

Metadata Driven Aspect Specification

Ricardo Ferreira
UNINOVA – Desenvolvimento de
Novas Tecnologias
2829-516 Caparica, Portugal
(+351) 21 294 9625
rmf@uninova.pt

Ricardo Raminhos
UNINOVA – Desenvolvimento
de Novas Tecnologias
2829-516 Caparica, Portugal
(+351) 21 294 9625
rfr@uninova.pt

Ana Moreira
CITI/Dept. de Informática
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
(+351) 2129 485 36
amm@di.fct.unl.pt

Abstract. Metadata provides the background information that describes the content, quality, structure and other appropriate data characteristics. This paper proposes the use of metadata to define a meta-aspect repository in the scope of early aspects¹. The repository stores the necessary information to support navigation over all stored information for future reuse, versioning control and traceability from requirements to implementation.

1. Introduction

The term *concern* represents a problem that a software program tries to solve. Thus, the expression “separation of concerns” refers to the ability to identify, modularize and use such parts of a software program. The advantages of such an approach are known to any software engineer: reducing software complexity, providing clear localization in case of code change and increasing the understandability of the whole software system.

However, some concerns are difficult to encapsulate since their implementation is spread throughout different modules. Security and logging are some examples of these “crosscutting concerns” which result in an increase on software complexity and a reduction in the code quality and understandability.

Several engineering techniques (e.g. [1, 2, 3]) have shown how to modularize crosscutting-concerns. Nevertheless, the proposed solutions are organized through a minimum set of fields with insufficient information to document decisions or to support traceability throughout the lifecycle. The present work provides a practical answer to contribute to the solution of these problems by proposing an open-source metadata repository that can conglomerate and store the necessary aspect-related information to support navigation over all stored information for future reuse, versioning control, and traceability from requirements to implementation through meta-relations.

Metadata provides the background information that describes the content, quality, structure and other appropriate data characteristics. It is used to organize the data in a structure that although extensible, offers all required information to the end-user.

This paper is organized as follows. Section 2 introduces other works related with our approach. Section 3 presents the meta-structure suggested to organize aspects, while Section 4 introduces a XML Schema to support such meta-structure. Section 5 suggests a meta-aspect repository with its main functionalities, showing how these can be put to practice. Finally, Section 6 draws some final remarks and proposes future work.

¹ Early aspects are crosscutting concerns that refer to the requirements elicitation, analysis and architecture design activities.

2. Background

The metadata definition proposed in this work results from the hybrid composition of two technologies: metadata repository (XML based) and aspect specification.

Metadata can be used for two purposes: documentation and operations. The advantages of a repository for storing and managing metadata were already assessed in the scope of the SEIS project [5, 6]. In this project, metadata described the parameters available in the system (around 850) and relations between them, besides operational configurations for each application / component of the system.

Since early aspects represent the “aspect core” from which implementation and architectural aspects can be derived from, the work presented here will focus on them. Within the scope of “early aspects”, a multidimensional approach was taken for the identification of crosscutting concerns since it provides a better way for handling functional requirements compared with the traditional bi-dimensional approaches based on a dominant decomposition using viewpoints, use cases or goals (e.g. [7]). As the work in [8] using a multidimensional approach proved to be quite complete and simple, it was taken as reference.

Previous experiences in the area of design pattern structuring [4] proved to be very useful, and some fields containing metadata information were inherited from this work.

Although the metadata definition described in the following sections for a meta-aspect repository is rather innovative, the idea of constructing a repository for holding aspect information has been previously addressed in [2]. Nevertheless, there are differences between the two approaches. While [2] defines a repository for storing component aspects, we suggest that early-aspects should be the core aspect information to store in a repository, completed with the definition of architectural and implementation aspects in future work. Finally, none of the validation schemes or navigation and traceability capabilities are available in [2].

3. A meta-template for aspect representation

This section discusses the information required for the representation and logical structure of early aspects. As mentioned in section 2, the approach presented in [8] is taken as reference, since it proposes a quite complete template to specify and document concerns.

