Aspect-Oriented Techniques for Web Services: a Model-Driven Approach[1]

Guadalupe Ortiz, Juan Hernández

Quercus Software Engineering Group
University of Extremadura
Computer Science Department
Spain
{gobellot, juanher}@unex.es

Abstract. In order to tackle the entire web service life cycle, it is necessary to face how to model systems based on service functionality and also how to add extra-functional properties to modelled services. In this regard, we propose first of all to use UML for modelling services based on the Service Component Architecture (SCA) specification, in order to provide a model environment in which extra-functional properties are added, also using UML. The implemented models based on these profiles will be independent of a final implementation language or platform, thus it is necessary to specify a particular type of model into which to convert the independent one in a subsequent step. In order to meet this requirement a JAX-RPC based specific metamodel is proposed for services and a soap tag, an aspect and a policy based ones are proposed as the intermediate step between the independent model and the final code.

Keywords. Extra-Functional property, web service, model-driven development, aspect-oriented techniques, WS-policy, service component architecture.

1 Introduction

Web Services provide a successful way to communicate distributed applications, in a platform independent and loosely coupled manner, providing the systems with great flexibility and easier maintenance. Although development middlewares provide a splendid environment for service implementation, methodologies for earlier stages of development, such us the modeling stage, are not provided in a cross-disciplinary scope, whereby, for instance, the automatic model-implementation transformation or the addition of extra-functional elements would be possible. At present, academy and industry are beginning to focus on the modeling stage, where it is also pursued to keep the loosely coupled notion and independence from the platform [10] [15]. Some rising proposals focus on representing the service as a component and others base the said model on WSDL elements; besides, the named approaches, component or

[1] This work has been developed thanks to the support of MEC under contract TIN2005-09405-C02-02.

WSDL-based may or may not propose a model-driven development; representative approaches are described below:

To start with, Service Component Architecture provides a way to define interfaces and references independently of the final implementation technology, which will be bound subsequently [3]. According to SCA, services are modeled as components, which are linked to a given interface which can be later specified in a particular one. Besides, they will show the needed references for their behavior to be completed. This way, a very high level model is defined, allowing the developer to implement it by using different approaches such as Java, BPEL, States Machine, etc., at a later stage. One additional example of a component-oriented proposal is the one from Smith et al. [12] where grid services are modelled as components stereotyped for grid environments.

As a second trend, we can find WSDL-centric proposals, where the service description is mainly based on the different elements which form the WSDL document. Most of the proposals try to find an appropriate way to model web services and their service compositions with UML based on the WSDL structure, as for instance [4] [6] or [13].

Additionally, many proposals are emerging in the literature where the Model-Driven Architecture (MDA) approach is being applied to Web Service Development. MDA has been proposed to facilitate the programming task for developers by dividing system development into three different phases: first of all, a *Platform Independent Model* (PIM) is proposed with the purpose of representing our system without coupling it to any specific platform or implementation language; secondly, a *Platform Specific Model* (PSM) represents our system based on a specific target platform; finally, *Code Layer* provides our final application code. Some of the component-oriented and WSDL-centric approaches may also propose a model-driven development. From those we mentioned earlier, for instance, the work from Smith et al. [12] consists of a model-driven development for grid applications based on the use of web services modeled as components; the research presented by J. Bezivin et al. [4] is also worth a special mention, where Web Service modeling, based on the WSDL structure, is covered in different levels, using *Java* and *JWSDP* implementations in the end.

Let us consider now that we want to provide our modeled services with extra-functional properties[2]. It is suggested by the SCA specification that this type of property may be modeled at a different level; the way to do so and to include them in additional stages of development has not been approached as yet. Alternatively, WSDL-centric proposals do not consider extra-functional properties in their models, since they are not part of the WSDL document elements. Besides, the named MDA proposals do not consider the way extra-functional properties may be included in modeled services. On the other hand, WS-Policies have emerged as a standardized way for describing extra-functional service capabilities by using the XML standard [14]. This allows properties to remain completely decoupled when described and

[2] By the term *extra-functional properties* we mean pieces of code which provide additional functionality to the services, but which are not part of their main functionality. This concept can be found in the literature as *extra-functional property*, *non-functional property* and even *functional aspects*. From now on we refer to this meaning when talking about extra-functional properties in this paper.

there is no need to establish dependences from the service description file (WSDL) to the policies ones; property description is not linked to a specific implementation, either, maintaining the platform's independent environment. However, WS-Policy does not determine how the properties are to be modeled or implemented, and an additional mechanism would be necessary so as to integrate property modeling and implementation with their description in service-based systems. We can mention the ASG (*Adaptive Services Grid*) project, which takes into consideration some specific extra-functional properties in their WSDL-centric model-driven development [11]; however, services and properties have to be initially described by a semantic language, and, being a WSDL-centric approach from the very beginning, the possibilities of implementation for the services described are limited.

