Guiding Service Composition in a Visual Service Creation Environment

Mathieu Braem, Niels Joncheere, Wim Vanderperren, Ragnhild Van Der Straeten,
and Viviane Jonckers
System and Software Engineering Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
{mbraem,njonchee,wvdperre,rvdstrae}@vub.ac.be,
vejoncke@ssel.vub.ac.be

Abstract

Current web service composition languages like WS-BPEL require in-depth knowledge of these languages by the service composition designers. In this paper we present a high-level, visual Service Creation Environment (SCE). This SCE provides service composition templates, verification of compatibility and guidelines, and advanced separation of concerns through Aspect-Oriented Software Development. Composition templates are abstract descriptions of reusable compositions containing several placeholders for services. Services are verified to be compatible with the composition template when a service is mapped onto a composition template's placeholder. Composition guidelines such as QoS constraints can be added to the SCE and verified. The modularization of crosscutting concerns is supported by the SCE through the general-purpose Padus Aspect-Oriented Programming language and the possibility to add concern-specific languages on top of Padus. The SCE generates the appropriate WS-BPEL processes given a complete and verified service composition.

1. Introduction

Over the last years, *web services* [2] 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 composition languages than by general-purpose programming languages. Today, the most popular language with respect 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) [33], has been proposed.

Although languages like WS-BPEL are designed specifically for web service composition as opposed to general-purpose programming languages, they still require a large amount of in-depth technical knowledge. For example, in the telecom world, due to the intensive competition, a fast introduction of new services is needed through innovative service creation mechanisms. Consequently, the service developer needs to be able to learn how to use these technologies easily. Some of the key principles of Business Driven Development (BDD) [21] are: (1) to cope with complexity, (2) to deliver incremental value to enable early and continuous feedback and (3) to focus continuously on quality.

Complexity can be reduced by working at a higher level of abstraction and by supporting modularization of so-called crosscutting concerns. The aim is to use higher-level tools and languages and as such reduce the amount of manual intervention. To enable early feedback to the user the composition process has to support, e.g. the verification of QoS requirements and the automatic generation of service compositions at an early stage of development. By providing support for guiding the composition process quality is ensured throughout the BDD life-cycle. To accommodate the above specified requirements, our approach supports:

- Verifying whether a service applied to a partner role is effectively able to function in that role. For instance,

the protocol of the service might be incompatible.

- Verifying whether QoS constraints can be met using the given services and workflow composition. This is especially important for real-time processes.
- Guiding the composition process by verifying other quality attributes of the process in typical application domains. For instance, certain activities need to be performed in parallel in certain applications.
- Supporting the modularization of concerns that cannot be nicely isolated in the main process description [20] such as billing [15] or access control [14]. Arsanjani *et al.* [4] show that such *crosscutting* concerns are a serious problem in web service development.

In order to facilitate and guide service composition a higher-level view above, e.g. WS-BPEL, and corresponding tool support is required. We therefore propose a visual *Service Creation Environment* (SCE), which allows user-friendly composition of services using reusable *composition templates*, and which supports *encapsulating crosscutting concerns* using both general-purpose and concern-specific aspect languages. This environment is implemented as a plug-in for the Eclipse [16] platform.

The research presented in this paper is conducted in the context of the WIT-CASE project, which is partly funded by Alcatel Belgium, a telecom company, and by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

The outline of the paper is as follows. The next section introduces the running example used throughout the paper. Section 3 provides an overview of the SCE and its main concepts. Afterwards, the composition process and verification support of the SCE is explained in more detail. Section 5 describes related work, and Section 6 states our conclusions and future work.

2. Running Example

This section introduces the *multi-party conference call* as a realistic example that will be used throughout this paper. Remark that the example is a simplified version of a use case delivered by Alcatel.

A *multi-party conference call service* orchestrates the planning of a multi-party conference call into the agenda of the participating parties. An *alert service* will initiate the multi-party conference call at the specified date and time. Upon unavailability of one of the parties, a notification will be sent to that party on the delivery channel and terminal as specified by that party.

The composition of the different services establishing a multi-party conference call is expressed in the Business Process Modeling Notation (BPMN). This notation has the potential to become the standard notation for business process modeling. In the context of service composition, it provides the service designer with a high-level graphical language for expressing service compositions. Therefore we elaborate the multi-party conference call example providing the corresponding BPMN diagrams.

