

# Dealing with crosscutting concerns in a Model Based Software Production Method

José Iborra

Oscar Pastor

Vicente Pelechano

Department of Information Systems and Computation  
Technical University of Valencia  
Campus de Vera, 46022 Valencia, Spain  
{jiborra, opastor, pele}@dsic.upv.es

## ABSTRACT

The OO-Method is a Model-based Code Generation Software Production Process that is based on object-oriented concepts. To face it from an Aspect-Oriented domain engineering point of view is the central goal of this paper. We want to do that for two main reasons: i) to fix which conceptual primitives should be required to accomplish aspect-oriented conceptual modeling ii) to define a precise subsequent map between aspect-oriented domain analysis concepts and their corresponding software counterparts in a given software architecture. This would make possible to define a Model Compiler based on aspects-based concepts, properly linking MDA-based proposals with Aspects.

With this objective in mind, the paper analyzes the OO-Method approach in the light of Aspect-Oriented Software Development (AOSD), with the concrete intention of finding out whether OO-Method deals with crosscutting concerns, and arguing on what could be gained by introducing techniques based on AOSD.

## 1. INTRODUCTION

The OO-Method is a model based software production method that appeared in the middle of the nineties. It enhances OASIS, an object oriented, formal specification language, with a notation, a methodology and an abstract execution model. On the basis of this execution model, [1] shows how an abstract architecture can be designed and a complete application can be generated.

In this document we analyze the OO-Method approach in the light of Aspect Oriented Software Development (AOSD), with the goals of finding out whether OO-Method deals with crosscutting concerns, and what can be gained by introducing techniques based on AOSD. The main paper objective is to fix how to see the OO-Method approach in terms of concerns, in order to define a functional mapping between those OO-Method concerns and their software representations, to minimize crosscutting and make easier the corresponding model-to-code transformation process provided by the OO-Method.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
ICSE'06, May 20-28, 2006, Shanghai, China.  
Copyright 2004 ACM 1-58113-000-0/00/0004...\$5.00.

Summarizing, we claim to present three main contributions: i) a paractical application of Aspect-Oriented-based techniques to a MDA-based software production process. ii) to identify OO-Method conceptual primitives as concerns. iii) to propose an optimized transformation tool, aspect-based, that should improve the current model transformation techniques provided by the method. In particular, the paper shows how this would work for a selected subset of the introduced OO-Method-based concerns.

In order to reach this goal, we must first obtain a catalog of the concerns present in OO-Method. These will be enumerated in section 2. Sections from 3 to 7 deal with the analysis of the crosscutting concerns, and in section 9 we present the conclusions of our work.

## 2. IDENTIFICATION OF CROSSCUTTING CONCERNS IN THE OO-METHOD

The OO-Method provides a set of conceptual constructs which enable analysts to capture the requirements of an Information System, based on a formal language called OASIS. The approach is based on the idea of providing a model-to-code automated transformation process, following a model compiling strategy.

1. [User] Identify the user
2. [System] Obtain the object system view
3. [User] Service Activation
  - 3.1 [System] Identify the object server
  - 3.2 [User] Introduce service arguments
  - 3.3 [System] Send the message to object server
  - 3.4 [System] Check state transition
  - 3.5 [System] Check preconditions
  - 3.6 [System] Valuations fulfillment
  - 3.7 [System] Integrity constraints validation in the new state
  - 3.8 [System] Trigger relationship test

**Figure 1 The OO-Method abstract execution model**

An abstract execution model fixes how to execute an OO-Method Conceptual Model. The template shown in Figure 1 describes the way in which users would interact with an OO-Method system, where we have outlined in every task whether the user or the system has the responsibility. As soon as a user is identified and has access to the system view, she can then activate any available services, which can be object queries (to explore the state), events, and transactions, and either locally or served by other objects. Any service activation has a number of steps as depicted on the figure.

Going back to our intention of linking conceptual primitives of OO-Method and concerns, the separation of concerns principle [2]

proposes encapsulating concerns in separate entities, in order to localize changes and deal with an important issue at a time. For example, the UML uses different models to deal with different properties of a problem domain separately. In this section we present a first decomposition of the concerns present in the OO-Method.

