Aspect-Oriented Analysis: a MDA Based Approach[1].

Carlos González
Quercus Software Engineering Group
University of Extremadura
Spain
carlosgc@unex.es

J.M Murillo
Quercus Software Engineering Group
University of Extremadura
Spain
juanmamu@unex.es

Pablo A. Amaya
Quercus Software Engineering Group
University of Extremadura
Spain
pabloama@unex.es

Abstract

Actually, there are many approaches to deal with crosscutting concerns in early stages in software development. One is MDSOC, which promotes the encapsulation of all relevant kinds of concerns (dimensions) and their integration. MDSOC starts from an initial problem decomposition to design the system. No discussion about how this decomposition should be carried is presented. Based on this deficiency, in this position paper we introduce the principles to make an aspect-oriented analysis in software development. The main concern of this work is the identification and analysis of the dependencies model between system properties. Furthermore, such interest is developed in the MDA framework and our approach is illustrated by a study case.

1. Introduction

AOSD (Aspect Oriented Software Development) is a paradigm of software engineering that leads to reduce the software systems complexity through the separation of crosscutting concerns [1]. Separation of concerns [2] is the ability to identify, encapsulate and manipulate in isolated way only those parts of software that are relevant to a concept, objective or intention given.

One of the most relevant current trends within AOSD is MDSOC [3] (Multi Dimensional Separation Of Concerns) which promotes the encapsulation of all relevant kinds of concerns (dimensions) and his integration to make the complete system. MDSOC also provides powerful composition capabilities, in order to allow developers to integrate these separated pieces. All the dimensions are dealt with in isolated way, allowing developers to be focused in a particular feature of the system being unaware of the others. By same kinds of concerns we mean concerns which decompose the software accordingly to a certain viewpoint or particular vision.

The main advantage of MDSOC based approaches, for example HyperJ [4], is that they provide a greater degree of concern independence than others, for example those based in AspectJ [5], where separately coded aspect can augment the base decomposition but cannot augment one another, so aspect are not composable.

MDSOC are intended to apply across the life cycle. However, much of the early work on MDSOC focused on design and implementation phases.

At design phase, MDSOC starts from an initial problem decomposition to model the system. As stated by the own authors [3], the identification and election of such decomposition has to be made very carefully because it will guide the rest of the design process. A system can have several decompositions each of them prioritizing some properties -which will be well-decomposed properties. As a consequence other properties will remain as crosscutting properties. We use the term *property* meaning any concern dealing with the system functionality. So, a property is a *crosscutting property* if the functionality it is concerned with overlap with some well-decomposed properties.

What MDSOC authors are really meaning is that the choice of the well-decomposed properties –that is, the system functionality that will guide the system decomposition, should be made carefully because it will determine what remain as crosscutting properties. However no discussion about how this selection process should be carried out is presented.

Based on this deficiency, in this position paper we introduce the principles to make an aspect oriented analysis in software development. Such analysis should be guided by (1) the identification of the system properties, (2) the identification of the dependencies between properties- that is, those dependencies making

[1] This work has been developed with the support of Spanish Ministry of Science and Technology under contract TIC2002-04309-C02-01.

that setting a property as a well-decomposed one some other will remain as crosscutting one, and finally (3) the identification of the more convenient properties to be chosen as well-decomposed ones –and consequently the crosscutting properties.

Thus, the main concern of this work is the identification and analysis of the dependencies (between properties) model of the system. Furthermore, such interest is developed in the MDA (Model Driven Architecture) framework [6]. MDA is an initiative by the OMG that emphasizes the importance of modelling for software development. MDA proposes the system development to be driven by the specification models of three different abstraction levels: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). To support models traceability MDA also provides mappings between models.

In the MDA framework, we propose to specify the dependencies model at the CIM. Its analysis and the selection of the well-decomposed properties will lead to the PIM. In this settings and being aware that the choice of the best decomposition of the system at early stages is usually not trivial we point out the possibility of managing several design options (PIM's) at the same time. All those independent PIM's will model the same system using different decomposition criteria. However, since all the PIMs refer to a single system they will include similar elements. So, we also propose to specify the correspondence between different PIMs. The different decomposition options can be maintained until a later stage in which a definitive (optimal) decomposition can be adopted.

Thus, Aspect Oriented Analysis becomes an independent, first-class activity of software development achieving some software engineering objectives [7]. For analysis and understanding, aspect oriented analysis reflect different design options and support new analyses of properties and application semantics. For evolution, aspect oriented analysis supports impact analysis, change propagation, and the evaluation of completeness, correctness, and consistency. For reuse, aspect oriented analysis themselves may be reused as sources of semantic information.

