AN XML-BASED LANGUAGE FOR SPECIFICATION AND COMPOSITION OF ASPECTUAL CONCERNS

Elisabete Soeiro[1], Isabel Sofia Brito[2], Ana Moreira[1]
[1] *Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal*

[2] *Escola Superior de Tecnologia e Gestão, Instituto Politécnico de Beja, Beja, Portugal*

Keywords: Aspect-Oriented Requirements, Aspect Composition, XML Schema.

Abstract: Separation of concerns refers to the ability of identifying, encapsulating and manipulating parts of software that are crucial to a particular purpose (Dijkstra, 1976). Traditional software development methods were developed with this principle in mind. However, certain broadly-scoped properties are difficult to modularize and keep separated during the lifecycle, producing tangled representations that are difficult to understand and to evolve. Aspect-oriented software development aims at addressing those crosscutting concerns, known as *aspects*, by providing means for their systematic identification, separation, representation and composition. This paper focuses on the representation and composition activities, by proposing an XML-based language to specify and compose concerns at the requirements level. An illustration of the proposed approach to an example supported by a tool is presented.

1 INTRODUCTION

Separation of concerns aims at identifying and modularizing parts of software that are relevant to a particular concept, goal or purpose (Dijkstra, 1976). Traditional approaches to software development, such as object-oriented and structured methods, have been created with this principle in mind. However, certain broadly-scoped properties are difficult to modularize and keep separated during the lifecycle, producing tangled representations that are difficult to understand and to evolve. Typical examples of such properties are security, synchronization, logging and tracking. Aspect-Oriented Software Development (AOSD) aims at addressing such crosscutting concerns by providing means for their systematic identification, separation, representation and composition. Crosscutting concerns are encapsulated in separate modules, known as *aspects*, and composition mechanisms are later used to weave them back with other core modules.

Traditionally, AOSD has focused mainly on the implementation phase of the software lifecycle, where aspects are identified and captured mainly in code (Kiczales, 1997) (Xerox, 2001) (Bergmans, 2001) (Lieberherr, 2001) (Tarr, 1999). Some work has also been carried out to incorporate aspects at the design level mainly through extensions to the UML meta-model e.g. (Clarke, 2001), (Suzuki, 1999), (Herrero, 2000). However, crosscutting concerns are often present well before the implementation, such as in requirements engineering (Brito, 2003) (Brito, 2004) (Rashid, 2003) (Moreira, 2005) (Clarke, 2005).

This paper builds on our previous work on Aspect-Oriented Requirements Engineering (AORE) (Brito, 2003) (Brito, 2004), focusing on two particular activities of the model presented therein: concern specification and composition. Concerns were described in terms of an informal template and compositions were defined informally with a minimum set of operators (cf. section 2.2). Concerns are all treated in a uniform fashion, independently of their crosscutting nature. In this paper we will present an XML-based language to specify and compose concerns.

We have chosen XML because it is a widely used standard that allows the description of any kind of data. XML Schema is another standard that allows the definition of a full specification for a XML document, enabling the creation of a set of rules to structure and form that XML document, as well as rules for data types and integrity. For these reasons our template for concern representation will be

Soeiro E., Sofia Brito I. and Moreira A. (2006).
AN XML-BASED LANGUAGE FOR SPECIFICATION AND COMPOSITION OF ASPECTUAL CONCERNS.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - ISAS*, pages 410-419
DOI: 10.5220/0002494704100419

mapped into a XML Schema, creating a *meta-template* that will function as a standard representation. Any instance that follows the meta-template must also be correctly validated with the defined XML Schema.

The contributions of this work are twofold: (i) define a XML-based language to define concerns; (ii) define a XML-based language to compose concerns at requirements level. A tool supports the end result where concerns and composition rules can be defined using pre-defined templates. The modules encapsulating the various requirements and composition rules are stored in eXist, a native XML database.

This paper is organized as follows. Section 2 presents some background on aspects and on the AORE model, serving as basis for our work. Section 3 gives an overview of XML representations of concerns and composition rules while Section 4 illustrates the ideas with an example. Section 5 presents some related work and Section 6 draws some conclusions, suggesting directions for future work.

2 BACKGROUND

2.1 What Are Aspects?

Aspect-Oriented Software Development (AOSD) attempts to aid developers in the separation of concerns, or the breaking down of system into distinct parts that overlap in functionality as little as possible. In particular, AOSD focuses on the modularization and composition of crosscutting concerns. The term *crosscutting concerns* refers to properties of software that cannot be effectively modularized using traditional software development techniques, such as object-oriented methods. Typical examples of such crosscutting concerns are non-functional requirements, such as security, fault tolerance, persistency. However, crosscutting concerns can also be functional requirements, such as auditing, or validation.

