus



Property	Value
Description	Allows displaying a notification on an IPTV device
Name	NotificationController

Figure 2: The SCE graphical user interface

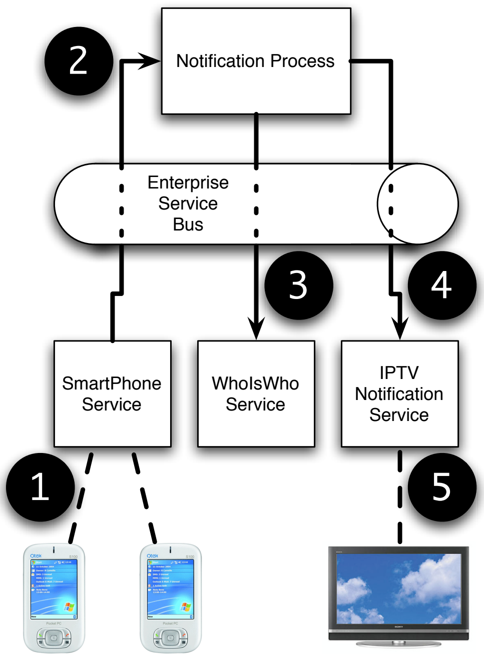


Figure 3: The IPTV notification use case

need arises. The results of guideline verification are shown at the bottom of the screen, in the problems view.

When the user is satisfied with the composition, the visual composition specification can be translated to executable BPEL code, which can then be deployed on any standard BPEL engine.

3. CASE STUDY

The case study we present in this paper is based on a use case which was developed in collaboration with our industrial partner. This use case is illustrated by Figure 3. In the first step of the use case, a smartphone is used to make a call to another smartphone. Before connecting the call, the smartphone service invokes the notification process (step 2). This process will then invoke the who-is-who service (step 3) in order to retrieve the details of the person placing the call based on his/her telephone number, and will instruct the notification service to display a notification on the television of the person being called (steps 4 and 5).

The typical way in which such a use case is developed using the SCE is as follows:

1. The BPEL process that will form the basis of the composition is implemented using the Eclipse BPEL editor,[2] which is integrated into the SCE. This process invokes a who-is-who service and a notification service.

2. The BPEL process is published as a composition template named "WhoiswhoNotification", which can be annotated with quality-of-service constraints.

3. A new composition is created by dragging the "WhoiswhoNotification" composition template onto the canvas. If necessary, the composition template's underlying BPEL process (which we specified in step 1) can be customized for this composition by editing it using the BPEL editor.

4. The placeholders of the composition template are filled in with concrete services.

5. The composition is augmented with crosscutting concerns, such as authorization or billing.

6. The code for the composition is generated and deployed on a BPEL process engine.

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

Each of these phases is explained in more detail below.

3.1 Creating the base process

When creating a new composition, the user has two options: either he/she can reuse a predefined composition template (see phase 3 below), or he/she can start from scratch, creating a new composition template by specifying its BPEL process. Such a process can be specified using the BPEL editor, which allows creating and editing BPEL processes in a visual way. It should be noted that specifying a BPEL process, even if it is done in a visual way, occurs at a lower level of abstraction than when creating a composition using existing composition templates. Hence, composition templates will typically be created by users with more technical knowledge than the average SCE user. The BPEL process for our use case can easily be created using a sequence of assignments and invocations.

3.2 Publishing the process as a composition template

After creating a new BPEL process, the user can publish it as a composition template. Optionally, the composition template can be annotated with quality-of-service constraints. The publishing function will copy the annotated BPEL process to the composition template repository and reload the palette. The new composition template will appear in the palette and can then be used to create new compositions.

3.3 Creating a new composition

A new composition can be created by dragging a composition template on the canvas and filling in its placeholders with concrete services. For example, as illustrated in Figure 2, the "WhoiswhoNotification" composition template contains two placeholders: "whoiswho" and "notificationcontroller". We can either drag services from the palette to these placeholders, in which case the compatibility of the current service with the composition template will be checked, or we can double-click the placeholders, in which case a dialog will show the names and descriptions of the services that are compatible with that placeholder, and allow selecting a service to be added to the placeholder. The "Directory" service is compatible with the "whoiswho" placeholder, and the "NotificationController" service is compatible with the "notificationcontroller" placeholder.

While the composition is being created, a number of guidelines are used to verify whether the composition is valid. Guidelines are objects that verify a composition and provide feedback concerning this verification process. The SCE provides a basic framework for defining such guidelines, as well as a number of predefined guidelines. For example, the quality-of-service guideline will verify whether the quality-of-service constraints we defined on the composition template are satisfied with respect to the concrete services we added.

