JOURNAL OF OBJECT TECHNOLOGY
Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2008

Vol. 8, No. 5, 2009

UML Extensions for Aspect Oriented Software Development

Francisca Losavio, Universidad Central de Venezuela, Venezuela
Alfredo Matteo, Universidad Central de Venezuela, Venezuela
Patricia Morantes, Universidad Nacional Experimental Francisco de Miranda, Venezuela

AOSD (Aspect Oriented Software Development) is an emerging discipline in Software Engineering. It focuses on the separation of concerns of the '70s, aiming at improving modularity, to ensure maintainability in the sense of extensibility, changeability and reuse, towards an evolutionary software product responding to a more dynamic environment. The AOP (Aspect Oriented Programming) paradigm of the '90s focuses on the separation of the crosscutting concerns, which are in general non functional requirements (quality requirements). It is now of general agreement that the entanglement and scattering of concerns must be handled early in the software life cycle. However, these aspects are inherent to all the stages of software development, from requirements elicitation to code implementation. Many concepts and mechanisms have been proposed to handle properly these issues; however terms are in general semantically slightly different in each discipline where they have been formulated and used, causing misunderstanding and confusion. This paper presents an AOSD UML core for early aspects (requirements, analysis and design disciplines), constructed gathering different modelling elements found in the literature, focusing particularly on the AOSD ontolgy document of the Common Foundation for AOSD of the European Community, and different UML extensions. Moreover, each notion is identified, clarified, presented by author, and associated to a discipline. We are particularly interested in early aspects for identification and management of functional and quality requirements, and the crosscuts, to justify and document the choices taken at architectural design. Despite the interest and the recognition of its importance, there is still a lack of appropriate techniques to identify quality requirements at software architecture design level. The risk is that potential aspects might be easily overlooked during the software architecture design and remains unsolved at the design and programming level. The case study provided illustrates an application of the core for an aspect-oriented architectural design, to define the initial architecture for a web application. The results on one hand contribute to the establishment of standards for a unified AOSD terminology, favoring understanding and easing communication. On the other hand they facilitate aspect-oriented architectural design.

1 INTRODUCTION

Software Oriented Aspect Development (AOSD) [17] is a new approach to software development originated from the previous works on Aspect Oriented Programming (AOP) [9]. It is now of general agreement that functional and non functional software system's requirements must be compliant with precise quality goals that the system must accomplish, often introducing new components or mechanisms to satisfy all these concerns in order to be considered a "quality" product. However, many of these concerns are considered or introduced late in the development process, and happen to crosscut the system functional or modular structure, making maintainability difficult, contradicting the object oriented paradigm. The central idea of AOSD as an emerging discipline of post-object technology, is to provide strong support to the separation of the repeated, scattered or entangled concerns at every stage of software development, introducing a new modular unit to encapsulate them to facilitate extensibility, changeability and reuse.
In the context of software engineering a *concern* is defined as a property or interest point of a system [17]. Concerns, from the system point of view, are defined as those interests belonging to the system and its operation, or other aspects which are critical or important for the stakeholders. That is to say, a concern is a kind of requirement needed by the system. Some concerns can be easily encapsulated within classes or modules, according to the chosen implementation language; however, others whose functionality affects several modules, are called *crosscutting concerns* and they are not easy to separate. They cannot be easily encapsulated into new functional units as "implicit functionality" because they crosscut the whole system and are implemented in many classes or modules producing an entangled or scattered code, difficult to understand and maintain. The goal of AOP is to encapsulate them into a modular unit, called aspect, to handle these requirements at implementation level. Recent research trends propose to use AOP notions and mechanisms also at early stages of software development to reduce development costs. We are interested in the so called "early aspects" approach which aims to the identification of the crosscutting concerns during the requirements and analysis disciplines to facilitate architectural design. In particular, the early identification of crosscutting concerns in the requirements discipline can reduce evolution costs [15]; in consequence a modelling language supporting aspect orientation is suggested.
The Unified Modelling Language (UML) [4, 14] is a graphic language notation widely used as a standard to support the object-oriented and component-based software development process. Since UML is provided with extension mechanisms to define new modelling elements [4], UML extensions for AOSD have been proposed to model the development process at early stages.
Several authors have examined the benefits of AOSD modelling: facilitate the implementation stage or the reengineering of existing systems, obtain more reusable and comprehensive components, document early architectural decisions related in general with requirements and maintain their consistency through all the software development stages.