A concern is described as *a thing in an engineering process about which it cares* [3]. Since we are limiting ourselves to the domain of OO-Method with the goal to come up with an architecture, we will consequently use a more concrete definition. We consider a concern to be a feature or set of features with a cohesion between them, among the set of features and constructs provided by the OO-Method modeling language.

A concern is crosscutting if *the implementation is scattered throughout the rest of an implementation* [3]. This definition is too bound to the implementation level for our interests, and thus we have looked for another definition more flexible in terms of the abstraction and representation level. [4] states that when using feature modeling in the analysis phase and an OO approach in the design phase, it is important to assure traceability. The same is stated in [5] but using use cases instead of feature modeling. In both cases the problem is the same: abstractions in the upper level don't find a direct mapping to abstractions in the level below. Generalizing, we can state that a concern is crosscutting if it cannot be properly modularized with the composition and decomposition mechanisms available in its level of abstraction.

A change of representation system (generally due to going down in the abstraction level) will thus probably increase the number of concerns which are crosscutting. This happens for instance in the transition from requirements to design & implementation, or as mentioned before, from domain analysis to domain design.

We start the analysis applying the well-known problem/solution space dichotomy. The OO-Method makes use of a structure of views to decompose the problem space. An original version provided four different views of the Conceptual Schema: the Object View, the Dynamic view, the Functional View and the Presentation View. Each of these partial views may at the same time have different sub-views. For instance, the Dynamic view deals separately with State transitions and Object Interactions. All together, they conform the Conceptual Model of the system domain.

It is our opinion that thanks to the high level of abstraction, the OO-Method does a fine job in helping the designer to work with one concern at a time. Hence, our goal in this section is not to find problems of crosscutting nature in OO-Method specifications (system specifications produced using the OO-Method). What we aim at is identifying those concerns that may become crosscutting when transitioning to the solution space.

In Table 1 we make a first decomposition of the concerns found in the OO-Method, based on the listing of features shown in Figure 2 and in the execution model. The concerns identified include thus both domain features and non-functional requirements. We have taken Java as the reference OO implementation language, but the results are applicable to other mainstream OO languages too.

The table shows that many of the abstractions in the OO-Method have a direct mapping to the OO implementation language, since both are using the same decomposition mechanism (objects). OO-Method complements these OO abstractions with i) a process

language, and ii) temporal logic for expressing constraints. These missing abstractions must be implemented in the solution space and will naturally tend to be a source of crosscutting concerns.

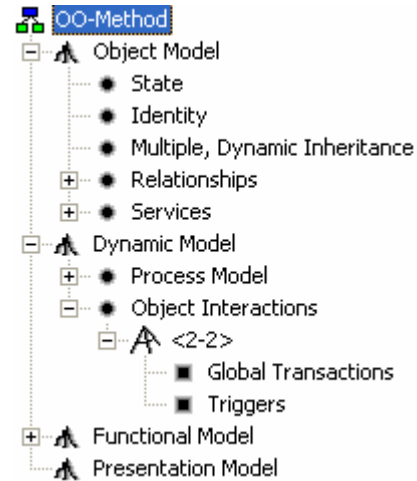


Figure 2 A listing of OO-Method features

In addition to these, the set of concerns coming from Non Functional Requirements (NFR), mainly from the execution model, prove to be crosscutting too.

In the context of Automated Software Production, the step from problem space to solution space is generally automated thanks to a model compiler, a term that denotes a mixture of model transformations and code generation. The development of the model compiler is a very complex task, due mainly to the impedance mismatch between the two spaces. This impedance mismatch, in our opinion, can be described as a set of crosscutting concerns, in the sense that if the mapping was direct, the generation of every concern would be independent with respect to the others, and composition should be trivial. In contrast, when dealing with crosscutting concerns the model compiler must take into account the interactions between them, and composition becomes much more involved.

The complexity rises exponentially when we introduce configurability in our compiler. For instance, with as few as three options for every concern the model compiler would have to deal with 243 (3<sup>5</sup>) possible combinations (we assume that only crosscutting concerns introduce complexity in this case).

