Evaluating the Use of AOP and MDA in Web Service Development

Guadalupe Ortiz
Quercus Software Engineering Group
Centro Universitario de Mérida, UEX
Mérida, Spain
gobellot@unex.es

Behzad Bordbar
School of Computer Science
University of Birmingham
Edgbaston, Birmingham, UK
B.Bordbar@cs.bham.ac.uk

Juan Hernández
Quercus Software Engineering Group
Escuela Politécnica, UEX
Cáceres, Spain
juanher@unex.es

Abstract— Model-Driven Architecture (MDA) is introduced to shorten the software development time, produce better quality of code and promote the reuse of software artifacts. On the other hand, Aspect-Oriented Programming (AOP) is motivated by the need to create decoupled systems, which are easier to maintain. As a result, it can be argued that adopting AOP and MDA side-by-side will provide advantages from both sets of techniques. However, adapting a new technology often entails extra cost and effort, including cost associated with training and support for the software tool. Therefore, it is crucial to evaluate the usefulness of applying such techniques. This paper presents a quantitative approach to evaluate the use of MDA and AOP in service-oriented environments. We shall start by presenting an outline of a method of implementation and maintenance of Web services, based on *both* MDA and AOP. Then, with the help of a case study we shall evaluate the advantages and disadvantages of applying the method, achieved by comparing two implementations of a prototype University Administration system; the first implementation is based on ad-hoc methods of Web service development, whereas the second implementation is carried out by applying MDA and AOP. We shall use various metrics to report on the maintainability, performance, percentage of generated and reused code resulting from the use of MDA and AOP.

Keywords: Aspect-Oriented Techniques, Web Services, Model-driven Development, Extra-Functional Property

I. INTRODUCTION

Modern businesses are increasingly adopting *Service-Oriented Architecture* (SOA) to become responsive to the rapid changes in the market. When adapting a service-oriented infrastructure to support business ideas, speed is of crucial importance. As a result, the application of techniques which aim at developing better quality software in a faster development time, whilst being easy to maintain, such as *Model-Driven Architecture* (MDA) and *Aspect-Oriented Programming* (AOP) have received considerable attention [4, 8, 12].

MDA [27] aims to promote the role of *models* in software development processes. Models in MDA are captured in MOF-compliant languages such as UML [26]. Central to MDA is the use of model transformation frameworks [1, 23], which allow the automatic generation of various software artefacts, such as code. Applying MDA techniques results in better quality of code created via automated techniques and reuse of software artefacts. This is expected to result in shorter development cycles and reduction of costs [12, 20].

During multiple development cycles, functionalities such as creating log files, encryption or login by users, are often incorporated to complement the original system functionality. We refer to such properties as *Extra-Functional Properties* (EFPs) [14, 15]. Implementing EFPs is a programming-intensive task with high maintenance cost, which often requires integrating related code into the system code for each property. Aspect-oriented programming can alleviate the burden of EFP implementation by allowing encapsulation and modularization of EFPs as *aspects* and by *weaving* them into the original implementation. Thus, using AOP is expected to improve the low coupling of service systems and reduce maintenance costs [10].

Considering the advantages of using AOP and MDA, it is expected that the use of both methods simultaneously will benefit from the two approaches. This paper adopts a numerical approach to the evaluation of the effect of using them in the implementation when implementing EFPs for Web services. We shall report on a case study of a Web service application development using MDA and AOP and we shall compare the result with conventional development methods. To do so, we have applied the method introduced by Ortiz prototype [14] in order to integrate EFPs in Web service-based systems. By applying performance analysis techniques and well-know metrics [7, 11, 22], the advantages and disadvantages of MDA and AOP are explored. We shall report on our case study to confirm that using AOP in conjunction with MDA may result in over 45% automated generated code which is well structured and modularized.

The paper is organised as follows. Section 2 provides an introduction to EFP in Web service development, MDA and AOP. In this section we shall also describe the outline of the method adopted in the development and maintenance of Web services based on [14]. Section 3 states the problem to be addressed in this paper. Then, we shall describe our case study in Section 4. Section 5 introduces the evaluation methodology, which has been followed in order to get the results shown in Section 6. Section 7 discusses the presented evaluation, whereas conclusions are summarized in Section 8.

II. PRELIMINARIES

Our research brings together three transversal areas of software engineering: extra-functional properties in Web service development, MDA and AOP. In this section we shall briefly

review these three, followed by an outline of a method for implementing EFPs with MDA and AOP [14].