Closely related to this is the main aim of this paper, which consists on offering a component-oriented model-driven methodology in order to deal with extra-functional properties in web service environments based on the standard modeling language UML. In order to do so we also provide a model-driven approach for the services themselves, although we expect the extra-functional property approach to be integrated with other web service model-driven approaches.

The rest of the paper is organized as follows: *Section 2* gives an overview of the whole process followed in this approach. *Section 3* shows how the PIM should be implemented, first of all presenting our profile proposed for service modelling in *Section 3.1*, then the one proposed for property modeling is explained in *Section 3.2* and finally showing the application of our approach to a case study in *Section 3.2*. *Section 4* explains the PSM stage, where *Section 4.1* shows the proposed specific metamodel for the services, *Section 4.2* explains the metamodels proposed for the properties and, then, *Section 4.3* shows the specific models obtained from the case study PIM. *Section 5* discusses the type of code to be generated from the platform-specific model. Finally, our main conclusions are presented in *Section 6*.

2 Model-Driven Transformations

In this section we are going to provide and depict a general overview of the presented approach, describing the order to be followed to face web service and their extra-functional property development from the platform independent model to code, which will be explained in detail in the next sections.

The first thing is to define the metamodel to be followed by the platform independent model. This can be done either by using MOF compliant metamodels (http://www.omg.org/mof/) or EMF ones (http://www.eclipse.org/emf/). We decided to use MOF-compliant metamodels at this level, which will be based in UML, thus facilitating the developer work by using common standard modeling tools at this stage of development. Particularly, we propose to use a UML profile based on the SCA specification for service models and an additional one for modeling extra-functional properties in a decoupled manner. Both are applied in this paper in order to obtain a case study model.

Afterwards, the specific metamodels have to be defined: in this case we decided for them to be EMF-compliant, which facilitates a graphical edition of the model element

attributes and their values, which permits easy consultation or modification, if necessary. In our approach, on the one hand, we propose a JAX-RPC metamodel for web services specific modeling, and on the other hand three different specific metamodels are provided for extra-functional properties, which will motivated shortly.

Subsequently, the transformation from PIM to the PSMs has to be defined. Several tools can be found for model transformations and code generation. We decided to use *ATL* (http://www.eclipse.org/gmt/atl/), which provides an Eclipse plugin and which has its own model transformation language. The ATL transformation file will define the correspondence between the elements in the source metamodel (PIM) and the target ones (PSMs) and will be used to generate the target model based on the defined rules and the input model. When the transformation rules are applied to the case study PIM, the case-study platform-specific models are obtained.

Finally, code can be generated from the specific models by applying additional transformation rules. In this case no target metamodel is needed since the output code files will be based on *Strings*. On the one hand, JAX-RPC web service code, to be deployed with the Java Web Service Developer Pack, will be generated from the service specific model. On the other hand, AspectJ will be used for the implementation of the property functionality, thus maintaining properties well modularized and decoupled from the services implemented; Java will be used to implement the code necessary for optional property inclusion. With regard to description, the WS-Policy [1] and WS-PolicyAttachment [2] documents are obtained for each property, which are integrated with the aspect-oriented generated properties. The usefulness of aspect-oriented techniques for extra-functional properties and its use in conjunction with WS-Policy is motivated and evidenced in [9].

3 PIM

In this section we are going to see the profiles proposed to model services and extra-functional properties and how they are used in a case study PIM.

3.1 The Service Profile

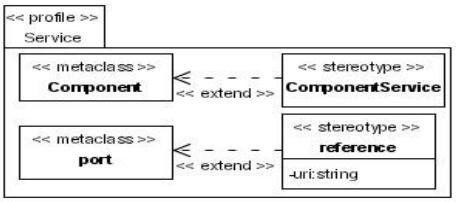

Fig. 1. Service profile.

As shown in *Figure 1*, the proposed service profile is very simple, since we aim to keep the simplicity and versatility of the SCA proposal. First of all, we can see the

serviceComponent stereotype which extends *component metaclass*. Secondly, we can see the *reference* stereotype which extends *port metaclass* and has the attribute *uri* to refer to the URI of the element needed to complete the service functionality. The elements *provided interface* and *required interface*, also used in the service model, are not defined in the profile as they already belong to the UML syntax. Further motivation and explanation on this profile can be found at [8].