Crosscutting concerns are encapsulated in separate modules, known as *aspects*, and composition mechanisms are later used to weave them back with other core modules, at loading time, compilation time, or run-time. However, aspects, as well as their compositions, also have an important role to play before the implementation activity. Aspects will allow the modularization of crosscutting concerns that cannot be encapsulated by a single use case (Jacobson 1992) or viewpoint (Finkelstein 1996), for example, and are typically spread across several of them. Composition, on the other hand, apart from allowing the developers to picture the whole system, allows them to identify conflicting situations whenever a concern contributes negatively to others (Rashid, 2003). This offers the opportunity to establish critical trade-offs before the architecture design is derived, supporting the necessary negotiations among the stakeholders.

AOSD aims at addressing such crosscutting concerns at the various levels of the software development process, by providing means for their systematic identification, separation, representation and composition.

2.2 Aspect-Oriented Requirements

The goal of this paper is to offer a systematic and rigorous mean to support the specification and composition activities of the Aspect-Oriented Requirements Engineering (AORE) model presented in (Brito, 2003) (Brito, 2004). This model is composed of three main tasks (see Figure 1): identify concerns, specify concerns, and compose concerns.

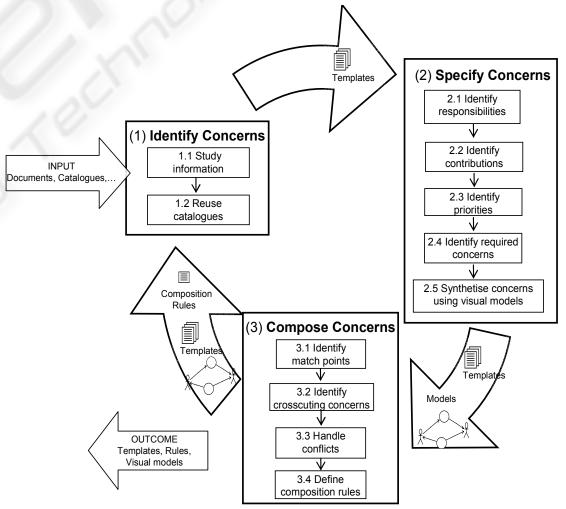

Figure 1: A model for Aspect-Oriented Requirements Engineering.

The task, *identify concerns,* aims at identifying all the concerns of a system, where a *concern* refers to a matter of interest which addresses a certain problem that is of interest to one or more stakeholders. Such a concern can be defined as a set of coherent requirements, defining a property that the future system must provide. This can be accomplished by analysing the initial requirements, transcripts of stakeholders' interviews, etc. Good sources for concern identification are the existing catalogues, such as the non-functional requirements catalogue offered by (Chung, 2000).

The task, *specify concerns,* provides a template, as shown in Table 1, which collects all the information about a concern.

Table 1: Template to describe concerns.

Concern	Description
Name	The name of the concern.
Description	Short description of the intended behaviour of the concern.
Sources	Source of information, e.g. stakeholders, documents, domain, catalogues and business process.
Classification	Helps the selection of the most appropriate approach to specify the concern. For example: functional, non-functional, goals.
Stakeholders	Users that need the concern in order to accomplish their job.
List of Responsibilities	
Responsibility #	List of what the concern must perform; knowledge or proprieties the concern must offer.
List of Contributions	
Contribution #	List of concerns that contribute or affect this concern. This contribution can be positive (+) or negative (-)
List of Priorities by Stakeholder	
Stakeholder #	Expresses the importance of the concern for a given stakeholder. It can take the values: *Very Important*, *Important*, *Medium*, *Low* and *Very Low*.
List of Required concerns	
Required Concern #	List of concerns needed or requested by the concern being described.

The *Name* field names the concern, while the field *Description* provides a textual explanation about the concern's importance. The *Sources* field states the origins of the concern, having several possible values, as stakeholder requirements, external catalogues of non-functional requirements (Chung, 2000), etc. The *Classification* field classifies the concern according to its type, e. g. functional, non-functional. The *Stakeholders* field shows which stakeholders interact with the concern. The *Responsibilities* entry lists the operations that the concern should provide, while the *Contributions* field offers a list of positive and negative interactions with other concerns. This field helps detecting conflicts whenever concerns contribute negatively to each other. These conflicts may be resolved through the *Stakeholder priorities* field, which assigns priorities to concerns from the stakeholders' perspective. Finally, the *Required concerns* field acts as a dependency reference to other concerns in the system. This field will be used to identify which concerns are crosscutting.

