Moving from Internal to External Services using Aspects

Martin Henkel[1], Gustav Boström and Jaana Wäyrynen

Department of Computer and Systems Science
Stockholm University and Royal Institute of Technology
Forum 100, SE-164 40 Kista, Sweden
[1] Contact author, Tel: +46 16 16 42, Fax: +46 8 703 90 25

martinh@dsv.su.se, gusbo@kth.se, jaana@dsv.su.se

Abstract. Service oriented computing and web service technology provide the means to structure an organisation's internal IT resources into a highly integrated network of services. In e-business and business process integration the internal services are interconnected with other, external organisations' resources to form virtual organisations. This move from using services internally to external use puts new non-functional requirements on the service implementation. Without any supporting technologies, meeting these new requirements can result in re-writing or changing a large part of the service implementation. In this paper we argue that aspect oriented programming is an important technique that can be used to facilitate the implementation of the new requirements that arises when moving from internal to external services. The suggested solution is illustrated by an example where quality of service metrics is implemented by using aspect oriented programming.

Keywords: Service oriented computing, Non-functional requirements, Interoperability, Aspect oriented programming

Topics: Separation of concerns, Connecting legacy applications to services, Monitoring service executions

1 Introduction

Service oriented computing, in particular the web service technologies, has drawn a lot of attention in recent years. The reason for this attention is multi-faceted. One reason is based on the view that service oriented computing is a natural step of evolution from object-oriented and component based computing. Another aspect that makes services interesting is that they can be used to structure and interconnect an organisation's internal IT systems. Even more importantly, services can be used externally to enable interconnection of enterprises [1], thus enabling the forming of networked or "virtual" enterprises [2].

The interconnecting of enterprises via service technology requires that part of an enterprise's internal systems must be made available to external organisations. This shift from internal to external use puts new requirements on existing IT systems. First of all, the systems must be able to communicate, regardless of differences in platforms and languages. This interoperability can be achieved by conforming to technical standards. The existing and upcoming web service standards such as SOAP [3] and WSDL [4] are important steps towards interoperable services. A second, somewhat overlooked part is that providing external services also puts new non-functional requirements on existing systems, such as scalability, security and quality. These requirements are sometimes also called "illities" [5].

Within the border of a company these requirements might be implicitly met by assumptions about the company's secure intranet, the well-defined number of users, the support departments ability to monitor the quality etc. However, exposing parts of the system to external partners will require that these implicit assumptions need to be converted into explicit, measurable, and monitorable implementations. Furthermore, the new non-functional requirements posed need to be implemented as part of the existing systems.

Implementations of new non-functional requirements often cuts across the entire system, i.e. a large part of the existing system code is affected by new non-functional requirements. Thus, when moving from internal to external service use, a potentially large part of the system code needs to be changed. The need to change large portions of code to implement new non-functional requirements is a problem, since it increases the risk of major redesign when moving from internal to external use of services. The question is how to overcome this problem.

Aspect Oriented Programming (AOP) is proposed as a technique to implement functionality that cuts across an entire system [6]. Thus, aspect orientation may be a solution that can facilitate the implementation of the non-functional requirements that arises when internal systems are to be exposed as external services.

In this paper we argue that aspect oriented programming can be a useful technique for moving from internal to external services. The paper begins with a short introduction to service oriented computing and aspect oriented programming. The introduction is followed by a description of the problem and the proposed solution. Then, we provide an example of how aspect-orientation can be applied in the context of a service that needs to be changed due to changed non-functional requirements. The article ends with a discussion of the proposed solution's applicability and a discussion of further work.

1.1 Service Oriented Computing

A *service* is an "act or performance offered by one party to another" [7]. Services offered by IT systems have been dubbed "e-Services" [8] and "software services". In this paper the term service denotes services offered by one system to another system, i.e. where both parties are software systems. This excludes services that are offered to (human) users via graphical user interfaces.