3.2 The Extra-Functional Property Profile

In order to maintain our system loosely coupled when adding extra-functional properties to the model, we propose the profile in *Figure 2*, whose elements will be explained as follows and are described thoroughly in [8]:

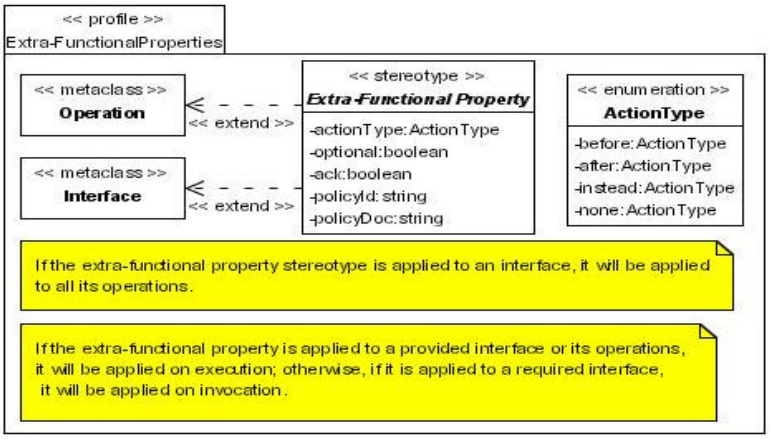

Fig. 2. Extra-functional property profile.

- First of all, we define the abstract stereotype *extra-functional property*, which will extend *operation metaclass* or *interface metaclass*. The extra-functional property provides five attributes: the first one is *actionType*, which indicates whether the property functionality will be performed *before*, *after* or *instead of* the stereotyped operation's execution – or if no additional functionality is needed it will have the value *none*, only possible in the client side. Secondly, the attribute *optional* will allow us to indicate whether the property is performed optionally –the client may decide if it is to be applied or not– or compulsorily –it is applied whenever the operation is invoked. Then, a third attribute, *ack*, is included: when *true* it means that it is a well-known property and its functionality code can be generated at a later stage; it will have the value *false* when only the skeleton code can be generated. Finally, we have two additional attributes, namely *policyID* and *policyDoc*. *PolicyId* contains the name of an existing policy or the name to be assigned to the new one in the service side; *policyDoc* allows the developer to reuse an existing policy document. If the attribute value is *null* then the WS-Policy document could be generated at code generation stage. The policy attachment

document would be generated in each case. These are the necessary attributes to define the main characteristics in any property, which may be complemented with the specific property attributes.

- In order to define *actionType*, an enumeration is provided with four alternative values: *before*, *after*, *instead* or *none*. In this sense, properties may include new functionality before executing the stereotyped operation, after it or they can even replace the operation's functionality. Some properties may be included from the client side without the need for any additional functionality, in which case its value would be *none*.

Once we want to use the profile in a specific case study, we will extend it with the specific properties to be used or we can have a pool of predefined properties, as for instance the one in *Figure 3*, which will be used in the next subsection.

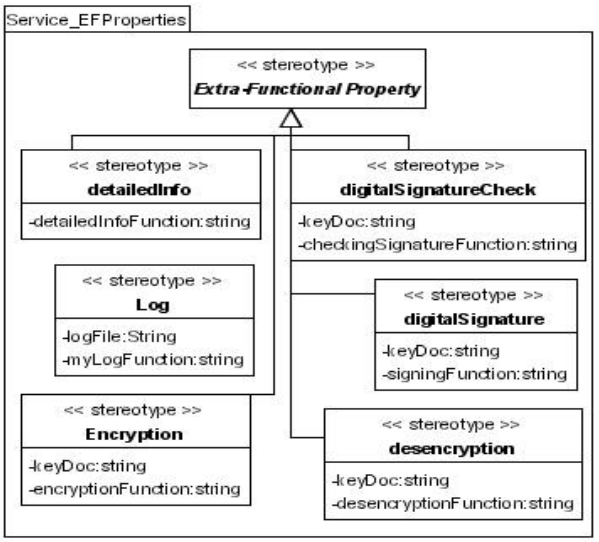

Fig. 3. Extension of the extra-functional property profile with specific properties.

3.3 Case Study PIM

Consider a simple case study in which we have a set of services related to the university administrative service (*Figure 4*). All the services are stereotyped as *ComponentServices*, which offer an interface. No references have been included as they are not relevant for property application. Let us imagine that we want to include some extra-functional properties to the services' model. We have devised four different sample properties, included in *Figure 4*:

- First of all, a *log* property, to be applied to all operations offered by the registration service to record received invocations.
- Secondly, a property called *detailedInfo*, which will be required discretionarily by the client when invoking *bringForwardExam* in *ExamOpportunityService*: exam dates and locations can be obtained when changing the semester in which the

student is going to take the exam; the change is regularly updated and no additional information is obtained.

