A Server Side SOA Meta Model for Assigning Aspect Services

Andreas Ganser, Stefan Hurtz, Horst Lichter

RWTH Aachen University, Research Group Software Construction, Ahornstr. 55, 52074 Aachen, Germany {ganser|lichter}@swc.rwth-aachen.de, stefan.hurtz@rwth-aachen.de

Abstract. Service oriented architectures allow heterogeneity in general and hence support the integration of legacy systems. But legacy systems must often be handled as black boxes and they usually lack additional requirements for new environments. The most important is security. Wrapping services and integrating them via enterprise service buses is one way to tackle this problem but it brings about proprietary and heavy weight software suites. Therefore, a lightweight design concept is missing which fosters encapsulating services at a platform independent level. Below, we present how aspect services can be assigned to services and underline how our concept becomes handy during design time. Additionally, we discuss tool support for assigning aspect services to services.

1 Introduction

One basic idea of service oriented architectures (SOA) is to join business division and IT division. Thereby, most of what IT does is challenged in terms of company's business goals. But, concerning technical aspects only and omitting business, organizational, and cultural points, SOA can be condensed into interfaces and is better characterized as "interface oriented architecture" [1]. Next, just caring about what is beneficial leaves out other "additional" aspects, e.g. logging or security.

These "additional aspects" are called cross-cutting concerns in aspect oriented modeling. Hence, we borrow a couple of terms from aspect oriented programming (AOP) and slightly alter some concepts.

In this paper we introduce a generic meta model for assigning cross-cutting concerns to services at server side. First, a short introduction illustrates the roots of our work taken from aspect oriented programming. Second, a guiding example motivates the necessity of our work and provides some design rationales. Third, the meta model describes a static view concerning the issues, and a typical message flow will depict further issues. Last, tools developed as feasibility prototypes show the meta model's applicability.

In a nutshell, our contribution to current research is a generic meta model for cross-cutting concerns in SOA environments. It provides the roots for handling dependencies and constraints between cross-cutting concerns and services during design time. Further, it enables designers to model effects on messages and interfaces during processing.

2 From AOP to Aspect Services

Cross-cutting concerns are requirements that are usually not of primary but of secondary interest. In other words, there is a second dimension in solving problems while building a program. But programming languages and the chosen design criteria are one dimensional! For example a program accesses a database. Let this be a primary requirement and, now, add logging as a secondary requirement. This means, every access to the database is supposed to be written to a file. The logging cross cuts whatever is done by this program because the database accesses are usually scattered all over the source code.

As a consequence, implementing cross-cutting concerns with programming languages has been cumbersome. Therefore, AOP introduced a concept to collect these concerns, here called aspects, statements at a single place [2]. By that, a separation of concerns is achieved and changing an aspect only results in changes to a single file. But the statements need to be propagated back to the places they belong. This distribution is done by a code weaver which inserts the aspect's source code at certain join points before the source code is compiled. (We do not consider run time mechanism here.)

For example, every access to a database is supposed to be logged. Therefore, calls of particular methods have to be monitored. Hence, all these methods are join points and the code weaver inserts a logging mechanism before these methods are called. In other words, the cross-cutting concern is realized as an aspect logging mechanism and is woven at compile time.

The objective, we are going to address, is cross-cutting concerns in a SOA environment and we refer to these concerns as aspects likewise AOP does. But aspects are slightly different in SOA since services have to be regarded as stateless black boxes in our environment. Due to this, join points can only exist outside of a service. This eases difficulties with respect to locating join points and catching messages but it hampers handling of more concerns in matters to order at one join point. Moreover, there will be no code weaver assuring the proper aspect distribution. But there will be assignments to be evaluated during run time. Lastly, it is reasonable to implement aspects as services themselves as the environment comprises of services. We call these services "aspect services".