In consequence, efforts been made towards the modelling of software elements at early stages using an aspect-oriented approach. In this work, a core for AOSD considering UML modelling elements with their stereotypes for early stages of development is obtained. In Figure 1, O represents all the concepts in the ontology document of the Common Foundation for AOSD of the European Community [3]. CO represents all the concepts in O required to model AOSD; $\overline{CO}$ (complement) is the set of other concepts identified for AOSD that are not considered in CO. Moreover, most of these concepts have stereotypes defined by different authors. The UML extensions core for AOSD is conformed by the concepts of CO required for modelling AOSD for which stereotypes exist, plus those of $\overline{CO}$. In philosophy, ontology is the study of conceptions of reality and the nature of being. In computer science, an ontology is a document defining formally terms and their relations; in a particular domain it provides a common vocabulary allowing a uniform interpretation of terminology, facilitating communication among working teams, reuse and sharing of conceptual information.
This paper is structured as follows: section 2 presents the basic aspects of AOSD modelling, section 3 gives the UML extensions for AOSD. Different profiles for aaspect-oriented UML are studied, obtaining an AOSD UML core, classified by development disciplines. Section 4 describes a case study, a movie theater portal to show the conceptual modelling of the initial architecture based on the modelling elements defined in the UML core for AOSD. Finally section 5 presents the conclusion and future work.

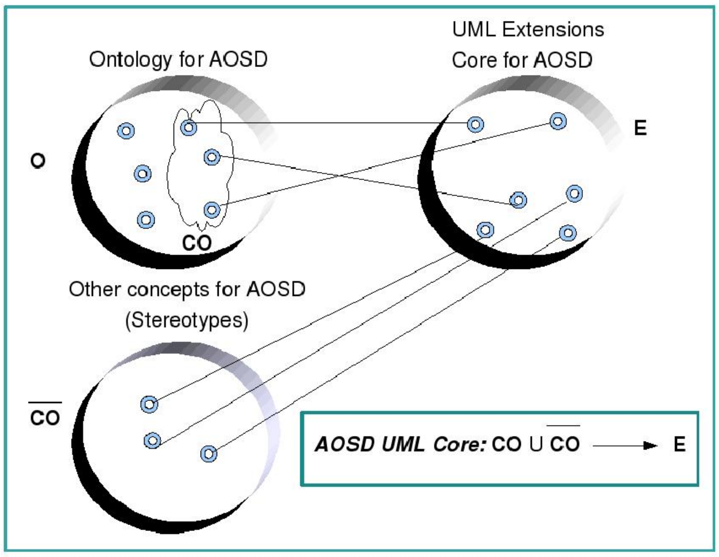

Ontology for AOSD
UML Extensions Core for AOSD
O
CO
E
Other concepts for AOSD (Stereotypes)
$\overline{CO}$
AOSD UML Core: CO U $\overline{CO}$ → E

Figure 1: AOSD UML Core

2 AOSD MODELLING

The use of UML as a standard implies communication without ambiguity. Nevertheless, if a new modelling element is introduced with a new semantics, UML can be extended in two ways, *metamodels* and *profiles*, to preserve the standard. A profile expresses the concepts specific to a certain kind of application domain. Its definition has evolved through the different UML versions [13]. A profile uses the same notation as a UML package and the keyword ≪profile≫ [14, 13, 7].
When the profile modality is used, extension mechanism, saying stereotype, tagged value and constraint are used. A *stereotype* is a class which defines how the existing or already stereotyped metaclasses should be extended, allowing the use of the terminology and/or notation of a particular application platform or domain. Some stereotypes are predefined in UML, others can be defined by the user [13]. A *constraint* imposes conditions on the metamodel elements which have been stereotyped. Constraints extend the semantics of model elements, allowing the addition of new rules or the modification of existing ones [14]. They can be specified in natural language or more formally using OCL (Object Constraint Language) [13]. A *tagged value* is an additional meta attribute associated to a metaclass of the metamodel extended by a profile. Every tagged value is a pair (Tag, Value) that can be used to assign information to any element or instance of a model. Like the stereotype, some tags can be predefined in UML, others can be user defined [13, 7].
A metamodel, which is a model of the modelling language, consisting of a set of basic concepts and rules enabling the construction of conceptual models in a given domain, defines the domain entities and their relations.
Recently, AOSD modelling elements have been frequently defined using UML stereotypes. This work focuses on the study of AOSD profiles.