- Additionally, invocations to *personalData* in *RegistrationService* must be encrypted. In order to enable this functionality the *desencyption* stereotype has to be applied to the offered operation.
- Finally, *sendPdf* from *PreregistrationService* can be invoked –optionally- with a digital signature. That is why the property *digitalSignatureCheck* stereotypes the named operation in the service model.

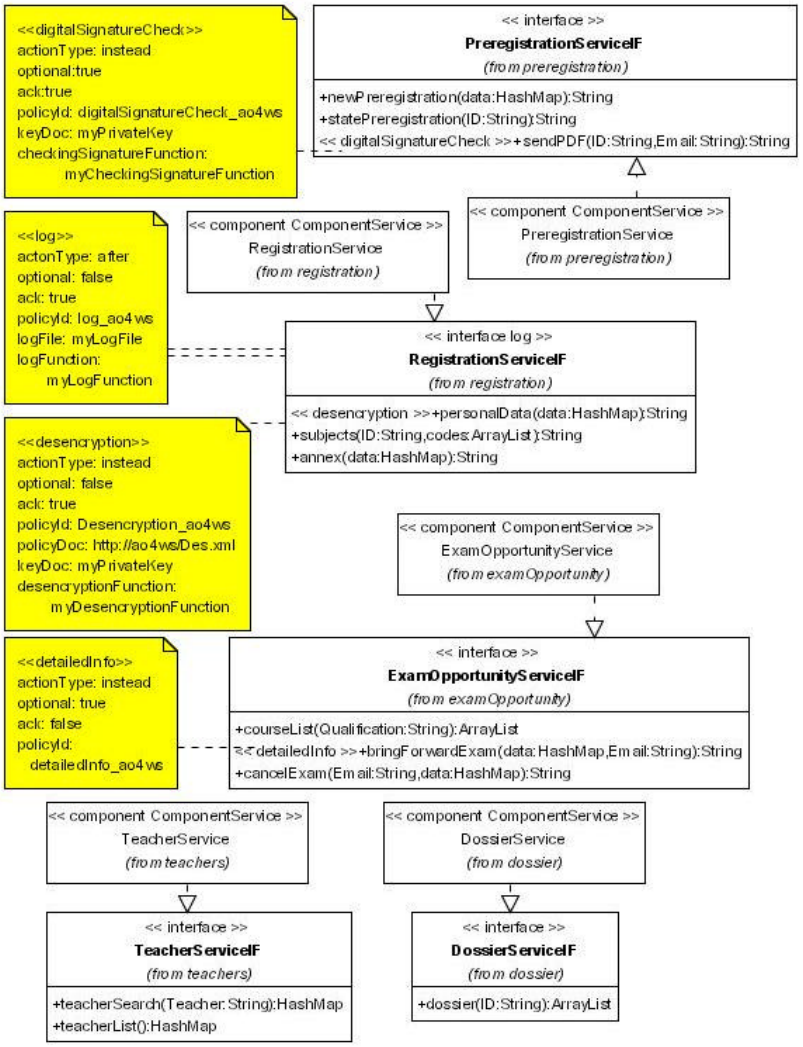

Fig. 4. PIM with extra-functional properties.

We have chosen, for instance, the *detailedInfo* property as a characteristic example for the remainder of the paper. In this sense, in order to provide *bringForwardExam* operation in *ExamOpportunityService* with *detailedInfo* in the PIM, we only have to stereotype the named operation with the << *detailedInfo* >> stereotype. Stereotype attributes are normally attached to models as tagged values, but they have also been included as comments in the illustration in order to show their values. In it the attributes for *detailedInfo* indicate that the property will be performed *optionally instead* of the execution of the named operation; it is not a well-known property; *policyID* is *DetailedInfo _ao4ws* and *policyDoc* is null (omitted for clarity). The remaining property values would be similarly interpreted; however, for those properties where *ack* is *true*, the function which will be invoked to include the additional functionality would also be specified.

4 PSMs

In this section we will show, first of all, the metamodel proposed for service specific models, then those proposed for extra-functional properties and, finally, the specific models obtained from the case study PIM will be discussed.

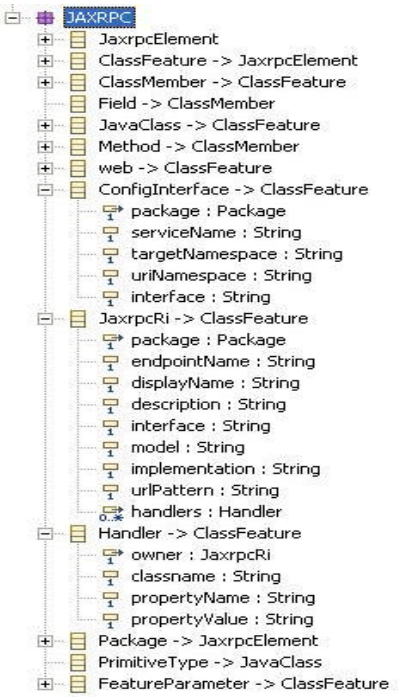

Fig. 5. Jax_Rpc services proposed metamodel.

