How to Model Aspect-Oriented Web Services

Guadalupe Ortiz
Juan Hernández
Pedro J. Clemente
Pablo A. Amaya
Quercus Software Engineering Group
University of Extremadura
Computer Science Department
gobellot@unex.es
juanher@unex.es
jclemente@unex.es
pabloama@unex.es

Abstract

Web Services provide a new and successful way of enabling interoperability among different web applications. In this paper, an MDA approach to modelling Web Services, in which aspect-oriented techniques are also applied, is provided. The UML profiles required to model aspects and Web Services independently from the platform (PIM) are presented. Once the system is modeled at this level of abstraction, transformation rules have to be applied in order to obtain the platform specific model (PSM). In this respect, two different approaches for the specific model are discussed, and benefits and shortcoming will be analyzed depending on whether the aspects' weaving is performed at design or implementation level.

1. Introduction

Web Services have become the new way to implement and compose applications through the Web, and they have had a great impact in the current way of developing applications.

Once Web Service technology seems to be highly consolidated, it is time to tackle how they can be modelled in order to be able to generate the necessary code automatically. Although it is influential to both evolution and maintenance in Web Services, the market has not proposed a suitable answer to this matter for the time being.

Furthermore, in spite of the importance of extra-functional properties' implementation, and interaction logic encapsulation in compositions in the area of Web Services, a unique and valid proposal has not yet been suggested. We have already proposed the encapsulation of both extra-functional properties and the composition interaction logic by using aspect-oriented programming (AOP) [1] as a suitable way to implement these aspects of Web Service development [2][3][4], but these issues must also be addressed in a more abstract level.

For this reason, we propose to model Web Services and their composition in a well modularized way, maintaining the logic decomposition of units by using a new profile to model aspects and another one to model Web Services altogether, independently of the platform. In addition, once we have the independent model, it has to be transformed into a platform specific one. Due to the presence of aspects, we find two different alternatives for the target specific model: firstly, to perform the weaving at transformation, so that no aspects remain in the specific model; secondly, to maintain the aspects in the specific model, keeping them abstract or translating them into a specific aspect-oriented language. Therefore, the main contributions of this paper are the definition of a PIM model for Web Services, in which aspect-oriented techniques have also been applied, and the analysis of two different approaches to the specific model depending on whether the aspects' weaving is performed at design or implementation level.

The rest of the paper will be arranged as follows: a case study PIM is presented in Section 2 to identify the various operations offered by the services and the way in which they are connected, highlighting how the aspects appear to deal with extra-functional properties and compositions' interaction logic. Section 3 specifies both the way to model Web Services and the one to model aspects and then moves on to outline the PIM appearance once it has been decided to include the

aforementioned aspects. Section 4 presents two alternatives for the aspect weaving, and the influence of the decision in later stages will be examined. To finish with, we discuss our proposal and future work in Section 5.

2. The Case Study

The case study is not detailed in depth in this paper as a common example was proposed for all the position papers submitted to this workshop. With the sole purpose of clarifying the main operations in the different services and the relations among them, a very general PIM is presented in *Figure 1*. To avoid confusion, points which had no relevance to our proposal have been omitted from the diagram. .

Consider now that we want to add extra-functional properties to the case study. The tangling resulting from the added properties causes difficulties when designing the model in a conventional way. The same problem is encountered when trying to model the compositions' interaction logic in an independent unit. Both issues have been studied and solved at programming level, as mentioned in the introduction, but not at design level. In this sense, our proposal aims at solving this inadequacy.

Therefore, various aspects can be included in our case study in order to model extra-functional properties and compositions' interaction logic in a modular and structured way. The *customer*, the *travel agent* service and the *bank* service need to communicate in a secure way, as these are the three elements which send information, such as credit card numbers, which must travel securely over the net. This security can be provided by the aspect *Encryption*. We can also use the *Logging* aspect in the travel agency service to maintain a file with all the operations invoked. Finally we have also decided to include the T*ravel_agComp* aspect, which encapsulates the composition interaction logic for the different services required to offer the travel agent behaviour. Similarly, the B*rokerComp* aspect may be included; its task would be to encapsulate the interaction logic of the different services required to offer the broker behaviour.

