Using an Aspect Oriented Layer in SOA for Enterprise Application Integration

Chinthaka D. Induruwana

School of Computer Science, University of Manchester,
Kilburn Building, Oxford Road
M13 9PL
induruwc@cs.man.ac.uk

Abstract. Service Oriented Architecture (SOA) is a new method for building information systems. Approaches for integrating different application services with SOA require the client application to be hard-wired to the service. For example, if a particular service talks to another service that uses a different XML standard, then the conversion of the XML between the two services is hard-wired. In large scale integration, it is typical to find different communication standards within the integration layer. This could be the result of incremental adoption or because a company wants to protect earlier investment. Moreover, the current SOA model is not able to encapsulate the scattered concerns that crosscut services which facilitate the integration (for example, concerns including logging, security, routing, fault tolerance and transactions). Therefore, to improve maintainability and achieve better flexibility in the integration of large numbers of heterogeneous application services, we propose an aspect oriented method, in particular using an Application Service Integration Layer (ASIL).

1. Introduction

Aspect Oriented Software Development (AOSD) [1] is a new and expanding field in computer science. The software development process is driven by modelling and implementing the interactions of various software *concerns*,[1] which can be both functional and non-functional. Functional concerns can be regarded as belonging to the core application; for example, concerns that specify the behaviour of a certain application, such as the function of a class, or the method used by the class. In contrast, non-functional concerns[2] are concerns such as logging, security, routing, fault tolerance and transactions. In

[1] Ramnvias Laddad defines a software concern as: "… a specific requirement or consideration that must be addressed in order to satisfy the overall goal" [2].

[2] Non-functional concerns that crosscut are referred to as crosscutting concerns in aspect oriented terminology and *aspects* can be used to encapsulate these concerns (Figure 1b shows an example).

this paper we have called these non-functional concerns *cross-service concerns.*

Aspect Orientation (AO) is a new software engineering paradigm, the purpose of which is to encapsulate non-functional software concerns that crosscut different software components. Aspect Oriented Programming (AOP) is the encapsulating mechanism that enables AOSD engineers to program these crosscutting concerns [2, 3]. Code weaving is the mechanism that turns an AO description into code that implements the concerns.

Service Oriented Architecture (SOA) is a new method for building distributed information systems. However, there are a number of limitations with the current SOA model. Existing service software contains pre-embedded information that is typically concerned with the environment in which it is to be deployed, such as the specification of the XML that it uses to communicate with the other services (this is referred to as *integration logic* in Figure 1). This pre-embedding of information, or hard-wiring, prevents different components from being used in other external contexts. Moreover, hard-wiring leads to code tangling and scattering as shown in Figure 1a. Logging and security are examples of crosscutting concerns, and Figure 1a shows how the code of these concerns can be both tangled[3] and scattered.[4]

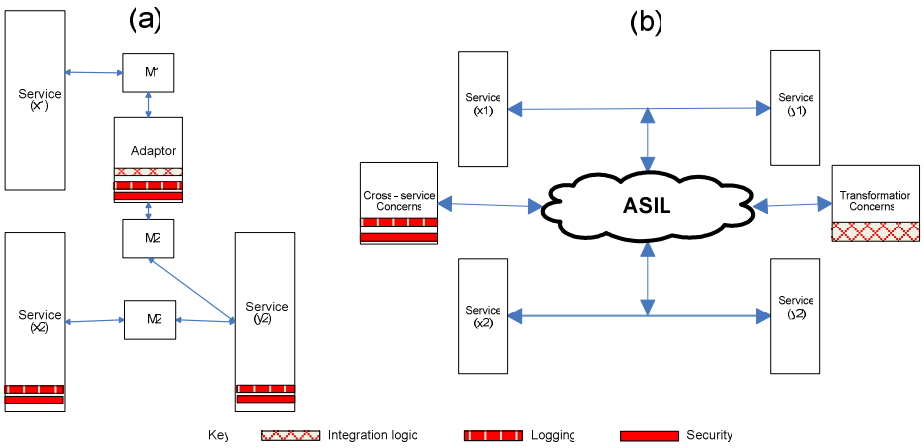

Figure 1: (a) Tangled implementation as a result of using an adaptor. (b) Application services integration layer (ASIL) overview.

In order to overcome the architectural limitations of the SOA model, there is a requirement for a new SOA model in which the services are independent of the protocols and their deployment environment. We expect

[3] Code tangling is when the code for a particular concern becomes intermixed with code for another concern.