This paper is organized as follows: First, we introduce a study case and the Hyperspace approach in section 2. In the following section we introduce our solution focused on a study case. Finally, in section 4 we discuss the conclusions and point out directions for future work.

2. Study case and the Hyperspace approach.

After introducing basic concepts, we will now show a real study case. The scenario is an administrative protocol in the government offices of Extremadura, Spain. The objective of this protocol is to manage all the sanctioning procedure within the administration. This procedure begins when a new file is created to denounce an offense. Later, a complex protocol is carried out until a final resolution is created. These system kinds are characterized by having many different users and interacting many heterogeneous systems. Furthermore, to carry out all the process, normally different parts of the process overlap in different government departments.

We begin initially with the following system requirements specification[1]:

> *The desired system supports the management of the sanctioning procedure within the administration. When an offense is denounced, a file is created. When the file is processed by an administration personal, a resolution proposal is created and it is notified to the accused. At this point, the accused will be able to liquidate the sanction with a discount of thirty percent. After a time period, a final resolution is created and it is notified to the accused. In this case, the accused must pay the complete sanction amount. When the accused liquidates the sanction, the file is closed and the payment information should be printed. Subsequently the file is closed. The file can be printed at any time. Therefore, the system should provide a printing service.*

In requirements statement we find the following coarse-grained properties which deal with system functionality[2]:

- File: It represents an entity on which all the process is carried out.
- Notification: the system should notify the accused when a resolution proposal or final resolution is created.
- Payments: the system should manage the payments. Payments management is carried out by two ways: On one hand, an early payment management must be possible. On the other hand, the full payment must be managed.

[1] This is a part of Requirements statement.
[2] This is a subset of requirement properties.

- Printout: The system should provide a printing service.

The Hyperspace approach [8] is a MDSOC based approach. It is a generic technique focussed on the decomposition and composing of software systems according to concerns. Relevant concerns of a concern type that are identified during system decomposition have to be represented using a particular dimension. Typically two dimensions are relevant for Feature Driven Software Development [9,10]: for design entities and for features. In our approach software properties are used to decompose the system, therefore two dimensions are relevant: for design entities and for properties. Such hyperspace is depicted in Figura 1.

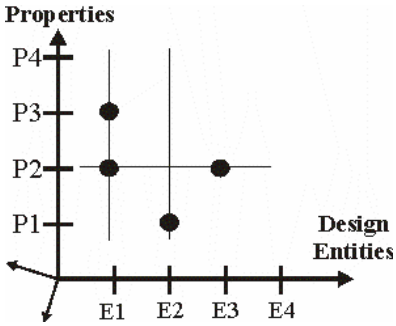

Figure 1. Hyperspace using properties.

Observing this multidimensional space of properties, there is a dimension (Design Entities dimension) which decomposes the system in a set of design entities. Such decomposition uses a decomposition criterion which is based on one or several software properties. These properties will be implicit in such criterion. Furthermore, these properties will be well-decomposed properties in our design because they are decomposed in independent design entities. Usually, this is referred as primary dimension. As a consequence, others system properties will remain as crosscutting properties, in the example, P1, P2, P3 and P4 properties. This means that these properties will be overlapped properties of different design entities and they will remain in auxiliary dimensions. Modelling the system in accordance with this approach, we found two limitations:

- A basis does not exist to establish an initial decomposition criterion. Usually, the most immediate decomposition is carried out, but others alternatives decomposition can exist.
- We cannot determine the impact in design of choosing this decomposition criterion in advance and/or we cannot determine the effect on other properties of taking a property as well-decomposed property.

The choice of a decomposition criterion, that is, the system functionality (property) that will guide the system decomposition is not a trivial process and it will guide the rest of the design process.

Based on these deficiencies, in the next section we introduce the principles to make an analysis before modelling the system. Such analysis will allows us to analyze certain semantics dependencies between system properties. These dependencies make that setting a property as a well-decomposed one some other will remain as crosscutting one.

3. Proposed solution.

As discussed in the previous section, the analysis must be guided by (1) the identification of the system properties (this is done in section 2), (2) the identification of the dependencies between properties- that is, dependencies such that decomposition by one property causes another property to be crosscutting, and finally (3) the identification of the more suitable properties to be chosen as well-decomposed ones.