The multi-party conference call service consists of two parts: a provisioning part, which is the part that describes what is done when the conference call is planned, and a real-time part, which is the part that describes what is done when the conference call is actually performed.

Five services are used in the provisioning part: the *ConferenceCallProvisioning* service, the *bridge and call controller* (BCC) service, the *agenda* service, the *messaging* service and the *alert* service. Figure 1 provides the BPMN Business Process Diagram (BPD) for the provisioning part of the example. In order to plan a conference call, the initiator of the conference call will contact the ConferenceCallProvisioning service. This service will first use the agenda service to add the call to the agenda of each of the attendees. After that, it will request an alert from the alert service for the planned time for the conference call, and configure through the bridge and call controller service possible supplementary services. If an error occurs during the execution of the "SetAlert" or "ConfigureBCC" tasks, a message will be sent by the messaging service to the initiator of the conference call indicating that something went wrong.

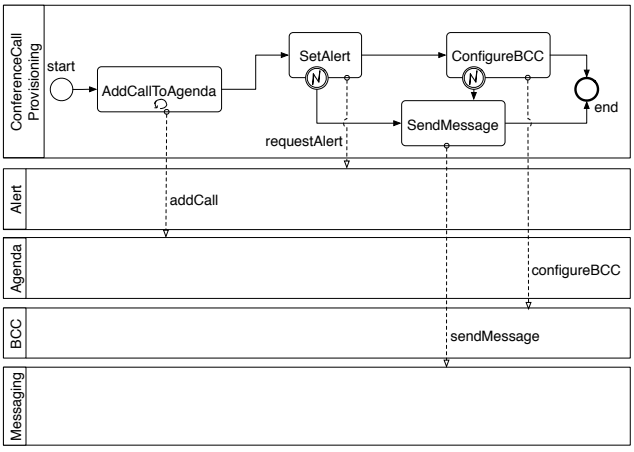

Figure 1. BPD for the provisioning part

Figure 2 provides the BPD for the real-time part. Five services are used in the real-time part: the *ConferenceCallRealTime* service, the bridge and call controller service, the agenda service, the alert service, and the messaging service. At the planned time for the conference call, the alert service calls the ConferenceCallRealTime service in order to start the real-time part of the conference call. This service will retrieve the call's attendees from the agenda service. If there

is only one participant, the process sends a message to this participant before terminating. If there are several participants the bridge and call controller service is called. This service will create the conference call and will invite the attendees. If an attendee is currently unavailable, the bridge and call controller service will send a message to the corresponding attendee containing the contact number for the conference call using the messaging service. After this, all available attendees can start the conference call.

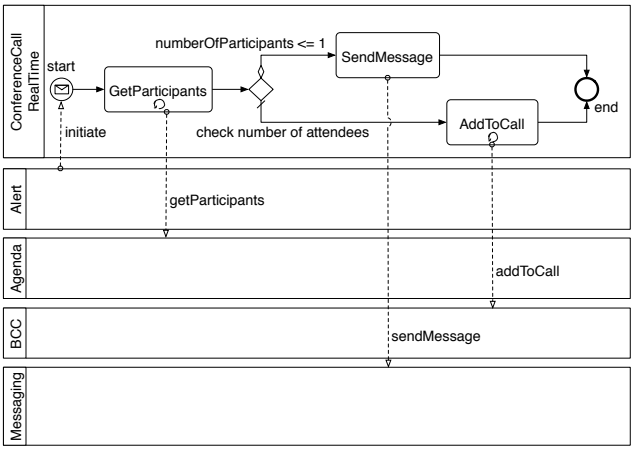

Figure 2. BPD for the real-time part

3. The Service Creation Environment

In this section, we introduce the main concepts and underlying tools and languages of the SCE. First, the main concepts of the SCE are explained. Secondly, we motivate the use of AOP principles and show how aspects can be defined in our AOP language Padus. The SCE allows the definition and usage of concern-specific languages that are built on top of Padus. We briefly describe an example of such a concern-specific language. Finally, the GUI of the SCE is presented.

3.1. Concepts of the SCE

Figure 3 gives an overview of the main concepts of the SCE. The SCE contains three repositories:

- A first repository contains a set of *documented services*. The services contained in this repository are the basic building blocks of the SCE. These services can be in-house or third-party services.
- Another repository contains a set of documented *composition templates*. A composition template is specified by one or several WS-BPEL processes in the SCE.