Two main structures are proposed for aspect representation: *System* and *Concern*². Both are defined through templates with three columns: field name, cardinality (i.e. number of occurrences) and description.

The structural representation for the *System* template is depicted in Table 1. At this level, the main objective is to offer a clear presentation for the system in terms of the context in which it is applicable and, on the other hand, define the consequences resulting from its usage. Most of the meta-fields presented in this template were inherited from the structure proposed in [4] for the representation of analysis patterns. Nevertheless, some changes were performed in order to keep information fields to a minimum. On the other hand two new fields were introduced: “Concerns” which holds information regarding all concerns related with the system, and “Similar Systems” that provides the possibility to navigate in the definition of other systems that are somewhat similar to the present one.

Table 1 - System template

| Field name | # | Description |
|-----------------|---|---|
| Name | 1 | System’s name. |
| Alias | N | Additional names that also identify the system. |
| Version | N | Chronological register of all previous versions of this system. The following fields are suggested: Date, Author, Reason and Changes. |
| Context | 1 | Description of the environment in which the problem and solution occur and for which the solution is desirable. |
| Motivations | N | Description of problematic situations intended to motivate the use of this system. |
| Concerns | N | List of concerns involved in the problem modelling. |
| Similar systems | N | Other systems similar to the current one. |
| Known uses | N | Describes known occurrences and applications of the system. |

Each System is identified by its *Name* that is unique within the scope of the repository and works as a primary key for referencing the System information. The *Alias* field represents a set of names that are commonly used when referring to the same system (considered as synonyms). The *Version* field provides versioning control regarding all changes performed previously on the system. Versioning information comprises four fields: the *Date* when the modification took place, the *Author* responsible for it, the *Reasons* that motivated the modification and a list of *Changes* that were performed over the template. The *Context* field presents the general environment from which the need of the modelled system emerges from, while the *Motivations* field provides a concrete set of examples where the usage of this system is advisable. The *Motivations* field can be seen as the

² Please note that only the logical information is presented here. Although the main ideas will hold, there will be differences to the implementation.

instantiation of several practical examples for an abstract system (similar to the results obtained from the usage of Product Lines [9]). The *Concern* field (each concern is defined in table 2) contains the list of concerns that compose the system. The *Similar systems* field offers a navigation mechanism to other systems that are somewhat related to this one and which may offer a more suitable solution for the developer needs. Finally, the *Known uses* field presents a set of “real world” applications that implement the system defined in the template, providing some “warranty” over the quality of the proposed system.

Table 2 presents the template for describing a concern. While the first nine fields were borrowed from [8], the remaining fields are new additions needed for the repository. Commonly to the *Name* field in the System template, the *Concern name* entry identifies a concern within the scope of a system and acts as a primary key.

Table 2 - Concern template

| Field name | # | Description |
|------------------------|------|--|
| Concern name | 1 | Concern name. |
| Source | N | Information source (e.g. domain, stakeholders, business process, documents and catalogues). |
| Stakeholders | N | Users that need the concern in order to accomplish their job. |
| Description | 1 | Short description of the intended behaviour of the concern. |
| Responsibilities | N | List of what the concern must perform; knowledge or properties the concern must offer. |
| Contributions | N | List of concerns that contribute or affect this concern. Contribution can be positive (+) or negative (-). |
| Stakeholder priorities | N | Expresses the importance of the concern for a given stakeholder. It can take the values: <i>Very Important</i> , <i>Medium</i> , <i>Low</i> , <i>Very Low</i> or <i>None</i> . |
| Priority | 1 | Expresses the importance of the concern within the system. |
| Required concerns | N | List of other concerns needed or requested by the concern being described. |
| Outcome | 1 | The outcome result for this concern at implementation level. |
| Similar concerns | N | Other concerns similar to this one. |
| Refinement concerns | N | Implementation or architectural concerns that can be used to refine the current concern. |
| Extensions | 0..N | Additional extensions to the definition of the concern. |

The *Source* field states the origins of the concern, having several possible values, as external catalogues of non-functional requirements [10]. The *Stakeholders* field shows which stakeholders interact with the concern, while the field *Description* provides a textual explanation about the concern’s importance. In the *Responsibilities* entry, information about the requirements related with the concern is provided, while in the *Contributions*

field a list of positive and negative interactions with other concerns is offered. This field helps detect conflicts whenever concerns contribute negatively to each other. These conflicts may be resolved through the *Stakeholder priorities* field, which attributes priorities to concerns from the stakeholders' perspective. The *Priority* field expresses the prioritisation of the concern within the system. For this field we have followed the MoSCoW rules expressed in [11].