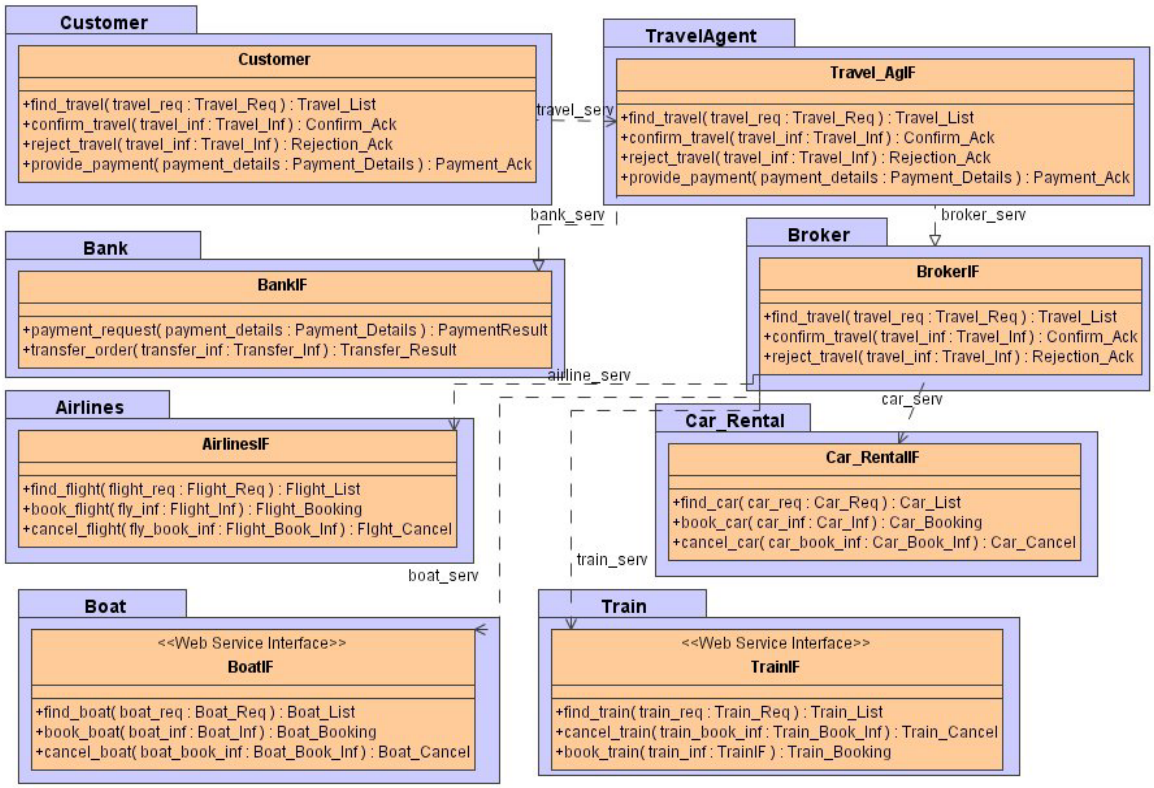

Figure 1. Case Study PIM

3. The Web Service and Aspect-Based PIM

In this section we are going to introduce our proposal in three steps: first of all, our proposal for modelling Web Services will be explained; secondly, our proposal for modelling aspects will be described and finally, the complete PIM proposed for the case study, based on previous premises, is presented.

3.1 Web Service Modelling

We can find many Web Service modelling-related proposals, such as [9] [10] [11]. From many of them it can be noted that most of the literature in this area tries to find an appropriate way to model service compositions with UML and most of them use the WSDL structure in order to model services.

The research presented by J. Bezivin et. al [12] is worth a special mention; in it Web Service modelling is covered in different ways, finally using *Java* and *JWSDP* implementations. Starting from this proposal, first of all, we decide to model our services representing the WSDL elements, but in order to simplify the model for our case study we will only show the *Web ServiceIinterface*, which would match the binding element in the WSDL metamodel representation in the said report. In this sense, we could extend the Web Services modelled in our case study to the rest of the elements in the WSDL metamodel.

.

3.2 Aspects Modelling

AOSD (Aspect-Oriented Software Development) has been widely studied in the specialised literature, and we can find different proposals on how to model aspects with current products. We can find various proposals mainly oriented to a specific language, such as *AspectJ* [5] [6], or more general ones [7] [8]. The first ones show *pointcuts* and *advices* to be clearly distinguishable, whereas the second ones are not so oriented to *AspectJ* terms, but only to the points to be intercepted and their associated behaviours.

Our goal is to obtain a model which is independent from the platform, so we have followed the more general proposal to make ours, where we define the aspects in a completely independent way, as depicted in *Figure 2*.

Aspect-oriented techniques describe five types of elements to modularize crosscutting concerns: firstly, we have to define the *join point* model which indicates the points where new behaviours could be included. Then a way to indicate the specific *pointcuts* needs to be defined, to specify in which points of the implementation we wish for the new code to be inserted. Next, we ought to determine how we are going to specify the new behaviour to be included in the *join point* referred to. We would then encapsulate the specified join points and their corresponding behaviours into independent units. Finally, a method to weave the new code with the original one has to be applied [1].