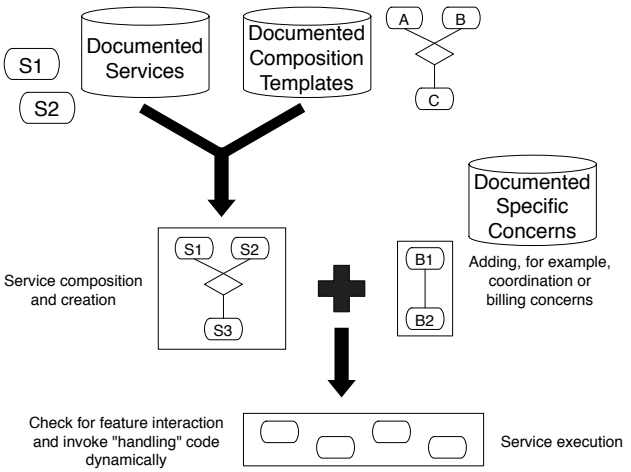

Figure 3. SCE architecture

- A third repository contains different *crosscutting concerns* corresponding to management concerns such as billing schemes. A crosscutting concern can be connected to services and composition templates visually and further refined through a pointcut language. These concepts are detailed in Section 3.3 and 3.4.

In Section 4, we explain how the service designer is supported in the creation of a service composition.

3.2. Services and Composition Templates

The services stored in the first SCE repository are documented by a WSDL [11] file. The WSDL file may contain a description of basic quality of service requirements. In the current implementation of our SCE, focus is on common QoS attributes like cost and average response time. Furthermore, a service needs to be documented by a WS-BPEL process that specifies the external protocol information (and that can be visualized by a BPD). Third-party services that are documented by other means than WSDL files and WS-BPEL processes are also amenable for verification by our SCE tool if it is possible to construct an appropriate WSDL file and WS-BPEL process from their documentation.

Composition templates are abstract descriptions of web service compositions and may contain one or more placeholders for services. These abstract descriptions of service compositions are visualized by BPDs. Figures 1 and 2 are examples of visualizations of such abstract descriptions of the conference call template. Remark that the conference call composition template contains a non-real-time part (the provisioning part) and a real-time part (the actual communication) resulting in multiple WS-BPEL specifications. The repository of composition templates can contain pre-defined composition templates, but the user can also define new

composition templates in the SCE. Consider again the BPDs in Figure 1 and 2. These BPDs can be drawn by the user in a BPMN editor and exported to WS-BPEL code. This WS-BPEL code represents abstract processes and is interpreted by the SCE as a composition template.

3.3. Aspects and Padus

Aspect-oriented software development (AOSD) has been proposed as a means of improving *separation of concerns* [26] in software. AOSD is based on the observation that a number of concerns in software (such as logging [19] and billing [15]) cannot be modularized using object-oriented software development: a program can only be decomposed in one way (e.g. according to the class hierarchy in object-oriented programming), 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" [25]. 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.

Initial research on AOSD has concentrated on applying its principles to the object-oriented programming paradigm. Arsanjani *et al.* [4] and others [10, 13, 32] have shown that AOSD has a lot of potential in a web services context too.

The SCE supports the modularization of crosscutting concerns through the Padus [6] language. Padus is our aspect-oriented process language based on WS-BPEL. A detailed explanation of Padus is out of the scope of this paper, but the interested reader is referred to [6]. We only introduce and explain the features of Padus relevant for 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 advices. A pointcut selects interesting points in the execution of the target WS-BPEL process (called joinpoints), and exposes target objects to the advice, which expresses extra behavior that should be inserted at that pointcut.

