Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 163 (2006) 31–44
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Towards a Subject-Oriented Model-Driven Framework

Pablo Amaya, Carlos Gonzalez, Juan M. Murillo[1,2]

Quercus Software Engineering Group
Department of Computer Science
University of Extremadura. Spain

Abstract

Model-Driven Architecture is an approach which tackles such problems as: the high availability that a software product requires to be ready for use, the high degree of evolution that a software system has nowadays, etc. However, in the development of large complex systems, the benefits of that approach have been diminished due to the size and complexity of models that describe these kinds of systems. At this point Aspect-Oriented Software Development (AOSD) appears to improve the understanding, reusability and adaptation of the software artefacts. Its mechanism is based on modularization of crosscutting concerns in well-identified isolated entities called aspects. For this reason we propose to use together AOSD and MDA in the hope of reducing the shortcomings of the latter. Thus, aspects like security, replication, real-time constraints, etc., will be modelled by specialist modellers independently throughout the MDA framework. Our proposal exploits a tool for checking the consistency between different models (aspects) at the same level of abstraction; supporting the traceability of UML elements, requirements, and concerns; and controlling the impact of changes throughout the MDA framework.

Keywords: AOSD, Subject-Oriented Modeling, MDA, Traceability

1 Introduction

Model-Driven Development [14] is a paradigm that tries to decrease the amount of responsibilities and work-load at the implementation time. For this reason, its objective is to change the classic code-centric development process by a model-centric one. Thus, the developer can focus on the semantics of software systems to model it without regarding the details relative to the underlying platforms.

An approach in this area is Model-Driven Architecture [20] from the OMG. This approach is a step forward in the Separation of Concerns principle [8] for separating

[1] This work has been developed with the support of the Spanish Ministry of Science and Technology under contract TIC2002-04309-C02-01 and with the support of the Junta de Extremadura under contract 2PR04B011.

[2] Email: pabloama@unex.es

doi:10.1016/j.entcs.2006.07.006

technological concerns into different abstraction levels (vertical separation of concerns) [19]. Thus, it establishes three abstraction levels called CIM (Computational-Independent Model), PIM (Platform-Independent Model), and PSM (Platform-Specific Model). Each of these levels focuses on different concerns of the software system being developed. The CIM models the real system independently of any computational system, that is, it makes up a domain model. The PIM models the system from a computational viewpoint independently of any underlying platform, and the PSM models the system for a specific platform. Also, between each pair of consecutive models are the transformations, another key mechanism of MDA and MDD [32]. Their aim is to establish mappings between elements from a source abstraction model to a more refined or abstract one. Thus, supporting traceability of requirements and elements between different levels of abstraction is achieved. Since this feature facilitates the system maintenance, it is very important for software development.

However, when MDA is used in the development of large complex systems, benefits promised by this framework (traceability, evolution, maintenance, etc.) diminish considerably. This problem arises because the system is specified by very large, complex, and monolithic models [30]. So, these models are difficult to maintain, evolve, extend, adapt, reuse, etc. In addition, transformations between the different abstraction models become very complex, large and less reusable. In this scope, traceability of elements across different abstraction levels is difficult because of the lack of alignment between these models [30]. This fact implies design and implementation of requirements being scattered over several design and implementation entities respectively. The final consequence is that tracing a requirement from CIM to code could produce too much traceability information which will be hard to manage.

On the other hand, the AOSD [3,26] has extended the Aspect Oriented Programming [12,24] benefits to the whole software development life-cycle. This approach supposes an advance in software modularization. So, it allows us to isolate in artifacts (called aspects) those properties whose specification is scattered throughout the system and whose isolation is hard to manage by conventional modelling techniques. In this way, AOSD techniques facilitate the traceability of concerns in a software system [17].

Trying to support traceability of requirements, UML elements, subjects, and concerns in the MDA framework, in this paper an approach of integration of both MDA and AOSD is presented. In this way, an algorithm for tracing a requirements from CIM to PSM is proposed. In our proposal each MDA level is constituted by a set of models -each of them corresponding to an aspect of the software system [3]. Such aspects (models) will be developed and transformed separately throughout the MDA framework in a collaborative development environment [4] [35]. Typical aspects in this context could be security, real time constraints, etc, and they will be specified

[3] The UML2 specification stated that a model is a partial specification of the software system. In this way, we consider an aspect as a partial specification of the software system.

[4] By collaborative development environment we mean the scenario in which several developers collaborate on building the same system each of them focused on one area of the system.

by an expert in the area focused by the aspect. Thus, keeping different concerns as different models at each abstraction level allow for clearer transformations and mappings and consequently for an improved traceability. Moreover, our proposal allows us to model aspects in a collaborative and consistent way in the MDA context. It uses xlinkit [7] for model coherence checking at each abstraction level (CIM, PIM or PSM). This feature also provides support for automatic analysis of the impact of changes in models at any abstraction level.