Finally, the task *compose concerns*, offers the possibility to compose a set of concerns, incrementally, until the whole system is obtained. Each composition takes place in a match point in the form of a composition rule. A match point tells us which crosscutting concerns should be composed with a given (non-crosscutting) concern. A composition rule shows how a set of concerns can be weaved together by means of some pre-defined operators. In order to accomplish this, we need to identify crosscutting concerns (those that are required by more than one other concern). At this point conflicting situations can be identified (if concerns that contribute negatively between them have the same priority and need to be composed in the same match point). Conflicts are solved by using a simple process to guarantee different priorities to those concerns (for more information, see (Brito, 2004)).

A composition rule takes the form:

<Term> <Operator> <Term>

where a *Term* can be a concern or a sub-composition (which is another composition rule) and the *Operator* represents the operator relating both terms. The operators we have chosen (*enable*, *disable* and *parallel*), were inspired in the LOTOS operators (Brinksma, 1988), where:

- *Enabling (denoted by C_1>>C_2)*: refers to a sequential composition and means that the behaviour of C_2 begins if and only if C_1 terminates successfully.
- *Disabling (denoted by C_1[>C_2)*: means that C_2 interrupts the behaviour of C_1 when it starts its own behaviour. This allows the representation of interruptions.
- *Full synchronization (denoted by C_1||C_2)*: refers to the parallel operator and means that the behaviour of C_1 must be synchronized with the

behaviour of C_2. It represents concurrent "execution" of concerns.

3 AN XML-BASED LANGUAGE TO DEFINE AND COMPOSE CONCERNS

This section presents two XML Schemas for a standard representation of concerns and compositions. We have chosen XML (W3C, 2004) because it is a widely used standard that allows the description of any kind of data. Since the XML schema language is extensible – it is based on XML itself – it is possible to define a full specification for a XML document, enabling the creation of a set of semantically meaningful tags, to structure and form a XML document, as well as rules for data types and integrity. Furthermore, XML provides a standard way to represent and manipulate source code, and represent meta-information that could be manipulated by another program.

For the reasons abovementioned, our template for concern representation will be mapped into a XML Schema, creating a standard representation. The resulting XML Schema is compliant with the logic present in the template previously defined. Similarly, we will define a XML Schema to represent the structure of a composition rule. In order to do that, we first need to study all the possible logical forms that a composition rule can take.

3.1 The Concern Type

The construction of the XML Schema has been done in a modular way, factoring out common definitions for better understanding. Not all the XML Schemas could be included here, due to lack of space. Those shown, however, illustrate the general idea. The structure of the complex type *Concern* can be mapped directly into the logical template illustrated in Table 1, where:

Name defines a unique name for the concern. Since all names in our solution are seen as keys, uniqueness is required.

Description presents a summary description of the behaviour of concern.

Classification indicates if a concern is functional or not functional.

ListOfSources lists the information sources that contributed to the concern creation. A list of four values (that may be expanded in the future if proved to be insufficient) is available: stakeholders, documents, domain, catalogues and business process (see Figure 2).

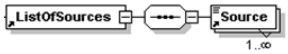

Figure 2: *ListOfSources* tag.

ListOfPrioritiesByStakeholder groups the *Stakeholders* and *Stakeholder Priorities* from the logic template (in Table 1). It is a list where each element aggregates one stakeholder together with his/her priority for the concern. The priority reflects the degree of importance of that concern to that stakeholder. This attribute can take the values "Very Important", "Important", "Medium", "Low" and "Very Low". Figure 3 presents its definition.

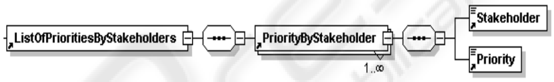

Figure 3: *ListOfPrioritiesByStakeholder* tag.

ListOfResponsibilities lists the information of what the concern must perform (see Figure 4).

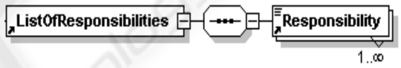

Figure 4: *ListOfResponsibilities* tag.

ListOfContributions defines the contribution relationship between the concern under study and other concerns. This contribution can be positive (+), negative (-), or "don't care" meaning that one concern might not have an explicit contribution with others concerns. Figure 5 illustrates its definition.