The pointcut language of Padus is a logic language based on Prolog, and is thus very expressive in the sense that it is declarative and naturally supports the specification of pointcuts. [17]. 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 AOSD-specific constructs. Advice code is defined in an XML element that specifies the type of the advice. For before, after and around advices, this is a WS-BPEL activity. In advices can be used to insert other WS-BPEL elements too. For around advices, the `<proceed>` activity can be used to include the original behavior specified by the joinpoint. Listing 1 shows an example of a before advice that logs all invocations of the `smsServicePT` web service. The extra behavior that is inserted, is a sequence of two activities: first, the log message containing the invoked operation is created; then, this message is sent to the logging service.

```
1  <before joinpoint="Jp"
2          pointcut="invoking(Jp,'smsService',
3                      'smsServicePT',Operation)">
4    <sequence>
5      <assign>
6        <copy>
7          <from>Logging invocation of operation
8                $Operation</from>
9          <to variable="logMsg" part="msg"/>
10       </copy>
11     </assign>
12     <invoke partnerLink="logging"
13             portType="log:loggingPT"
14             operation="logMessage"
15             inputVariable="logMsg"/>
16   </sequence>
17 </before>
```

Listing 1. An advice that logs all invocations of the SMS service

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 as specified in our visual environment. The main advantage of this approach is the compatibility with existing infrastructure, as the output can be deployed on any WS-BPEL-compatible engine.

3.4. Concern-Specific Languages

Our Padus language allows the implementation of aspects describing a certain crosscutting concern. Padus is a general purpose AOP language and programming a certain concern requires in-depth knowledge of the Padus language. This is in contradiction with our research objective to allow the description of management concerns in an intuitive, concise and declarative manner. Therefore, we need the ability to (visually) specify concerns on a higher level of abstraction. Our solution is to enable the definition of concern-specific languages on top of the Padus technology and integrated in the SCE.

An example of a crosscutting concern that occurs in many service compositions is *billing*. Billing can be as simple as deducting a fixed fee from a client's account after the execution of an operation, but it can also require complicated schemes based on the client's location, standing, which operation was executed, how long it took, etc. Therefore a dedicated concern-specific Billing language is provided to the service designer. The SCE also allows the defi-

```
1  <concern:billing type="time" name="billcall">
2    <!-- specify when billing should occur: -->
3    <start when="invoking(Service,Port,'connect',User)"/>
4    <end when="invoking(Service,Port2,'disconnect',User)"/>
5
6    <!-- specify what should be charged: -->
7    <advice>
8      <begin> <charge type="setup" context="User"/>
9      </begin>
10     <success> <charge type="time" context="User,$Time"/>
11     </success>
12     <fail> <!-- do nothing --> </fail>
13     <finally> <!-- do nothing --> </finally>
14   </advice>
15 </concern>
```

Listing 2. Billing example

nition of other concern-specific languages. These languages are XML-based languages too as they are built on top of the Padus general-purpose AOP language.

In the Billing language, there are three types of Billing modules: *event-based* modules are used to perform billing based on events that occur during the execution of a service (e.g. when a text message has been sent), *time-based* modules are used to perform billing based on the time that has passed between two events (e.g. between the start and the end of a telephone call), and *data-based* modules are used to perform billing based on the volume of data that has been exchanged between two services. Next, it is possible to specify for each kind of Billing module to specify *when* billing should occur, and *what* should be charged.

Listing 2 provides an example of a time-based Billing module (cf. line 1). Because our example is a time-based module, it specifies both when billing should begin (using the `start` element in line 3) and when billing should end (using the `end` element in line 4). The `when` attributes of the `start` and `end` elements are Padus pointcuts that select certain points in the execution of a service.

Each module specifies what should be charged in the `advice` element. This element has four children: the `begin` element specifies what should be done when the concern is activated, the `success` element specifies what should be done when the concern terminates successfully, the `fail` element specifies what should be done when an exception is thrown while the concern is active, and the `finally` element specifies what should be done when the concern terminates, regardless of whether it terminates successfully or not.

To apply a billing concern to the service process or service composition process, a user can select the relevant aspects, add them to the service process and concretize them.

3.5. SCE GUI

Figure 4 provides a screenshot of the SCE's user interface. 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, aspects expressed using Padus, and aspects expressed in a dedicated concern-specific language as they are loaded from the library. By double-clicking on a service or a service composition, the configured editor for that service or composition is launched. By default we use a BPMN editor but remark that other process modelling notations or languages can be used to visualize the composition and the internal flow of a service.

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

The next section explains in more detail how the service designer is guided through the composition process by verifying several service composition properties.

4. Guiding the Composition Process

In order to create a composition in the SCE, it suffices to drag a composition template on the composition canvas and fill all 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, in order to apply them to all the services that take part in this composition.