A concern may have four types of references to other concerns / refinement aspects:

- o The *Required concerns* field acts as a dependency reference to other concerns within the same System;
- o The *Similar concerns* field represents a navigation reference from an early aspect concern to another early aspect concern, not necessarily from the same System;
- o The *Refinement concerns* field holds two types of references used to further refine the early aspect concern: to an "Architectural Aspect"; or to an "Implementation Aspect".

The usage of the *Refinement concerns* field is closely related with the value present in the *Outcome* field. This contains information about the result of the concern (e.g. aspect, architectural choice, function). Finally, the *Extensions* field offers the developer a chance to include further information that does not fit in any of the previous fields. Any type of structure can be fit in this field since it has an "Any" data type. For this reason no validation scheme is available for the field content.

4. XML Schema for the meta-template

In this section we present a XML Schema compliant with the logic present in the meta-template previously defined. 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 has the power to define a full specification for a XML document, enabling the creation of a set of rules to structure and form a XML document, as well as rules for data types and integrity. For these reasons our meta-template for aspect representation will be mapped into a XML Schema, creating a standard representation. Any instance that follows our meta-template must also be correctly validated with the XML Schema defined in this section.

The construction of the XML Schema has been done in a modular way, factoring out common definitions for optimal understanding and future extensibility. The schema includes special types to represent possible navigation points between entities, providing navigation and traceability capabilities to the meta-aspect repository.

In the meta-template structure, we proposed two templates for the representation of *Concerns* and *Systems*, which shall be matched into two XML Schema complex types. In the following subsections these types and respective schemas are presented and justified.

4.1 The Concern Type

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*, for example, can be mapped directly into the logical meta-template.

Name. This field defines a concern name that shall be unique within the scope of the system to which it belongs. Since all names in our metadata solution are seen as keys, uniqueness is required. The basic meta-rule for creating unique concern names is followed: `<SystemName>.<ConcernName>.<Version>`

Where *SystemName* represents the unique system's name to which the concern belongs to and *Version* refers to the concern versioning value.

Description. A textual description of the concern.

Sources. This element states which information sources 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", "Concern", "Catalogues" or "System".

Priority. This element defines the priority of this concern within the system. The priority is properly enumerated with the MoSCoW: "Must have", "Should have", "Could have", "Want to have".

Stakeholders. Define the list of stakeholders that require the concern and the concern's priority for each stakeholder (Figure 1).

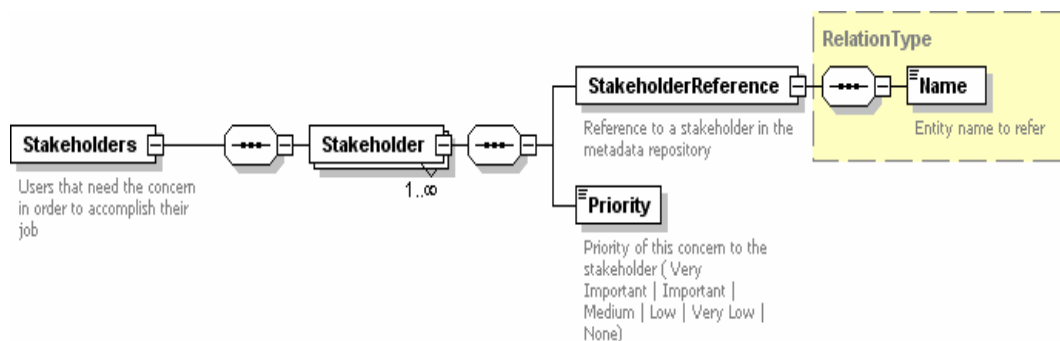


Figure 1: Stakeholders element

Multiple stakeholders can be associated to the same concern and each may have a different priority to each stakeholder. Priorities are properly enumerated from "Very Important" to "None". In this element we make use of complex type *RelationType* to inform the system that the stakeholder name should be a well-

known name in the repository. With this relation (besides being a validation procedure) it will be possible to navigate directly to the stakeholder definition.