Service Oriented Computing (SOC) builds on the component based development (CBD) principles of building systems by combining software parts. In contrast to components, a service is a run-time programming interface rather than a physical/binary entity that needs to be installed before use. This distinction between runtime provisioning and implementation is made by component based development methods [9]. The distinction might seem finicky, but it has a profound impact on how services are used. A provider of a service is responsible for the run-time availability of the service, whereas a component provider is only responsible for the construction and delivering of the binary component. Thus, building a component based system is about assembling software parts, while building a service-oriented system is about communicating with services offered by different providers.

The focus on run-time availability and provider responsibility makes services an ideal metaphor for interconnecting an organisation's IT-systems (internal use), where each system is a separate run-time entity. For the same reason, service oriented computing can play a major role in the interconnection of systems belonging to separate organisations (external use).

1.2 Aspect Oriented Programming

Aspect oriented programming is a paradigm that attempts to help in implementing concerns that are present in many modules and therefore crosscuts a system, that is *cross-cutting concerns*. Cross-cutting concerns are difficult to modularise using existing object-oriented techniques since there is no logical place in which to implement them. An illustrative example is logging of method-calls. Using existing object-oriented techniques the code for implementing this would be spread out in all methods that require logging. Changing the way logging is done, and especially, where it is performed is therefore difficult to accomplish without changing all methods that need to be logged. This poor modularisation leads to code that is difficult to maintain. The fact that you need to deal with several concerns, logging and business logic, in the same method also adds to the complexity of the code.

Aspect oriented programming provides a way to modularise these cross-cutting concerns in an efficient manner by factoring out logic belonging to a specific concern into an *Aspect* [10]. In this article we use AspectJ as a tool to implement aspect oriented programming [11]. An aspect in AspectJ consists of *Pointcuts*, and *Advice* (in AspectJ an aspect can also contain *Inter Type declarations*, but this concept is not used in this article). *Pointcuts* describe *where* the aspect should apply in terms of the object-oriented systems structure, e.g. the pointcuts of the logging aspect would describe *where* in the system logging should be performed in terms of the classes and methods of the system. *Advice* describes *what* should happen at these pointcuts, e.g. how the logging should be carried out. The AspectJ keywords *aspect*, *pointcut* and *advice* are added to the Java-syntax in order to support these concepts. The process of combining the aspects and the classes into an executable system is called *aspect weaving*.

1.3 Functional, non-functional and cross-cutting requirements

Functional requirements are statements of services that the system should provide. This can also be said as describing *what* the system should do. Non-functional requirements, on the other hand, are focused on *how* the system should perform the services [12]. An example of a functional requirement on an ATM-machine could for example be that an ATM-machine should be able to dispense money to the bank's customers. A non-functional requirement could be that this service has to be performed *securely* and with an acceptable *response time*. Other examples of non-functional requirements are performance, traceability, scalability and error handling. A problem with non-functional requirements is that they are often *crosscutting,* i.e. they affect many modules of the system. For example, security needs to be addressed in many parts of an ATM-system. It is therefore difficult to modularise crosscutting requirements. This can make systems difficult to maintain and evolve [13].

2 Moving from Internal to External Services

As stated in the introduction, service oriented computing promises to interconnect organisations. This is done by integrating and automating business processes that span across several organisations. Service oriented architectures (SOA) and service technologies, such as web services form the fundament for such integration. Integrating business processes and automating them relies on the integration of the organisation's IT systems. This in turn, requires that integration points between the systems need to be established. Systems, which previously only where used within a company, need to exchange information with external systems through these integration points. Interconnecting the processes of two organisations commonly do not require that all the IT systems need to be integrated. Rather than making an entire system available externally, a selection of functionality is made. This functionality is then exposed as services that can be used by external organisations [14]. As mentioned earlier, the exposed services commonly need to adhere to a new set of non-functional requirements. Examples of new non-functional requirements are security, quality of service measurements, and performance monitoring. Implementing support for these new requirements is instrumental in making the services available externally.