4.1 Service PSM metamodel

We intend for our extra-functional property proposal to be used for different service models and implementations; however, we provide one example of service specific model generation to show the proposal's value. We decided to generate a specific model oriented to JAX-RPC services to be compiled and deployed with Java Web Service Developer Pack. In this regard the metamodel will be formed by the service java interface and its implementation plus the necessary configuration files: web, config-interface and jaxrpc-ri; these elements and their corresponding attributes are shown in *Figure 5*. The metamodel, as shown in the said figure, is EMF-compliant instead of MOF-compliant, since it allows the developer to edit the generated EMF specific models to easily check and modify the property attributes when necessary.

The different elements shown in the named figure correspond to a simplified Java metamodel plus the three configuration files, which contain the main attributes necessary for their description.

4.2 Extra-Functional Property PSM Metamodels

As far as extra-functional properties are concerned, our specific models will be based, first of all, on an aspect-oriented [5] approach to specify the property functionality, secondly on what we have called a *soap tags*-based approach, to lay down the necessary elements to be included or checked in the SOAP message header and, finally, a policy-based one for property description.

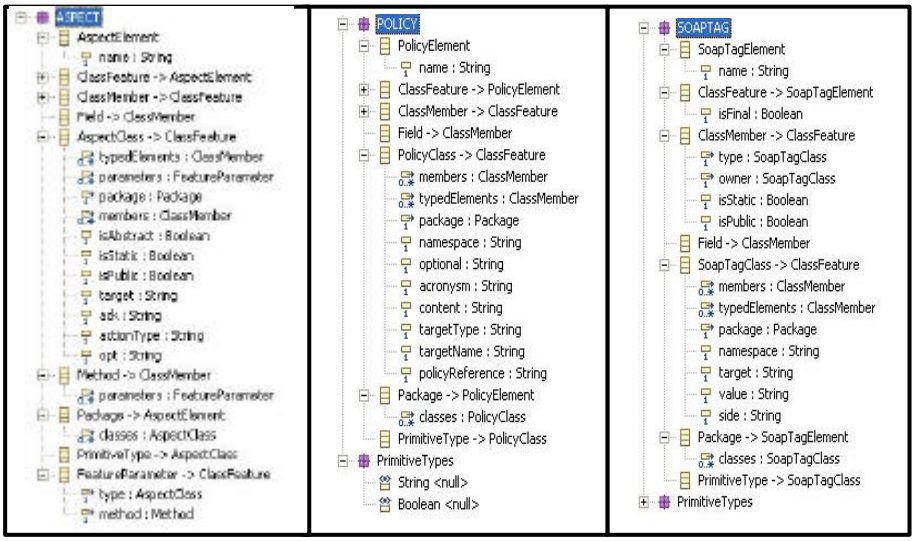

Fig. 6. PSM metamodels.

This way property implementation remains encapsulated and decoupled from the main functionality thanks to aspect-oriented techniques (further information on aspect-oriented techniques can be found at [5] and [7]). Besides, SOAP Tags will be

used to select optional properties and transfer the additional data necessary due to the property inclusions in a transparent way. To end with, policies allow us to describe the properties, independently of the implementation of the service and of the property itself and also remain decoupled from the service description file.

EMF-compliant metamodels are depicted in *Figure 6* and explained below:

- Every *aspectClass* will have an attribute *target* which indicates the method for the property to be applied, a second attribute, *actionType, which* informs of when it has to be applied; *ack* indicates whether the property is well-known or not and, finally, an *action* refers to the corresponding functionality. Besides, all additional characteristics from the particular property will be included as attributes.
- New tags are included in the SOAP Header to select –in the client side– or check –service side– relevant properties, when optional, or to deliver any other necessary information. Every *SoapTag* element will have an attribute *target* which instructs the method for the property to be applied, a second attribute, *value,* to indicate the tag to be included; finally, *side* indicates whether the tag is to be included by the client or checked by the service.
- A policy will be generated for each property. The policy element will contain the policy *name*, whether the property is *optional*, well-known or domain-specific (*ack*); *targetType* indicates whether the policy is to be applied to a *portType* or an *operation* and *targetName* gives the name of the latter.

