A PROPOSAL FOR HANDLING NON-FUNCTIONAL ASPECTS WITH A MODEL-DRIVEN ENGINEERING APPROACH

UNA PROPUESTA PARA MANEJAR ASPECTOS NO FUNCIONALES CON UN ENFOQUE DE INGENIERÍA DIRIGIDA POR MODELOS

DENISSE MUÑANTE

M.Sc, Universidad Nacional Mayor de San Marcos, UNMSM, Lima, Peru, denisseyessica.munantearzapalo@univ-pau.fr

PHILIPPE ANIORTE

Ph.D, Laboratoire d'Informatique de l'Université de Pau et des Pays de L'Adour, LIUPPA, Pau, France

Received to review March 09th, 2011, accepted January 27th, 2012, final version May 18th, 2012

ABSTRACT: Information systems (ISs) are composed of functional requirements (FRs) and non-functional requirements (NFRs). An NFR does not determine the function of the system itself, but the quality characteristics of an IS; for example, error handling, auditing, and access control. Non-functional requirements are often included in the coding phase of the IS, and these generally are present in various parts of the source code (i.e., they are scattered and tangled), which implies a difficult concept and even more difficult maintenance. In addition, we know that maintenance works are becoming more frequent due to both the technological and the functional changes of the IS. In this paper we present a proposal to define and include the NFR in the early stages of the analysis and the design of IS development. On the one hand, we use the aspect-oriented software development approach (AOSD) to model and maintain the NFRs as aspects. On the other hand, we use the model-driven engineering approach (MDE) to formalize this approach. For this, we create a unified modeling language (UML) profile. Then, we make use of MDE transformation mechanisms to obtain the complete model (with functional and non-functional aspects), and finally, a source code is generated; but this step is beyond the scope of this paper.

KEYWORDS: meta-modeling, aspect-oriented software development, model-driven engineering, UML profiles, weaving methods.

RESUMEN: Los sistema de información (SIs) están conformados por requerimientos funcionales (RFs) y requerimientos no funcionales (RNFs). Un RNF no determina una función del sistema en sí, sino encapsula una característica de un SI; por ejemplo, el manejo de errores, la auditoria y el control de acceso. A menudo los RNFs son incluidos en la fase de codificación del SI, y estos, por lo general, están presentes en diversas partes del código fuente; (i.e., están dispersos y enmarañados), lo cual implica una difícil concepción y mas aún un difícil mantenimiento. Por otro lado, sabemos que las labores de mantenimiento se hacen cada vez mas frecuentes debido tanto a los cambios tecnológicos como a los cambios funcionales del propio SI. En este articulo presentamos una propuesta para definir e incluir al RNF en fases tempranas de análisis y de diseño en el desarrollo de un SI. Por un lado, usamos el enfoque de desarrollo de software orientado a aspectos (AOSD según su acrónimo en inglés) para modelizar los RNFs como aspectos y facilitar las labores de mantenimiento. Por otro lado, utilizamos el enfoque de ingeniería dirigida por modelos (MDE según su acrónimo en inglés) para formalizar esta propuesta. Para esto creamos un perfil del lenguaje de modelado unificado (UML según su acrónimo en inglés). Luego, haciendo uso de mecanismos de transformación de MDE obtenemos el modelo completo (con los aspectos funcionales y no funcionales), el que finalmente se derivará al código fuente, pero este último paso está fuera del alcance de este artículo.

PALABRAS CLAVE: meta-modelamiento, desarrollo de software orientado a aspectos, ingeniería dirigida por modelos, perfiles UML, métodos de entrelazado.

1. INTRODUCTION

Today's information systems (ISs) support a variety of human fields such as medicine, commerce, learning, etc., where the ISs manage information, processes, and work-flows that support human activity. Therefore, it is important to require the quality of the systems. The quality properties of an IS are represented by non-functional requirements (NFRs) [13].

In the conception and elicitation of an IS, we obtain the functional (FRs) and non-functional requirements (NFRs). The first ones represent functions offered by the system, while the latter, the objective of our work, provide the quality attributes to the first, which will make the IS robust enough to deal with external problems. In many cases, the quality attributes are unnoticed by final users, but its absence often has negative impacts. These quality attributes could be: confidentiality, security, interoperability, maintainability, re-usability, verifiability, usability, or others [3].

Generally an NFR affects various parts of an application, being scattered and tangled, making it difficult to understand and maintain the system. The aspect-oriented approach (AOSD) deals with this phenomenon called cross-cutting concerns. AOSD focuses on the identification, specification, and representation of cross-cutting concerns in aspects [17]. An *aspect* is a modularized and isolated cross-cutting concern [3,9] for improving the understandability, maintainability, and re-usability of the system [10]. Due to these advantages, AOSD has been adopted by many researchers; so works on legacy, business, financial, distribution, and service systems [1,2,6,19] in analysis [12,13], design [4,6,14,19,22], development [1,3,18] and architectural [11,17] phases can be found.