The rest of the paper is organized as follows: in section 2 an overview of the proposal and how xlinkit is used to our goals are shown; section 3 presents the improvements put into the traceability and facility of evolution in MDA; section 4 shows how model consistency at the same abstraction level is managed; section 5 shows the related works; and finally, in section 6 the conclusions and possible lines of future work are presented.

2 Consistent Development with Model-Driven Architecture and Subject-Oriented Design

This section is organized in three subsections: the first presents some background about AOSD; the second gives an insight into our proposal with an example; and the third shows how xlinkit is used to check the consistency between models representing different aspects at the same abstraction level.

2.1 Background

The aim of AOSD is extending the AOP paradigm to all stages of software development. The key concept of AOP and AOSD is the separation of crosscutting concerns. After solving this issue at the implementation stage, AOP concepts are extended to all stages of the software life-cycle [26]. Thus, some approaches have been proposed for design stage [30,27], others for analyses stage [11] and some for requirements stage [4]. Almost all approaches model the systems using UML. This work is based on Subject-Oriented Modelling (SOM) [28,30]. The choice was motivated by the high degree of reusability and traceability that it provides in UML designs. SOM proposes that each requirement can be designed as a UML package called subject, and each subject will be implemented in Aspect/J or Hyper/J. Thus, when the system needs a change in its requirements, this change will only modify one subject so that the system maintenance task is improved. Moreover, SOM is very suitable for collaborative development due to its characteristic of symmetric paradigm for design. In [35] it is explained that symmetric approaches are suitable for collaborative development, and the asymmetric ones are suitable for development based on extensions.

2.2 Proposal Overview

This work assumes that the concerns crosscutting the system have been identified at early stages [5]. Such concerns will be modelled in isolation by specialist work-groups. The models will also be transformed independently keeping them separated along the development process. Thus, these work-groups will model and transform each of those aspects from the CIM to the PSM separately. The separation is managed by using the SOM approach. In this way, using our proposal to model these aspects collaboratively is allowed.

Having separate models at the same abstraction level makes it necessary to establish composition relationships between them. This is because it must be specified which semantics are shared between all facets of the system (described by different models), detect conflicts between these models, and to integrate them into a whole. These relationships are specified by a coordinator-modeller separately to the models so that the modellers focus on developing their aspects, unaware of other aspects of the system. The composition relationships are specified in a XML document. More details about this issue will be given in section 2.3 Xlinkit.

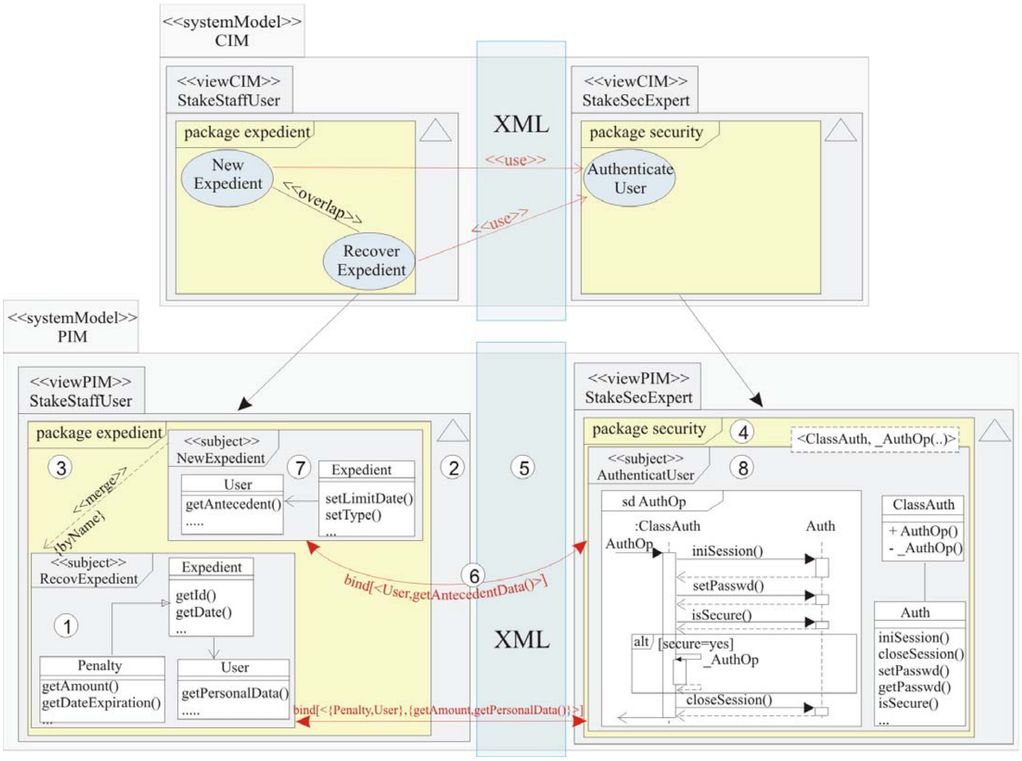


Fig. 1. CIM and PIM modelling two system aspects

Figure 1 shows the CIM and PIM level of our framework which is based on a case study of an e-government information system. This example deals with a sanctioning administrative protocol in our administrative council (Junta de Extremadura).