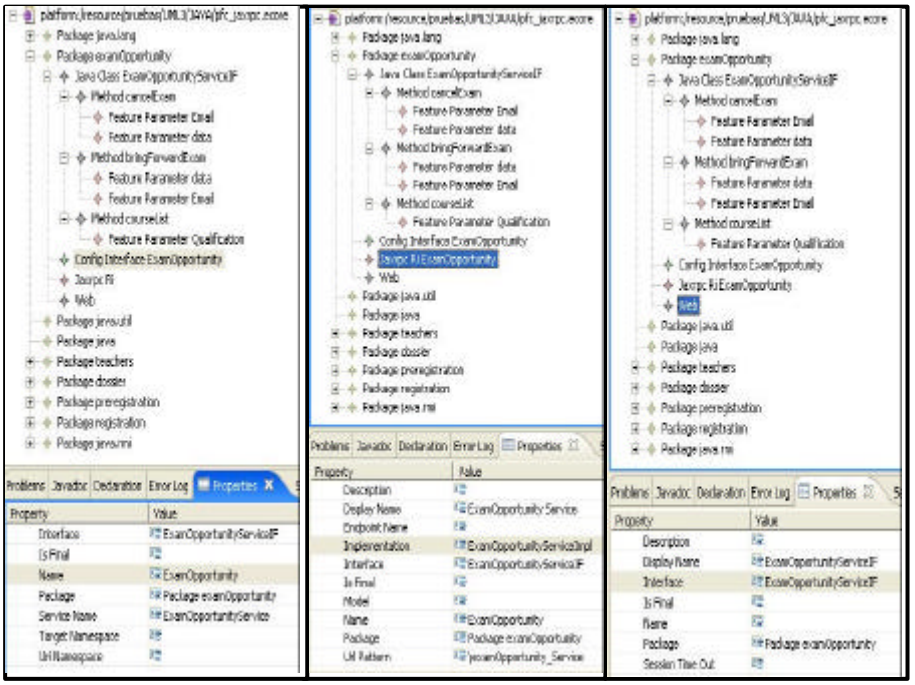

Fig. 7. Specific models of the case study web services.

4.3 Specific Models in our Case Study

Once the transformation rules are applied to the PIM, the specific models containing information on the services and properties are obtained. We show some of the model's elements and their properties in *Figures 7* and *8*, where service and property models can be examined, respectively.

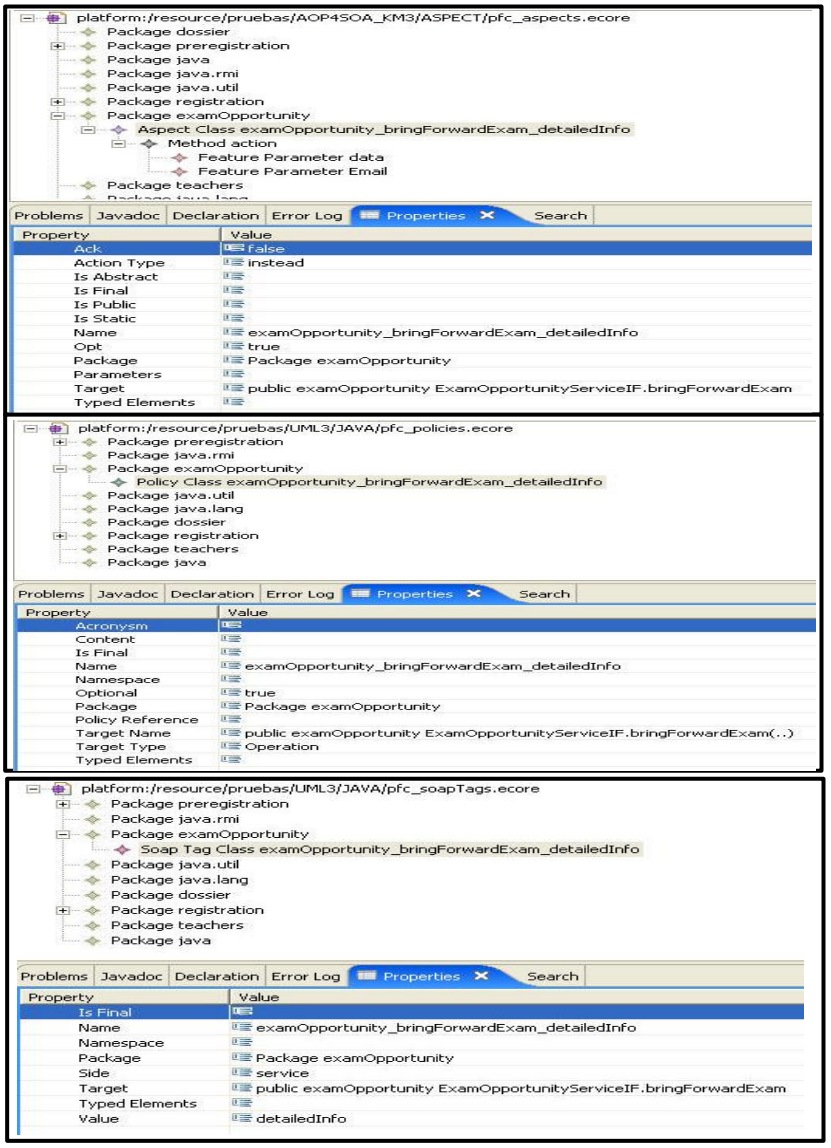

Fig. 8. Specific models of the case study extra-functional properties.