Non-functional requirements can be elicited in early software development [12,13,17], which allows us to take them into account at the right moment in the IS. We believe that the NFR conception has to be included in analysis and design phases; in order to do this, we have to use formal models. The model-driven engineering approach (MDE) systematically uses models as first class entities for all stages in the life cycle of software development [5]. An MDE case study to derive from a unified modeling language (UML) class diagram for Oracle9i entities is shown in [25]. The works [2,4,6,12,13,17] use MDE to formalize AOSD concepts with a static behaviour, while the proposals [14,19,22] have a dynamic behaviour which is closer to reality.

We present a proposal for handling NFRs in the design phase of software development, using the AOSD principles which are formalized by an MDE approach. Our proposal is the so-called *model-driven aspects design* (DADM, according to its acronym in Spanish). In the literature, there are similar approaches [3,4,6,11,14,17,19,22,24], but none of them allows us to model NFRs into design business models with a dynamic behaviour (sequence diagrams). This is why we create a UML profile as a meta-model. Meta modeling is a technique which defines the abstract syntax of models and relationships among its elements. For example, the UML meta model is defined in MOF of which the model exchange format is XMI [5], to define aspect models. The models represent the system requirements called *base models*, while those created from our proposal are called *aspect models.* Both models are defined separately; we use a weaving process to integrate them. The authors of [5,15,18] study *model to text* (M2T) transformation, while proposals [9,10,22] present research on *model to model* (M2M) transformation, which are close to our objective.

The remainder of this paper is organised as follows: In Section 2, we present our proposal DADM, Section 3 shows a case study to implement our work. The related works and conclusions are detailed in Sections 4 and 5, respectively.

2. MODEL-DRIVEN ASPECTS DESIGN (DADM)

There are three types of aspect-oriented approaches. This classification depends on the level of the weaving process: We have model-level, code-level, and executable-level weaving processes [4]. We consider that the first one has two major advantages: i) It is platform-independent; it does not depend on an aspect-oriented language (AspectJ, AspectWerkz, Aspect#, Aspyct, etc. are shown in http://www.aosd.net/) for its definition. Learning these languages takes time and is costly; moreover, they have performance problems [21]. ii) It is better understood by people with basic analysis and design knowledge. Non-functional requirements must be defined by architectural experts in the deployment and implementation of systems, but analysts and designers know precisely where and when to use them.

Therefore, we present our proposal, the so-called model-driven aspects design, using an aspect-oriented approach which is based on a model-level weaving process. Figure 1 shows the DADM overview. On the one hand, we have the *base model* (analysis and design diagrams) corresponding to FRs; that is, the reason of this system; and on the other hand, we have the *aspect model* (extended analysis and design diagrams) corresponding to NFRs. Both models are isolated which means that when one of them changes, it does not have an impact on the other. If an aspect needs to be updated, then we will update it only once; the combination of aspect and base models is an automatic and non-error prone process; because of this, we use a *weaving process* of which the final product is a *weaved model (base model +*

aspect model). Finally, we use a transformation process to obtain a source code and an executable.

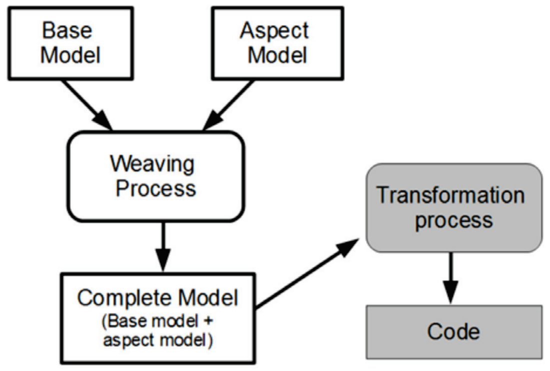

Figure 1. DADM overview

Our work uses the MDE principles to formalise the aspect-oriented concepts using models as primary entities. Therefore, *we create a meta-model* that provides a modeling language to represent aspects. Then, *we define the aspect models* based on the meta-model mentioned. Finally, *we create a weaving process* to combine base models (FRs) and aspect models (NFRs). An additional stage would be to create a transformation process, but this step is beyond our scope, as mentioned. These steps are derived from the ones presented in http://openembedd.inria.fr. OpenEmbeDD is an Eclipse-based MDE platform that supports each stage mentioned.

We detail each stage as follows:

2.1 Creating a meta-model

We use UML because it is the most widely used and it is easier for analysts and designers. A UML does not support the aspect definition (http://www.uml.org/), this is why we create a profile to add the aspect-oriented concepts. Then, we create and modify UML diagrams according to aspect concepts. These elements are built on OpenEmbeDD (see http://www.topcased.org/ for more details).