AOP is a new approach introduced at [6] which offers promising solutions to these problems. The application of AOSD techniques provides enhanced modularization mechanisms oriented at encapsulating these crosscutting concerns. [7] outlines a number of ways in which this kind of generative setup can benefit from AOSD. For instance, it can improve the maintainability of product line members and the product line definition itself. It can also provide the integration mechanisms for composing concerns together. The aspectual weaver can then relieve the model compiler from the involved composition described before. In fact, the benefit does not lie only in the weaving. AOSD provides a conceptual framework more appropriate to describe integration and dependency relationships between concerns. By having a concrete integration mechanism, we can now talk and reason about integration problems.

Name	Description	Crosscutting?
Object State	Defined in OASIS as the value of the set of attributes of the class	Not when using OO in the solution space. There is a direct mapping.
Object Identity	A tagged set of attributes of a class constitute its identity	Generally not when using OO in the solution space.
Relationships	Including association and aggregation relationships	Not when using OO. There should be a direct mapping.
Inheritance	OASIS v2.0 offers multiple, dynamic inheritance.	Mostly not.
Service Invocation	Means of communication between objects. In OO-Method this includes standard event invocation and implicit invocation	No.
State Changes (Valuations)	Means to produce changes in the state of an object. In OO-Method these means are called valuations, and can only affect the state of the current object. Valuations can only be defined in events.	No.
Invariants	Means to enforce a set of legal states. An object is never allowed to be in a non legal state. In OO-Method these means are called integrity constraints and are defined in first-order propositional logic.	Yes. It crosscuts the State Changes and the Object State concerns in order to validate a given change.
Preconditions	Conditions that may preclude the invocation of an event	No.
Agent Security	OO-Method provides an "is-agent-of" relationship which goes from a class to a class element (attribute or event). In the absence of this relationship, a class C cannot access the attribute or event desired.	Yes. It crosscuts the Service Activation concern in order to obtain the source of a message.
Dynamic model (Process Definitions / Triggers)	Several facilities within OO-Method deal with implicit Message Invocation. These include: <ul style="list-style-type: none"> <li>Triggers, conditions on state that trigger service activation</li> <li>Shared events, by which the invocation of an event in an object produces as a side effect the implicit invocation of an event in another object</li> <li>Process Model, where the lifecycle of a class is seen as a graph of event transitions, and the process model defines the legal transitions in the current state.</li> </ul>	Shared Events are not crosscutting. Triggers are, and the Process Model is too. There is no mapping from these OO-Method features to mainstream OO languages
Transactional events	For simple events, OASIS establishes that these have the property of non-observability of intermediate states. Additionally, it allows for grouping sets of events into transactions, which have the property of non-observability and an all-or-nothing policy. I.e. rollback if failure	In an executable translation of an OO-Method spec., transactionality crosscuts at least with Service Activation and Object State.
Persistence	OO-Method assumes that, in a running system, all instances are persistent from one run to another. It doesn't specify the kind of persistence system though.	Crosscuts with Object State, Object Identity, Service Activation, and Relationships concerns.
Presentation	An enhancement to the original OO-Method approach, the presentation model contains the information about the visual representation of the system, on a per-class basis.	This is probably not a single concern, but a set of them. We won't deal with it in this document.

**Table 1 OO-Method Concern catalog**

Along this paper we are going to use an example OO-Method conceptual model of a library management system from herein referred as BIB, a classical example [8] in the context of OO-Method. The BIB system encompasses four main concepts: student, book, loan and librarian, where a student can loan up to 10 books. A student has thus a number of loans, where each loan relates a student with a book. A book can participate in one loan at most. If it is not participating in any loan, the book is available for students. We can express this with a precondition saying that a book can only be borrowed when it is available. The class diagram in Figure 3 shows these relationships. Take into account

that this diagram is just an abstract representation: all the functional requirements are contained in an OASIS specification document, an excerpt of which is included in an appendix.