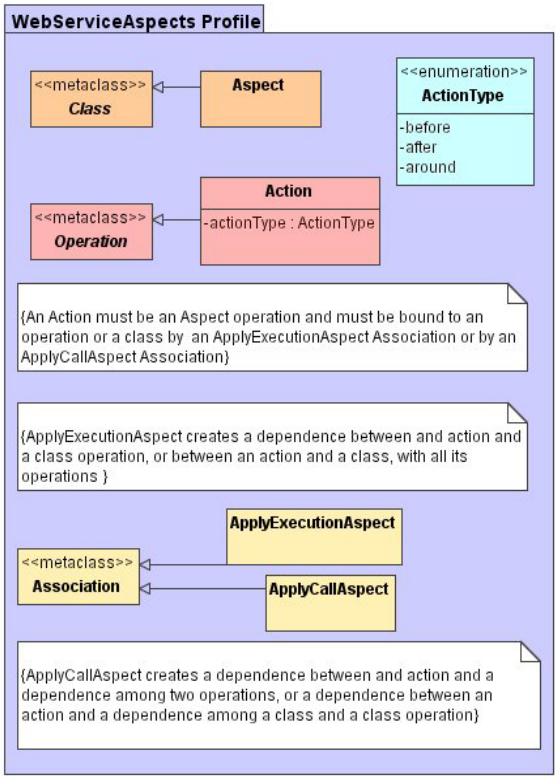

Figure 2. Aspect-Oriented UML Profile

Therefore, in *Figure 2* we can see the new stereotypes defined in order to model the five aspect-oriented types of elements just described, which are briefly going to be outlined:

- Due to the fact that services are *black boxes*, there will be two possible types of interaction points in Web Service applications: firstly, the point in which a service operation is to be executed, regardless of who made the invocation; secondly, when a service client (which may also be another service) invokes a specific operation in the target service. The first ones will be referred to as *executionAspects* from

now on in this article, as they are triggered when the service method is about to be executed; on the other hand, the second ones will be called *callAspects* since they are triggered when a specific operation is invoked by the client. Therefore, these two types of point in our application´s execution conform our join point model.

- In order to identify the pointcuts, two stereotyped associations have been added. In *executionAspects*, the association *ApplyExecutionAspect* binds the action, which implements the new behaviour to be included, to the operation whose execution is being intercepted; this association represents the pointcut in this type of aspect. For *callAspects* pointcuts, we have added the stereotyped association *ApplyCallAspect*, which binds *Action* to the invocation of a service method by a client one.
- The advices are represented by the *Action* stereotype which can be applied to operations and which will have a *tag Value* to indicate the type of action (*Before, After* or *Around*), that is, if the advice behaviour is going to be injected before, after or around the pointcut.
- The stereotype *aspect* contains *Actions*, which are linked to a service method by the *ApplyExecutionAspect* association or to a dependence between the client and a service method by the association *ApplyCallAspect*.
- Regarding *weaving*, two different alternatives will be studied in *Section 4*.

3.3 The Aspect-Oriented Case Study PIM

In this section, aspects and Web Service modelling are joined in the case study to illustrate our proposal. The complete case study model, in which various aspects have also been represented, is depicted in *Figure 7,* annexed at the end of this paper. Some details have been omitted in order to simplify model's presentation.

The structure of Web Services, as we mentioned before, limits the join point model [13] to the interface method execution on server-side and to their calls on client-side. The reason is that Web Services are implemented as *black boxes*, where only the interface with the offered operations is shown, thus limiting the logical join point model to those operations. Regarding actions associated to pointcuts, we have to indicate the action type, *before*, *after* or *around*, depending on the moment in which the action is to be executed in relation to the pointcut.

In order to illustrate how Web Services and aspect modelling are integrated in the case study, we have partitioned the complete PIM model into small pieces which allows us to explain the model definition in detail. In this respect, we are firstly going to study the application of an extra-functional property, which leads to the creation of a new *executionAspect*; then we will analyse the application of an extra-functional property which leads to the appearance of both an *executionAspect* and a c*allAspect* and, finally, the application of one aspect which encapsulates the composition logic of one service, which, as we will see, leads to an *executionAspect* ocurrence.

