A METADATA-DRIVEN APPROACH FOR ASPECT-ORIENTED REQUIREMENTS ANALYSIS

Sérgio Agostinho, Ana Moreira, André Marques, João Araújo
CITI / Departamento de Informática, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

Isabel Brito
Dep. de Engenharia, Instituto Politécnico de Beja, 7800-050 Beja, Portugal

Ricardo Ferreira, Ricardo Raminhos, Jasna Kovačević, Rita Ribeiro
UNINOVA – Instituto de Desenvolvimento de Novas Tecnologias, 2829-516 Caparica, Portugal

Philippe Chevalley
ESA/ESOC P.O. Box 299, 2201 AZ Noordwijk, The Netherlands

Keywords: XML, Metadata Repository, Aspect-Oriented Software Development, Requirements Analysis, Early Aspects.

Abstract: This paper presents a metadata-driven approach based on aspect-oriented requirements analysis. This approach has been defined in cooperation with the European Space Agency in the context of the "Aspect Specification for the Space Domain" (ASSD) project. ASSD aims at assessing the applicability and usefulness of aspect-orientation for the space domain (ground segment software projects in particular), focusing on the early stages of the software development life cycle. This paper describes a rigorous representation of requirements analysis concepts, refines a method for handling early aspects, and proposes a client/server architecture based on a metadata repository.

1 INTRODUCTION

Aspect-Oriented Software Development (AOSD) aims at providing improved modularisation and composition techniques to handle crosscutting concerns (Kiczales et al., 1997). Crosscutting concerns are encapsulated in separate modules, called *aspects*, and composition (or weaving) mechanisms are later used to weave them back to the base modules.

The ASSD approach proposed in this paper is an application of the metadata concepts introduced in previous work (Marques et al., 2007). The approach itself is a refinement of the Aspect-Oriented Requirements Analysis (AORA) framework (Brito & Moreira, 2003a, 2003b), a pioneer method in the domain. The ASSD approach addresses the identification, separation, representation and composition of crosscutting concerns at the requirements level. This early identification can provide a way to take into account options, tradeoffs and other decisions before the implementation or even the architectural design is derived.

The architecture and client tools that provide an actual concretization of the ASSD method are a refinement of previous work (Ferreira, Raminhos & Moreira, 2005). The architecture of the system relies on a Metadata Repository (Ferreira, Moura-Pires, Martins & Pantoquilho, 2005; Ferreira & Moura-Pires 2007) for supporting the approach definitions, to automatically generate specific documentation, and to provide the means for creating reusable catalogues (Chung, Nixon, Yu, & Mylopoulos, 2000).

This paper is organized as follows. Section 2 introduces the ASSD project and the base concepts that support the approach. Section 3 describes the

Agostinho S., Moreira A., Marques A., Araújo J., Brito I., Ferreira R., Raminhos R., Kovačević J., Ribeiro R. and Chevalley P. (2008).
A METADATA-DRIVEN APPROACH FOR ASPECT-ORIENTED REQUIREMENTS ANALYSIS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - ISAS*, pages 129-136
DOI: 10.5220/0001684101290136

ASSD model including the automatic document generation. Section 4 discusses the architecture implemented, i.e., infrastructure and client tools. Section 5 reports related work in the area and Section 6 discusses the case studies used in the validation of ASSD and draws some conclusions.

2 ASSD PROJECT AND MAIN CONCEPTS

The ASSD project (UNINOVA, 2007) aims at assessing the applicability and usefulness of aspect-orientation, focusing on the early stages of the software development life cycle. The project was developed for ESA in the scope of space domain, ground segment systems in particular, namely ESA Contract 19556/06/NL/JD/na.

ASSD was tested with two operational projects of the ground segment which followed a "traditional" object-oriented requirements analysis for their specification. A previous version of the Metadata Repository employed in the ASSD architecture was also analysed using the ASSD method. In order to exemplify some of the ASSD system functionalities and graphical capabilities the performed aspect analysis for the Metadata Repository component will be briefly described. A full analysis of this software component is outside the scope of the paper due to space limitations.

The ASSD approach is supported by XML technologies, namely XML Schema, which is used in the rigorous and non-ambiguous representation of supporting metadata concepts: *Stakeholder, Stakeholder Requirement, System Requirement, Concern, Decomposition Node, System*, and *Test Case* (Figure 1).

A *Stakeholder* describes an entity (e.g. person, company, university) responsible for defining the main capabilities (*Stakeholder Requirement*s) of a software application, usually during elicitation meetings. The concept's metadata is mainly descriptive regarding the contacts of the entity, e.g. address(es) and email(s).