3 UML EXTENSIONS FOR AOSD

The aspect oriented modelling concepts presented in what follows are extracted from the ontology document of the Common Foundation for AOSD of the European Community [3] and from the aspect-oriented extensions (profiles with stereotypes) presented by different authors which are not included in the ontology. [18, 1, 5, 12, 2, 19, 8, 10, 16]. These works were selected from the Bibliography of Aspect-Oriented Software Development by Filman [6].
In what follows, the entries of the tables show the author's main reference(s) where the concept has firstly appeared, the label of the stereotype if any, else only the name of the element is shown, the base UML metamodel element and finally the graphic representation.

Table 1: Concern

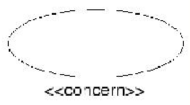
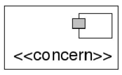
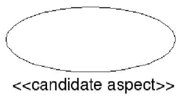
Author	Label	Base UML metamodel element	Represention
(Brito, et al, 2003) (Moreira, et al, 2002)	≪concern≫	classifier	
(Barra, et al, 2004)	≪concern≫	classifier	

Table 2: Crosscuting Concern

Author	Label	Base UML metamodel element	Represention
(Brito, et al, 2003) (Moreira, et al, 2002)	≪candidate aspect≫	classifier	

Concern

“A concern is a thing in an engineering process about which it cares. A concern is an interest, which pertains to the system’s development, its operation or any other matters that are critical or otherwise important to one or more stakeholders” (see Table 1).
The component graphical representation of concerns is useful in early aspect development to represent the architecture in the component diagram.

Crosscutting Concern

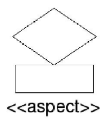
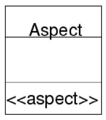
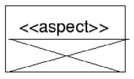
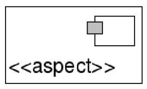
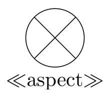
“A crosscutting concern is a concern for which the implementation is scattered throughout the rest of an implementation. A crosscutting concern is a concern, which cannot be modularly represented within the selected decomposition. Consequently, the elements of crosscutting concerns are scattered and tangled within elements of other concerns” (see Table 2).

In many approaches, a new use case is added to the use case diagram for each posible crosscutting concern, and it is called *candidate aspect*. It is represented as the use case labeled with the stereotype ≪candidate aspect≫.

Aspect

"An aspect is a modular unit designed to implement a concern. An aspect is a unit for modularizing an otherwise crosscutting concern" (see Table 3).
Notice that the graphic notation of component is required for early aspects development.

Table 3: Aspect

Author	Label	Base UML metamodel element	Represention
(Zakaria, et al, 2002)	≪aspect≫	classifier	
(Aldawud, et al, 2003)	≪aspect≫	classifier	
(Aldawud, et al, 2003)	≪aspect≫	classifier	
(Barra, et al, 2004)	≪aspect≫	classifier	
(Sousa, et al, 2004)	≪aspect≫	classifier	

Composition

"Composition is bringing together separately created software elements" (see Table 4).

Table 4: Composition

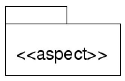
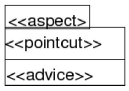
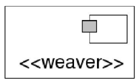
Author	Label	Base UML metamodel element	Represention
(Bash, M., Sanchez A. 2003)	≪aspect≫	package	
(Zhang, G. 2005)	≪aspect≫	package	

Table 5: Weaving

Author	Label	Base UML metamodel element	Represention
(Barra, et al, 2004)	≪weaver≫	classifier	

At implementation level, in AOP where the term first appears, composition means the integration of the aspects with the pointcuts found in the base classes of the system. To handle complexity, a huge system has in general to be divided into smaller units, to allow work on smaller pieces of information. The UML package construct offers a general mechanism to organize subsystems into groups of modelling elements. Composition at modelling level, particularly also for early aspects, encapsulates aspects into their own package, where the whole functionality of the aspect can be modelled. Packages include classic elements such as class and interaction diagrams; however for aspect-orientation, they can include pointcut and advice packages.

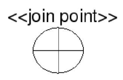
Weaving

"Weaving is the process of composing core functionality modules with aspects, thereby yielding a working system" (see Table 5).

Join Point

"A join point is a point of interest in some artifact in the software lifecycle through which two or more concerns may be composed" (see Table 6).