The composition shown in Figure 4 contains a composition template called "conferenceCall" with four service placeholders. Three services — a concrete "alert" service corresponding to the alert service (cf. Section 2), a concrete "SMS" service corresponding to the messaging service and a concrete "agenda" service corresponding to the agenda service — have been added to the composition template's placeholders, while one placeholder is still empty. This placeholder should be filled in before the composition can work, for instance using the "BCC" service available in the library. For each of the services it is indicated whether they belong to the real-time part or to the non-real-time part or to both parts in the conference call composition.

4.1. Protocol Verification

An important requirement of the SCE is that it supports and guides users in creating valid compositions. 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

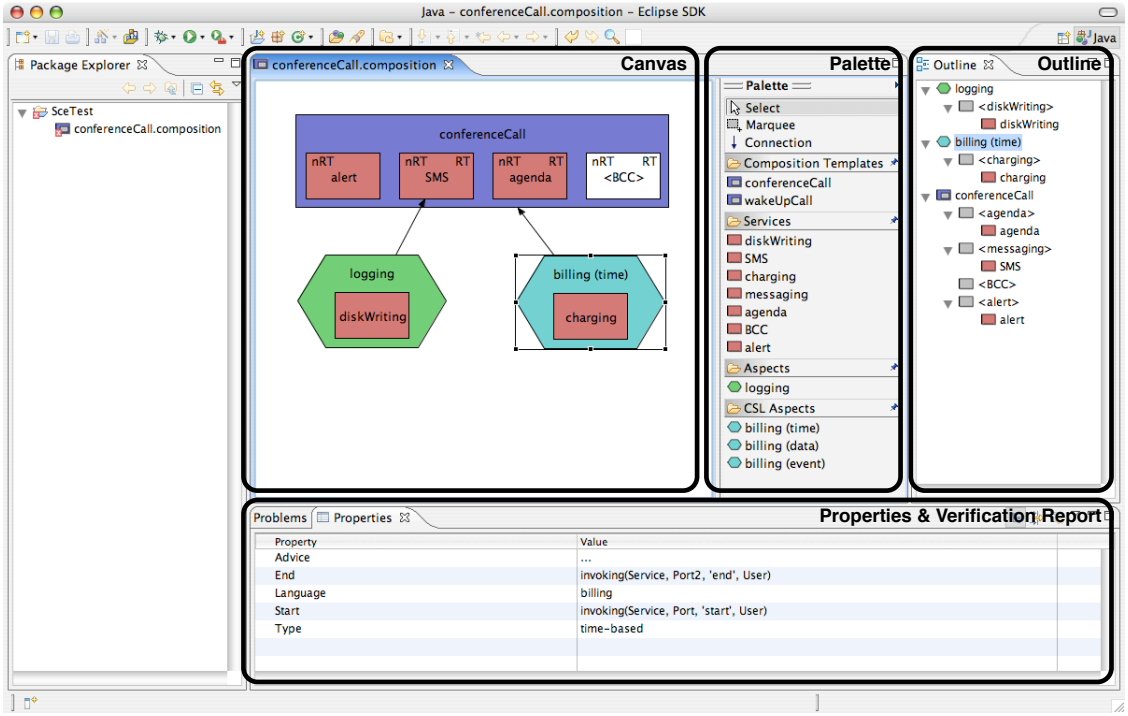

Figure 4. Screenshot of the SCE

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 [8, 22, 27, 31, 35]. Our verification engine is based on the PacoSuite approach [34], 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. More specifically, the PacoSuite algorithm consists of constructing an automaton that covers the complete composition by taking a specialized intersection of the composition template's automaton with each service's automaton. The composition is invalid when the resulting automaton is empty (i.e. no path from a start to stop state). In that case a report is generated that pinpoints which operations of which service does not follow the composition template's protocol. In case more detail is required, the user can also view the generated automaton.