[5] This task is out of the scope of this work

We present a small example which contains a set of expedients, citizens and magistrates. Every time that a citizen must be sanctioned for making an administrative fault, the system creates a new expedient and assigns it to a specific magistrate. Then, when the magistrate comes to a verdict, this expedient is a penalty that falls to the citizen.

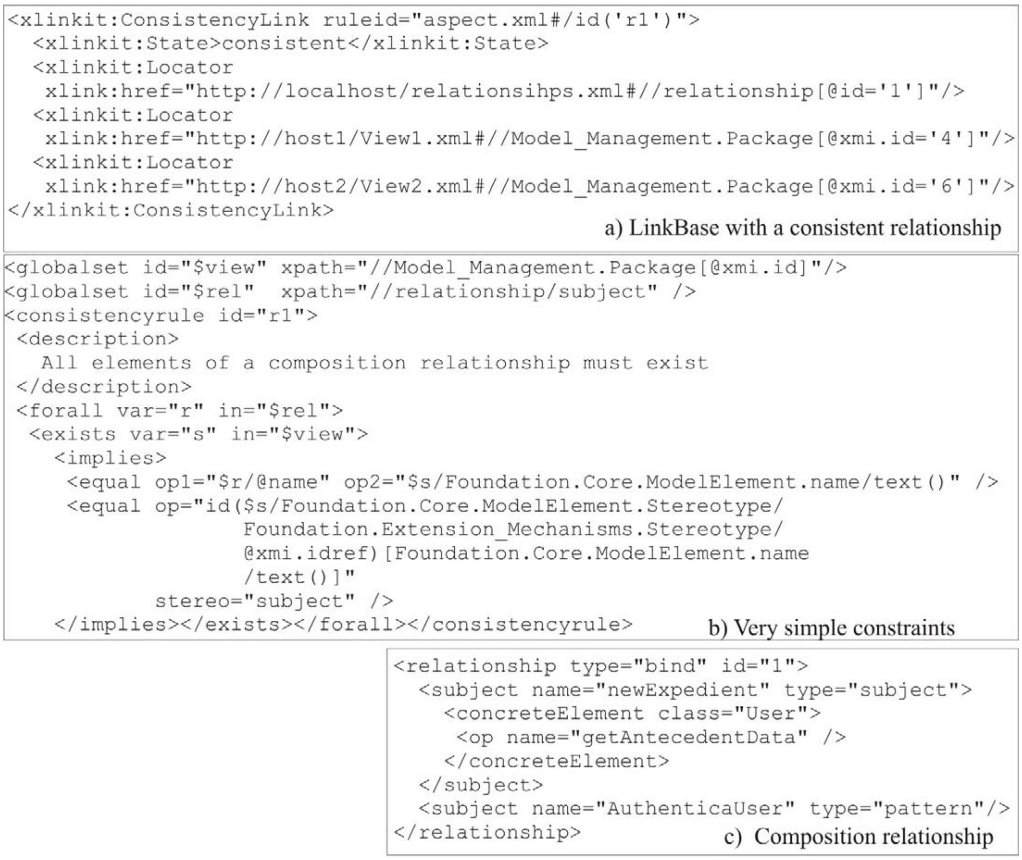
In this small example, we have identified three requirements in CIM. Using Subject-Oriented Design (SOD) the three requirements have been designed keeping them separately. If a conventional UML modelling was used instead the scattering and tangling problems described by Jacobson in [15] came up. Figure 1 shows the "Recover Expedient" use case that is designed by the subject "RecovExpediente" (1) in the model "StakeStaffUser" (2). In addition, two aspects (3)(4) have been modelled with the viewCIM and viewPIM stereotype of the UML2 model element [21]. In the same way, both the use cases and the two viewPIM models would remain separated at PSM level. We have adopted the SOM approach as it is, proposing a particular way for utilizing and realizing it. Unlike SOD, here the composition relationship between the expedient and authenticate concerns is specified in XML (5) by the coordinator-modeller. The XML specification is our own transcription which we have made of the three kinds of relationships proposed by SOD -*merge, override and bind*. This way of specifying composition relationships externally to the composed models supposes an advantage for two reasons:

- Firstly, because the modellers should develop the concerns with as little communication as possible between them [9], that is, a modeller should only concentrate upon his aspect being unaware of other aspects. Then, the composition relationships are established by an expert who is called coordinator-modeller.
- Secondly, using XML as the basis to specify relationship allows the use of tools for checking model consistency. In particular, we use xlinkit. It will be explained at length in the next section.