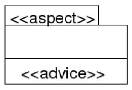
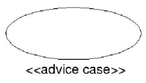
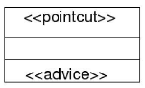
Table 6: Join Point

Author	Label	Base UML metamodel element	Represention
(Bash, M., Sanchez A. 2003)	≪join point≫	classifier	

Advice

“An advice is the behaviour to execute at a join point. An advice is an aspect element, which augments or constrains other concerns at join points matched by a pointcut expression” (see Table 7). From an AOSD point of view, an advice can be

Table 7: Advice

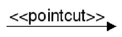
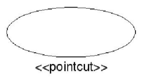
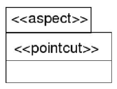
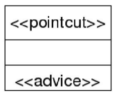
Author	Label	Base UML metamodel element	Represention
(Zhang, G. 2005)	≪advice≫	package	
(Kaewkasi, et al, 2003)	≪advice case≫	classifier	
(Steinmacher, 2003)	≪advice≫	classifier	

seen as a sequence of actions holding similar characteristics to a use-case, with the difference that it cannot be directly executed by an actor. From the implementation point of view of AOP, where the term was first used, an advice is an additional behavior that is added to the execution structure; it helps to define what to do, as a mechanism similar to a class method; it is used to declare what part of the code should be executed in a join point, which is captured by the pointcut. There are three types of advices: before advice (executed previously to the join point), after advice (executed after the join point) and around advice (deviate, continue or cause the execution in a modified context).

Pointcut

“A pointcut is a predicate that matches join points. More precisely, a pointcut is a relationship from JoinPoint → boolean, where the domain of the relationship is all possible join points” (see Table 8). A pointcut in AOP is designed to identify and

Table 8: PointCut

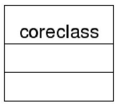
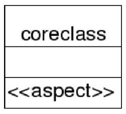
Author	Label	Base UML metamodel element	Represention
(Kaewkasi, et al, 2003)	≪pointcut≫	association	
(Zakaria, et al, 2002)	≪pointcut≫	classifier	
(Zhang, et al 2005)	≪pointcut≫	package	
(Steinmacher, 2003)	≪pointcut≫	classifier	

select a set of join points. Pointcut and advice conform the dynamic crosscutting rules: weaving of new behavior within the current program execution. From the AOSD pint of view, it can be seen as an association relation, where the stereotype is represented by a label indicating when the advice can be executed within a use-case. They are graphically represented by a package or an ellipse relating the aspect with the base class.
In what follows concepts that are not present in the ontology cited are presented. We have considered them here since we focus on the early stages of software development.

CoreClass

A class identified by the system designer as a main unit that encapsulates some cohesive functionality of a system, as opposed to a crosscutting unit, referred to as an aspect (see Table 9).

Table 9: CoreClass

Author	Label	Base UML metamodel element	Represention
(Zakaria, et al, 2002)	≪coreclass≫	classifier	
(Steinmacher, 2003)	≪coreclass≫	classifier	

Match-Point

A match-point is where the crosscutting concern should join the behavior of the functional concern (a non-crosscutting concern such as a class) it cuts across. This is an abstraction of the join point concept in AspectJ [5, 12]. No stereotype nor graphic representation has been defined for this concept.

Relations

The ≪crosscut≫ stereotype is used to model *crosscutting* relationships, where the aspect crosscuts the code of functional components, similarly to the join point concept. However, this stereotype is used in a slightly different way in AOSD; it denotes when a non-functional concern has a crosscutting relation with multiple use-cases (functional concerns) (see Table 10). There are other *aspect-class* relationships, where the candidate aspects affect the concerns: *overlapping, overriding and wrapping.* Many early aspect approaches consider that in the use-case diagram, crosscutting concerns are added as "included" use-cases called also ≪candidate aspect≫, by the classic "include" relation (see Table 11). With respect to the *aspect-aspect* relationship, the stereotype ≪dominates≫ specifies which aspect has priority on the other.

Finally, in Table 12 we select the elements for the AOSD UML core, grouped by the analysis and design development process disciplines. Notice that the modelling elements selected are particularly useful in architecture design. In this sense the component notion representing concerns, aspects and weaving is used in the component diagram, as we shall see in the following section.

Table 10: Aspect-Class Relations

Author	Label	Base UML metamodel element	Represention
(Aldawud, et al, 2003)	≪crosscut≫	association	$\xrightarrow{\text{<<crosscut>>}}$
(Sousa, et al, 2004)	≪crosscut≫	association	$\overset{\text{<<crosscut>>}}{\dashrightarrow}$