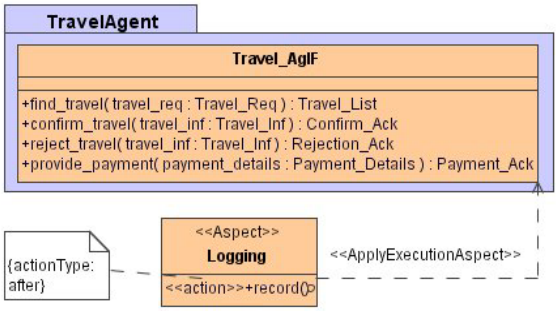

Figure 3. *Logging* Aspect Application Modelling

To start with, we can see how an extra-functional property, which solely affects one service, is modelled. As can be seen in *Figure 3*, a *logging* property is applied to the travel agent service. The Figure shows the stereotyped *Aspect Logging*, which contains *Action record*. This action has an associated note that specifies that it is an *after* action, that is, that the action will be executed after the intercepted method execution. Besides, the action is bound to *Travel_AgIF*, the service interface, by the stereotyped association *ApplyExeuctionAspect*, which means that the action will be applied to the whole method in *Travel_AgIF* class. As a result, every time any *Travel_AgIF* method is executed the record action will be triggered, after the method execution is complete.

Figure 4 shows how the *Encryption* property is modelled in our case study. This property, as can be observed in the above Figure, is applied to customer and travel agent through the aspect stereotypes *Server_Encryption* and *Client_Encryption*, with its associated stereotyped action *EncryptionAction.* As can be observed from the Figure, *Server_Encryption* is an *executionAspect*, thus it is modelled in the same manner as *Logging*: the action in *ServerEncryption*,

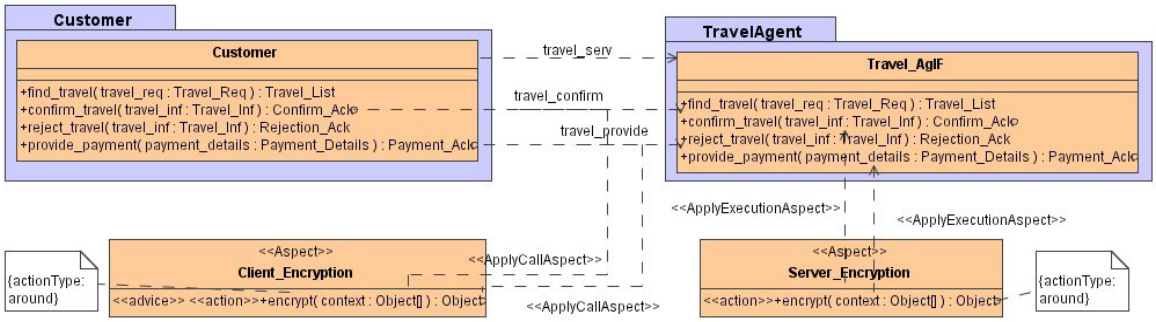

Figure 4. *Encryption* Aspect Application Modelling

this time an *around* action, is linked by the use of an *ApplyExecutionAspect* association to the methods *confirm_travel* and *provide_payment* in travelAgent. This means that any time these methods are executed, the *ServerEncryption* action will be executed *around* it (This action would desencrypt the parameters received in the invocation, then allow method execution with the desencrypted parameters and finally would re-encrypt the method result) The case of *Client_Encryption* is different as it is a *callAspec*. *Figure 4* shows how the action in *Client_Encryption* is bound to the dependence between *Customer confirm_travel* and *Travel:Agent confim_travel* and between *Customer provide_payment* and *Travel_Agent provide_payment*. In fact, it could be reduced to the dependences between *Customer* and methods *confirm_travel* and *provide_payment* in *TravelAgent*. This means that any time these two methods are invoked by the customer, the *Client_Encryption* action will be applied *around* (The call is intercepted, then invocation parameters are encrypted, after that the call goes on and, to finish with, when the result comes back, it is desencrypted before it is reused). We can now see how the same properties are reused in the case study to manage security between the travel agent and the bank. As can be observed in *Figure 5*, now the *Server_Encryption* action is also applied to *Bank payment_request*, thus any time this method is executed, the associated action will be executed *around*. Moreover, the named Figure shows that a