3.4 Adding crosscutting concerns

The SCE allows adding aspects to compositions by dragging them from the palette to the canvas and connecting them to a composition template. There are two kinds of aspects: Padus aspects and CSL aspects. Padus aspects are previously implemented and stored in the aspect repository; the SCE user can only (textually) modify the aspects' pointcuts. CSL aspects have a structure specified using the SCE's CSL framework; all of the CSL aspects' properties can be modified using the SCE's user interface. In our example, we add a Padus aspects which performs logging and a CSL aspect (that implements the Billing CSL) which performs billing. At the end of this phase, the composition for our use case would look like the one in Figure 2.

3.5 Code generation and deployment

The SCE's code generation and deployment process translates the composition that was visually specified into an executable BPEL file that can be deployed on a standard BPEL engine.

First, the billing aspect is translated to Padus using the CSL framework's compilation class for the Billing CSL. Second, a Padus deployment file is generated for the billing and logging aspects. Based on the aspects' pointcuts and on the composition template to which they are attached, this deployment file specifies where each aspect should be woven into the base process. Next, the aspects are woven into the base process by the Padus weaver, which results in a single executable BPEL file. Finally, this file is deployed on the BPEL engine (by default, the SCE uses the ActiveBPEL engine[3] — in this case a deployment descriptor is generated, which is packaged together with the BPEL file and the WSDL files of the who-is-who, notification, charging and logging services, and copied to the ActiveBPEL deployment directory).

4. IMPACT ON SOFTWARE ENGINEERING PROPERTIES

4.1 Comprehensibility

One of the advantages of the SCE is that it improves the comprehensibility of web service compositions by allowing compositions to be edited on several levels of abstraction. Developers who are not experts in BPEL programming can easily create and modify compositions in the SCE editor, while being supported by the SCE's automatic verification features. Adding crosscutting concerns to compositions can be accomplished straightforwardly by dragging existing aspects to a new composition. Aspects are presented in a way that can easily be grasped by people who are not familiar with aspect-oriented software development.

Developers who desire more control of the compositions they create are not limited to the SCE editor's high-level view on the current composition: by double-clicking a composition template the underlying BPEL processes can be opened in a BPEL editor, which provides a graphical representation of the actual BPEL workflow and allows editing all low-level properties of the current BPEL process. Developers who are even more knowledgeable can edit the BPEL process in a standard text-editor if they want to.

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

4.2 Reusability

The web services technology in itself aims to improve reusability by exposing heterogeneous software applications in a standardized way. The SCE improves on this by not only allowing reuse of the services that are used in web service compositions, but also allowing reuse of the composition logic itself (through the use of composition templates), and of the concerns that crosscut compositions (through the use of aspects). Note that when reusing an aspect in a new composition, the developer may need to modify the aspect's pointcut (which can easily be done in the SCE's properties view).

5. RELATED WORK

Several visual composition environments already exist in the context of component-based software development, which advocates plug-and-play composition of loosely-coupled, reusable, off-the-shelf components [23]. The main difference with our approach, apart from the focus on components instead of services, is that these composition environments do not support reusable encapsulation of composition logic. Furthermore, there is no support for verifying whether a certain composition is possible apart from syntactically checking messages and arguments. Another disadvantage is that they do not support modularizing crosscutting concerns.

Documenting components with protocol documentation is already well investigated in literature [11, 24, 17]. Similar to Yellin and Strom [26], Reussner [20], and Wydaeghe [25], our approach employs automata to document services.

In [19], the Component Workbench is presented. Components from different component models can be composed and combined with web services. The resulting composition can be exposed as a web service. The workbench supports multi-party connectors, however there is no support for modularizing crosscutting concerns.

In [1], an integrated, end-to-end service creation environment is presented. This environment allows for service matching, composition and deployment. As in our approach, common quality-of-service attributes are verified on the service composition. However, crosscutting concerns such as billing, access control, etc. are still tangled throughout the main composition logic, making it hard to add, modify or remove such concerns.

The Taverna workbench [15] allows the construction of complex workflows with focus on the bioinformatics domain. It can also be used to create web services. Their verification is focused on input and output compatibility, and they do not support modularizing crosscutting concerns.

The METEOR-S approach developed a quality-of-service model that allows for the description of non-functional aspects of web services [12].

Sirin *et al.* [21] report on an interactive, goal-oriented approach for web service composition. The composition relies on semantic annotation of web services using OWL-S.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we present a high-level service creation environment. Our approach guides users in creating valid web service compositions by verifying both hard constraints such as service compatibility and softer constraints such as quality-of-service guidelines. Our approach supports the modularization of crosscutting concerns using the Padus aspect-oriented programming language, and provides a framework for developing concern-specific languages on top of Padus. We describe a case study of the service composition environment based on a simplified use case from the telecom applicability domain, and discuss our approach's impact on two important software engineering properties.

Work on our approach is still in progress, and as such a number of improvements are worth investigating:

- Our approach supports the visual deployment of aspects on web service compositions. The pointcuts for these aspects, however, need to be defined using the Padus language. Describing pointcuts at a higher level of abstraction would be an improvement. We are currently experimenting with several existing pointcut visualizations [18, 14, 22].