Because the algorithm is based on taking intersections of automatons, the resulting performance is in worst-case exponential with respect to the size of the input automata. Notice that the algorithm is only executed at composition time and thus does not interfere with the running application. Furthermore, in practice the verifier seems to work in an acceptable time frame. For instance, a quite involved case study consisting of seven composition templates and about thirty services can be verified in under ten seconds on our test system.[1]

To demonstrate the idea we go back to the running example. The composition template expects a service that handles the scheduling logic: the agenda service. The vendor can choose from several alternatives in his service repository, each handling a different back-end. It is important for the composition that the protocol of the chosen service matches the protocol that is expected by the composition template. In this case we know from the service's WSDL file and from the protocol specification extracted from the service's WS-BPEL file that the operations `AddCall` and `GetParticipants` are supported and in which order these are invoked. Translating the protocol information to DFAs, we can verify that the service's protocol is compatible with the composition template and we allow it in the composition.

It is possible that an aspect adapts the external protocol of an existing service (e.g. by adding an invocation) so that it becomes incompatible with the composition template's

[1] Mac OS X 10.4, Intel Core Duo 2Ghz, 1GB DDR2-RAM

protocol. Our tool therefore employs the Padus weaver both on the composition template and on the services' WS-BPEL protocol specification before translating them to DFAs. As such, the effect of the aspect on the external protocol of the composition template and services is visible and can be taken into account by the verification engine.

4.2. Aspects and CSLs

The composition shown in Figure 4 contains an aspect called "logging", which is connected to the "messaging" service. A service called "diskWriting" has been added to the aspect's only placeholder. The result of this composition is that the conference call application works using the selected services, and that a logging aspect, which invokes the disk writing service, is deployed to the messaging service to log the messaging actions selected by the aspect's pointcut.

Notice that the aspects themselves define a declarative pointcut that selects interesting joinpoints in the target workflow for the aspect at hand. The user of the SCE can only visually select the deployment scope, i.e. a complete composition or a specific service.

Next to this "logging" aspect, the composition is also affected by a concern-specific billing aspect. The palette contains a library of templates for concern-specific aspects, which may be instantiated by dragging them on the canvas. The palette in the example contains three such templates, i.e., "billing (time)", "billing (event)" and "billing (data)", which correspond to the three types of billing that are identified above.

When a concern-specific aspect is dragged on the canvas and selected in the editor view, its properties appear in the properties view. A time-based billing aspect, for example, has five properties: "language", "type", "start", "end", and "advice". The first two properties simply show the language and the type of the aspect, respectively. The other properties, however, can be changed in order to define between which two points in the execution of the composition billing should occur, and how this billing should be achieved. Based on this information, the information for the corresponding Billing module is generated by the SCE.

4.3. Guideline Verification

In the context of the WIT-CASE project, the partners have identified a non-exhaustive list of conditions that can apply to service compositions in order to ensure efficient execution. We specified a number of programmable guidelines that statically check these conditions and detect bad smells in service compositions. These guidelines are optional and can be enabled and disabled by the service designer in the SCE. We list some of these guidelines here:

- **Quality-of-service** In the composition template we set constraints on placeholders for services, limiting the services that can be used to those that have an execution time bounded within these constraints.

- **Short lived real-time processes** This guideline is closely related to the quality-of-service guideline, but takes the execution time of the complete real-time part of the composition (instead of operations on single services) into account. The minimal execution time of the real-time process can be computed from the service composition modeled and the SCE generates a warning if this execution time exceeds a certain predefined amount of time.

- **Parallelism** Handling tasks concurrently improves efficiency because independent tasks do not have to wait for others to finish. This guideline verifies if activities are situated in parallel branches.

- **Asynchrony** The asynchrony guideline verifies that the invocations used in the composition are always asynchronous, i.e., non-blocking calls.

- **Subdivision** Two-way communication between a real-time and a non-real-time process should be limited and a warning is generated if the communication between these two parts exceeds a certain number of calls.

To illustrate the quality-of-service guideline we go back to the running example. The composition template requires a messaging service to notify the participants of the conference call of any errors. Imagine that the composition designer has two different services available that handle messaging: one works by sending a text message to the participants' phones, the other by sending an email to the participants. In the documentation for these services, it is declared that sending an SMS message takes 10 time units, while sending an email takes 5 time units. The composition templates states that using the messaging service can take a maximum of 5 time units, so only the email messaging service is accepted by the guideline.

Figure 5 shows a screenshot of the output of the guideline verification in the SCE. Two warnings are shown. The first one reports the violation of the quality-of-service by the concrete SMS service as explained above. The second warning indicates that the qsynchrony guideline is violated because a synchronous invocation is found in the composition specification of the conference call composition. The error item shown in the screenshot in Figure 5 specifies that the composition is not complete because the "BCC" placeholder is not yet filled in.