Responsibilities. Define the concern responsibilities, i.e. the requirements that the concern should fulfil. Multiple textual

entries are possible (one per requirement) and a minimum of one entry is mandatory.

Contributions. Define the relations between the current concern to others. Contributions between concerns respect an enumeration and must be either positive “+” or negative “-”.

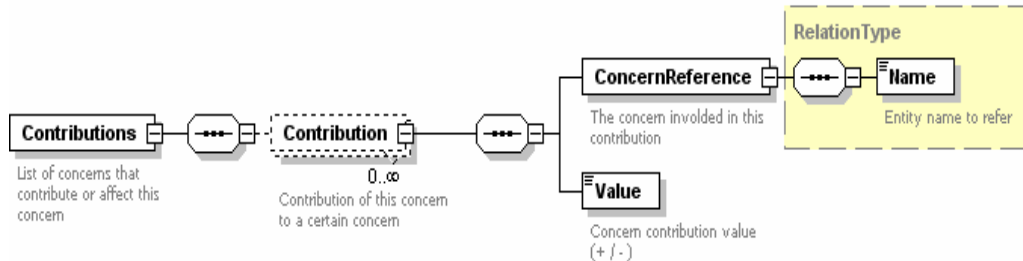


Figure 2: Contributions element

Required Concerns. Define a list of other concerns from which the current concern depends. This is accomplished, once again, recurring to relations to other concerns. However, in this case, the existence of such relations is not mandatory since a concern can be independent of all the others (see Figure 3).

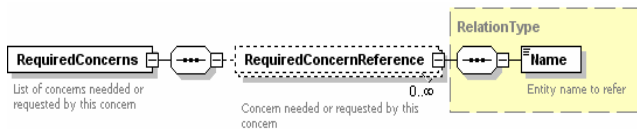


Figure 3: RequiredConcerns element

Similar Concerns. Define a set (possibly empty) of similar concerns to this one, in logic or behaviour, using relations between concerns. Its definition is similar to that in Figure 3.

The reader should notice the usage of the complex type *RelationType* when referring to another concern. Besides being used for validation and navigation, this relation will be used later in the resolution of conflicts between concerns (see Figure 2).

Outcome. Contains the definition on what the concern became in the system implementation. The *Refinement* element in our template is related with the outcome of the concern, therefore in the schema design that information has been included inside the *Outcome* element (Figure 4).

A description is provided for the concern outcome, and the *Refinement* element holds a list of *Implementation Aspects*, *Arquitectural Aspects* or other kinds of refinements. The use of *RelationType* enables future navigation capabilities allowing answers to questions like: “From what concern does this Architectural Aspect come from?” or “What were the requirements that originated this Implementation Aspect?”

Extensions. This is an optional open element (no validation shall be performed over the XML structure stored in this field) that the developer can use to store XML content describing some feature that was not possible to represent in the previous fields.