2.3 Xlinkit

Xlinkit is a tool to manage the consistency of distributed and heterogeneous documents in XML format that are crucial for the software development [7]. These documents are checked against a set of constraints implemented as rules. For example, a very simple rule could check if the classes' names of a Java implementation are consistent with their UML classes' names. This rule could force that for all classes in the design there must exist a class in the implementation with the same name. Xlinkit is based on XML, XPath and XLink for the generation of hyperlinks between distributed documents. The tool accepts a set of XML documents that represent models and another one which contains rules that elements of those models must satisfy. So, a XML document with pairs of links to pinpoint the consistent and inconsistent elements (LinkBase) as output is produced by xlinkit. That is, if two elements of two models satisfy a specific rule, the LinkBase shows hyperlinks pointing to the rule and the consistent elements.

The original objective of xlinkit is to manage the consistency between two mod-



```
<xlinkit:ConsistencyLink ruleid="aspect.xml#/id('r1')">
  <xlinkit:State>consistent</xlinkit:State>
  <xlinkit:Locator
   xlink:href="http://localhost/relationsihps.xml#//relationship[@id='1']"/>
  <xlinkit:Locator
   xlink:href="http://host1/View1.xml#//Model_Management.Package[@xmi.id='4']"/>
  <xlinkit:Locator
   xlink:href="http://host2/View2.xml#//Model_Management.Package[@xmi.id='6']"/>
</xlinkit:ConsistencyLink>
```

a) LinkBase with a consistent relationship

```
<globalset id="$view" xpath="//Model_Management.Package[@xmi.id]"/>
<globalset id="$rel"  xpath="//relationship/subject" />
<consistencyrule id="r1">
 <description>
   All elements of a composition relationship must exist
 </description>
 <forall var="r" in="$rel">
  <exists var="s" in="$view">
    <implies>
     <equal op1="$r/@name" op2="$s/Foundation.Core.ModelElement.name/text()" />
     <equal op="id($s/Foundation.Core.ModelElement.Stereotype/
                  Foundation.Extension_Mechanisms.Stereotype/
                  @xmi.idref)[Foundation.Core.ModelElement.name
                  /text()]"
            stereo="subject" />
    </implies></exists></forall></consistencyrule>
```

b) Very simple constraints

```
<relationship type="bind" id="1">
  <subject name="newExpedient" type="subject">
    <concreteElement class="User">
     <op name="getAntecedentData" />
    </concreteElement>
  </subject>
  <subject name="AuthenticaUser" type="pattern"/>
</relationship>
```

c) Composition relationship

Fig. 2. XML files

els. However, this work takes advantage of xlinkit just for:

- Checking aspects (models) at the same abstraction level together with their composition relationships. Usually, the semantic of checking in xlinkit is stored entirely in the constraint rules, but in our case, that semantic is shared between consistency rules and composition relationship because the latter specifies how elements of two or more models should be related. For this reason, we are developing a set of rules that validate and identify conflicts in the composition relationships between models.

- Checking consistency between a model and its transformation into another more abstract or refined one. In this case, the consistency rule should take account of the stored information in the transformation model about the mapping between two models.

- Using LinkBase as document to navigate into the composition relationships between aspects, as source to support automatic traceability between different models, and for assessing the impact of a change.

Following the previous example (Figure 1), the first step is to specify the composition relationships in XML by the coordinator-modeller so that xlinkit processes viewPIMs. Figure 2.c shows the bind[-User, getAntecedentData()-] relationship in

XML (Figure 1 (6)).

The second step is to create or select a set of rules for checking and establishing the different relationships specified in the XML composition document. This task can be accomplished by using the xlinkit workbench tool [31]. Figure 2.b presents a very simple rule that checks the previous bind relationship, verifying that the elements specified in the composition relationship exist in both models and are modelled by the *subject* stereotype. For example, another rule could validate that both parameters and elements linked by bind relationship are compatible and that neither one has been omitted.

The third step is to export viewPIMs to a XMI [22] document. Currently, these three previous steps are done manually.

Once the three previous steps have already been completed, xlinkit can be executed for processing the models and the composition relationships against the set of specified rules. Afterwards, the LinkBase is generated in XML format by xlinkit and it is divided into two parts:

- The first one contains those elements that are consistent between viewPIMs (a list with subject-relationships-subject).
- The second one contains inconsistent elements that have violated some of the rules against which they were checked.