2.1.1 Join points

The join points are where the concerns cross-cut system modules. Scattered, tangled, or crosscutting concerns are used for many system modules; therefore, a crosscutting concern has one or more joining points [9]. In DADM, this concept can be represented by the use case diagrams of a UML, where a use case corresponds to the "module" or the process of a system. In order to do this, DADM adds two elements: i) The *aspect use cases* that describe crosscutting concerns and describe the NFRs for this work. Those are represented by use cases with <<*aspect*>> stereotype. ii) The *join points links* that are associations between base use cases and aspect use cases, represented by associations with <<*joinPoint*>> stereotypes.

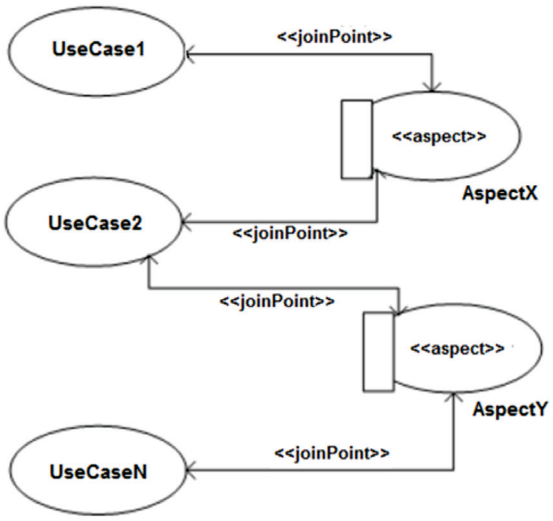

Figure 2. Aspect-oriented use case diagram

Figure 2 shows an example of a use case diagram with the two elements. For example, the "UseCase1" use case has a join point with "AspectX", that is to say, the use case uses this aspect. This diagram does not specify how and where to use the aspects; only use cases need aspects for their performance (use cases could work without aspects, but the aspects add some quality properties).

2.1.2 Pointcuts

Pointcuts are used to specify the exact places where an aspect cross-cuts a system. They use the previously-mentioned join points as a reference [9]. In DADM, this concept can be represented by UML sequence diagrams. These diagrams are more precise in specifying the place where an aspect cross-cuts a system. Model-driven aspects design stereotypes of the UML messages with <<*pointCut*>>, this stereotype can receive many parameters as it required (the parameters are detailed in the next section).

Figure 3 shows an example of a sequence diagram with pointcuts. We consider that this sequence diagram describes the design of the “UseCase2” use case (shown in Fig. 2) which has two join points with “AspectX” and “AspectY” stereotypes, therefore the sequence diagram can use both aspects. For example, “Method1” has a pointcut with “AspectX”, and “Method2” has a pointcut with “AspectY”. These pointcuts are interpreted as the crossing places between messages and aspects, but not as the crossing places’ behaviour.

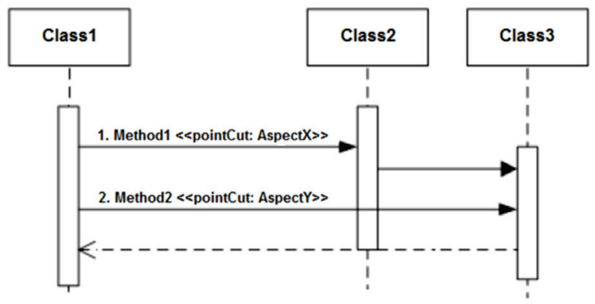
Class1
Class2
Class3
1. Method1 <<pointCut: AspectX>>
2. Method2 <<pointCut: AspectY>>

Figure 3. Aspect-oriented sequence diagram

2.1.3 Advice

A pointcut indicates where an aspect cross-cuts a system, but not the crossing’s behaviour. So, an advice concept was included. Advice can be attached before, after, instead of, or around a pointcut [9]. In DADM, this concept describes the aspects themselves. While the two previous concepts indicate where the interaction between aspects and a system takes place, this element specifically describes an aspect (i.e., what it does and how it behaves within the system). DADM adds a new diagram called the *weaving rule diagram*, this diagram is similar to the UML sequence diagram as shown in Fig. 4.

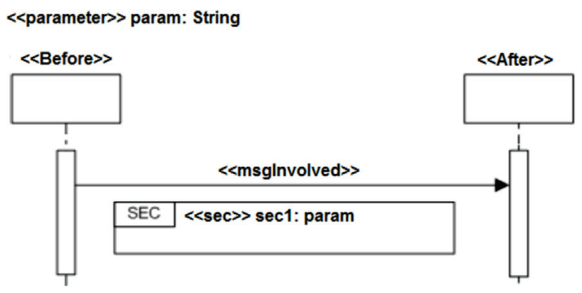

Figure 4. Weaving rule diagram

A weaving rule diagram shows how an aspect behaves within a system. The system is represented by two UML life line elements with *<<Before>>* and *<<After>>* stereotypes and a message with the *<<msgInvolved>>* stereotype. The three elements correspond to elements belonging to the sequence diagram with the *pointCut* stereotype (detailed above).