3 Guiding Example: A Document Retrieval Service

We studied a document retrieval service as a motivating and guiding example in order to illustrate how services and aspects might be related. With respect to services this document retrieval service is subdivided into operations. You can assign several cross-cutting concers to these operations. For example you might want to add an access control service or a billing service. Traditionally, these examples are manageable with AOP but they are conceptual challenging in a service oriented environment.

In more detail, the document retrieval service consists of two major operations: one operation is searching the documents (the seeker) and one operation is providing the documents (the stash). Therefore, a user first calls the seeker and receives a list of meaningless links. These links, which the stash is able to understand, point at some place in the stash. Consequently, the user employs these links to obtain the documents. So far, this system is easy, and, unfortunately, a black box.

Because the document retrieval service does not comply with any security requirements, we need to add them now. First, Bob might be allowed to retrieve a document or not. This surely depends on his authorization and hence an authentication beforehand. As a consequence, if Bob is authenticated the stash will display only the documents he is authorized to see. Second, Bob might be allowed to read the document but has to pay for it. In order to realize this, Bob must be authenticated and he must have paid beforehand. All these security aspects base upon encryption and hence we simplify matters to encryption.

4 Meta Model

The aforementioned document retrieval service uncovers many details that have to be taken into account while designing a suitable meta model. Therefore, we exploited this example to build the meta model and tried to find the generic and reusable parts. We did so, because we finally aim for a more generic model than for security issues. This is because there are other examples for aspect services like logging or billing that the meta model is supposed to cover.

We present the meta model in figure 1. There, we omit all attributes and condense a few concepts to simplify matters and to focus on the core concepts. These core concepts are ordered from left to right and hence the meta model is supposed to be read as such. As a consequence, one can follow a common sequence of processing by reading the core concepts from left to right. Additionally, we named the core concepts like the packages for emphasis. They are *aspect*, *aspect assignment*, *aspect service* and *service*.

In section 4.2 we will employ the meta model to model the relevant concepts during message processing and in section 5 we will show how the realization of single concepts looks like. In the latter, some concepts will emerge as files and some will as components.

4.1 Core Concepts of the Meta Model

Following our example from section 3 we have two operations to begin with, our seeker and our stash. Both *operation*s are to be regarded as black boxes and *apply to* an *interface*.

Now, security is under consideration: We decided to put security into an *aspect category* because single *aspect*s might be in this category. For example authorization, authentication or encryption are *aspect*s of the security aspect category. Picking out e.g. encryption, one possible realization encrypts complying with RSA, AES or IDEA. Therefore an *aspect operation* is required that belongs