A. Extra-functional properties in Web service development

In this paper, we are dealing with the modification of Web service implementations [5, 6] to incorporate extra-functional properties. The term "extra-functional", also called "*Non-functional"* is used in various contexts [2, 3, 9, 18]. The term "non-functional" can be slightly confusing, as it can be argued that non-functional properties, such as security, are indeed related to the system functionality. In this paper, extra-functional properties describe properties which are implemented as pieces of code to complement the main system functionality. For example, consider an online banking system: its main functionality is to provide a portal in which customers can manage their accounts. In this case, security is an EFP, as it complements the system's main functionality. Notice that an EFP such as security can be essential for the system; however in the case of a banking system encrypting the invocations is not the main functionality.

EFPs are sometimes referred to as *functional aspects*; however in order to avoid a possible misunderstanding between the functional aspect (the property itself) and aspect implementation (an option for property implementation with AOP) we avoid using the term *functional properties* in our approach. EFPS are sometimes called *policies*; we have decided against using this term to avoid confusion between the whole property and the property description by using WS-Policy.

B. Model-Driven Architecture

Model-driven development promotes the role of models, allowing us to focus on the essential aspects of the system, delaying the decision of the implementation technology for a later step. In model-driven development multiple models are used, where each will address one concern, independently of the remaining issues involved in the system's development; thus allowing the separation of the final implementation technology from the business logic achieved by the system. MDA [27] models are generally divided into three categories: *Platform-Independent Models* (PIM) representing the system without coupling it to any specific platform or language, *Platform-Specific Models* (PSM) expressing the system based on a specific platform, technology and programming languages, and finally, *Code Layer* provides the final application as code. A set of transformation rules may also be created in order to transform PIMs into PSMs and the latter into the final application code automatically [12, 20, 27].

C. Aspect-Oriented Programming

In many systems we may find it impossible to model several concerns into a structured decomposition of units by only using the *Object-Oriented Programming* (OOP) paradigm. For instance, in a system for the representation of geometric figures, we may have two different concerns: representing the type of figure and tracking its movement. AOP allows us to modularise these crosscutting concerns by encapsulating them into meaningful independent units called *Aspects* [13]. Afterwards, a method to weave the aspect code with the original one is applied [10].

D. Applying MDA and AOP to the implementation of EFPs

Figure 1 depicts the outline of the methodology presented by Ortiz [14] for integrating extra-functional properties into Web service model-driven development. "Model of the system" represents the platform-independent model of our system. Using an UML profile [17], EFPs such as *security* can be expressed as PIMs, denoted in Figure 1 by EFP1, EFP2, etc. The overall model consists of the original PIM "Model of the system" and a few models representing the EFPs.

Code creation involves three stages:

1. Generation of the code corresponding to the main functionality of the system, as depicted by a vertical arrow from the "Model of the system" to the "code of the system". This involves definition of *model transformations* from the PIM to PSM and then, from the PSM to code. After defining the model transformations, an MDA model transformation framework can execute the transformation to generate the final code.
2. Creation of snippets of code corresponding to extra-functional properties. This is represented in the Figure by several arrows from "EFP_i" to the "code for EFP_i". Similar to step 1, this transformation process is also conducted through the use of the MDA framework.
3. Finally, once the "code for the system" and the "code for EFPs" are available, we shall use an aspect compiler, such as AspectJ, in order to weave them [16].

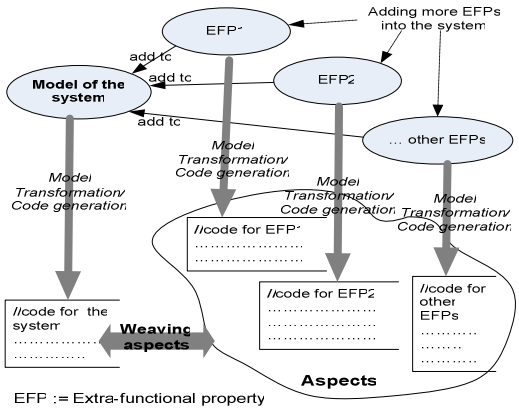

Figure 1. Using MDA and AOP for implanting EFPs

Implementation: As explained in details in Ortiz [14], a UML profile was used in order to model extra-functional properties. Each property is represented as a UML *stereotype* in the platform-independent model and can be used to stereotype the service interfaces or operations to which the property should be applied. We have used JAX-RPC metamodel for platform-specific modelling for the services. Moreover, an aspect-oriented metamodel and a policy-based one are supplied in order to model, at PSM level, the functionality provided by the properties and their description, respectively. Finally, A SOAP tag-based platform-specific metamodel is provided in order to model the new tags to be included in the SOAP messages for properties which require additional information to be supplied transparently, respectively. Platform-specific models are obtained automatically by the application of the provided PIM-PSM transformation rules to the platform-independent model.