Only some branches of the tree structure in the figures have been deployed in order to make the illustration easier to understand. All the elements can be examined in the upper sections of the illustrations, whereas at the bottom, only properties of the selected item are shown and can be modified.

Figure 7 shows the created web services with their corresponding generated elements, namely service Java interfaces and configuration files. We have deployed the *examOpportunity* service package, where we can see the Java interface *examOpportunityServiceIF* with its corresponding methods and parameters. We can also see the properties corresponding to the three configuration files in the three parts of the figures.

Figure 8 shows the property models obtained, where we have deployed the elements corresponding to the *detailedInfo* property – aspect, policy, soap Tags *exam_Opportunity_bringForwardExam_detailedInfo*, which are explained as follows:

- An aspect, *examOpportunity_bringForwardExam_detailedInfo*, will be generated for *detailedInfo* in the service side; its attributes *target* and *actionType* will have the values *examOpportunity.bringForwardExam* and *instead*, respectively and *ack* will be *false*.
- Regarding the policy element, its name will *be detailedInfo_ao4ws*; its *optional* value will be *true*. Finally, for policyAttachment, *targetType* will be *operation* and *targetName bringForwardExam.*
- Due to its optional nature, we ought to include code whose function is to check whether *detailedInfo* has been selected: the corresponding *SOAPTag target* will be *bringForwardExam*, its value *detailedInfo* and it will operate as a *side service*.

5 Generated Code

Once we have our models for the case study we may also apply additional rules to generate code from them. From the service specific model, where additional attribute values can be added or modified (e.g. *deployment endpoint*), the JAX-RPC service skeleton code for JWSDP compilation and deployment is generated. In this sense, the Java interface and implementation skeleton will be generated and complete configuration files created. *Figure 9* shows the java interface generated for *examOpportunityService*.

From the property models, transformation rules will generate skeleton code for the three extra-functional property model elements: *Figure 9* shows the code generated for *detailedInfo*. However, in the case of well-known or user-defined properties, a repository with specific code may be maintained to generate additional code for the three of them. In these cases, in which *ack* is *true*, it is possible to generate the advice functionality and further policy content.

Regarding property implementation, Java code will be generated to check if soap tags are included in the SOAP message and AspectJ has been chosen for the implementation of the property's functionality, consequently properties remaining well modularised and decoupled from implemented applications, as demonstrated in [7]. An AspectJ aspect will be generated for each aspect class in our model. AspectJ pointcuts will be determined by the execution of the target element. Concerning the

advice, depending on the *actionType* attribute value, *before*, *after* or *instead*, the advice type will be *before*, *after* or *around*, respectively; its name will be the one in the *action* attribute. With regard to property description, it is proposed to generate the WS-Policy and WS-PolicyAttachment documents for each property, which are integrated with the aspect-oriented generated properties as explained in [7]. In this sense, an xml file based on the WS-Policy standard is generated and attached to the service by the file based on the WS-PolicyAttachment standard. The policy is attached to the stereotyped element in the PIM model, which is represented in the policy specific model by the attribute *targetName*.

```
**************************************EXAMOPPORTUNITY SERVICE INTERFACE**********************
   package examOpportunity;
public interface ExamOpportunityServiceIF extends Remote {
    public ArrayList courseList (String Qualification) throws RemoteException;
    public String bringForwardExam(HashMap data, String EMail) throws RemoteException;
    public String cancelExam(HashMap data, String Email) throws RemoteException;}
```

```
*********************************************DETAILED INFO ASPECT*****************************************
public aspect opportunityExam_bringForwardExam_detailedInfo {

pointcut bringForwardExam_detailedInfoP (data: hashMap, Email:String):     execution(public *
            opportunityExam_bringForwardExam (HashMap, String)) && args(data, Email);

String around  ((data: hashMap, Email:String): bringForwardExam_detailedInfoP (data, Email){
  if  ((((String)opportunityHandlerHandler.operDetailedInfo.get("operationName")).
                                                    compareTo("bringForwardExam")==0)&&
   (((String)opportunityHandler.operdetailedInfo.get("propertyName")).compareTo("detailedInfo")==0)) {
              [...]   [functionality to be completed]   [...]}
              else result=proceed(data, Email)       [...]}
```

```
********** DETAILED INFO POLICY****
<wsp:Policy name=detailedInfo_ao4ws
xmlns:wsl="... "> <[to be completed]/>
</wsp:Policy>
<wsp:PolicyAttachment >
<wsp: AppliesTo>[...]
   <wsp:Operation Name=
          bringForwardExam/>
[...] </wsp:AppliesTo>
</wsp:PolicyAttachment>
```

```
/*********************SERVICE SIDE SOAP CODE************
if element.getElementName().getLocalName().equals
                                        ("operationName"))
  String operationName = element.getValue();
  operdDetailedInfo.put("operationName",operationName);

Iterator iter2= element.getAllAttributes() ;[...]
     If (name.getLocalName().equals("propertyName")) {
      String propertyName = Element.getAttributeValue(name);
      operDetailedInfo.put("propertyName",propertyName); } }
```