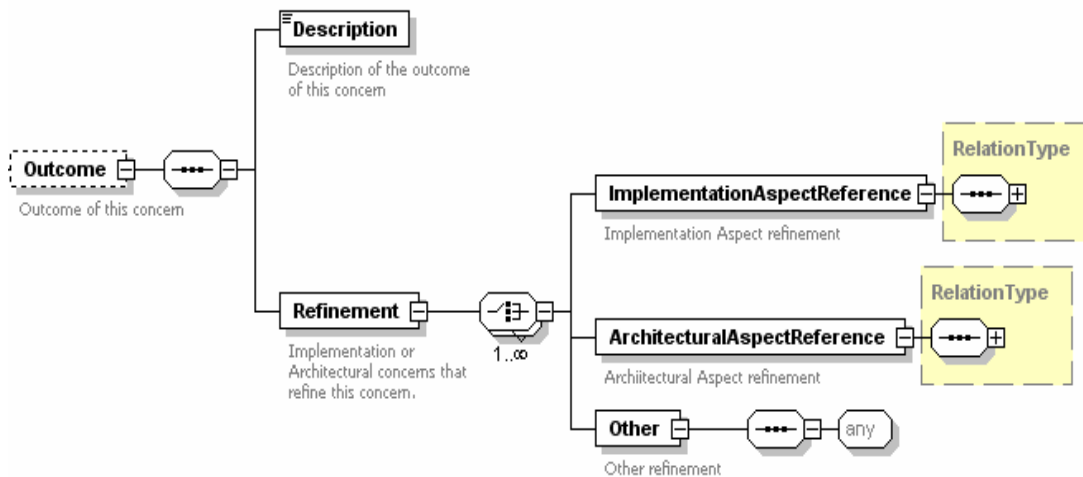


Figure 4: Outcome element

4.2 The System Type

Once again the reader can find a parallel between the structure of the complex type *System* and the logical meta-template.

Name. Mandatory element that is unique within the repository scope, acting as primary key.

Aliases. This optional element defines a set of synonym names for which the system is also known.

Context. Provides a textual description of the generic environment in which the problem occurs.

Motivations. Describe a set of application examples of this system in similar contexts.

Versions. Multiple versions are allowed for the system; each may contain a different set of concerns involved in it (Figure 5).

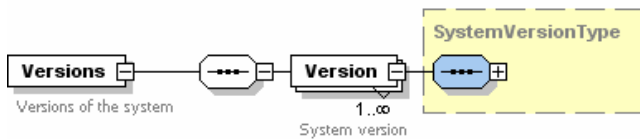


Figure 5: Versions element

Figure 6 presents the structure of the type *SystemVersionType* that represents one version of the system.