An additional set of transformation rules allows us to obtain automatically the service skeleton code on the one hand, and, on the other, the AspectJ for the implementation of the property functionality, WS-Policy [2] code for property description and Java code to implement a SOAP handler. This handler affords the inclusion of new tags in the SOAP message headers in the client side, or provides the code which allows the tags to be checked in the service side. For further information we refer the reader to previous work on the matter [14, 15, 16, 17].

III. DESCRIPTION OF THE PROBLEM

The assessment process involves evaluating the use of both MDA and AOP techniques. To the best of our knowledge there is no experimental evaluation used for MDA. There are various approaches [7, 11, 22] to the evaluation of the use and application of aspect-oriented techniques in different types of system. However, none of them are applied to model-driven Web service-based systems. We aim to study the following:

- The structure of the system's code and the dependences of EFPs' code from the main functionality one will be evaluated to ensure better system maintenance and evolution.
- The percentage of generated code, to evaluate the effectiveness of using MDA techniques.
- The performance of the automated generated code.

IV. UNIVERSITY ADMINISTRATION SERVICES: A CASE STUDY

Our case study consists of five Web services, which can be used in an ordinary Spanish University, such as the *Centro Universitario de Mérida* by students and course administrators: *PreregistrationService, RegistrationService*, *ExamOpportunityService, AcademicResultsService* and *TeacherService*. We have also designed a user-friendly interface for students to access these services. Both services and client are described in detail in [14]. The case study aims to integrate the following properties into the above services:

- *log*: producing log files is essential to ensure the possibility of tracking all the relevant invocations in the system. As a result, this property has been applied to the interface offered by the registration service and to the operation *bringForwardExam* in the *ExamOpportunity* service.
- *detailedInfo*: used in case a user requires extra information. For example, user may require to bring forward the date of an examination via *ExamOpportunityService*. He may also wish to get additional information on the exam room. In such a case, a *detailedInfo*-type EFP must be incorporated.
- *decryption*: security is an important EFP. In our case study, we considered it and, as a result, invocations to *sendPDF* will have to be encrypted compulsorily.

For the client side, the following must be taken into account:

- First of all, requests for the pre-registration pdf file to be re-sent in *preregistrationService* need to be encrypted.
- Secondly, the *detailedInfo* application is optional, so it may or may not be selected by the client. Our client will select the property to be applied on his *ExamOpportunityService* invocations to bring forward an exam.
- Finally, since the *log* property is client-independent, it does not have to be taken into account in the client side.

V. ADOPTED METHODOLOGY FOR THE EVALUATION

To evaluate the use of MDA and AOP we have applied the method described in section 2.4 in order to create an implementation of the case study EFPs using MDA and AOP. We have also created a hard-coded implementation of the system directly by including the implementation of each EFP into the system code. For example, to implement the EFP for *log*, we have included the relevant code in every single place of the system code where *log* is required. Therefore we have introduced the property code, which was encapsulated in the aspect, within the original main functionality code; the process to be followed for the remaining properties would be analogous. Then, the two created systems are compared.

In our model-driven development we are generating three types of code: the aspect-oriented one, the policy one and the SOAP tag-based one. Then we have to determine what type of evaluation is necessary:

- Concerning the policy code, we are simply generating XML code to describe the properties according to a proposed standard. It is known that XML is being used to provide a homogeneous and neutral description for Web services, therefore there is no question as to why to use XML instead of other possible description types at code level.
- Regarding the handlers created to add new SOAP tags to the message header in the client side and check their value in the service side, they are implemented in Java, since this is the final implementation of the system. Thus, there is no need to evaluate the code itself; moreover, as previously mentioned, the use of the SOAP header to provide information related to EFPs or to services' management is common practice [14, 19, 21], particularly if we intend not to include any intrusive code in the main functionality one [16].
- However, aspect-oriented programming may lead to some overhead in the applications' performance. This belief is probably originated by the first AOP proposals; nevertheless, AOP weavers, and specifically AspectJ ones, have evolved considerably and the latter community's aims for the performance of their implementation of AspectJ to be on par with the same functionality hand-coded in Java. In spite of this assertion, we are going to measure the performance of our aspect-oriented code to show how it does not suppose an overhead for the system. Furthermore, modularity, coupling and some more aspect-related properties of the system will also be measured.