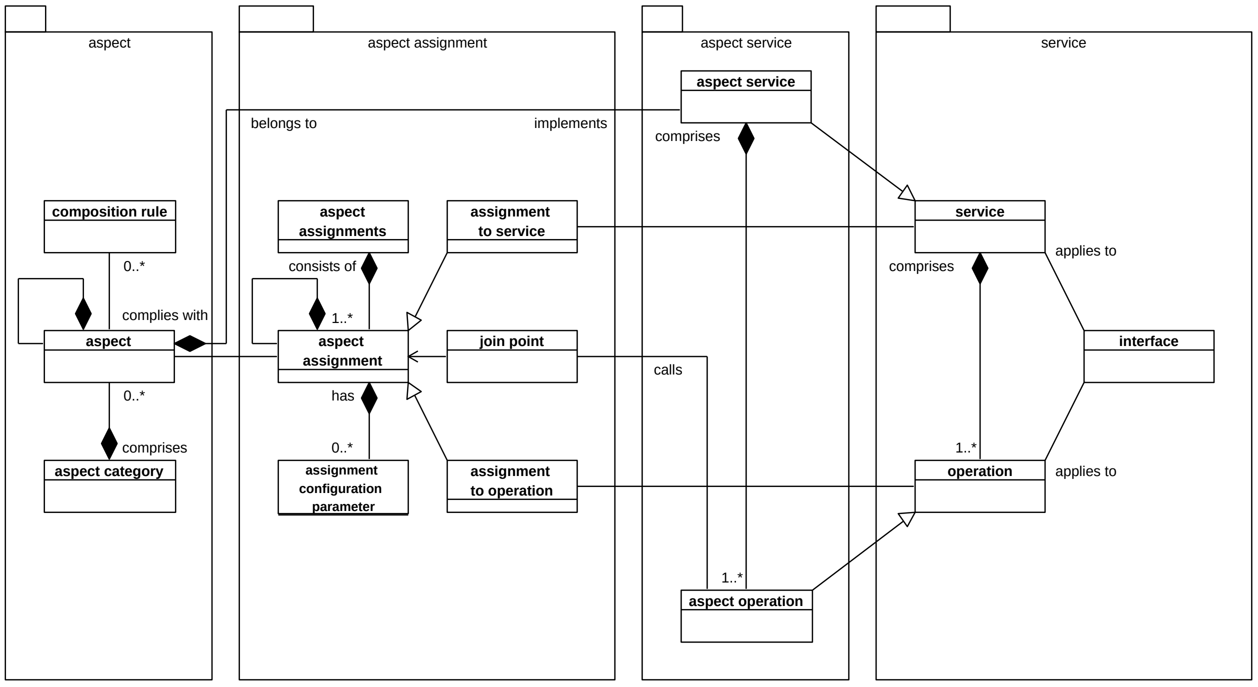


Fig. 1. Meta Model – Aspect Assignments

to our encryption *aspect service* which itself *belongs to* an *aspect*. Let's call our encryption operation IDEA-L.

Next, we need to relate our seeker and our stash to IDEA-L. This is achieved via *aspect assignments* which *consist of* single *aspect assignment*s. But our seeker and our stash are *operation*s, and, hence we need to apply a specialization of *aspect assignment*s: an *assignment to operation*.

Now, we need to apply our IDEA-L *aspect operation* at some time during run time. Fortunately, a *join point* belongs to an *aspect assignment* which is a concept similar to AOP because at this *join point* IDEA-L is called. Section 4.2 explains how this is realized in more detail.

Sometimes it is cumbersome to assign many *aspect operation*s to *operation*s. Luckily, *operation*s are usually subsumed under *service*s; like our seeker and our stash which are named document retrieval service. Thus we can assign IDEA-L to our document retrieval service by using an *assignment to service*.

Unfortunately, *aspect operation*s usually do not work immediately because some need *assignment configuration parameter*s. For example, IDEA-L might need a RSA public key for negotiating the IDEA-L key. This key as a parameter surely belongs to an *assignment to operation*. But it might be necessary to pass parameters to every *aspect operation* an *aspect service consists of*. For example you might want to set a default hash algorithm for all *aspect operation*s. This *assignment configuration parameter* must be part of the *assignment to service*.

Additionally, the execution of an *aspect* might have some prerequisites. For instance an authorization service makes no sense unless an authentication occurred before. Such prerequisites, which formulate dependencies between *aspect*s, belong to *composition rules*.

Further, the recursions in the meta model need more detailed explanation. First, an *aspect* might be related to other *aspect*s and second, *aspect assignment*s might be related to *aspect assignment*s. For example one might want to combine *aspect*s to a new *aspect*. In case we have digital signatures and logging as *aspect service*s then a combined version is a "verbose digital signature service". This new *aspect service* might be used for monitoring users. Consequently, we have recursion with the *aspect assignments* too. Another example why recursion is handy is presented in section 4.2. In our example we might add a logging service to IDEA-L. So, every step of IDEA-L is written to a log file.

To sum up, we made an *assignment to operation* by relating an *aspect operation* to an *operation* giving an *assignment configuration parameter*. In our example we assigned IDEA-L to our stash adding the public key for negotiating. This assignment takes place at the *join point*. Finally, we added a logging service to IDEA-L for e.g. debugging purposes.