The goal now is to produce an implementation that fulfills the requirements. There are of course many designs that will do, but in this context we are interested in an automated translation to a target programming language.

We have studied in detail all the crosscutting concerns enumerated in the table except Presentation, but due to space

constraints the focus is going to be on the Invariants, Agent, and Dynamic Model ones.

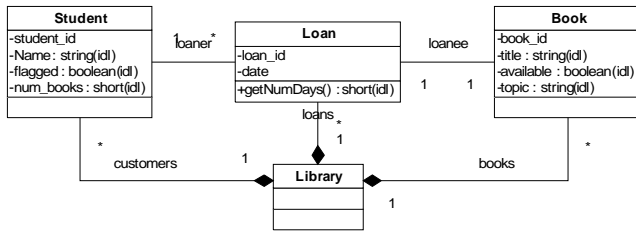


Figure 3 UML Class Model of the BIB system

### 3. THE INVARIANTS CONCERN

Classes in OASIS can have first order logic formulas attached expressing constraints over their state. The abstract execution model establishes that these constraints must be checked after every state change. For instance, the Student in the BIB system class has a constraint stating that it cannot own more than 9 loans.

In OASIS, events are the only means to produce changes on state. Events are implemented as methods in Java. Thus, methods realizing events must include specific calls to some method `validate_constraints()`. Additionally, since constraints can quantify over associations, it is necessary to verify the constraints after the occurrence of an event in any associated instance! Hence, the code for checking an invariant becomes scattered among a number of methods in different classes.

Design Patterns can be useful to help modularize this crosscutting code. In this case, the Template Method Pattern can be very helpful. We can encapsulate all the details about constraint enforcing in a template method `execute_event()` in a base class `OASIS_Class`. The abstract method `obtain_constraint()` will be overridden by implementors. The constraints are obtained from the closure of associated OASIS classes.

Let's anyway explore what can be gained using an Aspect instead of the Template Method. Note that we won't try to use an AOP rendition of the pattern here, but instead base the solution in AOP concepts and practices. Figure 4 shows an overview of the Invariants aspect, which implements the Enforce Constraints concern. The notation used is the AsideML modeling language [9]. This language extends UML with constructions for representing aspects. An aspect is represented by a package-like box with a diamond. It is composed of internal structure, crosscutting interfaces and template parameters. The internal structure declares the internal attributes and methods, it is represented as a UML class with no name in the central part of the aspect. Crosscutting interfaces are represented as UML interfaces with name. A crosscutting interface specifies when and how the aspect affects the bound/crosscut classes (more on binding later). Crosscutting interfaces are composed of additions, refinements and redefinitions.

Finally, the notation uses a dashed arrow to represent the crosscutting relationship, which relates/binds a crosscutting interface to one or more classes, instantiating the template parameters to concrete elements of the class.

In the figure we present an aspect that solves this concern in a generic way. Another option would be to generate individual aspects for every OASIS class, where every aspect would contain the constraints for its associated class. In the solution presented

here however, the constraints live in a metamodel repository. The aspect has the following participants:

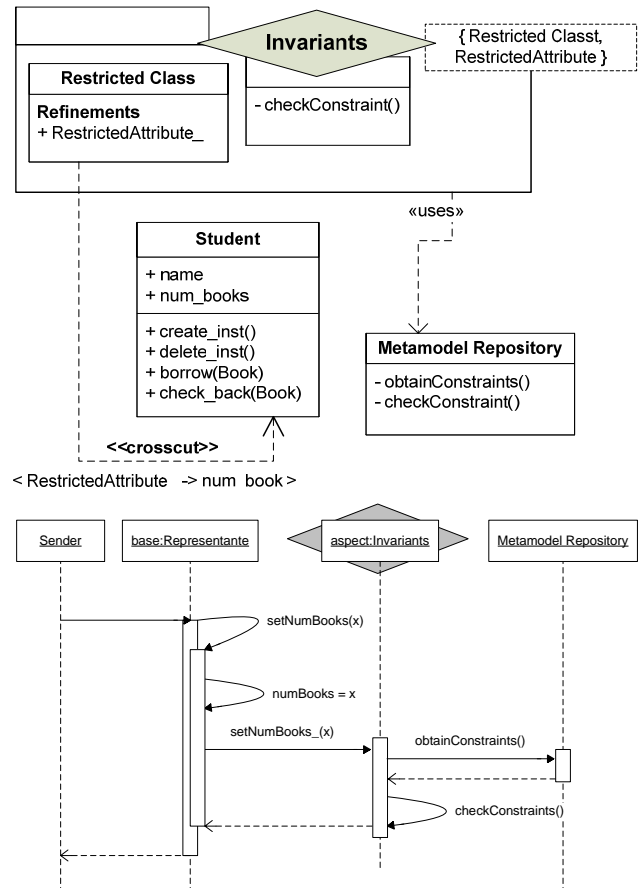


Figure 4 The Invariants Aspect

#### OASIS Representant

A crosscutting interface that represents a given OASIS class. We are interested in the restricted attributes, i.e. attributes which appear in the definition of some constraint.

#### Invariants Aspect

Implements the concern in a generic way.

#### Metamodel Repository

The metamodel repository contains an accessible representation of the conceptual model, including the information about the constraints that apply to a given attribute in an OASIS class. Its responsibility here is to extract the constraints present in all classes belonging to the reflexive and transitive closure of this class with respect to its associations, and determining which of those constraints affect the given attribute. Note that this computation can be done statically if needed.

The purpose of the Invariants aspect is to decouple the concern of enforcing constraints from basic object concerns. It accomplishes this objective by localizing all the constraints checking related code in the aspect. Whenever a constraint fails to check, it simply raises an exception and, since all events in OASIS are assumed to be transactional, we can expect the transactional concern to take care of restoring the object to its previous state.

The figure shows a binding to the num\_books attribute of Student, in order to enforce the previously mentioned constraint. More generally, it is necessary to create bindings with every class-attribute pair which has constraints associated.

#### 4. THE AGENT SECURITY CONCERN

OASIS has an access control mechanism in the agent construct. Without an agent declaration stating so, a class shouldn't be able to access the members of another class.

Java access control mechanisms on the other hand are mainly package-level, and cannot accommodate the agent construct directly. From our point of view, this turns the whole agents issue from a language feature into a security non-functional requirement of crosscutting nature.

First, it is necessary to know who the active agent is. In OASIS the active agent is the object sending the message in a service invocation, but in Java it is not possible to know which class is invoking a method. The OO-Method execution model introduces a different rendition of this concept by establishing that there is a login phase in the system, where the current agent is logged in and stored somewhere for the rest of the session. This is the agent used in all the future invocations in the system.

Once we have the current agent accessible in some context, every method representing an OASIS event must check that the current agent holds access rights. Also, accessor methods for properties must do the same. Hence, the code for this concern would be scattered around the code base. The Template Method pattern is again useful, but at this point we would have a highly tangled template method. And it does not fully modularize this concern, since we still have to deal with the authentication process in the GUI tier and the interface to retrieve the current agent.

Another approach is to localize this concern in the Interface Tier. A login component authenticates the user and then the System builds a GUI adapted to this user access rights, showing only the classes and methods that this user has access to. This is a less scattered solution, but it doesn't enforce security at the API level.

Figure 5 proposes a basic aspect that fully deals with the Security Concern. We proceed with the participants description:

##### The Restricted Class

A crosscutting interface that refines a class by enforcing an access control on some of its methods. In our particular case, restricted methods are those representing events or transactions, and also getters for OASIS properties.

##### The Entry Point

Where authentication takes place. It will probably reside in the GUI Tier. The crosscutting interface simply refines a method deemed an authentication point by monitoring and storing its result.

##### The Security Aspect

Solves the Agent Security concern in a generic way. In addition, it stores the current agent and provides a public OO standard interface to retrieve it.

The diagram shows how the Restricted Class crosscutting interface is bound to the Student class. The interface must be properly bound to every OASIS representant class in the code.