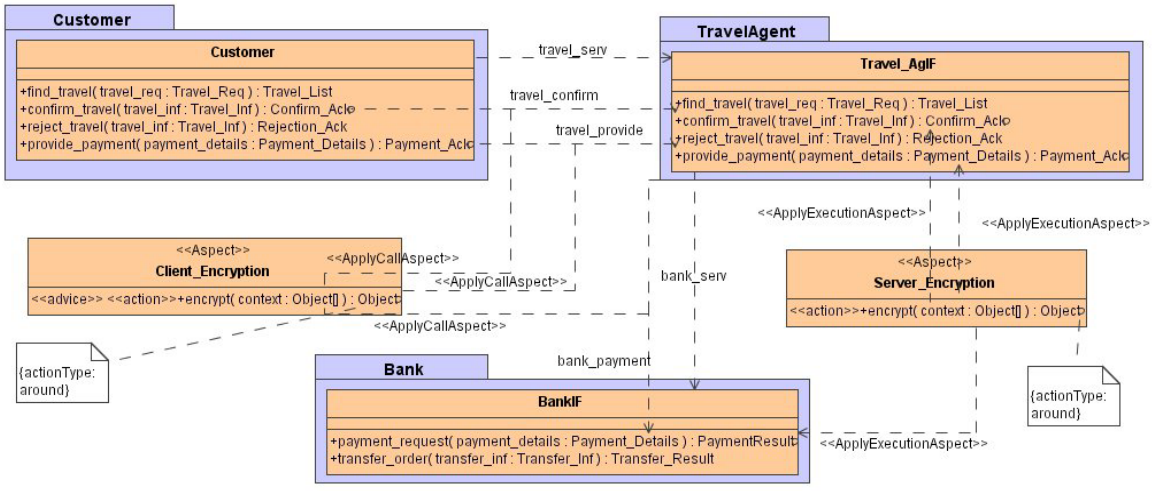

Figure 5. *Encryption* Aspect Reusing Modelling

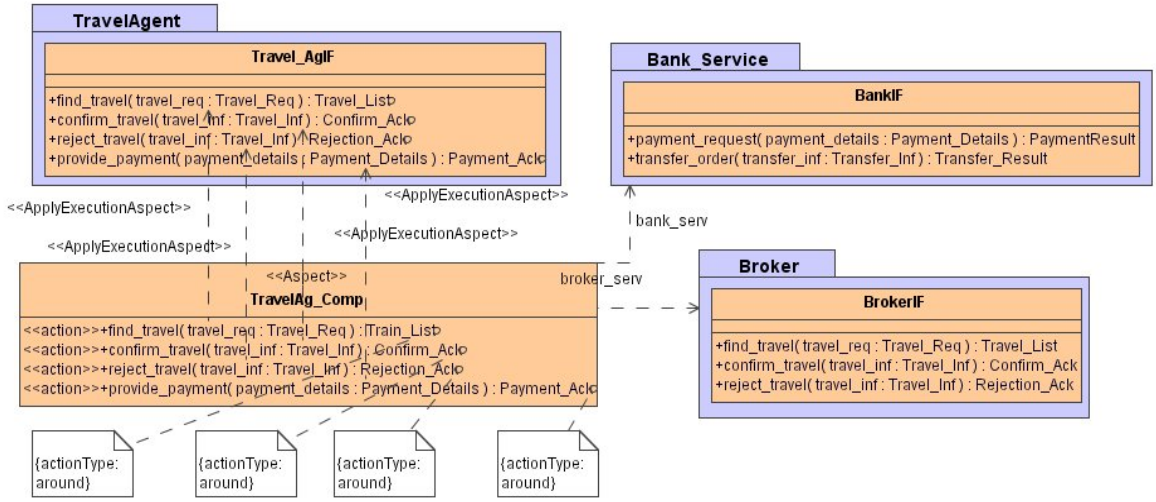

Figure 6. *Travel_AgComp* Aspect Application Modelling

Client_Encryption aspect is also associated to any invocation made by the *Travel_Agent* service to the *provide_payment* method in *Bank_Service*. In this case, the travel agent behaves as a client as it is the one making the call.

Finally, as we have previously explained, not only are extra-functional properties modelled into individual units, but also compositions' interaction logic, as depicted in *Figure 6*, where it can be seen that the *Travel_AgComp* Aspect is applied to the travel agent service. This aspect will define the interaction logic for composing the different services in the case study for the *Travel_ag* service to offer its behaviour. As depicted in *Figure 6*, the aspect actions are linked to different methods in *TravelAgent*, as there will be different actions for every method execution. Each action refers to the execution of one of the operations on offer: one will be associated to each method to offer its composition interaction logic. Therefore, four different actions can be found in *Travel_AgComp*.

To illustrate this idea, we can assume that the *confirm_travel* operation is invoked by the customer. The behaviour associated to the action *confirm_travel* in *Travel_AgComp* will be the one to trigger the bank service invocation to check the credit card provided and, depending on the result, invoke the broker to confirm this trip and cancel the rest, or ask the client to reenter credit card information. By modelling this interaction logic as an aspect action, all dependences among the composed services will be avoided, the action being the one which will compose them in a decoupled way.