There exist several solutions to this problem. The first solution that comes to mind is to rework the entire code to support the new requirements. This can be achieved by following common refactoring principles, such as those proposed by Fowler [15]. However, this solution requires a lot of work, since each method that should adhere to the new requirements has to be reviewed. More generic approaches have also been proposed, such as the addition of an extra layer to existing component-environments by using generated proxies controlled by proprietary description languages [16]. These generic approaches have a much better chance of reducing the amount of work required. They are, however, based on proprietary languages and technologies. The ideal would be a technique which is generic (to avoid too much rework), and at the

same time does not rely on proprietary technologies, servers, and languages. We propose that aspect oriented programming can be such a technique.

Aspect oriented programming separates the code required to fulfil the new requirements from the existing code. The new requirements can thus be separately implemented as aspects, without changing the existing code. These aspects can then selectively be applied (by aspect weaving) to the parts of the code that need to adhere to the new requirements. Applying the aspects does not require changing the original code. For a large system this can save a lot of time.

In the next section we will give an example of how aspect oriented programming can be used to implement non-functional requirements without a major rework of the original system.

3 An example: Adding QoS Metrics to Web services

The example in this section describes the steps necessary to extend a web service with quality of service metrics monitoring (QoS monitoring). The example elucidates the main point of this paper, that by using aspects, the move from internal services to external services do not require a major rework of the code. The need to extend web services with QoS metrics is selected as an example both because it is a likely scenario, and because it clearly demonstrates how non-functional requirements can be implemented using aspects. What makes the scenario likely is that enterprises starting to use the web service technology internally will need to further define, and monitor their quality of service when starting to use web service technologies as an external communication mean between enterprises, thus the need to add QoS metrics to web services.

3.1 Scenario

To illustrate how AOP will help in implementing non-functional requirements such as QoS metrics let's imagine a company that provides financial services such as mortgages and loans for cars. In this business it is essential to know your customers' credit worthiness. Credit worthiness is determined using the customers' credit history, income and other variables. Different financial services require different definitions and levels of credit worthiness. This information is used in determining whether to grant applications for both loans and mortgages. Since credit checking is an important part of this organisation's business, it is implemented as a web service that can be reused from all systems within the organisation. Figure 1, below, shows how the interface to this credit checking service might look like implemented in Java.

```
public interface CreditCheckingServiceInterface
{
  public boolean hasPaymentRemarks(String name);

  public boolean hasCreditHistory(String name);

  public boolean checkCreditForAmount(String name,int amount);
}
```

Fig. 1. The interface of the CreditCheckingService

The next step in the organisation's business plans could be to provide the credit checking service to external businesses, such as mobile phone operators and car leasing companies that also need efficient credit check processing. However, before using the credit checking from their systems, external businesses will require some form of quality guarantee. For example, a potential customer of the service would probably ask the following questions:

- How can it be ensured that the service paid for is reliable and running when needed?
- How can the organisation monitor that the performance is acceptable?

In short, the customers will require some form of agreement that states the intended quality of service. The agreement can include measurable limits for performance, cost, up time and other dimensions that affects the overall quality of the provided web service. For this example we use three QoS dimensions for web service processes as defined by Sheth et al. [17]: time, cost and reliability.

- Time is a measure of response time of the web service that is to be monitored. The response time is measured from request arrival to the completion of the request.
- Cost can be measured by either estimating an average cost for each service invocation, or by measuring the resources that are consumed to complete a request (such as processor time, cost of information storage etc).
- Reliability is a measure of technical failure rate, that is monitoring the reliability will discover how many times the service failed to deliver a response. Sheth et al. [17] suggest that reliability is to be measured as a ratio of successful executions/scheduled executions.

The credit checking web service mentioned above is not built with QoS metrics in mind, since it was designed for internal use only. Adding QoS metrics to the existing service can be a major undertaking, since code that monitors the metrics need to be inserted in all parts of the service. Without a technique that helps implement cross-

cutting, non-functional requirements such as QoS metrics, developers are running the risk of having to redesign a major part of the code. However, applying aspect oriented programming can reduce this risk. An example of how aspects can be applied in this case is described in the next section.