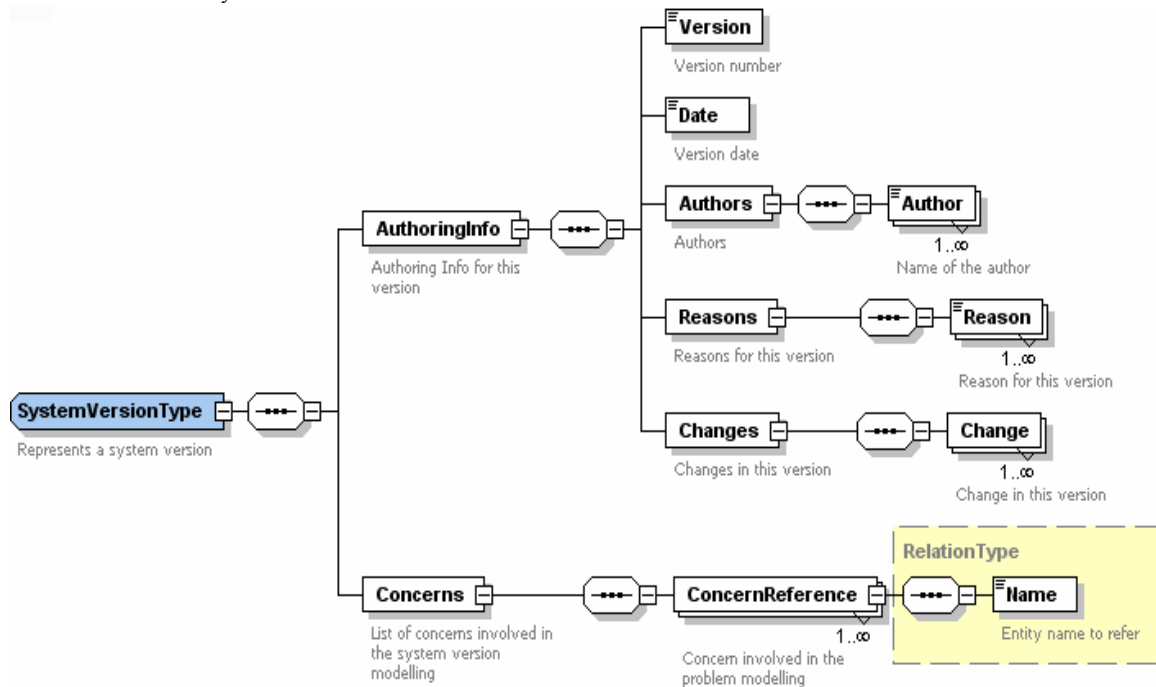


Figure 6: SystemVersionType

SimilarSystems. Provide references to other systems that may help the developer resolve his/her problem (Figure 7).

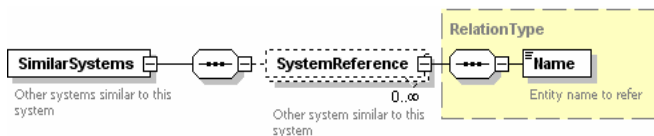


Figure 7: SimilarSystems element

KnownUses. Optional set of textual entries that describe the application of the current system in “real-world” situations.

5. Meta-Aspect Repository

The Metadata driven Aspect Repository aggregates the XML Schema specifications defined previously. The metadata entities in the repository (see Figure 8) are systems, concerns, stakeholders’ information and concern refinement aspects (architectural or implementation), where the “Relation Types” work as references between concepts. To ensure correctness, these are validated using the proposed XML Schema. Also, all the relations in these entities must be valid. Additional data structures

This type contains the element *AuthoringInfo* that gives all *Authors* that performed some *Changes* on the system having a set of *Reasons* as basis. Each system version also contains a set of concerns that are represented inside the *Concerns* element. Note the use of *RelationType* as it is crucial for the repository to know which concerns a certain system version is using.

will be introduced for managing these relations that allow bi-directional navigation between entities.

The repository acts as open infrastructure, where applications can be built on top of it to edit, visualize or simply to make use of metadata.

We propose building three applications on top of this infrastructure: an “Editor” a “Visualizer” and a “Validator”³.

5.1 Editor Application

This application will help the developer managing the whole repository metadata: editing systems, concerns, stakeholders, and refinement aspects besides ensuring versioning control. The application will act as a specialized editor of XML instances based on the XML Schema presented, helping the user inputting data, rather than editing a raw XML file.

³ At the time the paper was written the implementation of the supporting tools for the metadata repository had not started and the work done was at a conceptual level only.

5.2 Visualizer Application

This application provides a visualization of a system definition, featuring tracing and navigation capabilities within the system. Since our approach encloses the whole software development lifecycle (from requirements to implementation), it is possible to trace back the influence of several choices and requirements in the system. Knowing what originated a certain Implementation Aspect the user can trace back to the concern that originated it, check its stakeholders, and also its requirements. This navigation is possible because of the relation mechanism in the repository.