[4] Code scattering is when the code for a particular concern becomes scattered across the system.

that an AO approach can be used to encapsulate this non-functional and crosscutting information.

This paper investigates how the principles of AO can be applied to the SOA environment. In particular, it proposes the use of an AO-layer to achieve a more flexible Enterprise Application Integration (EAI)[5] [4] solution with the benefit of encapsulated integration policies.

2. A Novel Integration Layer in Application Services

In order to facilitate the integration of cross-service concerns that support the complete decoupling of the integration logic from the application service logic, we present the *Application Service Integration Layer (ASIL)*. The motivation of ASIL is to support an AO model for EAI with the aim of having a cleaner composition model than is achievable via the current SOA model. The advantage of a layered approach is that it enables the encapsulation of cross-service concerns, as shown in Figure 1b. As a result, ASIL can be used to achieve better flexibility, reliability and scalability with respect to EAI. ASIL is designed to enable the encapsulation of both the integration logic and the cross-service concerns into separate modules. The integration logic concern deals with the different specifications and standards arising from a heterogeneous integration environment. Currently adaptors are required to enable communications between heterogeneous services (Figure 1a). This integration logic would benefit from being encapsulated separately. The advantage of encapsulating the integration logic is that, if a particular set of application services are upgraded, or another one is integrated, and the XML message specification changes, only the transformation rule at one site needs to be altered. This makes it easier to integrate heterogeneous services.

The advantage of encapsulating cross-service concerns is that it enables the non-functional concerns of the system to be handled in a uniform and consistent fashion across the enterprise. Examples of cross-service concerns are as follows:

- **Logging.** The logging concern deals with the encapsulation of the logging behaviour. When certain points of the program execution are reached, the system log is updated to store a record of the program execution.
- **Security.** The security concern deals with different security aspects of the EAI, for example, the security mechanism which is used to communicate between the different services.
- **Fault tolerance.** The fault tolerance concern deals with component failure and exception handling, among other things.

[5] Enterprise application integration can be defined as the integration of processes across third-party applications as well as legacy systems to decrease the number of adaptors which have to be developed when connecting two systems [4].

3. Case Study and High Level Design

The following case study outlines a typical set of requirements for a simple enterprise integration scenario. In Section 4 we indicate how ASIL implements this example.

1. The integration of two application services, where service x1 is using a legacy version of XML specification, and x2 is using a newer XML specification. As a result the XML messages en route between the two services need to be advised[6] by ASIL to enable communication. If this communication policy is encapsulated as an aspect, when a new service is added that also requires communication with x1, then the same advice can be applied to support this communication without the need to write another adaptor.
2. The communications between services x1 and x2 require both logging and security. The enterprise has an encapsulated aspect repository that specifies the logging and security specifications for the entire enterprise as part of the ASIL. The ASIL layer can therefore be used to advise the communications between the two services. First, the message en route has to be encrypted before it is sent by one service to the other, and then logged. ASIL has to intercept the message invocation from the receiving service, decrypt the message and log it before delivering it. The cross-service concerns of logging and security are thus applied before and after interception.

The cross-service concerns described in this example are concerns that require advising both at the service and at the message interception levels. For example, the logging concern requires advising directly at the service level. This is because, if a fault occurs within the service, preventing it from sending back a message, the log would not be updated. An example of advising at the message interception level is security, as messages en route to the services can be intercepted and then encrypted or decrypted.[7]

4. Implementation

ASIL uses an interface definition language (ASIL-IDL) which is based on AspectXML [5]. This is needed for the task of composing the different cross-service concerns in a decentralised manner. Figure 2 shows an example of an ASIL-IDL descriptor. The example shows the specification of the aspect, advice and joinpoint. The aspect is the security cross-service concern. It

[6] An advice is the behaviour of an aspect at a particular join-point (Figure 2 shows an example). A join-point is a well defined place in the structure (execution flow) of a program where the additional behaviour of an aspect can be attached (advised).

[7] ASIL supports advice at the service statically or dynamically via advising messages en route between services.

advises at the message interception level, and the advice is the WS-Security specification.