To evaluate the use of MDA, we must measure *how much code is generated automatically* and how much is still necessary to complete the system's behaviour. This way, we are able to measure how much effort we have saved the system developer by some of the code being generated automatically. Hence, to make a comparison, we have to consider both the effect of using MDA and AOP techniques. Tables 1 and 2 describe the metrics that we have used to conduct the comparison. We have classified the measurement criteria in the tables into the following categories:

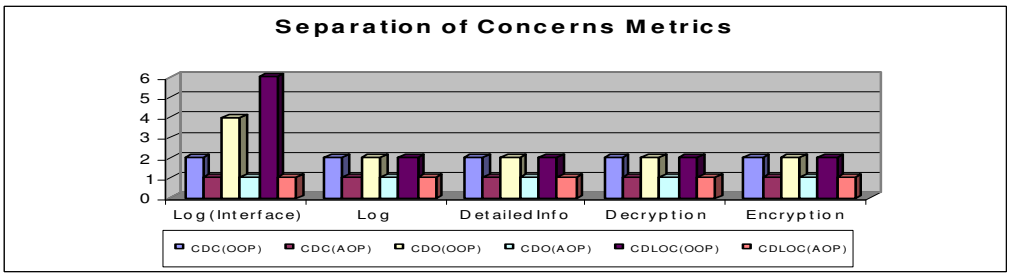

Figure 2. Separation of concerns representation.

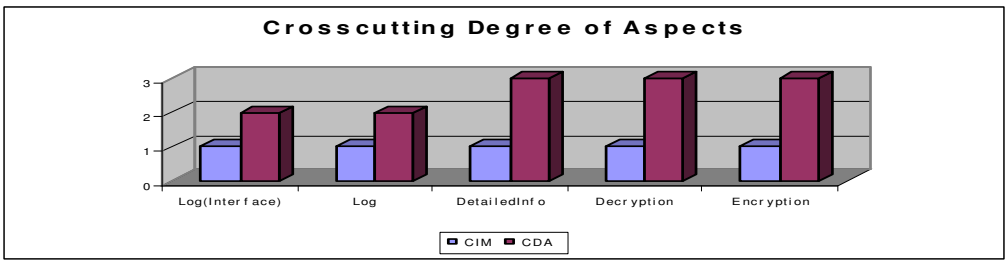

Figure 3. Crosscutting degree representation.

- **Separation of concerns metrics** (CDC, CDO, CDLOC) measure the degree to which a single concern is fulfilled in the system components (classes and aspects), operations (methods and advices), and lines of code. A low value for this metric indicates better system modularization.
- **Coupling** (CIM) measures the strength of dependences between the elements in the system. Lower coupling values imply better system modularization.
- **Crosscutting degree of aspects** (CDA) measures how much the aspect-oriented implementation encapsulates a functionality which may affect modules of the system.
- **Software size** is an essential measurement criterion. LOC will be used to obtain the total number of lines in both methods of implementation.
- **The ciclomatic complexity** number will be used to measure whether the aspect-oriented implementation increases or decreases the complexity of our system.
- **Performance** is used to evaluate the response time of the system to see whether it is affected by the aspect-oriented implementation in the performance or not.

TABLE 1. METRIC USED TO EVALUATE IMPLEMENTATION OF EFPS VIA ASPECTS.

Metric	Description
Concern Diffusion over Components (CDC)	Number of components in which there is code related to the implementation of the concern in question [11].
Concern Diffusion over Operations (CDO)	Number of operations in which there is code related to the implementation of the concern in question [11].
Concern Diffusion over Lines of Code (CDLOC)	Number of switches of concern through the lines of code [11].
Coupling on Intercepted Modules (CIM)	Number of modules named in the pointcut of a specific aspect [7].
Crosscutting Degree of Aspects (CDA)	Number of modules which may be affected by an aspect [7].
Lines of code (LOC)	Lines of code in the system's implementation [7, 11].
Ciclomatic Complexity Number (CCN)	Possible execution paths to be followed caused by control flow statements [22].
Performance	Response time of the system [22].