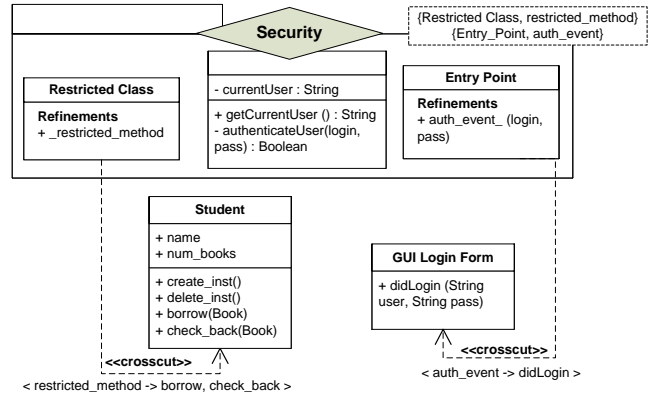


Figure 5 The Security Aspect

Note that this solution is highly reusable and independent of OO-Method<sup>1</sup>.

#### 5. THE DYNAMIC MODEL CONCERN

The dynamic model of an OASIS Class comprises two notions: its process model, which declares the set of valid states and state transitions, and its trigger declarations. A process model can be seen very much like a UML State Diagram. Figure 6 shows an example.

A trigger is an implicit invocation of a service when some predicate is fulfilled. These predicates can quantify only over the state of the object. Triggers can invoke services in the owner object, another related object, or the entire population of objects for a given class. In the BIB System, NotTrusted is a specialization of Student which has a trigger for invoking its service redeem whenever it has returned all its loans.

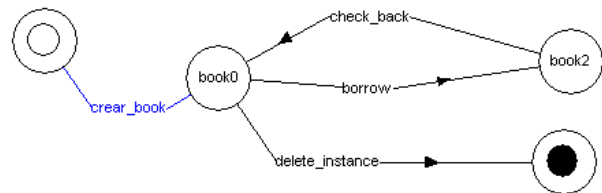


Figure 6 The Book class Process Model

The diagram in Figure 7 shows the aspect that realizes a basic implementation of this concern. Once again, we present for a generic solution comprising one generic aspect collaborating with a metamodel repository. Recall that the other option is using specific aspects for every OASIS Representative class.

The participants in the aspect are:

##### Process Model

A crosscutting interface that represents the Process Model for a given OASIS class. Every OASIS Representative class should be an instance of this crosscutting interface, which inserts a new attribute to keep track of the current state in the Process Model, and validates transitions between identified events. The figure shows it bound to the Student class, where the event parameter is bound to all the methods in the class, since in

<sup>1</sup> In fact, the security aspect proposed here has been inspired by the AWARE project at <http://docs.codehaus.org/display/AWARE>

this particular case all the methods happen to be representatives of OASIS events.

### Triggers Model

A crosscutting interface that represents the Triggers Model for a OASIS class. It does so by monitoring changes in the set of attributes affected by trigger definitions. The figure shows a binding to the NotTrusted class, monitoring the (inherited) num\_books attribute which appears in the trigger definition.

### Dynamic Model Aspect

The aspect that implements the Dynamic Model Concern in a generic way.

### Metamodel Repository

Same as in the Enforce Constraints Concern, the metamodel repository contains an accessible representation of the conceptual model.

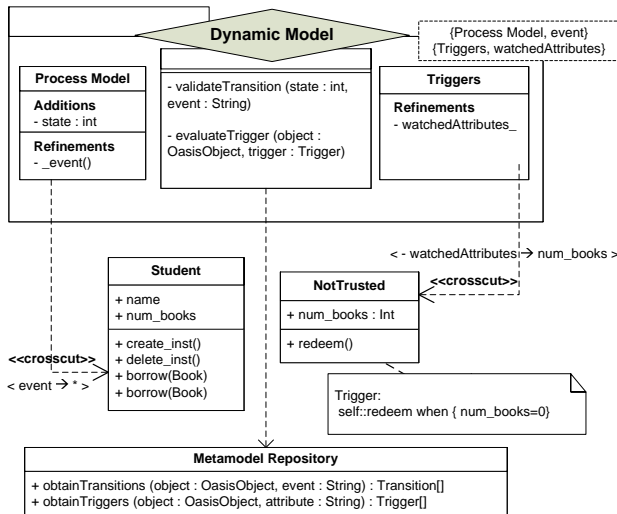


Figure 7 The Dynamic Model Aspect

We must say that this is a simplified implementation which hides some annoying details. In particular, the invocation of the effect of a trigger cannot happen until the current flow of execution has completed, i.e. until the current open transaction has been closed. This calls for some kind of context which keeps track of activated triggers and executes them on closing the transaction.