3.2 Applying Aspects

Let's look at how aspects could be applied in the described scenario. The three QoS dimensions time, cost and reliability define what is to be measured. Before implementing the actual metrics, it has to be decided where in the application code the dimensions should be measured. A basic approach would be to add code to register each metric in the beginning and end of each request, i.e. before and after each call to the web service. Without using aspects, this approach would require additional code that has to be inserted in *all* web service methods. However, using an aspect-oriented approach, adding QoS metrics to web services would only require the metrics *aspects* and their *join points* to be defined once, without any change to the original web service implementation.

To implement QoS metrics for the credit checking service, one aspect for each of the QoS dimension can be implemented. Thus, as an example we have implemented the aspects PerformanceQoSAspect, CostQoSAspect and ReliabilityQoSAspect .

```
public aspect PerformanceQoSAspect
{

   Timer timer=new Timer();

   pointcut timedMethods() : (
       execution(public * CreditCheckingService.* (..)));

   before() : timedMethods()
   {
       // Start timing
   }

   after() : timedMethods()
   {
       // End timing
   }
}
```

Fig. 2. Performance QoS aspect

3.3 Performance Aspect

The performance aspect is intended to measure the Time QoS dimension. Time can be measured by recording the request/method name, when the request arrived and when the response was sent. The implementation of this metric requires that two aspect join points are defined; one at the beginning of each method call and one at the end. These join points are defined within the AspectJ pointcut “timedMethods”, see figure 2, above.

3.4 Cost Aspect

Cost can be measured by recording the request/method name for each request. Using predefined cost for each type of request, the total cost can be calculated. The measurement of cost can be done by using a join point at the end of each web service method. The AspectJ example in figure 3 show how an aspect that logs each method call can be implemented.

```
public aspect CostQoSAspect
{

   pointcut costMethods() : (
       execution(public * CreditCheckingService.* (..)));

   after() : costMethods()
   {
       // Log the cost of the executed method
   }
}
```

Fig. 3. Cost QoS aspect

3.5 Reliability Aspect

Reliability can be measured by recording if the response of a request is a valid response or an error. In this case, a join point can be defined at the end of each method. The AspectJ implementation shown in figure 4 defines an aspect that logs every method call that ends with a non-application Exception.

```
public aspect ReliabilityQoSAspect
{
   pointcut reliabilityMethods() : (
       execution(public * CreditCheckingService.* (..)));


   after() throwing(Exception e): reliabilityMethods()
   {
       if(!(e instanceof ApplicationException))
       {
             // Log error
       }
   }
}
```

Fig. 4. Reliability QoS aspect

The example given above can be extended with more QoS metrics. This example illustrates the main points in using aspects for the implementation of non-functional requirements.

4 Conclusion

In this article we proposed that AOP could help the transition from internal to external services. By using AOP, non-functional requirements can be implemented without doing a major redesign of the existing system. The feasibility of the proposed solution has been demonstrated with a simple example written in AspectJ.

The example demonstrated that AOP could be a useful tool when an application needs to accommodate QoS metrics that have not been previously designed into the system. It also shows that this can be easily achieved using just a few lines of code. In fact, the bigger the application, the more amount of time will be saved by using aspects.

AspectJ was used in the example. However, there are other ways to implement non-functional requirements in an “aspect oriented” way. It could be argued that by using a component technology such as Enterprise Java Beans (EJB), QoS metrics could be provided by the application server (e.g. through the use of method interceptors in the JBoss EJB server [18]). The implementation of these interceptors however, are not currently standardised, they would therefore be different for each component server (Interestingly the latest version of the JBoss features heavy use of AOP to implement non-functional requirements). It would also require the application to be built as a component-based application from the start, which is often a lot more time-consuming and skill-intensive than using plain Java objects. Using design patterns such as the “proxy” pattern [19] could also alleviate the need for using

specific AOP technologies such as AspectJ. An example of this is provided by Filman et al. [5]. This approach, however, is considerably more time-consuming and therefore also more error-prone.