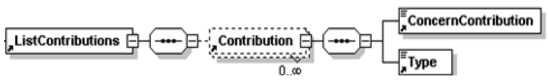

Figure 5: *ListContributions* tag.

ListOfRequiredConcerns defines a list of other concerns from which the current concern depends, i.e. the concerns required by the concern under study. This list can be empty, meaning that the concern does not need other concerns to fulfil its objectives (Figure 6).

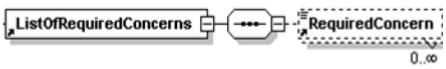

Figure 6: *ListOfRequiredConcerns* tag.

Figure 7 aggregates the above presented definitions to form the XML schema of the type *Concern* that defines one concern.

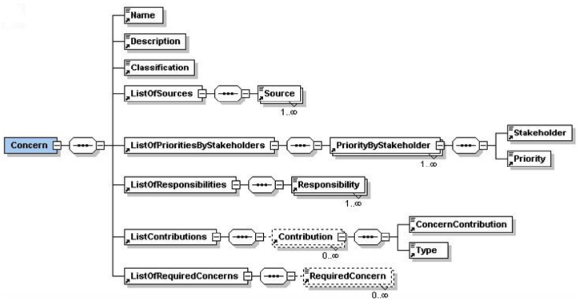

Figure 7: XML Schema of the definition of a concern.

The system will be represented as a list of all its concerns, together with a list of composition rules.

3.2 Composition Operators

A composition rule defines the order in which concerns will be applied in a particular match point. Figure 8 illustrates a simple grammar, where the terms between simple quotes represent the literals.

```
(1) Composition → Term Operator Term
                | (Composition)

(2) Term →  Composition

          | Concern

(3) Operator → '||'
             |'>>'
             |'[>'
```

Figure 8: Composition of concerns using EBNF.

These composition rules have been formally defined through the meta-language EBNF (Extended Backus-Naur Form) (Sebesta, 2003).

3.3 Defining XML Schema for Composition Specifications

The composition rules are based on the information defined in task 3 of the AORE model. Similarly to what has been done to define the *concern* type, the XML schema will be defined to represent composition rules. As mentioned before, a composition rule is defined for each match point by using the *MP* tag that encapsulates all its simpler composition rules. The *MP* tag is composed of: Name, ListOfConcerns, CompositionRule, ListOfCompositionRules.

Name. This element defines a unique name for the match point. Again, since all names in our solution are seen as keys, uniqueness is required.

ListOfConcerns. This element lists the concerns that compose a given *match point* (see Figure 9).

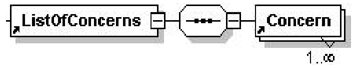

Figure 9: ListOfConcerns tag.

CompositionRule. This element is structured according to the rules defined in Figure 8. These rules are mapped into the XML Schema (with the additional attribute *Outcome*) depicted in Figure 10, where:

(i) a Term that can be a simpler CompositionRule or a ConcernName;
(ii) an Operator tag defining the operators described in Section 2.2. It can take the values: '>>', '[>' and '||';
(iii) an OutCome tag expressing the result of constraining the concerns' responsibilities by means of the operators described before. It can take the values: (a) *Satisfied*, used to declare that a concern/responsibility could be accomplish after the composition rule; (b) *Fulfilled*, used to declare that a concern/responsibility is successfully accomplish after the composition rule and (c) *Failed*, used to declare that a concern/responsibility is not accomplish after the composition rule.

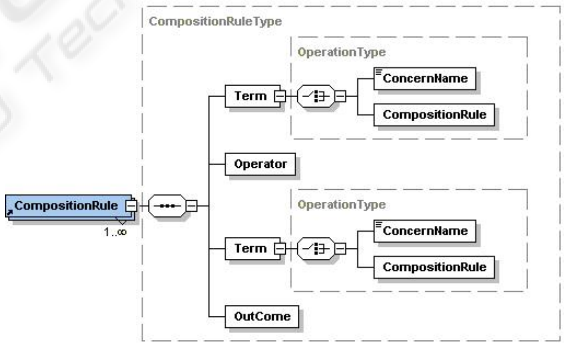

Figure 10: CompositionRule tag.

The CompositionRule tag includes the attribute *brackets* which value can be set to true or false. The value true means that open and close brackets are added to the composition, representing the term "(Composition)" in Figure 8.

Figure 11 represents the XML Schema for composition rules in a given match point.