Similarly, not represented in this Figure but available in the Appendix, the broker service has the aspect *BrokerComp* applied to it, where three different actions are defined for the execution of the three different operations on offer. Similarly to *Travel_AgComp Aspect*, *BrokerComp* also has one action associated to each method in order to develop the behaviour offered by the broker service through the composition of various services' invocations (*airline* service, *car_rental* service, *train_rental* service and *boat_rental* service).

4. Weaving Alternatives

Once we have our system aspect-oriented PIM we have to decide when we desire the weaving to be performed. We have two alternatives, which are going to be discussed in the next subsections: doing the weaving in the PIM-PSM transformation, and therefore eliminating aspects from the specific model, or letting aspects be in the specific model in order to undergo weaving at a later stage.

4.1 PSM Without Aspects

We may decide not to have aspects in our platform specific model, therefore the PIM aspects weaving

would be taking place in the transformation from one model to the other.

The main advantage of this proposal is that the obtained PSM does not need to also specify the metamodel of an aspect-oriented platform or language, thus resulting in a simpler model. We would just have to specialise the model to the platform and language used to define the services. Consequently, the specific model would look simpler than if we still had unweaved aspects; this allows the possibility of reusing any process of automatic code generation which may already have been developed for Web Service modelling; hence, we could use any previously implemented tool to generate code from the sequence diagram automatically, which could not be achieved with the second alternative shown in *Section 4.2*.

On the other hand, simplicity is also occasionally a drawback, as by obtaining a simpler model traceability problems arise, that is, we cannot recover aspects at a later stage by separating them from the rest of the elements. Furthermore, although we have no aspects in the specific platform class model, on executing aspects weaving their behaviour must somehow be reflected in the sequence or interaction diagrams. This must be performed in order not to lose dynamic information, which can be sometimes difficult at this level of abstraction.

4.2 PSM With Aspects

Once we decide to also have aspects in our platform specific model, we find two alternatives. The first one is to remain modelling them in a general form, in order to define them in a specific aspect-oriented language in a more refined PSM; the second option is to proceed to the last step straight away.

If the aspects are in the PSM, but they have not been linked to a specific language, we have a more complete model which allows us to maintain our model concerns, which crosscut the system, in a modularized way.

In any of the cases, extra-functional properties may be maintained in the PSM, therefore facilitating traceability and keeping our system well modularized. This fact provides us with the possibility of deciding if we want to use an aspect-oriented language to implement these properties at a future stage or not, depending of the appropriateness of the specific case study in the particular platform. The longer our extra-functional properties are kept modelled in a separate element, the easier it will be to code them in a modularized way, also opening the possibility of reusing them in different parts of the application.

Furthermore, the fact that there are aspects defined in this model in general terms offers us the possibility of using different tools for code generation, in order to obtain the target application for different aspect-oriented languages, thus offering great reusability and a wide range of target applications.

Moreover, in the case of the composition interaction logic modelling with aspects, we can defend the same viewpoint. If we maintain the interaction logic encapsulated in the specific model, it will offer us the possibility of implementing a more modularized application, in which we do not need to couple the different services composed from the earlier design stages. Besides, we do not have problems with traceability as the line between the code and all design models can be traced perfectly.

4.3 PSM Comparative Report and Discussion

We have made a comparative table of the different elements we can find when modelling Web Services within aspect-oriented techniques in three different models: the model that is independent from the platform (PIM), the one that is specific to the platform doing the weaving of the aspects in the transformation (PSM without Aspects) and the model specific to the platform when the aspects have not been weaved at transformation (PSM With Aspects). In the comparative report we have assumed that the transformation is done in *Java* and, when aspects appear, in *AspectJ* language.

As we can see in the table, the stereotypes defined for the main elements in Web Service modelling in the platform independent model, remain in both PSM models, just transformed into the target language, therefore obtaining the *Java Web Service Interface* and *Java Web Service Implementation* stereotypes.

For the aspect stereotypes, it is obvious that in the aspectless PSM they do not appear any more, but become behaviour which would be reflected in the sequence or interaction system diagrams, where the aspects would intercept the methods' execution to provide a new behaviour. When creating the PSM with aspects for the *AspectJ* specific target, the stereotypes are transformed into the new stereotypes defined for *AspectJ* syntax, maintaining the diagram structure.