Other elements are added to this work, they correspond to the definition aspect for defining the following types of aspect behaviour: before, after, instead of, or around. These elements are:

- *Operator with <<err>> stereotype*: used for error handling, similar to programming languages such as Java which uses *try catch finally* sentences.
- *Operator with <<sec>> nameSec: parameter* stereotype:* used to define a set of sequential messages with a given name and zero or more parameters (this diagram is similar to the UML sequence diagram). This sequence can be invoked from a weaving rule diagram or another sequence.
- *<<parameter>> namePar: typePar:* represents a parameter which can be used by a weaving rule. It is important to mention that all parameters which are used by the weaving diagram have to be indicated in the *pointCut* stereotype message in the sequence diagram when we invoke the aspect.
- *<<return>> valueReturn:* represents a return value of a sequence.

Figure 4 shows an example of a weaving rule diagram. The rule states that after executing the stereotyped message with *msgInvolved,* it needs to execute sequential messages represented by an *SEC* fragment named *sec1* and uses a *param* parameter (this sequence must be built using a UML sequence diagram that uses parameters. This kind of diagram is not shown because the example is only illustrative).

2.2. Defining aspect models

Our work considers three NFRs, which were designed based on the evidence presented. For reasons of limited space, we only show models for error handling. Audit and access control models are not presented in this paper.

Within the same method, we find at least two behaviours: normal and exceptional. Error handling covers three issues: detection, publishing, and the recording of exceptions [3]. According to this, we show a weaving rule diagram, a class diagram, and a sequence diagram to define error handling.

2.2.1. Weaving rule diagram

Figure 5 shows the *weaving rule diagram* for "*error handling*". Here, we use an *err* operator, where we invoke a "*Message1*" message in the *try* section. This represents the functional message (the *msgInvolved* stereotype is not visible for TopCased), and we invoke the "*ErrorHandling1*" sequence in the *catch* section, which designs the error handling itself (the sequence name is not visible for TopCased). This diagram is used as a template because its elements will be replaced by the "true" elements in the weaving process running. That is to say, "*Message1*" will be replaced by a functional message. Life lines with *Before* and *After* stereotypes will be replaced by classes which invoke and include functional messages, respectively. Finally "*ErrorHandling1*" will be replaced by the corresponding sequence.

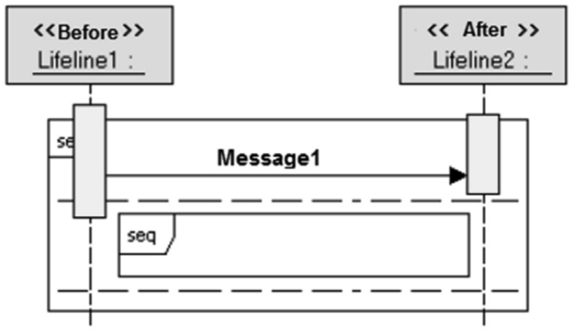

Figure 5. Weaving rule diagram for error handling

2.2.2 Class diagram

Figure 6 shows a *class diagram* for "*error handling*". This diagram has three classes: *Error* represents an error generated in a system, *LogError* represents a means of registering the error, and *InterfaceError* represents an interface which publishes the error. In addition, we also present datatypes to define error registry types (*TypeLog* can be: text file, system log, or a database record) and defines interfaces that publish the error (*TypeUI* can be: web page, desktop form, or e-mail).

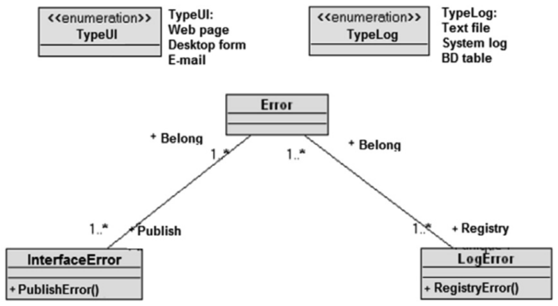
<<enumeration>> TypeUI
TypeUI: Web page Desktop form E-mail
<<enumeration>> TypeLog
TypeLog: Text file System log BD table
Error
+ Belong
1..*
1..*
+ Belong
1..*
+Publish
1..*
+ Registry
InterfaceError
+PublishError()
LogError
+ RegistryError()

Figure 6. Class diagram for error handling

2.2.3 Sequence diagram

Figure 7 shows the *sequence diagram* for "*error handling*" where the classes mentioned interact. When an error occurs, it is registered in LogError and shown by InterfaceError.