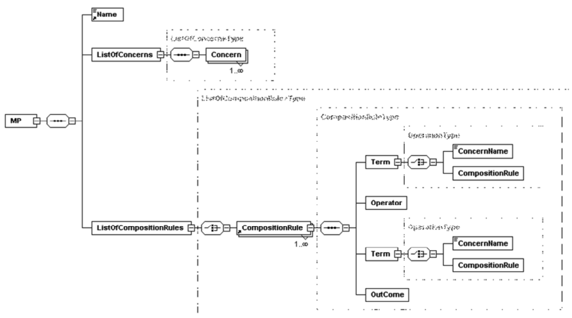

Figure 11: XML schema for composition in a match point.

The **ListOfCompositionRules** tag represents the set of all composition rules that take part in a match point.

3.4 A tool for AORE

A tool, called APOR (AsPect-Oriented Requirements tool), to support the model presented in Section 2 has been developed. This tool handles each concern as defined by the *concern* type metadata structure, helps identifying match points and defining composition rules on those points, generates the list of crosscutting concerns, and locates possible conflicting situations (on match points). This tool is composed of four views (see Figure 12): Concern View, Stakeholder View, MatchPoint View and Composite View.

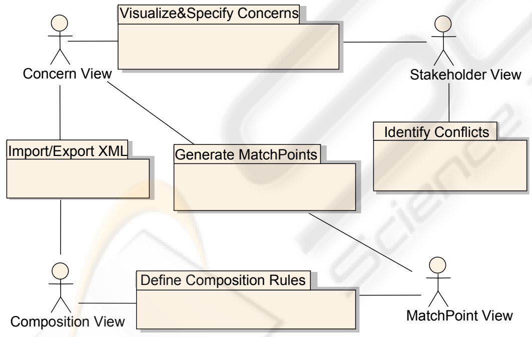

Figure 12: APOR tool main modules.

The Concern View supports the visualization of the concerns being specified, generates a list of match points needed within the composition task and offers import and export capabilities for XML documents describing concerns. This view also offers information regarding the crosscutting nature of the concern. The Stakeholder View allows the definition of priorities for each concern and identifies conflicting situations. The MatchPoint View supports the definition of composition rules for each match point, being also able to generate a list of both match points and crosscutting concerns. Finally, from the Composition View the user can also access and handle composition rules as well as the import and export features to manage XML documents defining composition rules.

The tool is developed in Java and JDOM (Hunter, 2005). The import and export features play an important role, since they offer the possibility to integrate the models and documentation produced by our tool with other tools.

4 AN ILLUSTRATIVE EXAMPLE

The example chosen is based on the Washington subway system taken from (Brito 2004):

"To use the subway, a client has to own a card that must have been credited with some amount of money. A card is bought and credited in special buying machines available in any subway station. A client uses this card in an entering machine to initiate her/his trip. When s/he reaches the destination, the card is used in an exit machine that debits it with an amount that depends on the distance travelled. If the card has not enough credits the gates will not open unless the client adds more money to the card. The client can ask for a refund of the amount in the card by giving it back to a buying machine."

Task 1: Identify concerns. Based on the requirements given above, we identify the following concerns: BuyCard, EnterSubway, ExitSubway, RefundCard and CreditCard. Functional concerns, as these, are usually not too difficult to identify. A closer look at the requirements can also suggest some non-functional requirements. For example, the text "special buying machines available in any subway station", suggests that "availability" is important. In fact, from our knowledge of the real world, the system should be available in 18/7 basis. Other concerns we should consider is Response Time, since the system needs to react in a short amount of time to avoid delaying passengers.
Other concerns can be identified based on the NFR catalogue (Chung, 2000). For each entry in the catalogue, we must decide whether it would be useful in our system or not. For example, the system