Table 1. Comparative Table Between PIM, PSM without aspects and PSM with Aspects

Compared Element	PIM	PSM WITHOUT ASPECTS	PSM WITH ASPECTS IN ASPECTJ
WEB SERVICE INTERFACE	Represented by the stereotype <<web service interface >>	Represented by the interface oriented to target language and platform, i.e. <<java web service interface>>	Represented by the interface oriented to the target language and platform, i.e. <<java web service interface >>
JOIN POINT MODEL	*Although* not represented explicitly in the diagram, as mentioned before, it is limited to service method execution and method calls from any element in the model to a service method.	This element would not appear in this model	This element would not appear in this model.
POINTCUTS	Represented by the stereotypes << ApplyExecutionAspect>> and <<ApplyCallAspect>>	This element would become part of the model behaviour representation. It could be represented as the connection point between services and aspects	Represented by stereotypes <<Execution-Pointcut>> and <<CallPointcut>>, where the different methods' execution or invocations would be linked to <<advices>>
POINTCUT RELATED BEHAVIOUR	Represented by the stereotype <<action>>	This element would become part of the model behaviour representation (idem). It would specify aspect behaviour.	Represented by the stereotype <<advice>>, which would be linked to a specific pointcut and would indicate the new behaviour to be injected in the intercepted pointcut.
ASPECTS	Represented by the stereotype <<aspect>>	It would be transformed into part of service specification and into the relations and interaction logic between them.	Represented by the stereotype <<aspect>>, formed by <<advice>> elements, which are bound to the rest of the elements by <<ExecutionPointcut>> and <<CallPointcut>> dependences.
ASPECT-SERVICE LINK	Represented by the pointcut stereotypes <<ApplyExecutionAspect>> and <<ApplyCallAspect>>, which link the action to <<Web Service Interface>>	It would be transformed into part of the behaviours services specification and into the relations and interaction logic between the named services	Represented by the pointcut[s] stereotypes <<ExecutionPointcut>> and <<CallPointcut>>

5. Discussion and Future Work

This paper has shown that we can model Web Services and their compositions by defining new stereotypes in UML. In this sense, we have defined the new necessary stereotypes for aspect inclusion at this stage of the development, in order to maintain our models well designed and our system well modularized.

Once demonstrated the importance of encapsulating Web Service composition code and the usefulness of AOP for this task, we affirm is time to face how to model it whilst maintaining the separation of concerns. As we have seen, our proposal permits Web Service modelling with a standard modelling language such as UML, by defining the necessary stereotypes, as well as aspect ones, regardless of the platform mode. Hence, it has been demonstrated that aspects can be easily modelled in service-oriented systems as extra-functional properties, and composition interaction patterns can also be encapsulated in aspects since modelling stage within the same domain.

We have also proposed two alternatives for the platform specific models, where we could choose between performing the aspects' weaving before or after we create the PSM. In this respect, there is still a lot to be done. As we have previously explained, when the weaving has been done at transformation the PSM is simpler than the one in which aspects remain. Once weaving is finished the aspects are reflected in behaviour diagrams, be it the sequence or the interaction diagram. However, in the PSM where the aspects' weaving has not been done yet it is not that simple as we have to maintain all the elements for the aspect classes and association description, but provides the privilege of allowing perfect traceability between the different models, although they may have undergone transformation. We leave this matter open for discussion in the workshop forum, in order to analyze the advantages and shortcoming of both proposals more in depth.

Once agreed on the best option for weaving, we still have to define all the process rules for automatic transformation from platform-independent into specific models. Transformation rules for translating from the specific model to the target code also have to be analysed. These are important parts of our work in the near future.

*Acknowledgment*s: This work has been developed thanks to the support of CICYT under contract TIC2002-04309-C02-01.

7. References

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

[2] Ortiz, G., Hernández, J., Clemente, P. J. *Decoupling Non-Functional Properties in Web Services: an Aspect-Oriented Approach*. Workshop EOOWS, ECOOP Conference, Oslo, Norway, June 2004

[3] Ortiz G., Hernández J., Clemente, P.J.. *Web Service Orchestration and Interaction Patterns: an Aspect-Oriented Approach*, Short Papers Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC). New York, USA, November 2004.

[4] Ortiz G., Hernández, J. Clemente, P.J. *Building and Reusing Web Service Choreographies by Using Aspect-Oriented Techniques*. Proc. of the WorkShop on Best Practices and Methodologies in Service-oriented Architectures: Paving the Way to Web-services Success at the Object-Oriented programming, Systems, Languages and Applications Conference (OOPSLA), Vancouver, Canada, October 2004

[5] Aldawud, O., Elrad, T., Bader, A. *A UML Profile for Aspect Oriented Modeling*. OOPSLA 2001 Workshop on Aspect Oriented Programming.

