Requirements for Applying Aspect-Oriented Techniques in Web Service Composition Languages

Mathieu Braem and Niels Joncheere

System and Software Engineering Lab (SSEL)
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
mbraem@vub.ac.be, njonchee@vub.ac.be

Abstract. In current composition languages for web services, there is insufficient support to explicitly separate crosscutting concerns, which leads to compositions that are hard to maintain or evolve. A similar problem in object-oriented languages is being tackled by aspect-oriented programming, and some work has been started to apply these techniques to web service composition languages as well. We identified some problems with these approaches. This short paper lists these limitations and offers a number of requirements to apply aspect-oriented techniques to workflow languages.

1 Introduction

Web services [1] are a popular way of making existing software available as external services over a network, through standardized protocols. In recent years, workflow languages [2] are used to express these web service compositions. One of the most popular languages today, is WS-BPEL [3].

Even though WS-BPEL, and other workflow languages, are very well suited to express these kind of compositions, its abstraction model doesn't explicitly support a good separation of concerns [4]. WS-BPEL processes are coded according to the requirements of the core functionality, which means that concerns other from the main concern, do not fit the decomposition, and end up scattered in the process.

A typical example of a non-core concern, is a business rule such as "billing" [5], which keeps some data during the execution of the process, e.g. the duration of invocations of external services, and charges the user proportionally. Without explicit support for separation of concerns, the billing concern cannot be cleanly separated from the base process, and is scattered in and tangled with the other concerns.

The same problem was originally identified in object-oriented systems, and aspect-oriented software development (AOSD) proves to be a valid solution [6]. The research community acknowledges this problem for workflow languages, and understands AOSD has potential in this context too [7]. There already have

M. Lumpe and W. Vanderperren (Eds.): SC 2007, LNCS 4829, pp. 152–159, 2007.

been proposals to approach the problem with AOSD solutions. However, they literally translate the existing AOSD approaches for object-oriented languages to workflow languages, without much regards for the specific needs of these workflow languages.

In this short paper the requirements for applying aspect-oriented techniques, to separate crosscutting concerns in web service composition languages, are outlined. Section 2 details the problem with current workflow languages. Section 3 gives an overview of the current approaches and their limitations. In section 4 we layout the directions for our future research, and list the features and requirements that we deem necessary in an AOSD approach to the problem. Section 5 concludes the paper with a short summary of the problem and our envisioned solution.

2 Crosscutting Concerns in Web Service Compositions

Web services are a relatively new standard that allow communication between distributed processes, based on existing internet protocols (e.g. http), with the addition of some required "middleware"-enabling protocols, e.g. WS-Security. Web services are of use in two different settings. They can be used to offer certain paid services on the internet, or they can realize the integration between several business processes. A web service is an interface, describing a number of operations that can be called by sending messages over a network. They are the foundation of the service oriented architecture (SOA). In this approach, applications use services that offer a certain function on the network. The clients are unaware of the specific implementation of the service, and only rely on its interface. This kind of loose coupling allows independent evolution of services, while maintaining compatibility with client applications.

2.1 Web Service Compositions

Web service composition is a powerful mechanism to create new web services. Imagine, for example, a travelling agent that offers separate services to book airplane tickets and hotel reservations. By composing these two services, we obtain a new service with which customers can book all-in trips. This kind of reuse removes a lot of complexity in designing advanced web services, as large parts of the service are handled by external services. They offer an added value over regular web services.

We can distinct two large families of web service composition languages. On one side, there are languages that define executable processes by means of web service orchestrations. They do so by fixing a well-defined order of interaction between web services. On the other side, web service choreographies only define the publicly visible behavior of the messages that are exchanged between services. They do not define the internal logic of a process, and as a result, they are not executable.

Choreographies are useful to verify the communication protocol of services, the specific order of messages that are to be exchanged. Consider for example a

web service where a user has to log himself in and out with the service, before and after further interaction. This protocol can be enforced with a choreography specification, without detailing the internal logic of the services. A number of high level composition languages exist that support web services standards. Most notably, there are WSCL [8], YAWL [9], en WS-BPEL [3]. However, none of these languages offers a way to separate crosscutting concerns.