TABLE 2. CRITERIA FOR EVALUATING THE USE OF MDA FOR ASPECTS GENERATION.

Metric	Description
Percentage of automated service implementation	Measure the percentage of code generated for services implementation.
Percentage of automated property implementation	Measure the percentage of code generated for property functionality implementation.
Percentage of automated property description	Measure the percentage of code generated for property description.
Percentage of automated property selection	Measure the percentage of code generated for optional property selection or when additional information is required.

We classified the measurement criteria in the tables into the following categories:

- **Separation of concerns metrics** (CDC, CDO, CDLOC) measure the degree to which a single concern is fulfilled in the system components (classes and aspects), operations (methods and advices), and lines of code. A smaller number of elements affected by a concern implies better system modularization.
- **Coupling** (CIM) measures the strength of dependences between the elements in the system. Lower coupling values imply better modularized of the system.
- **Crosscutting degree of aspects** (CDA) measures how much the aspect-oriented implementation encapsulates a functionality which may affect various modules in our system.
- **Software size** is an essential measurement criterion. LOC will be used to obtain the total number of lines of both approaches.
- **Ciclomatic complexity** number will be used to measure whether the aspect-oriented implementation increases or decreases the complexity of our system.
- **Performance** is used to evaluate the response time of the system to see whether it is affected by the aspect-oriented implementation in the performance or not.

VI. MEASUREMENT RESULTS FOR THE CASE STUDY

This section presents the outcome of the case study; all measurements are done via JavaNCSS [24] and JDepend [25].

A. Aspect-Oriented Metrics

- **Separation of Concerns**. Figure 2 depicts the separation of concerns metrics comparing the values for the adhoc and aspect-oriented implementations of the case study. It can be seen that concern diffusions are higher in the hardcoded implementation compared to the implementation according to our method. Specifically, diffusion over classes, operations and lines of code is slower when using AOP (i.e. *Log* concern appears in 2 classes in the adhoc implementation compared to 1 class in the AOP). This is because properties are encapsulated, avoiding references from the main implementation class of the service to the side classes which implement the EOP. Therefore, our measurement confirms that using an AOP implementation provides a better separation of concerns.
- **Coupling and Crosscutting Degree**. Figure 3 depicts the results related to crosscutting in the aspect-oriented implementation. The results related to coupling, which are not presented due to space limitations, are the same in both implementations. This is because a pointcut implies coupling to the target method (CAE). However, although coupling is the same for both AOP and OOP implementations, its direction is different: the main system functionality is dependent on the extra-functional property in the OOP implementation; however, when using AOP, the dependence source is in the property itself, therefore avoiding any intrusive code mixed with the system's main functionality. Furthermore, the measurements in Figure 3 show that coupling to intercepted modules is very low (one per aspect). This implies low coupling of the aspect regarding the application and therefore high aspect code reusability. Moreover, CDA measurements indicate that the aspects in the system indirectly affect a few more classes than those referred to in the pointcuts. Therefore, with regard to coupling we can conclude that low CIM values and higher CDA values show low coupling and good crosscutting modularization of the system.
- **Software Size**. Our measurement has revealed that both implementations produce roughly the same size systems. The property code is located in the aspect in the AOP implementation, while scattered over the application.
- **Ciclomatic Complexity**. This metric measurement maintains its value for both implementations in the case of *decryption*, *encryption* and the interface *log*. However its value is lower in the aspect-oriented implementation when *detailedInfo* is applied. This is due to the optional nature of the property. If no aspects are included, optional properties will result in additional complexity. Since complexity remains in the aspect in the AOP implementation, the system's main code remains unchanged.
- **Performance**. To establish the existence of any overhead attributed to the adoption of aspects, we have measured the invocations to the operations in *registrationService* (*Log*), to *bringForwardExam* in *OpportunityExamService* (*Log* and *detailedInfo*) and to *sendPDF* in *preRegistrationService* (the invocation is *encrypted* in the client and *decrypted* in the service). Measurements are carried out on an Intel Pentium Processor at 1.5 GHz with 1GHz RAM. The services have been deployed on the machine, which also contains the SQL database server and the client. All invocations have been made through the localhost. This will not affect our result, as we can assume that the effect of the net would reduce the difference between response times in both implementations. The invocations to services have been made from 1000 to 10000 times, in intervals of 1000. The average response time for one invocation to each service is calculated as the weighted arithmetic of all obtained response times –in thousands- from 1000 to 10000 executions as shown in Table 3.