[6] Stein, D., Hanenberg, S. and Rainer, U.: *A UML-based Aspect-Oriented Design Notation for AspectJ*. Proc. 1st Int. Conf. on AOSD, Enschede, The Netherlands, 2002

[7] Baniassad, E. Clarke, S. *Theme: An Approach for Aspect-Oriented Analysis and Design*. 26th Int. Conference on Software Engineer, Edinburgh, Scotland, UK, 2004

[8] France, R., Ray, I., Georg, G., Ghosh, S.. *An Aspect-Oriented Approach to Early Design Modeling*. IEEE Proceedings – Software, 2004

[9] Bordbar, B., Staikopoulos, A. *Modelling and Transforming the Behavioural aspects of Web Services.*

[10] Grønmo, R., Solheim, I *Towards Modeling Web Service Composition in UML.* 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure, Porto, Portugal, 2004.

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

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

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

Appendix . Case Study Web Service and Aspect-Based PIM

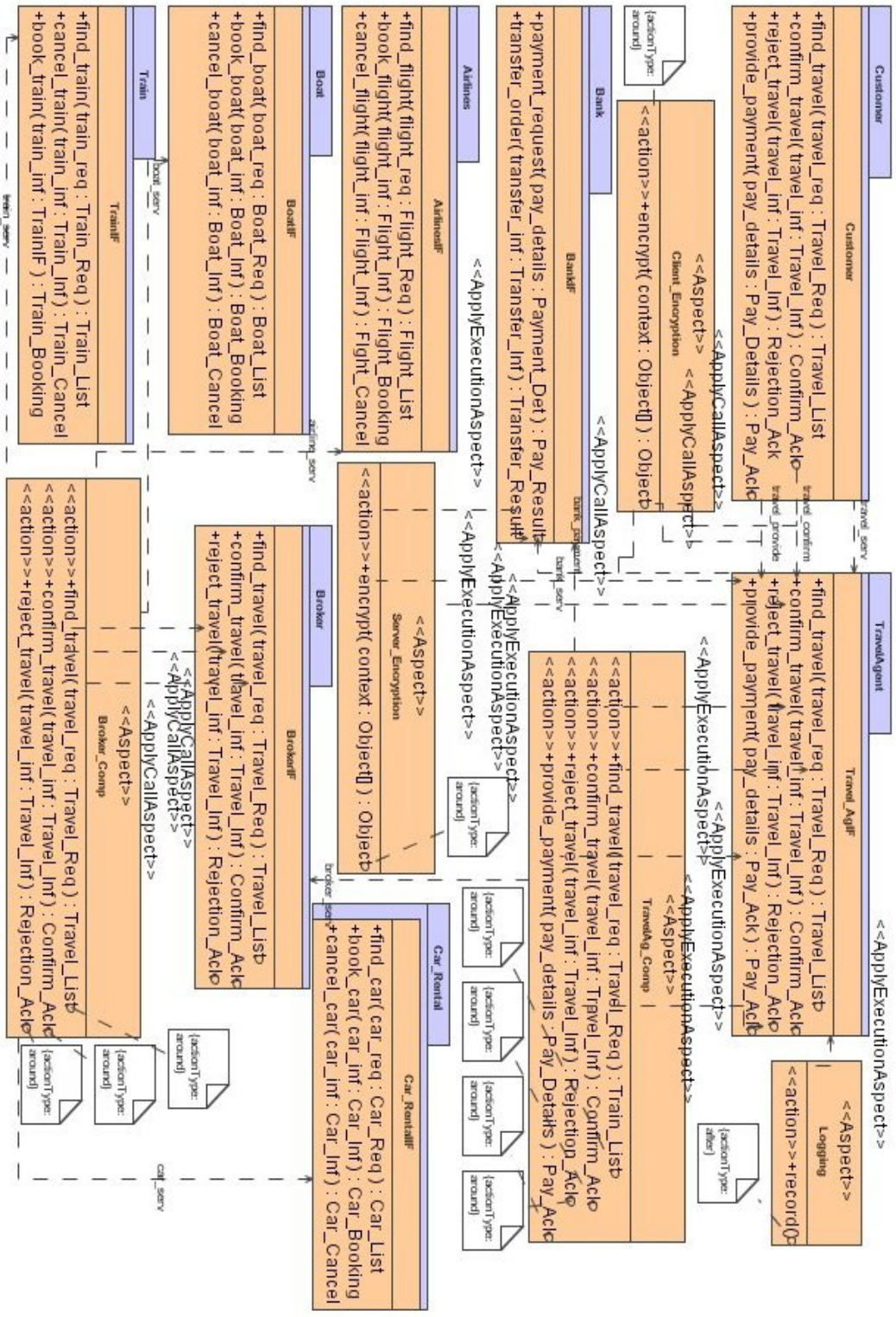

Figure 7. Case Study Web Service and Aspect-Based PIM.