2.2 Crosscutting Concerns

One important principle in programming languages, is that of separation of concerns, which states that properties of software systems must be dealt with separately. Doing so, those parts of the system can be more easily specified, changed, removed or reused.

In object-oriented programming, programs are built by making a class hierarchy of the problem domain. Properties that do not fit this modularization, end up scattered in the application, and entangled with each other. These properties, are the so-called cross-cutting concerns, and are a direct result of the tyranny of this dominant decomposition [10]. As noticed by Arsanjani et al. [7] and others [11,12,13], this problem also applies to web service composition languages.

In our example of the travelling agent, this could be a "Billing" concern, that measures the time it took to talk to the several web services, and charges the user accordingly. Coding this in the web service composition using the traditional mechanisms, leads to scattered and entangled code. An approach to this general problem is proposed under the form of aspect-oriented programming, and while most research is focused on its application to object-oriented programming, a few approaches for web service composition languages have come forward as well. The following section gives more details on these approaches.

3 Current Approaches

Current approaches to achieve better separation of concerns in web service compositions are designed to use aspect-oriented technology. This section starts with an overview of this relatively new programming paradigm, and continues to describe the current approaches and their shortcomings.

3.1 Aspect-Oriented Software Development

To support a better separation of concerns, aspect-oriented software programming (AOP) is proposed. Kiczales et al. note that with the existing methods, certain properties of an application have to be implemented in multiple modules of the system. The same logic is repeated over several modules, which means there is an amount of code duplication. This gives the typical problems that it is hard to add or remove these concerns, to change or even reuse them. AOP wants to achieve a better separation of concerns by moving these crosscutting concerns into separate modules, named aspects. Changing, adding or removing

concerns that are specified in aspects, no longer require changes in the rest of the system.

Several approaches to AOP exist. AspectJ is the first, and also the most well known, approach. This language is an extension to Java and allows programming of concerns in modules (aspects), separate from the classes of the base system. AspectJ introduces a few new concepts to achieve this goal. An aspect describes when it applies, by selecting a number of "join points". These join points are well-defined events during the execution of a program, e.g. the execution of a method. The expression that selects these join points is called a "pointcut". The aspect specifies in the "advice" what should happen on those selected join points. An "aspect weaver" activates the aspects on the places specified by the pointcuts.

3.2 Current Approaches Using AOSD on Web Services

- *AO4BPEL.* In [11], Charfi presents AO4BPEL, an aspect-oriented extension to WS-BPEL. It is a dynamic approach, meaning that aspects can be added to or removed from a composition at runtime, while the process is being executed. This approach requires a modified WS-BPEL engine, and is not compatible with existing software. In AO4BPEL each activity is a possible join point. Pointcuts capturing these join points are written in the low-level XPath language.
- *AdaptiveBPEL.* Another dynamic approach is AdaptiveBPEL [14]. This is a framework that allows managing dynamic aspects. In this approach it is possible to plug in and remove aspects into a core service composition to address quality-of-service concerns and adapt to changes in business rules. AdaptiveBPEL also relies on a modified WS-BPEL engine which applies aspects at runtime.
- *Courbis and Finkelstein.* An aspect-oriented extension resembling AO4BPEL is proposed by Courbis and Finkelstein [15]. In their approach they use XPath as a pointcut language and a modified WS-BPEL engine to allow dynamic addition and removal of aspects. Their advice language is Java.

3.3 Limitations of the Current Approaches

A first observation we made in studying the current approaches, is that they are all quite literal, direct translations of aspect-oriented solutions of object-oriented systems to the web service composition context. A result of this, is that these approaches suffer from a number of shortcomings. Some problems arise with the application of AOSD to the web service composition domain (problems 1 and 2). Others stem from the difficulties in object-oriented AOSD research (problems 3, 4 and 5). The issues there are also appearant in the web service composition domain, and it is worthwhile to investigate the application of the current techniques in this new context.