4.2 Processing Messages

The meta model presented in section 4.1 provides a static view on aspect assignments to services. Based on this meta model we want to explain the processing of messages in such an environment in more detail. Figure 2 depicts the most important concepts and roles that are involved in message processing. It is important to note that this figure only shows the involved concepts and roles. Furthermore, the *join point* comes up in two roles with respect to services. First, *invoke message*s are dealt with when the *join point* is a *mediator* during a *pre call*, and, second, *reply message*s are dealt with in case the *join point* acts as a *mediator* during a *post call*. In order to explain some more details let's follow a message flow and keep in mind the concepts in figure 2.

An encrypted *invoke message* is sent to the stash and this message is *received* by the *join point* since IDEA-L is assigned to the stash. The *join point* delegates the message for further processing to an *aspect operation*; let's say IDEA-L decryption during the *pre call*. This means, the *invoke message* is decrypted and can now be processed by a *service operation*, namely the actual document retrieval. But this can only happen if the *invoke message* is *conform* to the stash's *interface*. Otherwise the *invoke message* is malformed and an error message arises. But in normal progression, the *service operation* produces a *reply message* which contains the requested document delivered by the stash. Again, a *join point* receives a message. This time it is a *reply message* which the *join point* deals with as a *mediator* during *post call*. Hence, it delegates the *reply message* to another *aspect operation*. Let's say we sign the delivered document. This means the *outgoing message* is produced which needs to be *conform* to the *interface* as well.

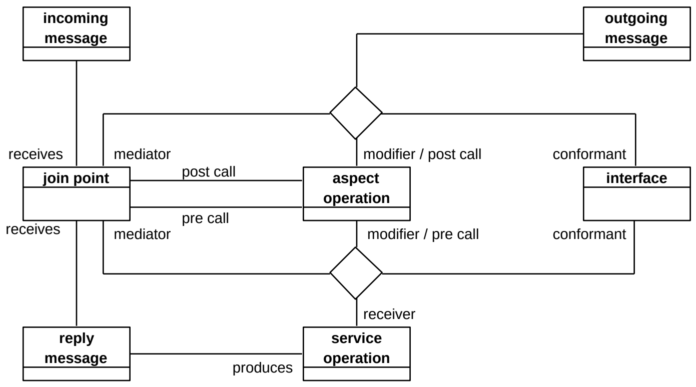


Fig. 2. Processing Messages

The above mentioned message flow considers two server side locations of join points with respect to service execution. And, since we are dealing with services as black boxes, we only inspected these two cases. One is located before and the other is located after the service execution. At first sight, both are as expected. At second sight, working purely on the incoming and outgoing messages is what one might refer to as filters, and such filters build on API message exchange patterns [3]. In any case, filters examine incoming and outgoing messages (or calls) and decide on roles (e.g. RBAC) how to continue.

Filtering before the execution of a service bears nothing unexpected. But, one needs to be careful with filtering data because an upcoming service has a certain interface. If the filter changes or omits data, the upcoming service might be harmed. This holds true for incoming and outgoing messages and must be checked during design phase already.

On the contrary, filtering outgoing data is clearly uncommon. Nevertheless, we had to keep this in mind as sometimes there is no way to execute a service but with super user privileges. This means a service returns whatever you request and this is not always good. Let's say our document retrieval service is implemented by means of SQL statements and needs to run with super user privileges due to database issues. Now, on the one hand, you cannot filter incoming SQL statements because they rapidly grow way too complex. But, on the other hand, trying to filter outgoing data is simply too late because all execution is done. For example a drop table returns only true and that's what you filter. But the harm to the database is already done. Hence, one needs to be very careful with this kind of filter.

Now, one can imagine orchestration of services with incoming and outgoing filtering. We call this I/O filter and, of course, this is the difficult part of filtering.

Additionally, this caused us to extend the meta model with a mechanism for parameterizing the assignments between services and aspects. Moreover, this is why we need self-references with *aspects* and *aspect assignments* in the meta model introduced in section 4.1.