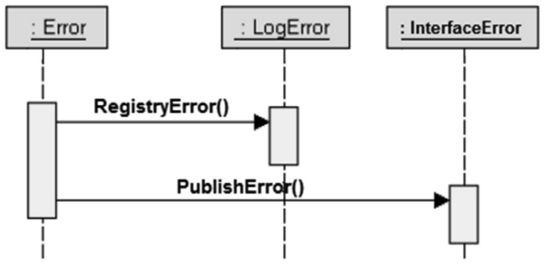

Figure 7. Sequence diagram for error handling

2.3. Creating a weaving process

It is necessary to create a weaving process which defines how to combine base models and aspect models to obtain a weaved model that includes both. Table 1 shows a rule for each DADM defined stereotype. These rules are built in OpenEmbeDD, according to its syntax (see http://www.eclipse.org/atl/ for more details).

Table 1. Weaving rule for each stereotype

Stereotype	Base class	Weaving rule
aspect	use case	Use cases with this stereotype must not be included.
joinPoint	association	Associations with this stereotype must not be included.
pointCut	message	This message needs to be replaced by the corresponding weaving rule diagram.

before	life line	This life line needs to be replaced by the life line that invokes the message with the pointCut stereotype.
after	life line	This life line needs to be replaced by the life line where the message with the pointCut stereotype comes from.
msgInvolved	message	This message needs to be replaced by the message with the pointCut stereotype.
err	fragment	This operator remains in weaved model.
sec	fragment	This operator needs to be replaced by the referenced set of sequential messages.

3. CASE STUDY

Before formally presenting the case study, it is important to present a scheme that shows how DADM works. Figure 8 shows how we separate the base module and the aspect into models; on the other hand, we also see how to include the aspect-oriented concepts like Aspect, Join Point, and PointCut. Finally, we simulate a weaving process to combine both models. This is represented by arrows: the objects where the arrows begin are replaced by the objects where the arrows end. We reiterate once the weaving process is executed and we will obtain a new model which combines functional and non-functional concerns.

The weaving process, as mentioned, runs automatically, but there are stages which are manual and these are not given in detail here. These steps are performed by an analyst and will be used to develop this case study. Therefore, we present a methodology for using DADM with four steps: The first step is *modeling the base functionality* that represents the FRs of the system. The second step is *including the aspects* using join point and pointcut elements on the use case and sequence diagrams of the base model, respectively. The third step is executing the weaving process to obtain a complete model that includes NFRs and FRs. Finally, the last step is *executing the transformation process* to obtain a source code that corresponds to the system. The last step will not be considered for this case study because we are not elaborating a transformation process. It is not within the scope of our work.

Now that we know how our proposal works and how to use it, we proceed to the development of the case study:

3.1. Case Study Definition

We have chosen the case study proposed by aosd.net (http://aosd.net/): This system helps to identify, evaluate, and handle a crisis situation (such as an accident, a robbery, a natural disaster, etc.) orchestrating communication between all entities in order to handle the crisis.

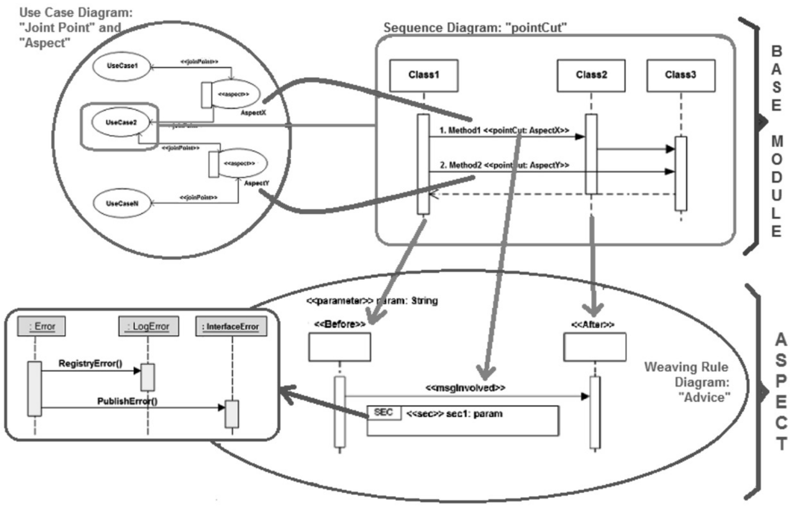

Figure 8. Running DADM

We develop the "Execute Mission" functionality of a "Crisis Management System for Car Crash" to reduce the scope of the case study. This functionality allows the super observer to store and execute a mission to rescue victims [23].

3.2. Development of the case study using DADM

3.2.1. Modeling the functionality base

We model the base functionality "execute the mission" for a "Crisis Management System for a Car Crash". This functionality is defined by a use case diagram, a class diagram, and a sequence diagram shown in Fig. 9. (These diagrams were proposed by aosd.net [23]).

3.2.2. Including the "error handling" aspect

We include the aspect into the mentioned base functionality; because of this, the use case diagram and the sequence diagram shown in Fig. 9 are modified to include the error handling aspect as shown in Fig. 10. Note that due to limited space, we have cut the diagrams of Fig. 9 since our objective is to show the new elements for inclusion of the aspects. In Fig. 10a, it is easy to distinguish the *aspect* and *joint point* elements, while for Fig. 10b we added message properties to distinguish the *pointCut* element.