In order to keep the diagrams manageable, we are not showing a dependence that the Dynamic Model aspect has with regard to the Security concern. Evaluation of transitions and triggers requires the knowledge of the current agent of the system, which is provided by the Security concern.

## 6. EMBEDDED DSL'S WITH AOP

In this section we briefly comment on an approach to handling the bindings of the aspects discussed in the previous sections. The solution proposed here follows an innovative approach, which was first sketched in non-academic contexts by [10] and [11]. A similar approach is put in practice in the upcoming EJB 3.0 specification<sup>2</sup>.

It revolves around the idea of extending the implementation language with the concepts and composition mechanisms present in the OO-Method level. This would be done ideally at the meta level of the language if available, but in the cases considered, the programming languages do not provide a meta level. Java and C# for instance provide a remarkably powerful reflection mechanism, but access is limited to read-only.

There is another capability provided by both of these languages that can be very useful: metadata. Java and C#, among others, allow attaching (properly typed) annotations (also called tags) to

```
@Test(groups={"checkintest", "broken"})
public void testMethod2() {
    ....
}
```

Listing 1 An example of Java 5 annotations

several meta objects, such as method declarations, fields and classes. These can be used to, in a way, extend or customize the language, although these annotations do not have executable semantics. Listing 1 shows a Java method with a Test annotation.

Cutting edge AOP tools have incorporated this kind of metadata into their join point model. It is possible to quantify over elements of the program using these tags and attach advice to them, providing in this way an operationalization of the semantics underlying the tags. It is also possible to easily introduce new tags in selected join points in an oblivious way.

The proposal is to create a kind of Embedded Domain Specific Language (EDSL<sup>3</sup>) which closely reproduces the expressive capabilities of OO-Method, but with executable semantics. We believe that this technique is interesting, since traditionally the mentioned OO languages have not been well suited for the task of EDSL creation<sup>4</sup>.

In our case, the idea is to produce a set of annotations that helps to complement a Java Class declaration with the features present in OO-Method. We want to tag some methods as being *events*, while others will be *transactions*. Every method will be able to include a set of *preconditions*. In the same line, a class will be able to define *constraints* and *triggers*. Formulas will be used in these definitions, employing the same first-order logic used in OO-Method. Finally, means are provided to associate a process description to a class. Note that this makes unnecessary the use of an external model repository.

The generic aspects we have created automatically bind over these annotations, giving them executable semantics as said.

To see how the snippet in Listing 2 illustrates our proposition of this being a kind of EDSL, compare it with the excerpt of the OASIS Specification included at the appendix.

We admit that we have not yet taken this approach far enough to offer comments on the results. All we can say is that it looks promising, but we still have to test it on more complex cases. We have also to fix until what degree this body of annotations is able to replace the need for a true model repository in complex cases.

<sup>3</sup> For an introduction to EDSLs, see [12]

<sup>4</sup> Examples of languages that are well suited for the construction of EDSLs are Lisp, Scheme and Haskell. In the OO paradigm we should cite Ruby.