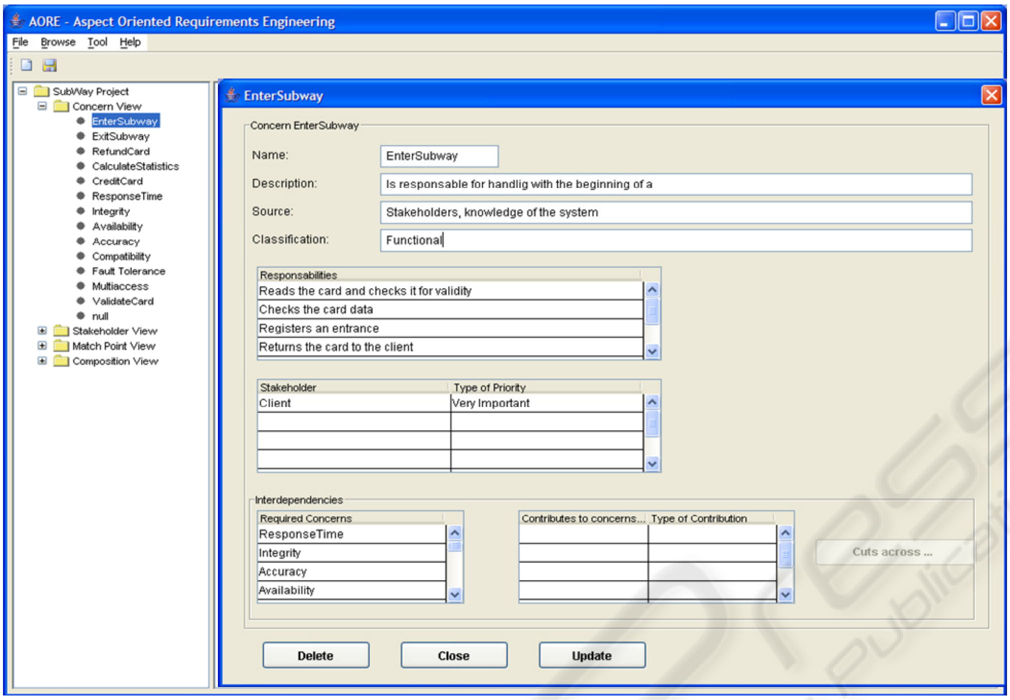

Figure 13: Defining EnterSubway concern.

can be used by many passengers at the same time, then Multi-access is an issue that the system needs to address. Other concerns identified based on this catalogue are: Accuracy, Security, Compatibility and Fault Tolerance. A more detailed analysis of the NFR framework catalogue for security, for example, would suggest us to also include in this list its decompositions, which are Availability and Integrity. We already have availability, so we also added integrity. Also, from the catalogue, we can see that response time is an element of performance.

In a second iteration, and during a more refined analysis of the templates and while building the use case diagram, we decided that it would make sense to factor out from EnterSubway, ExitSubway, CreditCard and RefundCard the common behaviour ValidateCard.

Task 2: Specify concerns. In this task we need to complete the specification of each concern by filling in an instance of the meta-template in Figure 7. This is achieved by using the APOR tool. Figure 13 illustrates part of the views (left hand-side) supported by the tool, together with the EnterSubway specification template (right hand side). In particular, the concern view shows the list of concerns of the subway system. This interface is conformant with the XML schema defined for the concern type.

Task 3: Compose concerns. The goal is to compose all concerns to obtain a picture of the whole system. A composition rule will be defined per match point. Its structure is according to the meta-template specified in XML in Figure 11.

Figure 14, an instance of the definition in Figure 11, shows an extract of the composition rule for the match point EnterSubway.

This rule is encapsulated in an *MP* tag and has an identification attribute "id" that is unique. The *Name* tag indicates the match point name (line 2) while lines 3-12 list the concerns in the match point. The *ListOfCompositionRules* tag (line 13) is composed of seven simpler *CompositionRule*. Such composition rules are encapsulated by a *CompositionsRule* tag (line 14). Each *CompositionRule* is composed by two terms, one operator and one outcome. Moreover, a *Term* encapsulates a *ConcernName* or a *CompositionRule*. Lines 15-21 compose "Availability" (*ConcernName* tag) in *parallel* (*Operator* tag) with "MultiAccess" (*ConcernName* tag). Note that attribute *brackets* has value true, which means that open and close brackets are added to the composition. The *outcome* of this composition states that EnterSubway is satisfied, through the attribute *action*. Similar rules are defined between this rule and the remaining concerns in the match point. For example, lines 25-34 compose a *CompostionRule* (*composition-rule_id=5*) in *sequence* with Integrity (c*oncernname_id=5*). The *outcome* is the fulfilment of EnterSubway.

```
<MP id="1">
  <Name>MP EnterSubway</Name>
    <ListOfConcerns>
      <Concern id="1" name="Performance.ResponseTime" rank="6"/>
      <Concern id="2" name="EnterSubway" rank="3"/>
      <Concern id="3" name="ValidateCard" rank="4"/>
      <Concern id="4" name="Accuracy" rank="5"/>
      <Concern id="5" name="Integrity" rank="2"/>
      <Concern id="6" name="MultiAccess" rank="1"/>
      <Concern id="7" name="Security.Availability" rank="2"/>
      <Concern id="8" name="FaultTolerance" rank="7"/>
  </ListOfConcerns>
  <ListOfCompositionRules>
    <CompositionRule id="1“ brackets="true"> >
       <Term>
          <ConcernName concern_id="7“>Availability</ConcernName>
       </Term>
       <Operator name="||"/>
       <Term>
          <ConcernName concern_id="6">MultiAccess </ConcernName>
       </Term>
       <OutCome action="Satisfied" concernName="EnterSubway"/>
    </CompositionRule>
          ...
    <CompositionRule id="6“ brackets="true >
       <Term>
          <CompositionRule compositionrule_id="5"/>
       </Term>
       <Operator name=">>"/>
       <Term>
          <ConcernName concern_id="5">Integrity</ConcernName>
       </Term>
       <OutCome action="Fulfilled" concernName="EnterSubway"/>
    </CompositionRule>
          ...
  </ListOfCompositionRules>
</MP>
```