3.2.3. Executing weaving process

Finally, we execute the weaving process to obtain a resulting model in an xmi file. We compare this result with another xmi file output from a model that includes the base functionality and the aspect, which was built manually. The result of this comparison indicates that it is similar to our DADM proposal, although executed manually. The advantage is that our proposal makes maintenance easier, is less time consuming, and more readable.

4. RELATED WORKS

[3] shows a study about including quality attribute applications in the developing phase. Our work does it in design phase.

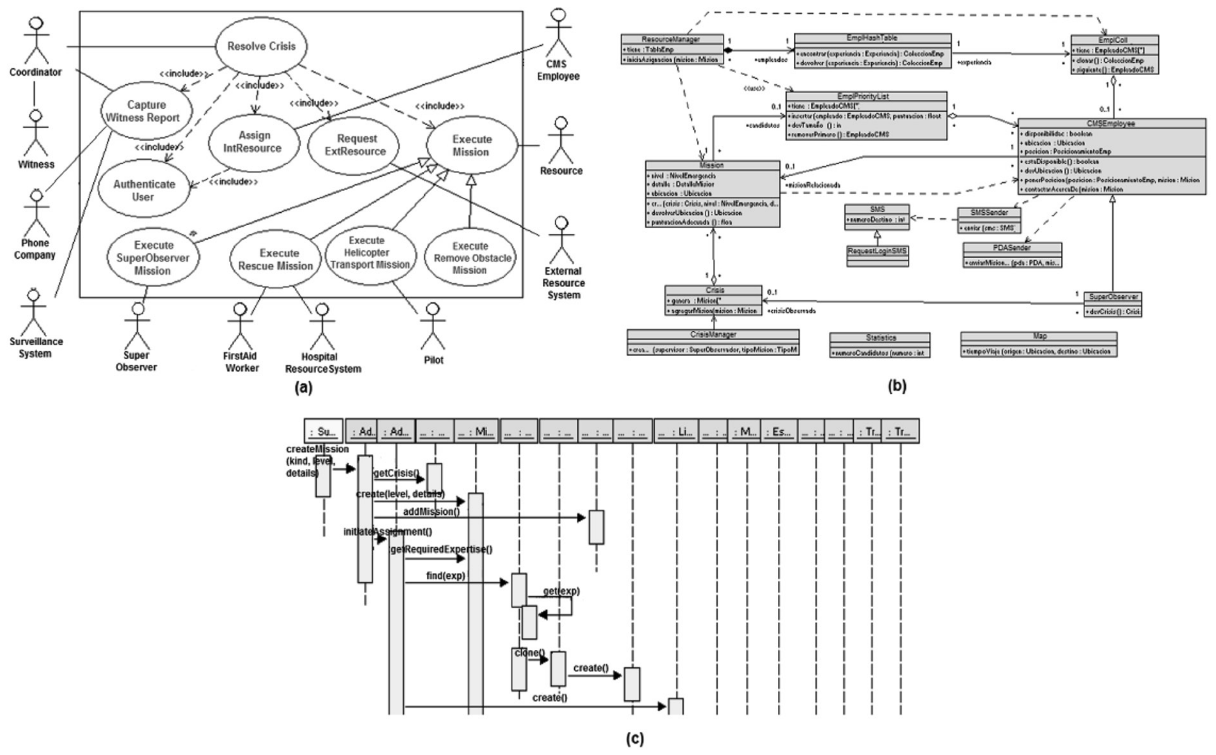

Figure 9. Diagrams for "Mission Execution" functionality of the "Crisis Management" system: (a) use case diagram, (b) class diagram, (c) Sequence diagram