Figure 2.a shows a simple LinkBase that contains two consistent elements between two viewPIMs. These elements [@xmi.id=4] and [@xmi.id=6] (Figure 1 (7) and (8)) and the composition relationship ([@id=1] Figure 1 (6)) are consistent with the r1 checking rule.

At this point, two strategies can be followed for obtaining the whole system implementation:

- Generating the code of each model (aspect) for Hyper/J [13]. In this case, the LinkBase and the composition XML are used in order to derive the composition relationship between Hyperslices and Hypermodules.
- Composing or weaving the models (aspects) at PSM level and later generating the code of a usual PSM [10].

We have chosen the first option because it is less complex that the second one. The second strategy should do a compositional transformation for weaving the aspect models and a model transformation for generating the code from PSM [18]. Moreover, since our proposal both generates aspect-oriented code and checks models (and composition relationships) at PIM and PSM level by using xlinkit, then the model compositions are not necessary. This feature is very important, because the model composition is a complex and hard task [5].

Currently, we are mapping models (aspects) and composition relationship manually from PSM to code. This mapping is based on rules stated in [29]. However, we have already started to use tools for automatically transforming these entities but we have not obtained results yet. These tools are based on QVT [25].

3 Checking Consistency and Supporting Traceability

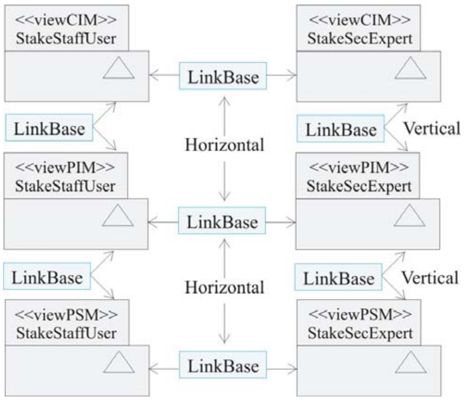
In this section, it is shown how using xlinkit consistency between models can be checked. The checking process produces the LinkBase documents that will serve as an entry for the automatic traceability of requirements from CIM to PSM.

Having done the steps described in the last sections, consistency between models can be checked. In particular, the checking process should be performed in the next situations:

- Before transforming a model into another (more refined) one, it is convenient to check its consistency with those models at the same abstraction level. This step guarantees that the source model is correct.

- After executing a transformation, the consistency between source and target models should be checked to verify the correctness of the transformation.

- When a new aspect (model) is added to the system, checking whether the resulting model is correct is necessary.



<<viewCIM>> StakeStaffUser
<<viewCIM>> StakeSecExpert
LinkBase
LinkBase
Horizontal
LinkBase
Vertical
<<viewPIM>> StakeStaffUser
<<viewPIM>> StakeSecExpert
LinkBase
LinkBase
Horizontal
LinkBase
Vertical
<<viewPSM>> StakeStaffUser
<<viewPSM>> StakeSecExpert
LinkBase

Fig. 3. LinkBases for checking partial consistencies

Checking consistency with xlinkit produces a set of LinkBase documents. As it can be seen in figure 3, having separated models allows for a partial manageable consistency checking. Instead, without the separation proposed in this work only global consistency checking is allowed which is sometimes neither possible nor desirable when large complex systems are being developed [6].

In addition, the LinkBase documents can be used as entry to an algorithm for tracing UML elements, concerns, requirements, and subjects within our framework. For example, if the Java programming language has been chosen to implement the system and a failure is obtained when a java class is generated. Then it would be very useful to be able to trace "where that class comes from", that is, what PIM and PSM elements (even CIM) are the "causes" of that class.

The traceability can be accomplished by processing the LinkBase documents in a simple downwards and upwards way. This process could be as follows: first the source element to be traced is located in the LinkBase, then its pair in next model in the path is determined. The pair is traced in that model and located in the next LinkBase document and so on. Algorithm 1 shows how to trace a use case from the

CIM to the PSM. For instance, we use the XMI models and vertical LinkBases of Security View (right part on Figure 3) for tracing the Authenticate User use case (Figure 1) from CIM to PSM. In more detail, the algorithm runs the following steps:

(i) It initializes a trace list for storing the elements to be traced from CIM to PSM.

(ii) Next the use case XMI identifier is searched in the CIM (in XMI format). It uses the //UML:UseCase/@name XPath for extracting the node which contains information about that use case, and then the searched XMI identifier is obtained by using the //UML:UseCase/@xmi.id XPath. The use case name and identifier are stored in the trace list together with the CIM's name.