Table 11: Concern-CandidateAspect Relations

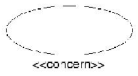
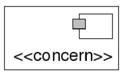
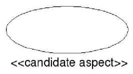
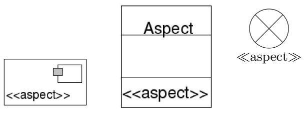
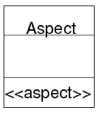
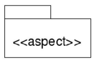
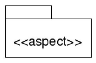
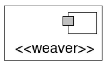
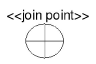
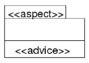
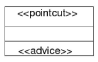
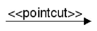
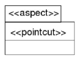
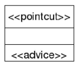
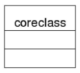
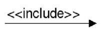
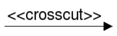
Author	Label	Base UML metamodel element	Represention
(Moreira, et al, 2002)	≪include≫	association	$\xrightarrow{\text{<<include>>}}$
(Brito, et al, 2003)	≪wrappby≫	association	$\xrightarrow{\text{<<wrappby>}}$
(Brito, et al, 2003)	≪overlapby≫	association	$\xrightarrow{\text{<<overlapby>>}}$
(Brito, et al, 2003)	≪overriby≫	association	$\xrightarrow{\text{<<overriby>>}}$

4 CASE STUDY: MODELLING THE INITIAL ARCHITECTURE OF A MOVIE THEATER CHAIN PORTAL

The AOSD UML core defined will be used to construct the initial architecture for the problem studied. The deployment view, the use-case and component models of the analysis discipline will be developed. The deployment diagram illustrates the basic architecture for the application domain obtained from the domain knowledge, which corresponds to the physical view of the architecture. A composition table [5] will be used to transform the use-case diagram into the component diagram. It shows the relation among the crosscutting concerns and ≪candidate aspects≫. In the use-case diagram, the concern stereotype is used to express the functionality and the candidate aspect stereotype expresses the potential crosscutting concerns (quality requirements).

The problem: a Movie Theaters Chain asks for a system providing a massive use of functionality for public access. It should offer mainly integrated on-line tickets

Table 12: AOSD UML Core

Concepts	Disciplines		
	Requirements	Analysis	Design
concern			
candidate aspect (crosscutting concern)			
aspect			
composition			
weaving			
join point	≪match point≫		
advice			
pointcut			
core class			
relations concern - candidate aspect			
relations aspect - class			

sales facility. Tickets can be bought, reserved or payed on-line. It should provide also facilities to promote the different movie theaters on the chain. Main facilities considered by the system are: on-line ticket bookings, payment, consultation and cancellation, user registration, transaction statistics.

The solution: a portal is proposed, with a commercial browser and internet facility.

Domain analysis for a Portal application: a Portal is a web application that uses mainly transactional and portal services. For this kind of application, general non functional concerns (quality requirements) are identified. Moreover, main functional requirements and the architectural style are also known [11]. They are presented in what follows.

Quality requirements

- Transactional
  - Functionality
    - security (integrity)
    - accuracy
  - Reliability
    - availability
  - Efficiency
    - time behavior
    - resource utilization
- Portal
  - Portability
    - adaptability: scalability
  - Efficiency
    - time behavior
    - resource utilization

Functional requirements
The main functionalities of a Portal application are: data exchange, access control and encrypting, since the portal uses transactions; consult and access are needed.

Initial architecture
A classic 3-tiers client-server style, to guarantee separation between the user-interface on the browser (Web Client) and the Data Base (Data Base Server); the application logic tier (Application Server) mediates between the client and server tiers. The communication is assured by the TCP/IP protocol.

In conclusion, from the domain analysis, the information obtained is: functional requirements, quality requirements and initial architecture.

From the problem statement, the following functional concerns can be identified: consult of portal web-pages (consult, access), on-line user registration (data exchange, access control, encrypting), on-line ticket booking (data exchange and access control), on-line ticket payment (data exchange, access control and encrypting). Notice that all the functionality of the application in this particular case correspond to the functionalities already identified in the domain analysis for this family of Web applications.
We must now look for the crosscutting (non functional) concerns related (which crosscuts) with the functional concerns and identify the points where the composition will take place. We notice that efficiency (response-time), precision and security are required by on-line tickets booking and payment; availability is required by consult of portal. On the other hand, security is also needed for on line user registration (see Table 13). Notice that all these crosscutting concerns have already been identified as quality requirements from the information obtained in domain analysis. To construct the use-case model, we present the use-case diagram using the ≪include≫ relation stereotype to indicate the crosscut relationship as an included use-case (See Fig. 2). We follow the early aspect approach of including all the quality requirements as candidate aspects, i.e. potential crosscutting concerns[5].