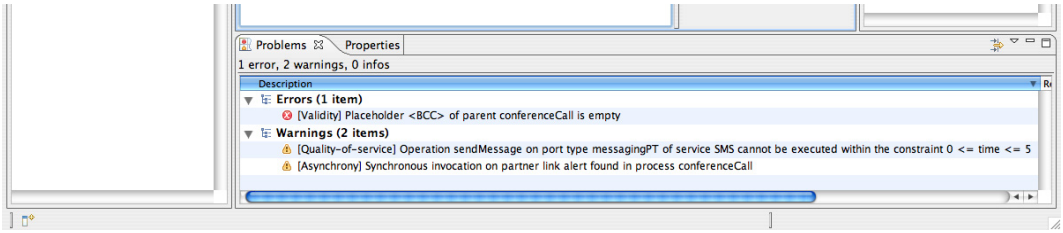

Figure 5. Guideline verification report

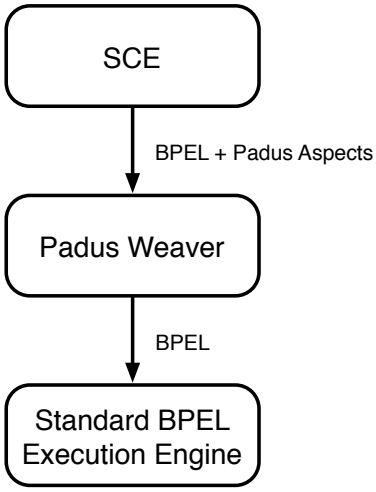

Figure 6. Code generation flow

4.4. Code Generation and Deployment

When the composition is complete and verified, the user may choose to generate the resulting composition and deploy it on a WS-BPEL engine. This will start the code generation process (of which the flow is denoted in Figure 6), 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. Remark that for the aspects written in a concern-specific language there is an extra step of indirection. These aspects are first translated into regular Padus aspects, and then woven into the code.

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 for the new service, this process is also automated.

The SCE also includes a built-in WS-BPEL engine that can be used to execute immediately a resulting composition. This feature is meant to be able to assess quickly 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.

5. Related Work

Several visual component composition environments already exist in the context of Component-Based Software Development (CBSD). CBSD advocates reusable and loosely-coupled components in order to realize flexible plug-and-play component composition of off-the-shelf components [30]. The main problem in CBSD is that wiring components together requires writing glue-code manually in order to resolve syntactic and semantic incompatibilities. A visual component composition environment allows to visually compose the components and supports the (semi-)automatic generation of glue-code that implements the composition. Current practice component composition environments, like for example JBuilder from Borland allow already some form of automatic glue-code generation from a given component composition. The main difference with our approach, apart from the focus on components instead of services, is that they do not support a 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. Campbell and Habermann [8] introduced the idea of augmenting interface descriptions with sequence constraints already in 1974. More recent work includes the Rapide system [22] or the PROCOL system [31]. 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 [35], Reussner's CoCoNut project [27] and PacoSuite [34] 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.

In [24] 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 QoS attributes are verified on the service composition. However, in [1] 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 [18] 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 QoS model that allows for the description of nonfunctional aspects of web services [9]. To automatically compute the overall QoS of web service composition, a mathematical model has been designed and implemented. An issue of future work is to investigate whether we could use this algorithm in our approach. Sirin *et al.* [28] report on an interactive, goal-oriented approach for service composition. The composition relies on semantic annotations of services with OWL-S. Extending our verification techniques with semantic information provided by marking up content is an issue of future work.

6. Conclusions and Future Work

In this paper, we present a high-level Service Creation Environment for composing services. Our approach guides the service composer in creating service compositions by verifying both hard constraints such as service compatibility and softer constraints such as quality guidelines. Furthermore, the SCE supports the modularization of crosscutting concerns through the Padus aspect language. The SCE also provides a framework for developing concern-specific languages on top of Padus.

Our work is still in an early phase and as such several improvements are possible:

- Our approach supports the visual deployment of aspects onto concrete services. The pointcuts still have to be defined programmatically in Padus. Describing pointcuts at a higher level of abstraction would be an important contribution to our work. We are experimenting with existing pointcut visualizations such as Theme/UML [12], Join Point Designation Diagrams [29] and AOSF [23] to solve this problem.