Fig. 9. Code obtained from the transformation.

6 Conclusions

This paper has shown a model-driven approach for web services and their extra-functional properties. Services are described at PIM level by using an UML profile based on the Service Component Architecture specification and properties are included by using an additional UML profile. Additionally, thanks to an ATL transformation, the initial PIM has been converted into four specific models which conform to four provided metamodels, the first one to model JAX-RPC-based services and the rest, based on soap tags information, aspect oriented elements and policy based ones, for property selection, implementation and description, respectively. Moreover, an ATL transformation can also be used to generate the skeleton code related to the JAX-RPC services on the one hand and on the other to

their extra-functional property implementation with AspectJ, their property description with WS-Policy, and their property selection by Java.

Thus, this proposal allows properties to remain well encapsulated and decoupled at all stages of development, providing the possibility of generating properties and services automatically and independently. Besides, disunion of concerns lets us benefit from good traceability, since any extra-functional property in the model can be located in the code clearly, avoiding its being tangled with the main functionality code and vice versa.

7 References

[1] Bajaj, S., Box, D., Chappeli, D., et al.. Web Services Policy Framework (WS-Policy), ftp://www6.software.ibm.com/ software/developer/library/ws-policy.pdf, September 2004

[2] Bajaj, S., Box, D., Chappeli, et al. Web Services Policy Attachment (WS-PolicyAttachment), ftp://www6.software.ibm. com/software/developer/library/ws-polat.pdf, September 2004

[3] Beisiegel, M., Blohm, H., Booz, D.,et al. Service Component Architecture. Building Systems using a Service Oriented Architecture. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-sca/SCA_White_Paper1_09.pdf, November 2005

[4] Bézivin, J., Hammoudi, S., Lopes, D. et al. An Experiment in Mapping Web Services to Implementation Platforms. N. R. I. o. Computers**:** 26, 2004

[5] Elrad, T., Aksit, M., Kitzales, G., Lieberherr, K., Ossher, H.: Discussing Aspects of AOP. Communications of the ACM, Vol.44, No. 10, October 2001.

[6] Grønmo, R, Solheim, I Towards Modeling Web Service Composition in UML. Int. Workshop Web Services: Modeling, Architecture and Infrastructure, Porto, Portugal, 2004.

[7] Kiczales, G. Aspect-Oriented Programming, ECOOP'97 Conference proceedings, Jyväskylä, Finland, June 1997

[8] Ortiz G., Hernández J., Toward UML Profiles for Web Services and their Extra-Functional Properties, Proc. Int. Conf. on Web Services, Chicago, EEUU, September 2006 (awaiting publication).

[9] Ortiz, G., Leymann, F. Combining WS-Policy and Aspect-Oriented Programming. Proc. of the Int. Conference on Internet and Web Applications and Services, Guadeloupe, French Caribbean, February 2006

[10] Papazoglou, M. Van Den Heuvel, W. Service-oriented design and development methodology, International Journal in Web Engineering and Technology, Volume 2, Issue 4, 2006.

[11] Roman, D et al. Requirements Analysis on the ASG Service Specification Language. Deliverable D1.1-1, DERI Innsbruck, 2005.

[12] Smith, M., Friese, T. Freisbelen, B. Model-driven Development of Service-Oriented Grid Applications. Proc. of the Int. Conference on Internet and Web Applications and Services, Guadeloupe, French Caribbean, February 2006

[13] Thöne, S. Depke, R, Engels, G.. Process-Oriented, Flexible Composition of Web Services with UML. Int. Workshop on Conceptual Modeling Approaches for e-business: A Web Service Perspective, Tampere, Finland, 2002

[14] Weerawarana, S. Curbera, F. Leymann, F., et al. Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More, Ed. Prentice Hall, ISBN 0-13-148874-0, March 2005

[15] Zimmermann, O., Krogdahl P, Gee, C. Elements of Service-Oriented Analysis and Design, http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/, May 2004