1. *Confused layers of abstraction.* It is important to stay on the same level of abstraction as the base WS-BPEL process. In more recent versions of AO4BPEL it is possible to alter SOAP messages on the engine level, which allows expressing concerns that can not (easily) be written in a WS-BPEL process. But doing so, this is no longer an approach to separate concerns on workflow level. This extension is also specific to implementation details, by being only applicable on WS-BPEL engines that uses SOAP as the transport protocol. In the approach of Courbis and Finkelstein, the abstraction layers also fade. Their advice code is written in Java, and, unfortunately, is not only used to obtain a clean separation of concerns, but also to execute arbitrary Java code in WS-BPEL processes. Code expressed in the Java language is of a lower abstraction-level than WS-BPEL.
2. *Missing workflow-specific concepts.* The current approaches lack concepts that are specific to web service composition languages. The advice model should support workflow specific concepts, such as synchronization, skipping activities, etc.
3. *Limited pointcut language*
   (a) The pointcut language in AO4BPEL and the approach of Courbis and Finkelstein is XPath, which is a language to navigate an XML structure. The pointcuts in these approaches are no more than XPath expressions that point to a direct path in a WS-BPEL XML document. Being tied so closely to the structure of an XML file, limits the reusability of the pointcuts. They are very fragile, and easily break when the process would evolve.
   (b) None of the approaches support quantification of the join points based on protocol. If we for instance want to express that an aspect can only activate on the third execution of a certain action, we should be able to express this in the pointcut language.
4. *Reusability.* One of the most appealing advantages of aspects is that they are reusable. A concern implemented in an aspect, should be applicable to multiple compositions. However, current approaches require the aspect to be changed to the specifics of the composition it is applied to. Some techniques to solve this are proposed in current research in the AOSD field, but are not considered in the approaches for composition languages.
5. *Aspect interaction.* When multiple aspects are applicable on a single join point, the so called "feature interaction" problem occurs. Aspects are not necessarily orthogonal, and as such, the order of execution matters. It is furthermore desirable to express more advanced aspect compositions than a simple ordering. Current approaches for composition languages either do not consider this problem, or try to tackle it in a very limited way.

4 Towards Better Separation of Concerns in Web Service Compositions

In this section we list the requirements for a better aspect-oriented approach to separation of concerns in web service composition languages. The list is based on

the limitations we identified in existing approaches and introduces some novel ideas.

1. *Workflow-specific advice.* We extend the advice language with concepts that are specific for workflow languages. Examples of these specific concepts are listed here.
   (a) *Advice without order.* In advising web service compositions it occurs that a branch has to be added in a construction where the order is not of importance. For example, an extra activity to be executed in a parallel split, or an extra condition in a merge activity. This kind of advice is named "in" advice [16]. Another kind of workflow specific advice is the "while" advice, that specifies that a certain activity is to be executed during the execution of the selected join point.
   (b) *Skipping advice.* Aspects can specify that certain activities are no longer to be executed. Consider for example an aspect that specifies that a certain part of a web service composition is no longer of use, because some decisions are made by the client beforehand.
   (c) *Synchronize activities.* Aspects can alter workflows in such a way, that new synchronization points are to be introduced in the composition. In future research we plan to look into how this can be easily expressed, and how this can be efficiently implemented.
2. *Declarative pointcut language.* Using a pointcut language based on logic programming offers advantages over traditional pointcut languages. Pointcuts are no more than logic rules that select join points. It is very easy to create new pointcuts by reusing the old ones. Furthermore, it is possible to write recursive pointcuts. Introducing an expressive pointcut language, addresses problem 3a.
3. *Protocol based quantification.* Web service compositions express processes, and precisely in such program specifications it is interesting to express the applicability based on a sequence or a protocol of events (problem 3b). Consider for instance a workflow where a user has three chances to complete a password request before he is denied further access to the service. This can be encoded in a separate aspect, if there is support for remembering prior events in the execution of a workflow. This is also known as "statefull aspects" [17,18].
4. *Deployment constructs.* Aspects are often applicable on more than a single web service composition, and therefore we want them to be as reusable as possible. As shown in problem 4, current approaches have little support for this. In order to achieve this, we plan to use a dedicated layer between the aspects and the compositions to which they apply. In this layer deployment specifications are made. In the deployment constructions we also express the aspect composition specifications that address the feature interaction problem (as shown in problem 5). Lastly, the deployment construct is a good place to specify how the aspects are to be deployed: e.g. one per workflow instance, globally, etc.