<sup>2</sup> Final draft for JSR-220 at <http://jcp.org/en/jsr/detail?id=220>

We have the support of the company that commercializes the OO-Method model through the OlivaNova Model Execution tool, to analyze these issues in depth in further works.

```
@OasisClass(constraints="num_books<=10" )
class Student {
  @OasisId int student_code;
  .....
  @Event
  @Precondition ("num_books > 0")
  @State (from="STUDENT1" to="STUDENT1")
  public void checkBack(Book book) {...}
  .....
}
```

**Listing 2 An OO-Method annotated Java class**

## 7. CONCLUSION

AOSD formalizes a powerful set of techniques. By means of this case study, we are trying to take advantage of them for improving the construction and design of model compilers and execution models. It is our belief that Aspects can improve the current software production process based on Model-Transformation techniques. To do that, it is needed to fix how the conceptual primitives used for Conceptual Modeling purposes can be seen from the point of view of aspects. In this paper we report on our preliminary concrete results on this research issue. We have analyzed the conceptual constructs of a MDA-based method (the OO-Method) in terms of aspect-based concepts, with three main objectives: i) a practical application of Aspect-Oriented-based techniques to a MDA-based software production process. ii) to identify OO-Method conceptual primitives as concerns. iii) to propose an optimized transformation tool, aspect-based, that should improve the current model transformation techniques provided by the method.

Using aspects to go from the Problem Space to the Solution Space makes things clearer, both conceptually and practically. We have checked that using Aspects, composability of final software components is much more amenable, because i) since they have a well defined structure, new aspects can be introduced without altering the existing ones. ii) they offer mechanisms to explicitly regulate composition issues, such as precedence.

Next steps will be oriented to accomplish a realistic empirical validation, to de bone with the support of the company that is commercializing the tool that implements the OO-Method.

## APPENDIX: OASIS SPECIFICATION FOR THE STUDENT CLASS

```
class student
  identification
    student_id:(student_id);
  constant attributes
    student_id : nat;
    nombre:string towards(1,1)
  variable attributes
    num_books : nat towards(1,1);
    avisado : bool towards(1,1);
  constraints
    { num_books <=10 ; }
  events
```

```
alta_student new;
check_back ;
borrow;
valuations
[alta_student] avisado:=false, num_books:=0 ;
[borrow] num_books:= num_books +1 ;
[check_back] num_books:= num_books -1 ;
preconditions
check_back if { num_books > 0 } ;
protocols
student:
student = alta_student.student1;
student1= baja_student + (borrow +
check_back).student1;
end class
```

## REFERENCES

- [1] Pastor O., Gómez J., Insfrán E., Pelechano V.: The OO-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. *Information Systems* 26(7): 507-534 (2001)
- [2] Parnas, D.L., On the Criteria To Be Used in Decomposing Systems into Modules. *Comm. ACM*, 1972. 15(12): p. 1053-1058.
- [3] Berg, K.v.d., J.M. Conejero, and R. Chitchyan, Public Ontology of aspect orientation. 2005, AOSD-Europe.
- [4] Nyssen, A., S. Tyszberowicz, and T. Weiler. Are Aspects useful for Managing Variability in Software Product Lines? A Case Study. in *Aspects and Software Product Lines: An Early Aspects Workshop at SPLC-Europe 2005*. 2005. Rennes, France.
- [5] Clarke, S.a., et al. Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code. in *Proc. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. 1999. Denver.
- [6] Kiczales, G., et al. Aspect-Oriented Programming. in *11th European Conf. Object-Oriented Programming*. 1997: Springer Verlag.
- [7] Siy, H., M. Zand, and V. Winter. The Role of Aspects in Domain Engineering. in *Aspects and Software Product Lines: An Early Aspects Workshop at SPLC-Europe 2005*. 2005. Rennes, France.
- [8] Pastor, O. and I. Ramos, Oasis 2.2: A Class-Definition Language to Model Information System Using an Object-Oriented Approach. Vol. 95.788. 1996, Valencia: SPUPV.
- [9] Chavez, C., et al. Taming Heterogeneous Aspects with Crosscutting Interfaces. 2005.
- [10] Colyer, A. The New Holy Trinity. [Blog] 2005 March 05 [cited; Available from: <http://www.aspectprogrammer.org/blogs/adrian>.
- [11] Laddad, R., AOP@Work: AOP and metadata: A perfect match, Part 2--Multidimensional interfaces with metadata. 2005, IBM Developer Works.
- [12] Hudak, P., Building domain-specific embedded languages. *ACM Comput. Surv.*, 1996. 28(4es): p. 196.