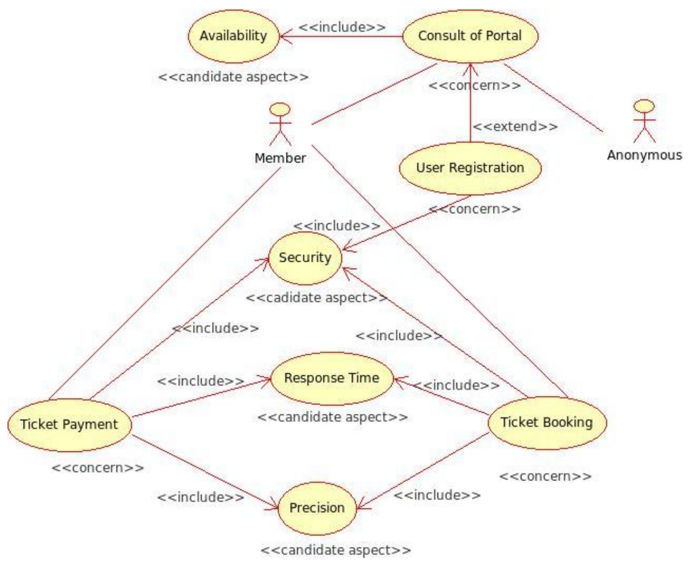


Figure 2: Use-case diagram for Movie Theater Chain Portal

Architecture for the Movie Theater Chain Portal
We proceed now to construct an initial architecture for the Movie Theater Chain Portal application, on the basis of the information provided by the domain analysis

Table 13: Composition points identification

Candidate Aspect	Functional Concern			
	Consult of Portal	User Registration	Ticket Booking	Ticket Payment
Response Time			X	X
Precision			X	X
Security		X	X	X
Availability	X			

and the Use-case diagram.

Initial architectural configuration

It is obtained directly from the initial architecture of the domain analysis (Fig. 3). The Application Server supports the application logic component which is where the major development effort has to be made and it is modelled by the use-case diagram Fig. 2.

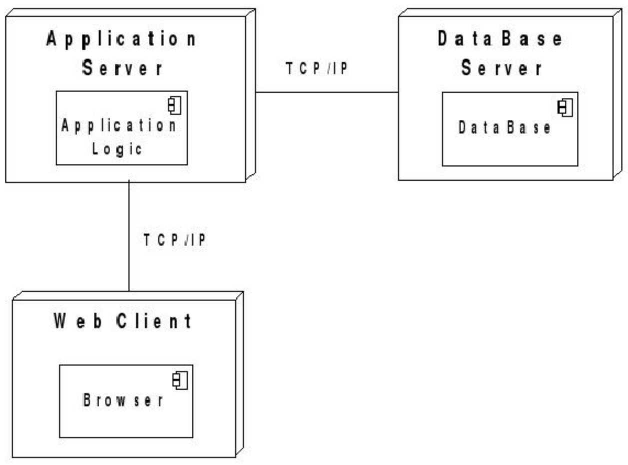


Figure 3: Initial architecture for the Movie Theater Chain Portal application: deployment diagram

Architecture refinement

The application logic component is refined assigning a component to each use-case in the use-case diagram (Fig. 2). Notice that concerns components *require* aspects components, according to the UML 2.0 standard notation to express the component diagram (Fig. 4). From the composition table (see Table 13), we observe that "availability" is a candidate aspect that crosscuts only the "consult of portal" functionality, hence it is considered a concern component, i.e. it is an "implicit functionality". It can be solved by introducing a mechanism such as replication. The remaining candidate aspects which in the composition table appear to crosscut the functionalities are expressed as aspect components. Notice that each concern component that has been introduced must respond to precise quality requirements: User Registration requires security, Ticket Booking and Ticket Payment require security, precision and response time.

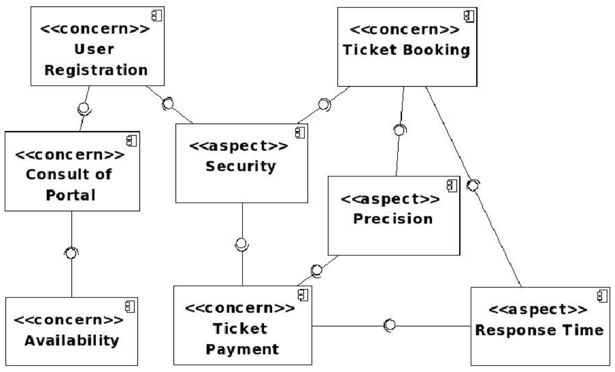

<<concern>> User Registration
<<concern>> Ticket Booking
<<concern>> Consult of Portal
<<aspect>> Security
<<aspect>> Precision
<<concern>> Availability
<<concern>> Ticket Payment
<<aspect>> Response Time

Figure 4: Architecture refinement of the Application Logic component: component diagram with concerns and aspects

Component Weaving.