5 Conclusion

Current web service composition languages lack support to explicitly separate crosscutting concerns from each other and the base workflow. This problem was already found in object-oriented programming language, and the solution that is proposed for these languages can be applied to workflow languages as well. However, this approach can not be literally translated to this other paradigm, as workflow languages have their own specific needs. Current approaches approaches do not take this into account, and as such suffer from some shortcomings.

In this short paper we identified these limitations of the current approaches. We also listed a number of requirements for future work to tackle these problems. We already started the development of an aspect-oriented extension to WS-BPEL, which we named Padus [16]. This approach strives to a better reusability of aspects and uses several techniques to accomplish this. Padus uses a declarative pointcut language, based on logic programming. In this pointcut language, it is possible to select join points in a describing way, rather than explicitly listing the points. Padus also supports explicit aspect deployments for further reuse. In addition to the regular before-, after- and around advice, the advice model of this language already supports "in" advice to allow including new branches in orderless structures like parallel splits (flow activities). Padus, however does not yet support some of the requirements listed in the previous section, e.g. the "while" advice and support for protocol based quantification in the pointcut language.

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)
4. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Comm. ACM 15(12), 1053–1058 (1972)
5. D'Hondt, M., Jonckers, V.: Hybrid aspects for weaving object-oriented functionality and rule-based knowledge. [19] 132–140
6. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. Technical Report SPL97-008 P9710042, Xerox PARC (1997)
7. Arsanjani, A., Hailpern, B., Martin, J., Tarr, P.: Web services: Promises and compromises. Queue 1(1), 48–58 (2003)
8. Banerji, A., Bartolini, C., Beringer, D., Chopella, V., Govindarajan, K., Karp, A., Kuno, H., Lemon, M., Pogossiants, G., Sharma, S., Williams, S.: Web Services Conversation Language (WSCL) 1.0. W3C Note 14 March, 2002, World Wide Web Consortium(2002), `http://www.w3.org/TR/2002/NOTE-wscl10-20020314/`

9. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language (revised version). QUT Technical Report FIT-TR-2003-04, Queensland University of Technology, Brisbane, Australia (2003)
10. Ossher, H., Tarr, P.: Using subject-oriented programming to overcome common problems in object-oriented software development/evolution. In: Proc. 21st Int'l Conf. Software Engineering, pp. 687–688. IEEE Computer Society Press, Los Alamitos (1999)
11. Charfi, A., Mezini, M.: Aspect-oriented web service composition with AO4BPEL. In: Zhang, L.-J, Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 168–182. Springer, Heidelberg (2004)
12. Cottenier, T., Elrad, T.: Dynamic and decentralized service composition with Contextual Aspect-Sensitive Services. In: WEBIST 2005. Proceedings of the 1st International Conference on Web Information Systems and Technologies, Miami, FL, USA, pp. 56–63 (2005)
13. Verheecke, B., Vanderperren, W., Jonckers, V.: Unraveling crosscutting concerns in web services middleware. IEEE Software 23(1), 42–50 (2006)
14. Erradi, A., Maheshwari, P.: AdaptiveBPEL: A policy-driven middleware for flexible web services composition. In: MWS 2005. Proc. of the EDOC Middleware for Web Services Workshop, Enschede, The Netherlands (2005)
15. Courbis, C., Finkelstein, A.: Towards aspect weaving applications. In: Inverardi, P., Jazayeri, M. (eds.) ICSE 2005. LNCS, vol. 4309, pp. 69–77. Springer, Heidelberg (2006)
16. 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: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, pp. 113–128. Springer, Heidelberg (2006)
17. Douence, R., Fradet, P., Südholt, M.: Composition, reuse and interaction analysis of stateful aspects. In: [19] pp. 141–150.
18. Vanderperren, W., Suvée, D., Cibrán, M.A., De Fraine, B.: Stateful aspects in JAsCo. In: Gschwind, T., Aßmann, U., Nierstrasz, O. (eds.) SC 2005. LNCS, vol. 3628, pp. 167–181. Springer, Heidelberg (2005)
19. Lieberherr, K. (ed.): AOSD 2004. Proc. 3rd Int' Conf. on Aspect-Oriented Software Development. ACM Press, New York (2004)