The proposed solution is applicable when the move from internal to external services poses new non-functional requirements. Clearly, if no additional non-functional requirements need to be fulfilled, the need to introduce aspect-oriented concepts is not as obvious. Furthermore, the proposed solution presumes that the non-functional requirements can be implemented in a generic, separated way using aspects. In the case that not all new requirements can be implemented in this way, aspects can still contribute to the implementation of some of the requirements. Thus, we believe that the use of aspect-oriented programming can be a valuable technique when moving from internal to external services.

5 Further work

In this paper we examined how aspect oriented programming can be used to tackle the non-functional requirements when moving from internal to external services. However, when integrating processes it is likely that other changes need to be implemented in parallel with the new non-functional requirements. For example, when integrating processes there might be a need for further process automatisation, i.e. new functional requirements. A possible future extension of our work might include principles guiding the combination of aspect-oriented programming with traditional refactoring techniques to implement both non-functional and functional requirements.

Another interesting question is whether AOP could prove useful for solving other “architecture breaking” problems. There are several indications that this could be the case. De Win et al. [20] have shown how AspectJ can be used to help implement security features in an application. Filman et al. [5] have described how AOP can be used for inserting “ilities”, such as stability and reliability.

References

1. Fremantle, P., Weerawarana, S., Khalaf, R., Enterprise Services. Communications of the ACM, October 2002, Vol. 45, No 10 (2002)
2. Yang , J., van den Heuvel, W. J., Papazoglou, M. P., Service Deployment for Virtual Enterprises. Australian Computer Science Communications, Vol. 23, Iss. 6 (2001)
3. Gudin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H. F., SOAP Version 1.2. W3C Candidate Recommendation. (2002)
4. Chinnici, R., Gudin, M., Moreau, J., Weerawarana, S., Web Services Description Language (WSDL) Version 1.2. W3C Working Draft (2003)
5. Filman R., et al., Inserting Ilities by Controlling Communications. Communications of the ACM, Vol. 45 (2002)

6. Duclos F., Estublier J., Morat P., Describing and Using Non Functional Aspects in Component Based Applications. 1st International Conference on Aspect-Oriented Software Development (2002)
7. Lovelock, C., Vandermerwe, S., Lewis, B., Services Marketing. Prentice Hall Europe (1996)
8. Piccinelli, G., and Stammers, E., From E-Process to E-Networks: an E-Service-oriented approach. OOPSLA Workshop on Object-Oriented Web Services (2001)
9. Allen, P., Frost, S., Component-Based Development for Enterprise Systems: Applying the Select Perspective. Cambridge University Press (1998)
10. Elrad, T., Filman, R., Bader, A., Aspect-oriented Programming an Introduction, Communications of the ACM Vol. 44 (2001)
11. AspectJ, www.aspectj.org. Accessed in April 2003
12. Cysneiros, L., do Prado Leite, J., Non-Functional Requirements: From Elicitation to Modelling Languages. International Conference on Software Engineering (2002)
13. Moriera, A., Araújo, J., Brito, I., Crosscutting Quality Attributes for Requirements Engineering**.** 1st International Conference on Aspect-Oriented Software Development (2002)
14. Georgakopolos, D., Schuster, H., Cichocki, A., Baker, A., Managing Process and Service Fusion in Virtual Enterprises. Information System Vol. 24, No6 (1999) 429-456
15. Fowler, M., Refactoring – Improving the Design of Existing code. Addison-Wesley (1999)
16. Becker, C., Geihs, K., Quality of service and object-oriented middleware - multiple concerns and their separation. International Conference on Distributed Computing Systems (2001) 117 -122
17. Sheth A., Cardoso J., Miller J., Kochut K., Kang M., QoS for Service-Oriented Middleware.. Proceedings of the Conference on Systemics, Cybernetics and Informatics, (2002)
18. JBoss, www.jboss.org, Accessed in November 2004
19. Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
20. De Win, B., Vanhaute, B., De Decker, B., How Aspect-Oriented Programming Can Help to Build Secure Software. Informatica Journal, Vol. 26, (2002) 141-149