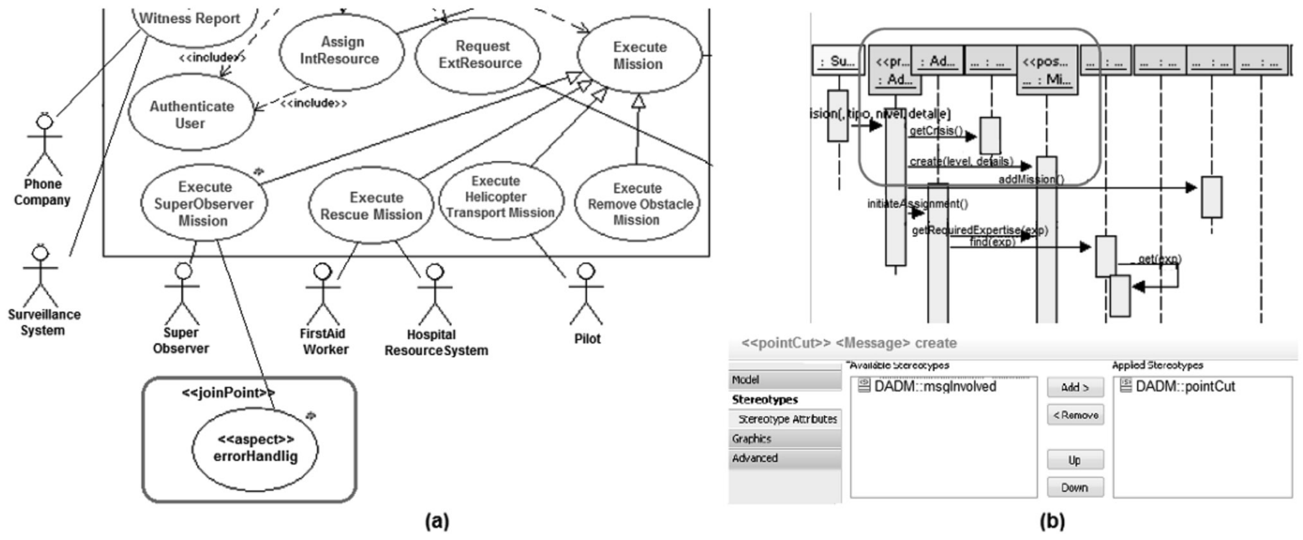

Figure 10. Inclusion of aspects into base functionality: (a) Inclusion of "Aspect" and "JoinPoint" concepts into use case diagram presented in Fig. 9a. (b) Inclusion of "PointCut" concept into sequence diagram presented in Fig. 9b

[6,19, and 25] present frames to model aspects for dispersed services, web services, and learning systems, respectively. Our proposition allows for one to model aspects for the NFRs of an IS.

[4] presents a UML profile for AspectJ language. This profile gives a complete integration of all AspectJ's elements, whereas we take aspect-oriented concepts to abstract them into a profile and to model language-independent aspects.

The work presented in [14] shows a UML notation to design aspect-oriented applications. That paper uses UML class diagrams to include aspects, while we use UML sequence diagrams to allow one to define the dynamic behaviour related to aspects.

[17] takes non-functional requirements into account, decomposing the system. This is similar to DADM, but based on architectural concepts. We focus our work on design elements (sequential diagrams).

Our work is based on the MATA proposal [22] which presents a graph transformation meta-model to model aspects. The MATA helps us to work the aspects' dynamic behaviour. We add specificity to MATA to abstract aspect-oriented concepts such as: aspects, joint points, pointcuts, and advice.

5. CONCLUSIONS

In general, NFRs are issues that cross-cut various parts of an IS. This is why they are difficult to conceive and even more difficult to maintain. Creating NFRs in development, deployment, or implementation phases makes conceiving and maintenance more difficult. Often they are not close to the expectation of final users. It is true that NFRs are well-defined on platforms, but they could be conceived in early phases where an analyst or designer knows precisely where and when to use them.

To avoid conception errors, we believe that it is necessary to give suitable mechanisms to add earlier NFRs separately from the functionality system to improve performance in maintenance work. Therefore, we have proposed an approach which uses aspect-oriented principles—to provide independence between the system concerns—formalized according to MDE. Due to this, our work gives a UML profile such as the DADM meta-model to create and include aspects in the analysis and the design phases according to the life cycle of software development. In addition, we also give a weaving process to allow the combination of aspect or non-functional models and base or functional models. Beyond that, we think it is appropriate to keep analysts up to date with the different steps of this approach.

Finally, we have presented a formal case study to validate our proposal. The obtained results indicate that we obtain the same final model either by using the proposed DADM or by doing it manually. The DADM advantage is that a DADM offers easier maintenance; for example, using DADM will update the base model (for functional changes) or aspect model (for non-functional changes) only once, while doing it manually would be tedious and error prone. It should be noted that the MDE transformation model—although beyond the scope of this work—could be used to obtain a source code for a specific platform. This would make maintenance work, due to technological changes, easier.

To conclude, our proposal simplifies maintenance work from an early conception onward. Furthermore, early NFRs improve error detection and correction. We know that correcting errors is less costly in the analysis and design phases than in the implantation phases. Several studies demonstrate that the detection of errors in the development stage costs 10 to 200 times more, since the efforts required to correct such errors are very important [26].

While our work has given good results, further testing on other systems on more complex NFRs is needed in order to evaluate the new results and to update our proposal, if necessary.

ACKNOWLEDGEMENTS

Many thanks to the BILIMERI Convention since it allowed for the cooperation necessary between UNMSM (Peru) and LIUPPA (France) to finalise this paper.

REFERENCES

[1] Bram, A., Co-Evolution of Source Code and the Build System: Impact on the Introduction of AOSD in Legacy Systems. Gent - Belgium: Ph.D. Thesis, University of Gent, 2007-2008.

[2] Tavares, A., Methodological Approaches and Techniques for Model Driven Development of Software Product Lines. Braga - Portugal: Ph.D. Thesis, University of Minho, November 2007.