Figure 14: XML for EnterSubway match point.

Currently, APOR is being changed to incorporate a parser that checks the correctness of each composition rule and edits it in the following form:

```
((Availability || Multiaccess)
   >> ( (ValidateCard >> EnterSubway)
   ||
    ResponseTime
   ||
   Accuracy)
  >> Integrity)
[> FaultTolerance
```

This composition rule expresses the order in which each concern must be satisfied. The functional concern EnterSubway requires prior satisfaction of ValidateCard. These two concerns, however, can only be satisfied if Avaliability and MultiAccess are guaranteed (in parallel). The resulting composition is then offered in parallel with ResponseTime and Accuracy. And only after the successful satisfactions of this composition will Integrity be satisfied. If something goes wrong with any of the above concerns, FaultTolerance interrupts the behaviour of that concern and starts its own behaviour.

5 RELATED WORK

The increasing interest for the aspect oriented software development has taken researchers to define modeling languages for aspect models (Chavez, 2004) (Han, 2004). These languages extend object languages to portray the same type of concepts at different abstraction levels. (Bakker, 2005), for example, defines a language to handle transversal characteristics at the requirements level. (Moreira, 2002), (Rashid, 2003) and (Sousa, 2004) use templates to represent candidate aspects and to show the impact of concerns over others. The approach in (Rashid, 2003) is based on separating the specification of aspectual requirements, non-aspectual requirements and composition rules in modules representing coherent abstractions and following well-defined templates.

Our approach differs from the above by offering a set of operators that are simpler to understand than those in (Rashid, 2003). Also, our language is at a more operational level, showing explicitly the order of composition. Moreover, our concern template is a lot more complete, offering more information.

A metadata driven approach for the creation of a repository for aspects is discussed in (Ferreira, 2005). A XML schema is used to represent this structure in order to guarantee information independent from representation, traceability of aspects through the development phases and versioning control. Our goal is not the creation of a general repository for aspects. The concerns in a given project are stored in eXist, a XML native database.

The Theme approach provides support for aspect-oriented development at analysis and design levels (Baniassad, 2004)(Clarke, 2005). At the analysis level, Theme/Doc is carried out by first identifying a set of actions in the requirements list which are, in turn, used to identify crosscutting behaviours. At the design level, Theme/UML allows a developer to model features and aspects of a system, and specifies how they should be combined. Our approach differs from this one since Theme does not offer a well-defined concern specification language neither does it offer the possibility of composing themes together to study the impact of each crosscutting concern on the system.

6 CONCLUSIONS AND FUTURE WORK

This paper focuses on the representation and composition activities of AORE approach, by proposing an XML-based language to specify and compose concerns at the requirements level. The use of XML ensures that the approach remains adaptable to other applications and extensible to incorporate new concerns and composition rules. The reason why XML schema has been chosen is because it guarantees information independency between representations, promoting traceability of concerns through the software development phases. The approach is supported by the APOR tool, facilitating the specification of concerns, identification of crosscutting concerns, generation of the match point table and definition of composition rules.

Currently we are rebuilding the tool to adopt an Model-Driven Development strategy through the use of the AORE metamodel, together with the XML Schemas defined here. At the same time, we are refining the composition rules to define constraints at the responsibility granularity level. We are also exploring how fuzzy logic can be applied to help solving conflicts that can arise when concerns that contribute negatively to each other need to coexist in the same match point. In the near future we plan to integrate with the method, and the tool, a reference model to support forward and backward traceability.

ACKNOWLEDGEMENTS

This work is supported by the Portuguese FCT Grant SOFTAS (POSI/EIA/60189/2004).