In the same configuration, the weaving process is represented using the weaver component (Fig. 5). We have chosen to represent a weaver for each crosscutting concern: the Security weaver component, which "provides" the security service to User Registration, Ticket Booking and Ticket Payment; the Response Time weaver, providing the response Time service to Ticket Booking and Ticket Payment, and finally the Precision weaver, handling the precision service also for Ticket Booking and Ticket Payment. Notice that another choice could have been to represent only one weaver component, composed by the three weavers mentioned.

Further refinements.
Since the crosscutting concerns have already been identified, we can proceed to look in more detail at each concern components, using known architectural design techniques. The team work can be eased from the fact that a quality requirement can be handled properly, when dealing with the involved concern component, at

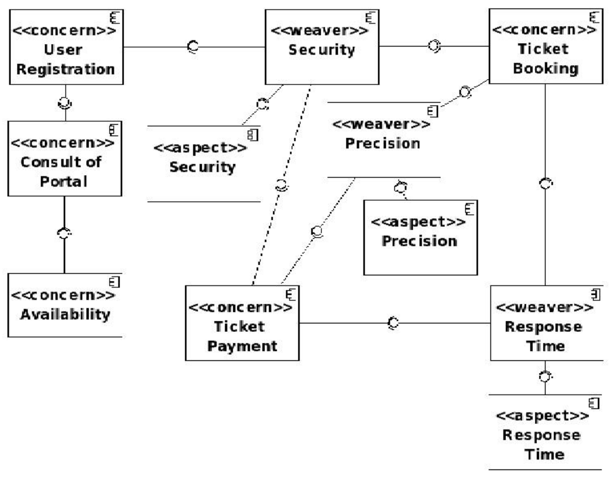


Figure 5: Architecture refinement of the Application Logic component:component diagram with weaving components

a very early stage of develoment. It is clear that we can use all the stereotypes proposed in the AOSD UML core, for a more detailed aspect-oriented architectural design during the analysis and design disciplines.

5 CONCLUSIONS

AOSD is a new Software Engineering paradigm, considered now as a suitable alternative to software evolution and to improve the development process of modern complex systems. This work has studied the concepts of the ontology document of the Common Foundation for AOSD of the European Community, besides numerous UML extensions for aspect orientation proposed by different authors. From this study, an AOSD UML core for early aspects (requirements, analysis and design disciplines), conformed by the modelling elements and their stereotypes has been proposed. Moreover, each notion is identified, clarified, presented by author, and associated to a discipline. We are particularly interested in the requirements and analysis disciplines to identify early the functional and quality requirements for architectural design, reducing the gap between the requirements, use-case and component models. These results contribute in general to the establishment of standards for a unified AOSD terminology, favoring usability of architectural design techniques. The case study provided illustrates an application of the core, to define the initial architecture for a web application. We have presented a practical and straightforward approach to build an initial architecture on the basis of domain knowledge, within an early aspect approach. The decision taken are justified and documented, showing the practical usage of AOSD techniques to detect potential crosscutting concerns.

Our approach has been applied and tailored in different academic projects. Ongoing works are on one hand to enrich the core with dynamic modeling, for example sequence diagrams, and the usage of the core in an MDE (Model Driven Engineering) context, for the requirements and analysis disciplines,to define the input and output models to the transformations. On the other hand, to revise and improve further the approach to define an initial step that can be included in product-line aspect-oriented architectural design approaches, considering requirement's priorities and trade-offs analysis, including the support of automated tools.

ACKNOWLEDGEMENT

This work has been partially supported by the Consejo de Desarrollo Científico y Humanístico of the Universidad Central de Venezuela

REFERENCES

[1] O Aldawud, T Elrad, and A Bader. A uml profile for aspect oriented software development. In *The Third International Workshop on AO Modeling*, 2003.

[2] M Bash and A Sanchez. Incorporating aspects into the uml. In *International Conference on Aspect-Oriented Software Development*, 2003.

[3] K Berg, J Conejero, and R Chitchyan. AOSD ontology 1.0 public ontology of aspect orientation. *Common Foundation for AOSD*, 2005.

[4] G Booch, J Rumbaugh, and I Jacobson. *El Lenguaje Unificado de Modelado.* Addison Wesley Iberoamericana, Madrid, 1999.

[5] I Brito and A Moreira. Towards a composition process for aspect-oriented requirements. In *Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, workshop of the AOSD*, Boston, USA, 2003.

[6] R Filman. A bibliography of aosd version 2.0, 2005.

[7] L Fuentes and A Vallecillo. Una introducción a los perfiles de uml. *Novática*, 168:6–11, 2004.