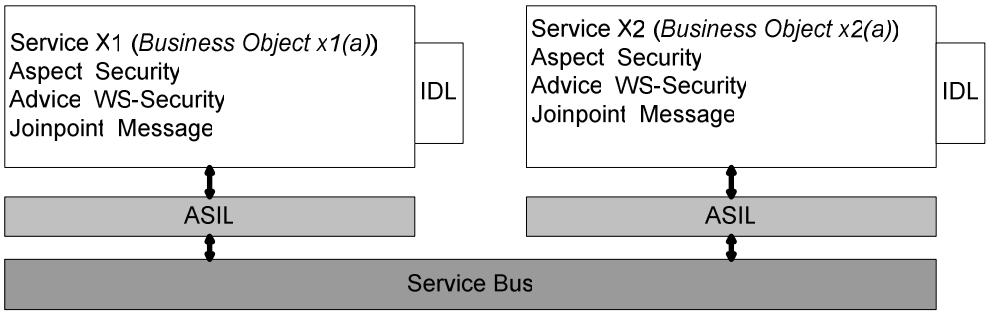

Figure 2: Interface Definition Language (IDL) Descriptor.

A detailed discussion of implementation is beyond the scope of this paper. However, a detailed description of the implementation specifications will be presented at the workshop.

5. Related Work

There are currently a number of academic and industrial research efforts trying to link AO with middleware technologies. The JBoss framework [6] is one of the most popular Application Platform Suite (APS)[8] middleware technologies. Others include J2EE, JAsCo [7], JAC [8] and Caesar [9]. However, these systems allow only implicit association of an aspect implementation.[9] Moreover, these technologies are dependent on the vendor and their aspects are restricted to the ones offered by that particular vendor. Therefore, these systems are not suitable for EAI.

There are also a number of compositional approaches;[10] these include orchestration languages, such as AO4BPEL [10]. AO4BPEL is an AO extension to BPEL4WS; it enables a more modular and dynamically adaptable web service composition. However such languages have a number of limitations. One of the main limitations is that it may not be possible for the orchestration code at a particular service to be modified. This could be due to the fact that the integration involves a legacy version of the orchestration

[8] Application platform suites are middleware technologies that are dependent on the vendor. Therefore, they only function within certain environments.

[9] By this we mean that aspects of services currently depend on a particular framework and, as a result, they have to be modified when they are reused within a different framework. With ASIL this is not the case.

[10] Service composition involves orchestration and choreography. Orchestration languages, such as BPEL4WS, describe the internal behaviour of a service within collaboration. Choreography languages, such as those used for the semantic web, can describe the interactions between services using ontologies. However, they are not mature and are not widely supported by industry.

language or because the service can not be taken offline. Another limitation of orchestration languages is that they only support static composition. A dynamic composition mechanism would be beneficial as certain services and service interactions can only be known at run-time.

The advantage of ASIL is that it is a dynamic layer which can be placed on top of heterogeneous platforms to facilitate integration. As a result it can be used to integrate legacy services, which may or may not use standards such as WSDL or BPEL4WS, whilst utilising the advantages of AO.

6. Conclusion

This paper has introduced the ASIL framework, which is an AO layered approach for EAI. ASIL allows the decoupling of concerns, such as security, from the core of the application within the SOA. The paper also suggests how a dynamic, AO, XML-based, implementation-independent pointcut language is well suited for achieving this decoupling.

7. References

1. Gregor Kiczales, J.I., John Lamping, Jean Marc Loingtie R, Cristina Videira Lopes, Chris Maeda, Anurag Mendhekar, Aspect-Oriented Programming. Proceedings of the European Conference Object-Oriented Programming, 1997: p. 220-243.
2. Laddad, R., AspectJ in Action. 2003: Manning Publications.
3. Robert E. Filman, T.E., Siobhan Clarke, Mehmet Aksit, Aspect-Oriented Software Development. 2004: Addison Wesley.
4. Sayavedra, A.L.a.L., EAI Business Drivers. EAI Journal, 2003. 2: p. 27-29.
5. AspectXML open source community project, http://www.aspectxml.org/.
6. JBoss AOP 2005, http://www.jboss.org/products/aop.
7. María Agustina Cibrán, M.D.H., Davy Suvée, Wim Vanderperren, Viviane Jonckers, JASCO for Linking Business Rules to Object-Oriented Software. System and Software Engineering Lab Vrije Universiteit Brussel, 2003.
8. A Renaud Pawlak, L.S., A Laurence Duchien, Gerard Florin, A Fabrice Legond-Aubry, A Laurent Martelli, JAC: An Aspect-based Distributed Dynamic Framework. Softw. Pract. Exper., 2004. 34(12): p. 1119--1148.
9. Mira Mezini, K.O., Conquering Aspects with Caesar. Proceedings of the International Conference on Aspect-Oriented Software Development, 2003. Boston, USA.: p. 90 - 99.
10. Anis Charfi, M.M., Aspect-Oriented Web Service Composition with AO4BPEL. The European Conference on Web Services, 2004. Erfurt, Germany.