(iii) Then, it looks for subjects which model the use case functionality at PIM level by using the CIM-PIMs LinkBase and the identifier found in the previous step. It uses the //xlinkit:locator/@ xlink:href XPath for extracting nodes which contain consistency links (inconsistent or consistent) between the use case and subjects. Therefore, we can obtain the siblings of that node which store the subject XMI identifiers. Moreover, it searches the subjects' names (//UML:Package/@name) by using the previous subject XMI identifiers and the //UML:Package/@xmi.id XPath. The subjects' names and identifiers are stored into the trace list together with their PIM's name.

(iv) Once the subjects at PIM level have been obtained, the next step is to obtain the subjects at PSM level. The algorithm searches PSM subjects which implement the PIM subjects by using the PIM-PSMs LinkBase and the identifiers found in the previous step. It uses the //xlinkit:locator/@ xlink:href XPath for extracting nodes which contain consistency links (inconsistent or consistent) between these subjects. Therefore, we can obtain the siblings of that node which store the subject XMI identifiers of the PSM. Again, it searches the subjects' packages (//UML:Package/@name) in the PSM by using the previous subject XMI identifiers and the //UML: Package/@xmi.id XPath. The subjects' names and identifiers are stored into the trace list together with their PSM name. This trace list can also be used for other aims such as: printing a report, storing a historical traceability, etc.

Since the previous steps are used for tracing a use case from CIM to PSM, the needed changes for tracing another kind of element such as subjects or classes are minimal. For instance, in order to trace a class from PIM to PSM, the third and fourth steps should only change the use case reference by a class reference and the //UML:Package string by the //UML:Class string in the XPaths. Moreover, our framework could have other extra PIM or PSM levels. In this case, the trace algorithm should only repeat the third and fourth steps for tracing from the CIM to the lower PSM.

At any rate, if the use case is not mapped onto well-modularized entities (subjects in our proposal) at the PIM and PSM level, that is, its functionality is scattered over several classes, then the information about traceability is too large because the mapping is not as lineal as in our proposal. This information is even larger and

Algorithm 1 .

```
trace-list :=initialize
with CIM in XMI
    xmi-id-CIM := get the ID for the searched use-case
    add to trace-list the xmi-id-CIM and use-case's name
end-with
with the CIM:PIM LinkBase
    forall xlinkit:locator with xmi-id-CIM
        xmi-id-PIM := get the element ID for the sibling of this node
        with PIM in XMI
            add to trace-list the element with id=xmi-id-PIM and element's name
        with the PIM:PSM LinkBase
            forall xlinkit-locator with xmi-id-PIM
                xmi-id-PSM := get the element ID for the sibling of this node
                with PSM in XMI
                    add to trace-list the element with id=xmi-id-PSM and element's name
            end forall
    end forall
end-with
```

Algorithm 1. Tracing a use case throughout the MDA framework

more complex for handling it at PSM level.

This algorithm is able to trace those elements that have been checked using xlinkit, that is, whatever kind of element which appears in the LinkBase.

One of the benefits of traceability is the ability to predict the impact of change [17]. Once the system has been developed, if a change is needed either in requirements, in design, or in an element, it would be desirable to know what elements in lower and upper levels will be affected by that change. Since our proposal can trace elements from top to bottom and bottom to top, by means of processing the LinkBase, the elements of other level which could be affected by such change can be obtained. For example, if a requirement is removed at the CIM level, this change could bring on several changes for removing subjects at PIM and PSM levels, adapting composition relationships, modifying mappings between abstraction levels, etc. Therefore, these changes could be too costly and they could be performed, delayed or cancelled.

The same procedure can be used to trace and control changes in a horizontal direction. In this case, we process the LinkBase which relates different models at the same abstraction level, that is, the horizontal LinkBases.

Summarizing, on one hand, SOM provides a good alignment between abstraction levels (CIM, PIM and PSM), and therefore identification of concerns and requirements which are affected after a change in any abstraction level of the MDA framework is facilitated. On the other hand, the LinkBases can be used for tracing requirements, concerns, elements, and subjects throughout the MDA framework.

Also, model transformations automate and make agile changes in the system. Thus this work integrates all these technologies in a suitable way for Model-Driven Development.

4 Towards a consistent incremental development

Another impotant feature of the work presented here is the support of the incremental development process of large complex systems by integrating Subject-Oriented Modelling and MDA. This is due to SOM being able to add or modify behaviour and structures in a model already implemented additively instead of invasively. For example, in our case study, once the three abstraction levels of the security aspect (viewCIM, viewPIM, and viewPSM) have been modelled, the system may need a change in the specification of its requirements: "the access control will be made on a secure flow by SSL". This modification will involve creating a new use case that "extends" the previous one of security. Also, this change implies the modification of PIM and PSM entities, but these are accomplished additively.

Thus, a new subject will be designed for appending the new security behaviour on "AuthenticateUser" subject (Figure 1 (8)) without modifying the existing one. The same process is repeated exactly for the PSM. Therefore, this supposes an improvement in the evolution and maintenance of the software system by making changes additively.