Based on identified properties in section 2, we motivate a more detailed study of properties to identify the dependencies among them and it is a requirements based study.

In requirements statement we found the following details:

Notification property overlaps with the File property because the notification depends on the file type. That is to say, depending on file type, notification is carried out in a way or another. However, Notification property does not overlap with the Payments and Printout properties because they are isolated properties. Payments and Printout properties are carried out in an independent way. Table 1 shows how the properties overlap some with others.

Table 1. Property interactions.

	File	Notification	Payments	Printouts
File		Yes	Yes	Yes
Notification			No	No
Payments				Yes
Printouts				

As described in [11], there are a set of fundamental relationship kinds that reflect the basic semantics of properties type. Examples of semantic relationships

among software properties can be contribution relationships. These relationships suppose that properties can affect positively or negatively the way in which one property is addressed. In our study case, Notification and File properties are overlapped properties and we denoted this as contribution relationships. In the next research stages, we will investigate if these contribution relationships are negative or positive and we will determine how this can affect to the system decomposition. On the other hand, we say that there are no contribution relationships between Notification property with Payments and Printout properties because there is no overlapping. This is illustrated in Figure 2.

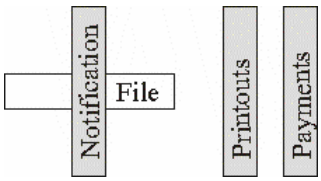

Figure 2. Notification dependencies.

On the other hand, the File property is the most overlapped property because it represents an entity on which all the process is carried out. Therefore, a contribution relationship exists between File property and all another properties (see Figure 3).

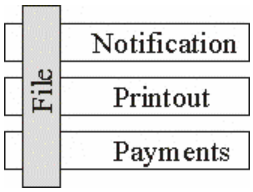

Figure 3. File dependencies.

These contribution relationships (dependencies) mean that to using one or several properties as decomposition criterion (i.e., well-decomposed properties) will make some other property a crosscutting one. This decomposition criterion will suppose in a design entities set. Once the dependencies between properties are specified, we can determine different system decompositions. At first sight, File property is candidate to decompose the system because it is the main functionality in the domain model. Observing the File property dependencies, we can deduce that if we decompose the system using this property, all another properties will remain as crosscutting one because there are contribution relationships among these properties and File property. Such system decomposition is illustrated in Figure 4.

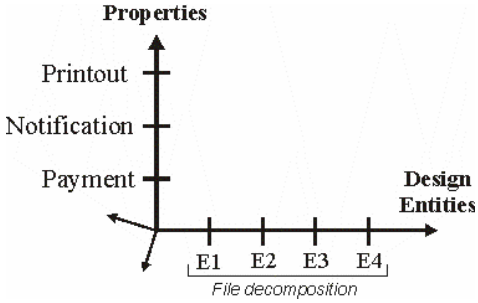

Figure 4. System decomposition using File property.

However, usually the best system decomposition is not trivial in this software development stage. Therefore, we point at the possibility of taking into account different design options using the MDA framework. This will be object of future work and we propose to maintain the dependencies model at CIM level. Its analysis and the selection of the different well-decomposed properties will lead to the independent PIM´s which will model the same system from different viewpoints.

Let us suppose that the Payment property is used to carry out the initial problem decomposition. Observing the Payment dependencies (see Figure 5), there is no a contribution relationship between Payment and Notification properties.

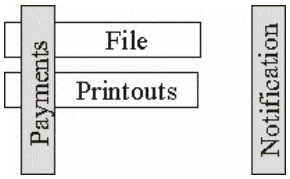

Figure 5. Payments dependencies.

This means that the Notification property does not overlap with the Payment property, because Payments and Notification are independent process. Therefore, Notification will remain as well-decomposed property due to such relationships. However, File and Printouts properties will remain as crosscutting one because they have a contribution relationship with Payment property. These relationships exist because Payments are carried out based on File type and the Printouts process depends of Payments process to printing one documentation or another (See Figure 6).

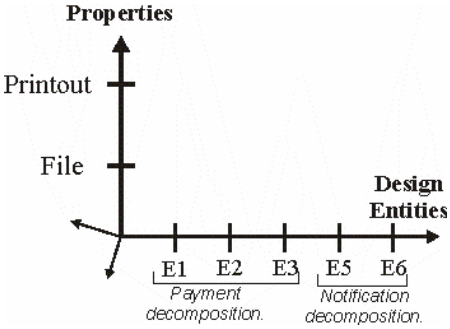

Figure 6. System decomposition using Payment property.