5 Tool Support

The realization of the meta model is subdivided into a coarse grain and a fine grain editor as can be seen in figure 3. The coarse grain editor is a GUI editor for editing *aspect assignment*s by assigning *aspect service*s to *service*s. Further, it gives an overview which *aspect assignments* exist. The fine grain editor is a GUI editor for setting the *assignment configuration parameter*s. Both tools work hand in hand in order to achieve a deployable configuration.

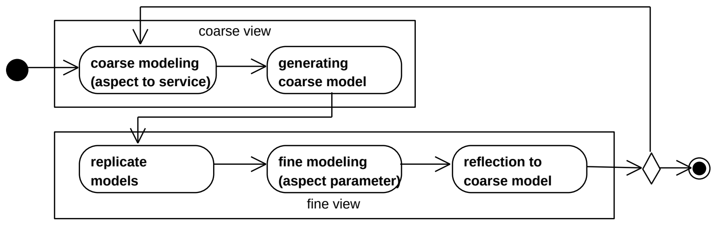

coarse view
coarse modeling (aspect to service)
generating coarse model
replicate models
fine modeling (aspect parameter)
reflection to coarse model
fine view

Fig. 3. Concepts Involved Configuring Assignments

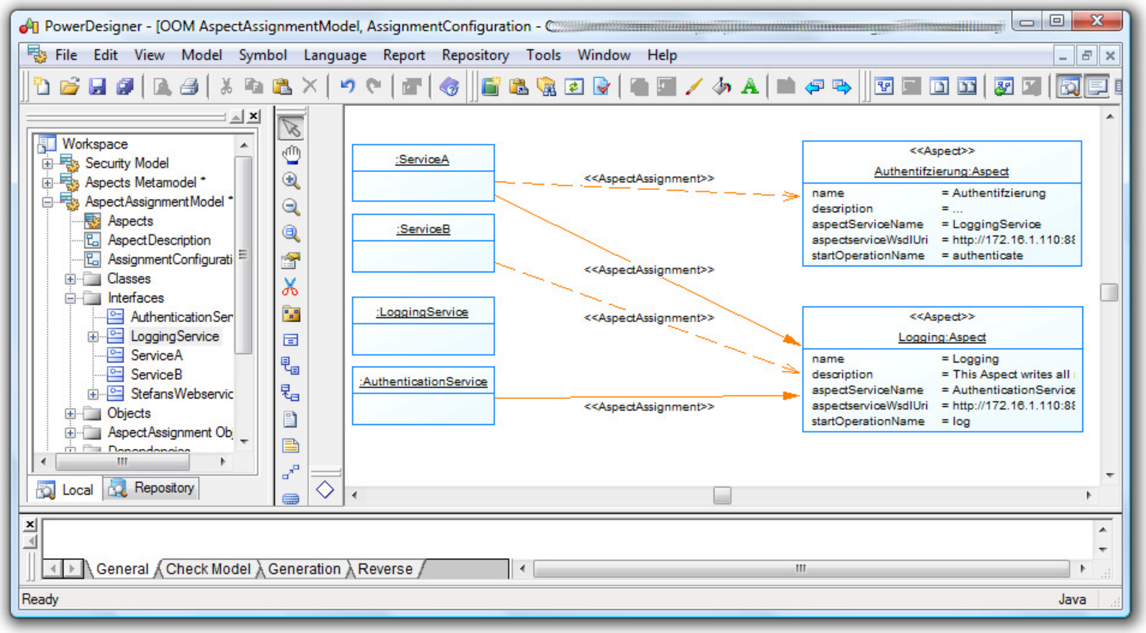
But the foundations of the editing are three repositories that comprise *aspect*s, *composition rules* and *join point*s. First, the *aspect*s are as mentioned in section 4. Second, the composition rules describe possible prerequisites for certain aspects. For instance, authorization does not make much sense, if no authentication occurred beforehand. Third, the join points configure how the join points are realized in a particular target platform.