REFERENCES

Baniassad E., Clarke S., “Theme an Aproach for Aspect-Oriented Analyssis and Design”, International Conference on Software, Engineering 2004, Edinburg, Scootland, 2004.

Bakker J.,Tekinerdogan B.,Aksit M.; Characterization of early aspects approaches; Proceedings of the Early Aspects Workshop at AOSD’05, 2005.

Bergmans L. and Aksit M., "Composing Crosscutting Concerns using Composition Filters", *CACM*, 44(10), 2001.

Brinksma E. (ed): Information Processing Systems - Open Systems Interconnection – LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, ISO 8807, 1988.

Brito, I. and Moreira A. “Advanced Separation of Concerns for Requirements Engineering”. Jornadas.de Ingenieria del Software y Bases de Datos, Alicante, Spain, 2003.

Brito, I. and Moreira A. “Integrating the NFR Approach in a RE Model”. Early Aspects Workshop at AOSD’04. Lancaster, UK. 2004.

Chavez C.; A Model-Driven Approach to Aspect-Oriented Design; PhD Thesis, Computer Science Department; PUC-Rio; Brazil, April 2004.

Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers, 2000.

Clarke, S., Walker, R.J.: Composition Patterns: An Approach to Designing Reusable Aspects, Proceedings of ICSE’01, Toronto, Canada, 2001.

Clarke, S., Baniassad E., Aspect-Oriented Analysis and Desing: The theme Approach, Addision-Wesley, 2005

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

Ferreira, R., Raminhos, R., Moreira, A.: Metadata Driven Aspect Specification, Workshop on Aspect Oriented Modeling, 8th Int. Conf. on MoDELS, Jamaica, October 2005.

Finkelstein, A. and Sommerville, I.: "The Viewpoints FAQ." *BCS/IEE Software Engineering Journal*, 11(1), 1996

Han Y., Gunter Kniesel G. e Cremers A.; A meta model for AspectJ; Technical Report IAI-TR-2004-3, Computer Science Department III, University of Bonn. ISSN 0944-8535. October 2004.

Herrero. J., Sánchez F., Lucio F., Toro M.: Introducing Separation of Aspects At Design Time, Workshop on Aspects and Dimensions of Concerns, ECOOP’00, France, 2000.

Hunter, J., McLaughlin, B., Easy Java/XML integration with JDOM, http://www.javaworld.com, 2005

Jacobson, I., et al: Object-Oriented Software Engineering –a Use Case Driven Approach: Addison-Wesley, 1992.

Lieberherr, K. J.; Orleans, D., Ovlinger, J.: Aspect-Oriented Programming with Adaptive Methods, CACM, Vol. 44, No. 10, pp. 39-41, 2001.

Kiczales G., Lamping J. , Mendhekar A., Maeda C., Lopes C., Loingtier J.-M., and Irwin J.. Aspect-oriented Programming. In ECOOP'97, LNCS 1241, pp 220-242, Finland, 1997

Moreira, A., Araújo, J., Brito, I.: Crosscutting Quality Attributes for Requirements Engineering, 14th International Conference on Software Engineering and Knowledge Engineering, ACM Press, Italy, July 2002.

Moreira, A., Rashid A., Araújo, J., Multidimensional Separation of Concerns in Requirements Engineering, 13th International Conference on RE, IEEE Press Paris, France, August - September 2005

Rashid, A., Moreira, A., Araújo, J.: Modularization and Composition of Aspectual Requirements", 2nd

International Conference on AOSD, ACM Press, pp. 11-20, Boston USA, 2003.

Sebesta, Robert, Concepts of Programming Languages, Sixth Edition, Pearson Education, 2003

Sousa G., Soares S., Borba P., Castro J.; Separation of crosscutting concerns from requirements to design: Adapting the use case driven approach, Proceedings of the Early Aspects Workshop at AOSD'04, UK, 2004.

Suzuki J. and Yamamoto Y., Extending UML with Aspects: Aspect Support in the Design Phase, ECOOP Workshop on Aspect Oriented Programming, Lisbon, Portugal, 1999.

Tarr P. L., Ossher H., Harrison W. H., and Sutton S. M., "N Degrees of Separation: Multi-Dimensional Separation of Concerns", ACM, pp. 107-119, ICSE, Los Angeles, USA, 1999.

W3C Recommendation, World Wide Web Consortium, “XML 1.0. (2004) Extensible Markup Language (XML) 1.0 (Third Edition)”, http://www.w3.org/TR/2004/REC-xml-20040204.

Xerox Parc, AspectJ home page, Technical report, http://www.aspectj.org/, 2001.