- The support for developing concern-specific languages for the SCE is currently limited: apart from a framework consisting of a set of common tools and visualizations, defining and implementing a new concern-specific language still largely happens in an ad-hoc manner. A more in-depth solution based on existing research [5, 10] is subject to future work.

Acknowledgments

This research is partly funded by Alcatel Belgium and the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT-Vlaanderen) through the WIT-CASE project — a project which is performed in collaboration with the DistriNet research group of the Katholieke Universiteit Leuven.

7. REFERENCES

[1] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava. A service creation environment based on end to end composition of web services. In *Proceedings of the 14th International World Wide Web Conference (WWW 2005)*, pages 128–137. ACM Press, 2005.

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, editors. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, Heidelberg, Germany, 2004.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, version 1.1, May 2003. `http://www.ibm.com/developerworks/library/ws-bpel/`.

[4] A. Arsanjani, B. Hailpern, J. Martin, and P. Tarr. Web services: Promises and compromises. *Queue*, 1(1):48–58, 2003.

[5] D. Batory, V. Singhal, J. Thomas, S. Dasari, B. Geraci, and M. Sirkin. The GenVoca model of software-system generators. *IEEE Software*, 11(5):89–94, 1994.

[6] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol, version 1.1. W3C Note 08 May 2000, World Wide Web Consortium, May 2000. `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`.

[7] M. Braem, N. Joncheere, W. Vanderperren, R. Van Der Straeten, and V. Jonckers. 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, July 2006. Elsevier.

[8] M. Braem, N. Joncheere, W. Vanderperren, R. Van Der Straeten, and V. Jonckers. Guiding service composition in a visual service creation environment. In *Proceedings of the 4th European Conference on Web Services (ECOWS 2006)*, Zürich, Switzerland, December 2006. IEEE Computer Society.

[9] M. Braem, K. Verlaenen, N. Joncheere, W. Vanderperren, R. Van Der Straeten, E. Truyen, W. Joosen, and V. Jonckers. Isolating process-level concerns using Padus. In *Proceedings of the 4th International Conference on Business Process Management (BPM 2006)*, Vienna, Austria, September 2006. Springer-Verlag.

[10] J. Brichau. *Integrative Composition of Program Generators*. PhD thesis, Programming Technology Lab (PROG), Vrije Universiteit Brussel, Brussels, Belgium, September 2005.

[11] R. Campbell and A. Habermann. The specification of process synchronisation by path expressions. In *Proceedings of an International Symposium on Operating Systems*, pages 89–102, April 1974.

[12] J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 1(3):281–308, 2004.

[13] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language, version 1.1. W3C Note 15 March 2001, World Wide Web Consortium, March 2001. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[14] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design — The Theme Approach*. Addison-Wesley, 2005.

[15] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: A tool for building and running workflows of services. *Nucleic Acids Research*, 34, 2006.

[16] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. Technical Report SPL97-008 P9710042, Xerox PARC, February 1997.

[17] D. Luckham, J. Kenney, L. Augustin, D. Vera, D. Bryan, and W. Mann. Specification and analysis of system architecture using Rapide. *IEEE Transactions on Software Engineering*, 21, 1995.

[18] M. Mahoney, A. Bader, T. Elrad, and O. Aldawud. Using aspects to abstract and modularize statecharts. In O. Aldawud, G. Booch, J. Gray, J. Kienzle, D. Stein, M. Kandé, F. Akkawi, and T. Elrad, editors,

The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004, October 2004.

[19] J. Oberleitner and S. Dustdar. Constructing web services out of generic component compositions. In *Proceedings of the 1st International Conference on Web Services — Europe (ICWS-Europe 2003)*, pages 37–48. Springer-Verlag, 2003.

[20] R. H. Reussner. Automatic component protocol adaptation with the CoCoNut tool suite. *Future Generation Computer Systems*, 19(5):627–639, 2003.

[21] E. Sirin, B. Parsia, and J. Hendler. Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4):42–49, 2004.

[22] D. Stein, S. Hanenberg, and R. Unland. Expressing different conceptual models of join point selections in aspect-oriented design. In *Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD 2006)*, Bonn, Germany, 2006. ACM Press.

[23] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press and Addison-Wesley, New York, NY, USA, 1998.

[24] J. van den Bos and C. Laffra. PROCOL: A concurrent object-oriented language with protocols delegation and constraints. *Acta Informatica*, 28:511–538, June 1991.

[25] B. Wydaeghe. *PacoSuite: Component Composition Based on Composition Patterns and Usage Scenarios*. PhD thesis, System & Software Engineering Lab, Vrije Universiteit Brussel, Brussels, Belgium, November 2001.

[26] D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, March 1997.