TABLE 3. AVERAGE RESPONSE TIME (MS).

(Average)	log (Interface)	log + detailedInfo	encryption+ Decryption
OO	46.64	6041.26	33.27
AO	43.51	6384.98	32.28
AOP Penalty (%)	-6.7	5	-0.97

In order to measure response times for *RegistrationService*, we have taken into account the three operations in the interface since *Log* is applied to all of them. Hence, every measured time corresponds to one invocation for each service operation. As shown in the last row of Table 3 the aspect-oriented implementation improves the average response time by 6.7%. The differences between the execution rates in the object-oriented implementation and the aspect-oriented one are below 10%, which in general can be regarded as insignificant [11].

B. Model-Driven Development Measurements

We have also measured the percentage of automatically generated code to implement various properties on the services. The percentage of generated code for the services fluctuates around 40%. We need to take into account that this percentage of generated code includes the configuration, compiling and deployment files; thus for larger services the percentage might decrease. We have also measured the number of code lines of property implementation and description (not shown due to space restrictions) separately. In the case of *Log* we have 100% of code generation, due to the fact that it is a well-known property to the system and we can generate the full code from the properties repository. For *detailedInfo* we also get a high percentage of generated code (28.57%); however for *decryption* we obtain a low rate (14.8%). For every property the aspect skeleton is generated.

Regarding the policy description the percentage of generated code is very high in all cases. This will also depend on the complexity of the property: domain-specific properties may not require too many specific data to show and thus these are the ones where we get a higher rate of generated code. On the contrary, common properties for which established description standards are provided (such as encryption) may imply a more detailed and complex description. In this regard, the more complex the information to be provided is, the lower rate of generated code we will get in the policy description.

Finally, 100% of the code necessary to include optional properties in the client, plus the one that checks whether optional properties are included in the service are generated, as well as 100% of code needed to add new information to the client SOAP header or to retrieve it from the service.

VII. DISCUSSION

The main focus of this paper is to use numerical metrics to evaluate AOP and MDA. Further advantages of using AOP and MDA that we come across are as follows.

- Modularity: using the approach outlined in section 2.4, the implementation of various properties remains separated from the implementation of the main functionality.
- Encapsulation: as a consequence of the system's modularity, various properties are implemented in an encapsulated way, enhancing re-usability.
- Traceability: also a consequence of the previous characteristics, systems traceability is maintained along the development process since any property located in a stereotype in the PIM is accurately located in an aspect in the aspect-oriented PSM, in a policy in the policy-based PSM and in a SOAP tag in the SOAP-tag-based PSM, when necessary. Moreover, these PSM elements are totally located in an AspectJ aspect, a WS-Policy description and a Java SOAP handler in the code, respectively. This process path can also be followed reversely, from code to PIM. Therefore our properties are completely traceable through all stages of the development process.
- Simplicity: in our experience, the transformation in the model-driven development is simpler if the properties are not mixed with the main services, since independent elements can be generated separately.
- Maintainability: since properties are separated from the main functionality, it is easier to add a new one or to delete or modify existing ones without affecting the main service functionality at all.

There are additional metrics regarding AOP which could have been used, such as Coupling Between Components (CBC), Deep of Inheritance Tree (DIT) or Lack of Cohesion in Operations (LCOO) [7, 11, 22]. These metrics have not been considered in this paper, since they are not relevant to the implementation of EFPs in the Web services scope.

VIII. CONCLUSIONS AND FUTURE WORK

This paper reports on a case study for evaluating the use of MDA and AOP techniques in the implementation of extra-functional properties for Web services. The aim of the paper is to show the advantages of such techniques by using well-established measurement metrics. The results obtained with the applied metrics have provided us with the information required in the problem statement: the measurements indicate that using AOP results in better separation of concerns and coupling, while the decline in the performance of the generated code resulted from the use of aspects is negligible. Using MDA techniques results in substantial saving of resources and a noticeable percentage of automated generated code. In this sense, we have observed that the system generated by using AOP and MDA techniques is modular, encapsulated and traceable. Moreover, using aspects also results in simpler MDA transformations, as the latter are defined on sub-modules of the system which are simpler in structure.