[8] C Kaewkasi and W Rivepiboon. Aspect-oriented extension for capturing requirements in use-case model. In *CAISE: The 15th Conference on Advanced Information Systems Engineering*, Austria. CEUR. 74, 2003.

[9] G Kiczales, J Lamping, A Mendhekar, C Maeda, C Lopes, J Loingtier, and J Irwin. Aspect-oriented programming. In M. Aksit and S. Matsouka, editors, *ECOOP 1997 Object-Oriented Programming, 11th European Conference*, number 1241, pages 220–242, Berlin, 1997. LNCS, Springer-Verlag.

[10] I. Krechetov, B. Tekinerdogan, and A. Garcia. Towards an integrated aspect-oriented modeling approach for software architecture design. In *DSOA 2006*, 2006.

[11] F. Losavio, A. Matteo, and R. Rahamut. Characterization of web services domain based on quality standards. In *Conferencia IADIS, Ibero-americana*, 2006.

[12] A Moreira and J Brito. Crosscutting quality attributes for requirements engineering. In *14th International Conference on Software Engineering and Knowledge Engineering SEKE2002*, pages 167–174, Italy, 2002. ACM Press.

[13] OMG. *Revised submission for MOF 2.0 Query/View/Transformation RFT*, 2003.

[14] OMG. *UML 2.0 Infrastructure Specification Object Management Group*, document ptc/03-09-15 edition, 2003.

[15] A Rashid, P Sawyer, A Moreira, and J Araujo. Early aspects: a model for aspect-oriented requirement engineering. In *IEEE Joint Conference on Requirements Engineering*, Essen-Germany, 2002.

[16] G Sousa, S Soares, P Borda, and J Castro. Separation of crosscutting concerns from requirements to design:adapting an use case driven approach. In *Early Aspects: Aspect-Oriented Requirements Engineering and Architecture*, volume 1, pages 1–10, Lancaster, 2004.

[17] S Sutton and P Tarr. Aspect-oriented design needs concern modeling. In *1st International Conference on Aspect-Oriented Software Development*, Enschede, 2002.

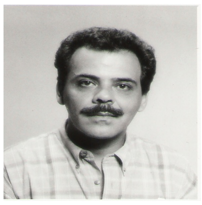
[18] A Zakaria, H Hosny, and A Zeid. A uml extension for modeling aspect-oriented systems. In *UML 2002 Workshop on Aspect Oriented Modeling*, 2002.

[19] G Zhang. Toward aspect-oriented class diagram. In IEEE Computer Society, editor, *12th Asia-Pacific Software Engineering Conf. (APSEC'05)*, volume 00, pages 763–768, 2005.

ABOUT THE AUTHORS

Francisca LOSAVIO received the Doctor degree in 1991 and a 3ème. Cycle Doctor Degree in 1985, both in Computer Science and from the Paris-Sud University, Paris XI, Orsay, France. She also obtained a MSc. Degree also in Computer Science in 1983 from the Simon Bolivar University, Venezuela. She is a Titular Professor at the School of Computer Science, Faculty of Science, Venezuela Central University, Caracas, where she works at the ISYS (Software Engineering and Systems) Research Center, coordinating the MoST (Models, Software & Technology) Laboratory. She has participated in national and European Community research projects. Her main research axes are software architecture, software quality, quality standards and software development process.

flosav@cantv.net



Alfredo MATTEO received the Doctor degree in Computer Science for the Paul Sabatier University, Toulouse, France in 1984. At present he is Titular Professor at the School of Computer Science, Faculty of Science, Venezuela Central University, where he has coordinated the TOOLS Laboratory of the ISYS (Software Engineering and Systems) Research Center, being now part of the research staff of the MoST (Models, Software & Technology) Laboratory of ISYS. He is now responsible of the Postgraduated Studies in Computer Science. His research includes software engineering, requirements engineering, architectures, methodologies and model driven engineering.

almatteo@cantv.net

Patricia MORANTES received her degree in Informatics Engineer at the Central-West "Lisandro Alvarado" University in 1995. She is a Doctor Candidate in Computer Science at the Venezuela Central University, Caracas, Venezuela. At present she is Assistant Professor of the Physics and Mathematics Department at the National Experimental University "Francisco de Miranda", Coro, Venezuela. Additionaly, she works at the MoST (Models, Software & Technology) Laboratory of the ISYS (Software Engineering and Systems) Research Center, and her research includes software engineering, requirements engineering, architectures, methodologies and model driven engineering.

pmorantes@cantv.net