In addition, these kinds of additive changes can be easily managed by xlinkit. Xlinkit allows us to do an incremental analysis of the consistency, that is, it extracts the differences between a XMI model before and after its modification by analyzing only those elements that could have been inconsistent after these modifications. Therefore, as the change introduced in the system is well identified and isolated, then xlinkit will only check the new aspect and the elements related to it.

5 Related Works

Reina et al. [1] propose the using of different aspect oriented modelling proposals at PSM level. The reason that the authors argue for this is that these proposals are platform specific. Thus, they suggest to use Domain-Specific Languages (DSL) for each aspect that is modelled at PIM level. The problem is that for each new system aspect it is necessary to use a new DSL (based on meta-model extensions or UML profiles), and therefore, developers must work with several languages at the same abstraction level. In addition, this approach proposes a set of models that are related to Web technology (presentation, navigation, security, etc) at PIM level and really it could seem that this level is not technology independent.

Ivar Jacobson analyzes in [15,16] the problems of tangling and scattering in component diagrams during use cases guided software development. He solves these problems using multi-dimensional separation of concerns. The dimensions that he establishes are use cases and classes. However he does not give details about composition, nor transformations of models, rules of composition, structural relationships, etc.

Kulkarni et al. [33,34] integrate separation of concerns into MDD for facilitating traceability, reusability and evolution in a software system. In order to carry out this separation, an abstract template meta-model is used to separate system concerns in a hierarchical way at model and code level. But the abstract template itself couples some aspects to others.

The work presented in this paper is similar to the Theme approach [11]. In that approach, software requirements are specified with Theme/DOC using Action Views, and analysis and design stages are modelled by Theme/UML using themes (subjects and Composition Patterns). Thus, this approach should compose models for checking and validating them, but this task is not necessary in our proposal due to the use of xlinkit. Moreover, they do not propose anything on aspect or model transformations, and they only present analysis and design stages without focusing on possible intermediate stages or refinement of models. That is, our proposal explicitly separates the software system design into two stages, one technology independent stage and another specific one for MDA compliance.

Our proposal is very similar to the approach, presented by Robert France et al. [2,10], which is also based on MDA. The most important difference is that it distinguishes between a core model and other aspect models that will be applied to the former, therefore, it is an asymmetric aspect approach and our proposal is a symmetric one [23] [35]. Thus, this proposal is highly influenced by AspectJ [12], while our work is closest to the multi-dimensional separation of concerns [24].

6 Conclusions a Future Works

In this work we have presented a MDA framework by proposing aspects of a system as different models keeping them separated from the CIM to the PSM. In addition, the viewModels (an aspect developed for the three abstraction levels) can be developed by different specialist modellers in a consistent and incremental way by using the xlinkit tool. Also, the proposal integrates a flexible and external mechanism for automating traceability of concerns, requirements, and other abstract artefacts on MDA. So, the software system maintenance and evolution can be carried out in a controlled way through the identification of elements that can be affected after a change in the system. In this case, the Subject-Oriented Modelling allows us to design these changes additively instead of invasively.

We argue that model composition is a hard and complex task that can be too costly. Thus, we propose to generate aspect-oriented code and use xlinkit for checking models in order to avoid model compositions.

An important open question is to study how ViewPIMs and ViewPSMs internal organization could change if other kinds of diagrams to model system requirements in CIM are used: activity diagrams, workflows, domain models, mixtures of these, BPMS, etc. In addition, we can look for the most appropriate way to separate and make "slices" of each model accordingly to the system requirements.

Nowadays, we are working on a viewModels repository that covers all MDA levels. That is, our aim is to have aspect models repositories that cover the three abstraction levels for reusing them in different systems in the same domain.

As already indicated, we are making a catalogue with rules on constraints of subject compositions in order to execute a strong checking at model level. This will validate the composition at that abstraction level so that the code generated from this model won't have consistency problems.

References

[1] A.M. Reina, J. Torres, and M. Toro. Towards developing generic solutions with aspects. Workshop in Aspect Oriented Modelling held in conjunction with the UML 2004 Conference, oct 2004.

[2] A. Solberg, D. Simmonds, R. Reddy, S. Ghosh, R. France, "Using Aspect Oriented Technologies to Support Separation of Concerns in Model Driven Development", Accepted in the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005), Edinburgh, Scotland, July, 2005

[3] Aspect-Oriented Software Development Web Site, http://www.aosd.net, 2005

[4] A. Rashid, A. Moreira, J. Arajo. Modularisation and Composition of Aspectual Requirements, AOSD 2003, Boston, USA, 17-21. March, 2003.