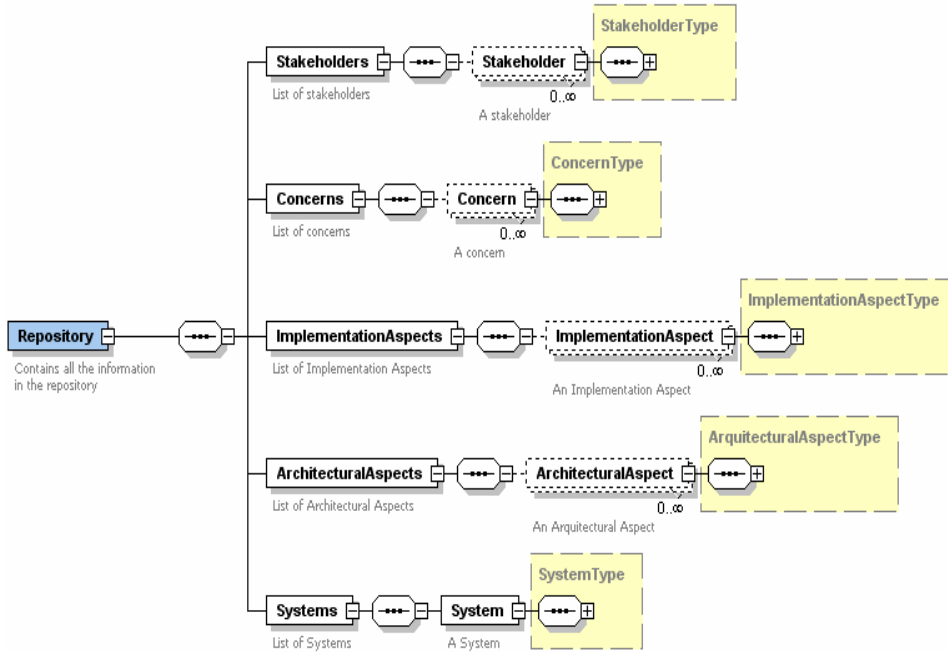


Figure 8: Metadata Repository entities

5.2.1 UML Requirements Analysis Model

Based on the concern information it is possible to create an UML use case model. Each concern is mapped into a use case and the concern's stakeholders are mapped into actors. Using the "Required Concerns" field it is possible to introduce relations between use cases, namely <<include>>. We can also create new stereotypes, such as <<contributes>> to show contribution between use cases.

Table 3 – Visualizer Outputs

| Output | Description |
|----------------------------------|--|
| UML analysis model (use cases) | UML use case model where each concern is mapped into one use case. |
| Match Point Identification table | Table containing the point where the composition will take place. |
| Stakeholder Match Point table | Match Point table in a Stakeholder perspective. |
| Composition Rules | Defines the order in which the concerns will be applied in a particular match point. |

Navigation is also possible at the concern level, checking similar or required concerns. The same is applicable to the system layer, where several systems can be compared to one another.

Four types of outputs are possible using the Visualizer (Table 3).

Extensions to these outputs can be generated only with the available metadata information. For instance, it is possible to have a comparison view between the state of the system after the requirements phase, and its state at the end of the implementation phase. Other possible output is to generate system/concern tables, possibly with different formats (e.g. HTML, PDF).

5.2.2 Match Point Identification Table

Based on the "Required Concerns" field, a bi-dimensional table of "Concerns versus Required Concerns" can be generated. Cells are marked if a given concern at the column "Concerns" is required by the concern at the row "Required Concern". So, a match point [8] for a certain concern gives which other concerns must be composed with it (Table 4).

Table 4 – Match Point Identification Table

| | | Concerns | | |
|-------------------|----|------------------|----|------------------|
| | | C1 | C2 | C3 |
| Required Concerns | C1 | | | |
| | C2 | ✓ | | ✓ |
| | C3 | ✓ | | |
| Match Points | | MPC ₁ | | MPC ₃ |

Looking at the table two match points were detected: $MP_{C1} = \{C2, C3\}$ and $MP_{C3} = \{C2\}$.

5.2.3 Stakeholder Match Point Table

This output provides a bi-dimensional table with all the match points detected in the previous output in one dimension and

system's stakeholders in other dimension [8]. Since a match point is composed as a set of concerns, a mapping between these concerns and the stakeholders is made (Table 5).

Table 5 – Stakeholder Match Point Table

| | | Match Points | |
|--------------|-------|------------------|------------------|
| | | MPC ₁ | MPC ₃ |
| Stakeholders | Stk 1 | | |
| | Stk 2 | C3 | |
| | Stk 3 | C2 | C2 |

5.2.4 Composition Rules

Composition Rules can be created (one per Match Point) taking as input the Match Points previously defined. Conflicts are detected if within the same Match Point exist at least two concerns that contribute negatively to each other (this information is in Contributions field). The “Stakeholder Priority” field can resolve the conflict if different priorities are set to these concerns. In this case the higher priority concern prevails. Otherwise, if the concerns have the same level of priority it will be necessary to negotiate with the Stakeholder involved in the conflicting concerns (previously identified in Table 5).