As conclusion, the analysis of software properties allows us to have a semantic basis between properties (dependencies model). With this dependencies model, some design decisions can be made. We can take into account several system decompositions each of them prioritizing some properties. Additionally, we can determine what properties will remain as crosscutting properties owing to decomposition criterion in an automatic way.

4. Conclusions and Future Works

At the moment, there are approaches that promote the multidimensional modeling of concerns. A multidimensional space of concerns is made encapsulating relevant kinds of concerns in dimensions. Some approaches, like MDSOC starts from an initial problem decomposition to decompose the system. However, a system can have several decompositions each of them prioritizing some properties which will be well-decomposed properties. As a consequence other properties will remain as crosscutting one. We use the term property meaning any concern dealing with the system functionality. It really meaning that the choice of the well-decomposed properties –that is, the system functionality that will guide the system decomposition, should be made carefully because it will determine what remain as crosscutting properties However, no discussion about how this selection process should be carried out is presented. Designing a system based on MDSOC approach, we find two limitations:

- A basis does not exist to establish an initial decomposition criterion.
- We cannot determine the effect on other properties of taking a property like well-decomposed property.

Based on these deficiencies, this position paper is based on establishing an aspect oriented analysis stage. It should be guided by (1) the identification of the system properties, (2) the identification of the dependencies between properties- that is, those dependencies making that setting a property as a well-decomposed one some other will remain as crosscutting one, and finally (3) the identification of the more convenient properties to be chosen as well-decomposed ones –and consequently the crosscutting properties.

Being aware that the choice of the best decomposition of the system at early stages is usually not trivial, we point out the possibility of managing several design options at the same time. In the MDA framework, we propose to specify the dependencies model at the CIM. Its analysis and the selection of the well-decomposed properties will lead to the different and independent PIM´s. The different decomposition options can be maintained until a later stage in which a definitive (optimal) decomposition can be adopted. This will be object of future work.

As a future line of research we are working on carrying out a more detailed properties study. Furthermore, we are working on using some formalism to specify properties. The objective is to find more types of dependences among them. Classification, generalization, motivation and more properties, which are declared in [11], will be study. Furthermore, the contribution relationships will be study in a detailed way. Additionally, we are working on solving different conflicts among properties. These conflicts occur when some properties have a contribution relationship in the same dimension.

As conclusion, in our approach Aspect Oriented Analysis becomes a first-class activity of software development. The dependencies model supposes a source of semantic information which promotes system evolution and maintainability.

10. References

[1] Aspect-Software Oriented Development, 2005. http://aosd.net.

[2] E.W. Dijkstra, "A Discipline of Programming,", *Prentice-Hall*, 1976.

[3] H. Ossher, P. Tarr, W. Harrison and S. M. Sutton Jr, "N Degrees of Separation: Multi-Dimensional Separation of Concerns", In Proceedings of International Conference on Software Engineering ICSE, Los Angeles CA USA, 1999.

[4] H. Ossher and P. Tarr, “Using multidimensional separation of concerns to (re)shape evolving software”, Communications of the ACM, October 2001, pp. 43-50.

[5] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. Griswold, “Getting started with AspectJ”, Communications of the ACM, October 2001, pp. 59-65.

[6] MDA (Model Driven Architecture). March 12, 2002 http://www.omg.org/mda.

[7] S. M. Sutton Jr and P. Tarr, “Aspect-Oriented Design Needs Concern Modeling”, Proceedings of the Aspect Oriented Design (AOD) Workshop in AOSD Conference, Enschede The Netherlands, April 2002.

[8] H. Ossher and P. Tarr, “Multi-Dimensional Separation of Concerns and The Hyperspace Approach”, In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development, Kluwer, 2000.

[9] M. Riebisch, “Towards a more precise definition of Feature Models”, *M. Riebisch, J. O.Coplien, D, Streitferdt (Eds.): Modelling Variability for Object-Oriented Product Lines*, Norderstedt, 2003, pp. 64-76.

[10] I. Philippow, M. Riebisch and K. Boellert, “The Hyper/UML Approach for Feature Based Software Design”, In Proceedings of 4th AOSD Modeling With UML workshop in UML Conference, San Francisco CA, 2003.

[11] S. M. Sutton Jr and I. Rouvellou, “Modelling of Software Concerns in Cosmos”, Proceedings of the 1st international conference on Aspect-oriented software development, Enschede The Netherlands, April 2002, pp. 22-26.