[5] B. Baudry, F. Fleurey, R. France, R. Reddy. Exploring the Relationship between Model Composition and Model Transformation. In Workshop on Aspect Oriented Modelling held in conjunction with the AOM 2005 Conference, oct 2005.

[6] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging Inconsistency in Software Development. IEEE Computer, 33(4):24-29, April 2000.

[7] C. Nentwich, W. Emmerich, A. Finkelstein and E. Ellmer, "Flexible Consistency Checking," ACM Transactions on Software Engineering and Methodology, vol. 12, pp. 28-63, 2003.

[8] Dijkstra, E.W., A Discipline of Programming, Prentice-Hall, 1976.

[9] D. L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. In Communications of ACM, Vol. 15, issue 12. ACM, December 1972.

[10] D. Simmonds, A. Solberg, R. Reddy, R. France, S. Ghosh. An Aspect Oriented Model Driven Framework. "An Aspect Oriented Model Driven Framework", Accepted to Ninth IEEE "The Enterprise Computing Conference" (EDOC 2005), Enschede, Netherlands, 19-23 September, 2005.

[11] E. Baniassad and S. Clarke. Theme: An approach for Aspect-Oriented Analysis and Design. In Proceedings of the 26th ICSE, 2004.

[12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, and J. Irwin. Aspect-oriented programming. In ECOOP'97Object- Oriented Programming, 11th European Conference. LNCS 1241, pages 220- 242, 1997.

[13] 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

[14] IEEE Software. Special issue on Model-Driven Development. Volume 20, number 5. September/October 2003.

[15] Ivar Jacobson: Use Cases and Aspects Working Seamlessly Together. In Journal of Object Technology, vol. 2, no. 4, July-August 2003, pp. 7-28.

[16] I. Jacobson, Pan-Wei Ng. Aspect-Oriented Software Development with Use Cases. Addison Wesley Professional, 2004.

[17] J. Bakker. Traceability of Concerns. Master's thesis, University of Twente, April 2005.

[18] Krzystof Czarnecki and Ulrich W. Eisenecker. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Boston, 2000

[19] M. Aksit. Systematic analysis of crosscutting concerns in the MDA design approach. Symposium How Adaptable is MDA?. University of Twente, May 2005.

[20] OMG. MDA Guide V1.0.1. Document – omg/03-06-01

[21] OMG. UML 2.0 Superstructure. Document – ptc/04-10-02

[22] OMG. XMI 2.0. Document- formal/03-05-02

[23] P. Amaya, C. Gonzlez, J.M. Murillo. MDA and Separation of Aspects: An approach based on multiple views and Subject Oriented Design. In Workshop on Aspect Oriented Modelling held in conjunction with the AOSD 2005 Conference, mar 2005.

[24] P. Tarr, H. Ossher, W. H. Harrison, and S. S. Jr. N degrees of separation: Multi-dimensional separation of concerns. In Proceedings of the ICSE, pages 107119. IEEE Computer Society Press,1999.

[25] QVT-Merge Group. Revised submission for MOF 2.0 Query / View / Transformation RFP (ad/2002-04-10). OMG, 2005.

[26] R. Filman, T. Elad, S. Clarke, M. Aksit (Editors) "Aspect-Oriented Software Development" Addison-Wesley, 2005.

[27] R. B. France, I. Ray, G. Georg, and S. Ghosh. An aspect-oriented approach to design modeling. IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 151(4), August 2004.

[28] S. Clarke. "Extending standard UML with model composition semantics" in Science of Computer Programming, Volume 44, Issue 1, pp. 71-100. Elsevier Science, July 2002.

[29] S. Clarke, Robert J Walker. "Separating Crosscutting Concerns across the Lifecycle: From Composition Patterns to AspectJ and Hyper/J". [TCD-CS-2001-15], Trinity College, Dublin and UBC-CS-TR-2001-05, University of British Columbia. May 2001

[30] S. Clarke, W. Harrison, H. Ossher, P. Tarr. "Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code". In Proceedings of OOPSLA) Denver, Colorado U.S., November 1999.

[31] Systemwire Web Page. Xlinkit Rule Workbench. http://www.systemwire.com, 2005

[32] S. Sendall, W. Kozaczynski. "Model Transformation: The Heart and Soul of Model-Driven Software Development," IEEE Software, vol. 20, no. 5, pp. 42-45, September/October 2003.

[33] V. Kulkarni, S. Reddy: Integrating aspects with Model Driven Software Development. In International Conference on SERP'03. Las Vegas, USA. June 2003

[34] V. Kulkarni, S. Reddy: Separation of Concerns in Model-Driven Development. IEEE Software 20(5): 64-69 (2003).

[35] W. Harrison, H. Ossher and P.Tarr. Asymmetrically vs. symmetrically organized paradigms for software composition. Technical report, IBM, 2002.