- The support for integrating concern-specific languages is currently quite limited. Apart from a framework consisting of a set of common tools (such as XML parsing and transformation tools) and a visualization template language, defining and implementing a new concern-specific language still largely happens in an ad-hoc manner. A more in-depth solution based on existing work (e.g. Babel [7] or GenVoca [5]) is subject to future work.

- The verifier now only checks as soon as a service is dragged on a placeholder. A more user-friendly approach would already pre-check compatible services for each placeholder and only allow to select from the list of pre-approved services. We plan to alter our prototype tool to be able to support this.

- A case study of a real deployment is needed to validate our approach in an industrial setting.

Acknowledgments

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] 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 WWW 2005*, pages 128–137. ACM, 2005.

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

[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] 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 S. Dustdar, J. Fiadeiro, and A. Sheth, editors, *Proceedings of the*

4th International Conference on Business Process Management (BPM 2006), volume 4102 of *LNCS*, pages 113–128. Springer, 2006.

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

[8] 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, Apr. 1974.

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

[10] A. Charfi and M. Mezini. Aspect-oriented web service composition with AO4BPEL. In L.-J. Zhang, editor, *Proceedings of the 2nd European Conference on Web Services (ECOWS 2004)*, volume 3250 of *LNCS*, pages 168–182. Springer, Sept. 2004.

[11] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL), version 1.1. W3C Note 15 March 2001, World Wide Web Consortium, Mar. 2001.

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

[13] T. Cottenier and T. Elrad. 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)*, pages 56–63, Miami, FL, USA, May 2005.

[14] B. De Win, W. Joosen, and F. Piessens. Developing secure applications through aspect-oriented programming. In R. E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors, *Aspect-Oriented Software Development*, pages 633–650. Addison-Wesley, Boston, 2005.

[15] M. D'Hondt and V. Jonckers. Hybrid aspects for weaving object-oriented functionality and rule-based knowledge. In K. Lieberherr, editor, *Proc. 3rd Int' Conf. on Aspect-Oriented Software Development (AOSD-2004)*, pages 132–140. ACM Press, Mar. 2004.

[16] The Eclipse platform. http://www.eclipse.org/.

[17] K. Gybels and J. Brichau. Arranging language features for pattern-based crosscuts. In M. Akşit, editor, *Proc. 2nd Int' Conf. on Aspect-Oriented Software Development (AOSD-2003)*, pages 60–69. ACM Press, Mar. 2003.

[18] 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. *Nucl. Acids Res.*, 34(suppl_2):W729–732, 2006.

[19] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In J. L. Knudsen, editor, *Proc. ECOOP 2001*, volume 2072 of *LNCS*, pages 327–353, Berlin, June 2001. Springer.

[20] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *11th European Conf. Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer Verlag, 1997.

[21] P. Kroll and W. Royce. Key principles for business-driven development, 2005. http://www-128.ibm.com/developerworks/rational/library/oct05/-kroll/index.html.

[22] 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.

[23] 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*, Oct. 2004.

[24] J. Oberleitner and S. Dustdar. Constructing Web Services out of Generic Component Compositions. In M. Jeckle and L. Zhang, editors, *Proceedings of the International Conference on Web Services Europe*, volume 2853 of *LNCS*, pages 37–48, Berlin, Heidelberg, New York, et al., August 2003. Springer.

[25] H. Ossher and P. Tarr. Using subject-oriented programming to overcome common problems in object-oriented software development/evolution. In *Proc. 21st Int'l Conf. Software Engineering*, pages 687–688. IEEE Computer Society Press, 1999.

[26] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(12):1053–1058, Dec. 1972.

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

[28] 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.

[29] D. Stein, S. Hanenberg, and R. Unland. Expressing different conceptual models of join point selections in aspect-oriented design. In *AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development*, pages 15–26, New York, NY, USA, 2006. ACM Press.

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

[31] 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.

[32] B. Verheecke, W. Vanderperren, and V. Jonckers. Unraveling crosscutting concerns in web services middleware. *IEEE Software*, 23(1):42–50, Jan. 2006.

[33] S. A. White. Business Process Modeling Notation (BPMN), version 1.0, May 2004.

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

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