The coarse grain editor can be any arbitrary modeling tool that allows certain extensions. For instance, we have chosen Sybase PowerDesigner 12.5 due to its popularity in the application area [4]. Hence, the designers already have service repositories which exist in PowerDesigner they are now able to reuse.

The actual assigning is easy. It is done by simple drag and drop actions and the assignment is illustrated as a line between two symbols as depicted in figure 4. Further configuration of this assignment is then done by the fine grain editor. Consequently, the coarse grain editor only needs to call the fine grain editor and needs to pass the information about the assignments.

We are aware of the drawbacks of this GUI approach. Especially the problem to keep things clear and orderly quickly occurs when models grow. But at the moment we do not expect too much complexity and therefore we have chosen this approach. Second, all changes in either model must be transparently reflected back to the other editor. But this ”round trip” compatibility was comparably easily realized by locking mechanisms.

One target platform we have taken into consideration as a server side has been the Oracle SOA Suite [4]. This platform is a heavy weight suite with a lot of tools. In particular the agent concept has become handy for realizing the join points and hence these agents call sentinels which are configured by our assignments. They do so in order to call aspect services. Of course, the Oracle SOA Manager can do something similar but it is too deeply integrated into the Oracle Software and lacks flexibility; especially with respect to integrating legacy systems.



```
name                  = Authentifzierung
description           = ...
aspectServiceName     = LoggingService
aspectserviceWsdlUri  = http://172.16.1.110:8
startOperationName    = authenticate
```

```
name                  = Logging
description           = This Aspect writes all
aspectServiceName     = AuthenticationService
aspectserviceWsdlUri  = http://172.16.1.110:8
startOperationName    = log
```

Fig. 4. Coarse Grain Editor taken from [4]

6 Related Work

A good point to start working is to get an idea of an overall SOA meta model from Linthicum Group [5]. This model gives a good idea on how subdividing issues in a SOA environment might work and how certain concerns crosscut the architecture. Two important points to mention are governance and security. Taking a closer look at security issues, some technical terms in information security might be helpful [6]. Using these, a brief view in the field of possible threats and challenges underlines the necessity of research [7]. A state of the art response to these challenges, in terms of programming, is given by Kanneganti by realizing security as services in a SOA environment [8].

Now, thinking about model driven development, a constructive meta model is required. One proposal aiming at code generation, which means it comprises a lot of attributes, has been published by Everware-CBDI [9]. The model's perspective is from a business point of view and services are not regarded as black boxes. Therefore, this meta model does not support recursion of concepts.

Back to security, one way to do e.g. authorization control is described as a pattern called secure service agent [10]. This pattern bases upon a secure service proxy and an intercepting web agent. Further, it aims at the integration of legacy systems as black boxes in general. Since this is only one approach how to deal with these issues a look into the "Analysis of the Security Patterns Landscape" gives a more precise overview [11].

Taking into account employed patterns and model driven development, one can build a meta model for supporting security issues by extending MOF [12]. Another way of building a meta model is extending OCL [13]. Both models have in common that they are considering services as black boxes; what distinguishes both approaches is the different stages in development they are applicable to.

We are not dealing with MOF nor OCL but AOP, and therefore a piece of related work is a proposal dealing with cross-cutting concerns by introducing a "Web Service Management Layer" [14]. This layer is an "adaptive middleware between applications and Web services" that takes care of management responsibilities at the client side [14]. There the employed approach complies with dynamical AOP and facilitates service integration during run time.

Additionally, a "Service Creation Environment" has been built to support visual editing for web services [15]. This tool is an Eclipse plug-in and employs "documented services" which can be used with "service composition templates" for modeling [15]. Finally, these compositions can be assigned to cross-cutting concerns by an XML-based language called Padus to be statically woven.