[3] Paez, N., Utilización de programación orientada a aspectos en aplicaciones enterprise. Buenos Aires - Argentina: Engineering System Thesis, University of Buenos Aires, November 2007.

[4] Evermann J., A Meta-Level Specification and Profile for AspectJ in UML. New Zealand: University of Wellington, published in Journal of Object Technology, August 2007.

[5] Ruscio, D., Specification of Model Transformation and Weaving in Model Driven Engineering. Aquila - Italy: Ph.D. Thesis, Università de L'Aquila, 2007.

[6] Simmonds D., Reddy, R., France R. y Ghosh S., Developing Distributed Services Using an Aspect Oriented Model Driven Framework. Colorado - EEUU: Published in International Journal of Cooperative Information Systems, December 2006.

[7] Reina, A., Torres, J, y Toro, M., Hacia Lenguajes de Metamodelado Orientados a Aspectos. Sevilla - Spain: Universidad de Sevilla, published in Aspect-Oriented Software Development, related to Software Development and Database XV Workshop, October 2006.

[8] Figueroa, P. y Isaza, A., Programación por Aspectos. Una Introducción. May 2006.

[9] Lengyel, L., Levendovszky, T. y Charaf, H., Aspect-Oriented Techniques in Metamodel-Based Model Transformation. Budapest - Hungary: Sixth International Symposium of Hungarian researchers in Computational Intelligence, November 2005.

[10] Lengyel, L., Levendovszky, T. y Charaf, H. Aspect-Oriented Constraint Management in Metamodel-Based Model Transformation Steps. Budapest - Hungary: Department of Automatize and Applied Computing, 2005.

[11] Kong, J., Zhang, K., Dong, J. y Song, G. A Generative Style-driven Framework for Software Architecture Design. Texas: Published in 25th Software Engineering Workshop of IEEE/NASA, 2005 (SEW'05).

[12] Gonzales, C., Murillo, J. y Amaya, P., Un modelo de propiedades y dependencias para el análisis orientado a aspectos en MDA. Extremadura - Spain: Software Engineering of Quercus Group, University of Extremadura, published in Aspect-Oriented Software Development, 2004.

[13] C. De souse, G. y Brelaz De castro, J. Towards a Goal-Oriented Requirements Methodology Based on the Separation of Concerns Principle. Pernambuco - Brasil: Published in Requirement Engineering Workshop, 2003 (WER'03).

[14] Pawlak, R., Duchien, L. y Florin, G. A UML Notation for Aspect-Oriented Software Design. Paris - France: CEDRIC-CNAM, LIFL, LIP6 and AOPSYS laboratory, March 2002.

[15] Lin, Y., A Model Transformation Approach to Automated Model Evolution. Ph.D. Thesis. Alabama – EEUU: University of Alabama at Birmingham, 2007.

[16] De souse, F. Modelog: Model-Oriented Development with Executable Logical Object Generation. Pernambuco - Brazil: Ph.D. Thesis, Federal University of Pernambuco, February 2007.

[17] Perez, J. Prisma: Aspect-Oriented Software Architectures. Valencia - Spain: Ph.D. Thesis. Polytechnic University of Valencia. December 2006.

[18] Lengyel, L., Levendovszky, T., Mezei, G. y Charaf, H. Model Transformation with a Visual Control Flow Language. Budapest - Hungary: published in International Journal of Computer Science, 2006.

[19] Ortiz, G., Hernandez, J., Clemente, P., Amaya, P., How to Model Aspect-Oriented Web Services. Extremadura - Spain: Software Engineering of Quercus Group, University of Extremadura, published in 5th International Conference on Web Engineering, 2005 (ICWE'05).

[20] Chevrel, R., The AMMA Platform and the DotNET environment. Nantes - France: INRIA, University of Nantes.

[21] Storzer, M., Impact Analysis for AspectJ. Passau - Germany: Dissertation, University of Passau, 2007.

[22] Whittle, J. y Jayaraman, p. Mata,: A Tool for Aspect-Oriented Modeling based on Graph Transformation. Published in Aspect-Oriented Modeling Workshop Models, 2007.

[23] Kienzle, J., Guelfi, N. y Mustafiz, S,. Crisis Management Systems A Case Study for Aspect-Oriented Modeling. Proposal for participants of aosd.net Workshop, 2009.

[24] Montenegro, C., Gaona, P., Cueva, J y Martíez, O. Application of Model-driven Engineering (MDA) for the construction of a tool for Domain-specific Modeling (DSM) and the creation of modules in Learning Management Systems (LMS) Platform Independent. Medellín – Colombia: Published in 169th DYNA, Journal of University National of Colombia, October 2011.

[25] Arango, F., Gómez, M. y Zapata, C., Transformation from UML class model to Oracle9i using the MDA guidelines: A study case. Medellín – Colombia: Published in 149th DYNA, Journal of University National of Colombia, October 2006.

[26] Johnson, J. Chaos: The dollar drain of IT project failures. Application Development Trends 2.