Composition rules are created using the LOTOS notation [12] expressing the order in which each concern must be satisfied. This language offers sequence and parallel operators for the representation of priorities in the satisfaction of each concern.

Since the logic that needs to be expressed in this rule is highly dependent on the developer solution, our application will only provide a header describing the conflicting concerns and propose a rather simplistic solution based on the sequence of all involved concerns. Taking this as the starting point, the developer should define the correct composition behaviour.

5.3 Validator Application

This application will validate an entire system version. Part of the validation process depends on solving possible conflicts between concerns, which can be easily solved using the “Match Point Identification” and “Stakeholder Match Point” tables as previously described. If the system is validated correctly then all the outputs described in Table 3 can be provided to the user. Otherwise information is returned regarding the conflicting situation and the Editor application is opened automatically in the XML specification that might be wrong. After correcting the problem the validation process is restarted (iterative process). This validation process is interactive in the sense that it requires user feedback on step 1 (“Use Case Model”) and step 4 (“Composition Rules”) for determining the correctness of the output suggestion. Steps 2 (“Match Point Identification Table”) and step 3 (“Stakeholder Match Point Table”) are completely automated.

6. Conclusions and Future Work

This paper discussed a metadata driven approach for the structural representation of crosscutting concerns. Since “early aspects” were considered as an important “aspect core” from which all other (architectural or implementation) aspects and components derive from, an existing approach to aspect-oriented requirements

engineering was taken to serve as a foundation for the identification and structuring of the meta-aspect repository. The templates needed for the repository were inspired in a previous template created for the area of design patterns. The XML schemas for these templates were presented and a meta-aspect repository discussed showing the advantages of such representation – information independent from representation, traceability of aspects through the software development phases, incorporation of relations between aspects (e.g. conflicts or complementary) and versioning control.

For future work we intend to validate and refine the proposed XML schema with the representation of a higher spectrum of examples. Although the benefits of metadata architecture have been already proved within the scope of the SEIS project, we intend to implement a first prototype for the proposed meta-aspect repository, proving the feasibility and usefulness of such approach. Finally, it is our goal to implement the editor, the visualizer and validator applications.

References

- [1] Clarke, S., Walker, R., *Composition Patterns: An Approach to Design Reusable Aspects*, ICSE'01, Canada, 2001
- [2] Grundy, J., *Storage and Retrieval of Software Components using Aspects*, Australian Computer Science Conference, Australia, 2000
- [3] Tarr, P., Ossher, H., Harrison, W., Sutton, S., *N Degrees of Separation: Multi-Dimensional Separation of Concerns*, ICSE'99, 1999
- [4] Pantoquillo, M., Raminhos, R., Araújo, J., *Analysis Patterns Specifications: Filling the Gaps*, In Viking PloP'03, 2003
- [5] Pantoquillo, M., Viana, N., Ferreira, R., Pires, J., Donati, A., Baumgartner, A., Marco, F., Peñin, L., Hormigo, T., *SEIS: A Decision Support System for Optimizing Spacecraft Operations Strategies*, 2005
- [6] Ferreira, R., Pires, J., Martins, R., Pantoquillo, M., *XML Based Metadata Repository for Information Systems*, 12th Portuguese Conference on Artificial Intelligence, Portugal, 2005
- [7] Rashid, A., Moreira, A., Araújo, J., *Modularisation and Composition of Aspectual Requirements*, AOSD'03, USA, 2003
- [8] Brito, I., Moreira, A., *Integrating the NFR framework in a RE model*. Workshop on Early Aspects, at AOSD'04, UK, 2004
- [9] Ardis, M., Daley, N., Hoffman, D., Siy, H., Weiss, D., *Software Product Lines: a Case Study*, Software Practice & Experience 30, 2000
- [10] Chung, L., Nixon, B., Yu, E., Mylopoulos, J., *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, 2000
- [11] Stapleton, J., *Dynamic Systems Development Method*, Addison-Wesley, 1995, Pag. 28, 29
- [12] 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