It is part of our future goals to generate the full code for a set of predefined properties, so that no functionality code would need to be added by the developer and better evaluation would be obtained from the MDA perspective. Another area for future work is to include properties dynamically in order to provide the possibility of adding properties to a deployed system. This would imply using dynamic aspect-oriented techniques and therefore an additional evaluation for performance would be necessary.

IX. ACKNOWLEDGEMENTS

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

REFERENCES

[1] Akehurst, D.H. Boardbar, B., Evans, M., Howells, W.G.J., McDonald-Maier, K.D. SiTra: Simple Transformations in Java .Proc. ACM/IEEE I.C. on Model Driven Engineering Languages and Systems, 2006

[2] 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.

[3] Baresi, L. Guinea, S. Plebani, P. WS-Policy for Service Monitoring. VLDB Workshop on Technologies for E-Services, Norway, 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] Cauldwell, P., Chawla, R., Chopra, V., Damschen, G., Dix, et al.: XML Web Services. Wrox Press (2001)

[6] Alonso, G., Casati, F. Kuno, H. and. Machiraju, V, Web services-Concepts, Architectures and Applications, Springer-Verlag, 2004

[7] Ceccato M., Tonella, P. Measuring the Effects of Software Aspectization. Workshop on Aspect Reverse Engineering at the Working Conference on Reverse Engineering Delft, The Netherlands, November 2004.

[8] Charfi, A., Mezini, M., Using Aspects for Security Engineering of Web Service Compositions, I.C. on Web Services, Orlando, Florida,, 2005.

[9] Duclos, F., Estublier, J., Morat, P.: Describing and using non functional aspects in component based applications. I. C. on Aspect-oriented software development, ACM Press, Enschede, The Netherlands (2002)

[10] Elrad, T., Aksit, M., Kitzales, G., Lieberherr, K., Ossher, H.: Discussing Aspects of AOP. Communications of the ACM, V44 No. 10, October 2001

[11] García, A., Sant 'Anna C, Figuereido E.m Uirá K., Lucena, C., von Staa A.. Modularizing Design Patterns with Aspects: A Quantitative Study.Transactions on AOSD I. LNCS 3880 pp 36-74, 2006.

[12] Grønmo, R. Skogan, D. Solheim, I. Oldevik, Jon Model-driven Web Services Development. 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-04), Taipei, Taiwan 2004

[13] Kiczales, G. Aspect-Oriented Programming, ECOOP'97 Conference proceedings, LNCS 1241, June 1997.

[14] Ortiz G. Integrating Extra-Functional Properties in Model-Driven Web Service Development. PhD Thesis, University of Extremadura, April 2007.

[15] Ortiz, G. Hernández, J. Clemente, P.J.: How to Deal with Non-Functional Properties in Web Service Development, I.C. on Web Engineering, Sydney, Australia, 2005.

[16] Ortiz, G., Leymann, Ff. Combining WS-Policy and Aspect-Oriented Programming. I.C. on Internet and Web Applications and Services. IEEE Computer Society Press 0-7695-2522-9/06 2006.

[17] Ortiz, G., Hernandez J. Toward UML profiles for Web Services and their Extra-Functional Properties. IEEE I C on Web Services, Chicago, 2006.

[18] Röttger, S. Zschaler, S. Model-Driven Development for Non-functional Properties: Refinement through Model Transformation. LNCS 3273, 2004.

[19]Saltz R., XML.com 2002, http://www.xml.com/pub/a/ws/2002/07/17/salz.html.

[20] Smith, M., Friese, T. Freisbelen, B. Model Driven Development of Service-Oriented Grid Applications. I.C. on Internet and Web Applications and Services. IEEE Computer Society Press 0-7695-2522-9/06 2006.

[21] Verheecke, B., Vanderperren, W, Jonckers, V. Unraveling Crosscutting Concersn in Web Services Middleware. IEEE Software, V 23 Iss. 1, 2006.

[22] Zhang C. Hans-Arno, J. Quantifying aspects in middleware platforms. I. C. on Aspect-Oriented Software Development. Boston, 2003.

[23] ATL http://www.inria.fr/rapportsactivite/RA2006/atlas/uid15.html

[24] JavaNCSS http://www.kclee.de/clemens/java/javancss/

[25] JDepend http://www.clarkware.com/software/JDepend.html

[26] Meta-Object Facility http://www.omg.org/mof/

[27] Model Driven Architecture. http://www.omg.org/mda