Another piece of related work is a paper from Tomaz et al. [16]. It gives a detailed description on how an idea, which is very similar to ours, was implemented in a web services environment. But, this paper does not take into account too much conceptual roots or design aspects. Instead, it focuses very much on the employed target platform and the XML documents used.

7 Summary and Outlook

To sum up, we examined a server side SOA environment and tried to find a light weight approach for dealing with cross-cutting concerns. We used this approach to build a design tool for easy assignment of aspect services to black box services at design level. But, one needs to be aware of some constraints which might occur when messages are affected by aspect services. Hence, we examined this in more detail and we analyzed filtering mechanisms in particular.

In addition, we tried to make the assignments transparent for the user and to integrate the approach into existing design tools. For this reason, we built a platform independent GUI tool that can be integrated into the designer's standard tool. In our terms we build a fine grained editor that can be used by the coarse grain editor. As a consequence, the GUI tool neither depends on the designer tool nor on the deployment platform.

References

1. Josutti, N.: SOA in Practice: The Art of Distributed System Design. O'Reilly Media (2008)
2. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In Akşit, M., Matsuoka, S., eds.: Proceedings European Conference on Object-Oriented Programming. Volume 1241. Springer-Verlag, Berlin, Heidelberg, and New York (1997) 220–242
3. Starke, G.: SOA Expertenwissen. dpunkt.verlag (2007)
4. Hurtz, S.: Modelling of Security Concerns in a Service Oriented and Model Driven Environment. Diplomarbeit, RWTH Aachen (Sept. 2008)
5. Group, L.: SOA Meta-Model. http://www.linthicumgroup.com/Linthicum(2007)
6. Savolainen, P., Niemela, E., Savola, R.: A Taxonomy of Information Security for Service-Centric Systems. Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on (Aug. 2007) 5–12
7. Schwarz, J., Hartman, B., Nadalin, A., Kaler, C., Davis, M., Hirsch, F., Morrison, K.S.: Security Challenges, Threats and Countermeasures. http://www.ws-i.org/Profiles/BasicSecurity/SecurityChallenges-1.0.doc (May 2005) Web Services-Interoperability Organization.
8. Kanneganti, R., Chodavarapu, P.: SOA Security. Manning (2008)
9. Everware-CBDI-SAE: A Meta Model for SOA - Version 2.0 (2007)
10. Emig, C., Schandua, H., Abeck, S.: SOA-Aware Authorization Control. Software Engineering Advances, International Conference on (Oct. 2006) 62–62
11. Heyman, T., Yskout, K., Scandariato, R., Joosen, W.: An Analysis of the Security Patterns Landscape. Software Engineering for Secure Systems, 2007. SESS '07: ICSE Workshops 2007. Third International Workshop on (May 2007) 3–3
12. Delessy, N.A., Fernandez, E.B.: A Pattern-Driven Security Process for SOA Applications. Availability, Reliability and Security, 2008. ARES 08. Third International Conference on (March 2008) 416–421
13. Alam, M., Breu, R., Breu, M.: Model driven security for Web services (MDS4WS). Multitopic Conference, 2004. Proceedings of INMIC 2004. 8th International (Dec. 2004) 498–505
14. Verheecke, B., Vanderperren, W., Jonckers, V.: Unraveiliny crossoutting concerns in Web services middleware (Jan.-Feb. 2006)
15. Braem, M., Joncheere, N., Vanderperren, W., Straeten, R.V.D., Jonckers, V.: Concern-Specific Languages in a Visual Web Service Creation Environment. In: Proceedings of the Second International Workshop on Aspect-Based and Model-Based Separation of Concerns in Software Systems. Volume 163-2., Amsterdam, The Netherlands, The Netherlands, Elsevier Science Publishers B. V. (2007) 3–17
16. Hmida, M.M.B., Tomaz, R.F., Monfort, V.: Applying AOP Concepts to Increase Web Services Flexibility. In: NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices, Washington, DC, USA, IEEE Computer Society (2005) 169