A *System Requirement* refers to a technical requirement that allows an efficient mapping between a possible non-technical Stakeholder Requirement to a requirement that is non-ambiguous in its interpretation and can be understood by the development team. Depending on the elicited Stakeholder Requirements, these may be collapsed in one, or expanded in multiple System Requirements. Both concepts contain prioritization attributes that classify the requirement's importance within the devised System.

A *Concern* groups similar, or closely, related System Requirements. A concern refers to a property which addresses a certain problem that is of interest to one or more stakeholders and which can be defined as a set of coherent requirements. The concern's metadata contains a textual description, specifies navigation relations to similar concerns and some may have negative and positive contributions to others. Since concerns may be as abstract as required (e.g. Security, Performance), concerns can be decomposed based on the definition of *Decomposition Node*. The Decomposition Node concept is used to define a tree-based structure where the root node indicates a possible *operationalization* – an actual concretization of a possible abstract concern (Chung et al., 2000) – and where each child node indicates a refinement for that node. Simple logic operators may be used for defining the operationalization tree.

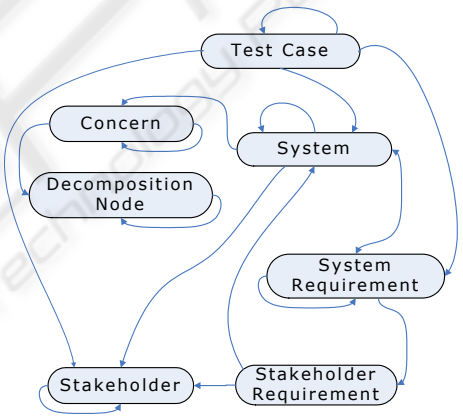

Figure 1: Concept's main relations.

The *System* concept represents an abstraction for a real software application or component, functioning as an integrator concept within ASSD; it joins together information about the concerns the application responds to, the stakeholders involved and the concern priorities defined by each stakeholder. Other metadata describes the context in which the system is applicable, the consequences resulting from the system usage and presents a set of examples whether the system had been previously applied with success. Furthermore, the System concept holds information describing how the system will be tested and validated using the Test Case concept.

A *Test Case* defines the tests to be performed to guarantee that the System Requirements and Stakeholder Requirements have been correctly

implemented. It specifies the profile for the tester as well as the requirements that shall be validated by the test execution. In spite of being successfully used in the project, this concept is out of the scope of early aspects, and therefore it is no further discussed in this paper.

The main relations for the proposed concepts are depicted in Figure 1, where each arrow represents a reference relation. The System concept, representing a collection of interacting and interrelated elements, links together Concern, Stakeholder, Stakeholder Requirement and System Requirement, making all system dependent information stored in the main System concept. All references linking the System concept to the independent concepts are intended to promote reusability.

3 PROPOSED APPROACH

The proposed ASSD approach model is composed by the following tasks. The development approach is iterative. The main tasks and respective subtasks are explained in the next sub-sections. This model is an evolution of AORA, with some differences and extensions that are explained at Section 5. The resulting specification from the proposed approach is stored in an XML-based Metadata Repository (Ferreira & Moura-Pires, 2007). XML was chosen as the document format since it provides flexibility to represent any kind of information.

3.1 Identify Stakeholders and Requirements

The analyst obtains stakeholders and their requirements from the provided documentation and interview transcripts with the involved entities in the project.

Identify Stakeholders. A stakeholder may be any person, organization and application with an interest in the system. Stakeholders may have different roles; those with a direct interaction with the system correspond to UML Use Case actors.

Identify Stakeholder Requirements. Stakeholders define their requirements for the system, which should be addressed and implemented by the developers. Requirements can be classified according to Sommerville (2006) and have assigned priorities, using the MoSCoW rules (Stapleton, 1997). These artefacts are usually referred in the literature as “user requirements”.

3.2 Elicit System Requirements

System requirements detail the services provided by the system and also the constraints that the system must satisfy. They are classified according to their visibility (*internal* or *public* to the analysis and development teams), and have a priority. A system requirement may comply with one or more stakeholder requirements and a stakeholder requirement may comply with one or more system requirements. These requirements are summarized in a *System Requirements Document* (Sommerville, 2006) that serves as a contract between developer and client. It is a good practice to fulfil all Stakeholder Requirements with at least one System Requirement, unless properly justified. This mapping is sometimes expressed in a requirements traceability matrix.

3.3 Identify Concerns

This task is divided in two parallel steps: *Elicit concerns*, and *Reuse catalogues*. Concerns are elicited based on the understanding of the system domain, interview transcripts and existing documentation. Concerns can be specialized into sub-concerns, if necessary, and classified as functional or non-functional (Sommerville, 2006). To promote reusability, the use of concerns catalogues (Chung et al., 2000), is proposed.

3.4 Specify Concerns

This task is divided into six sub-tasks. Most of the analysis results will be based on the outcome of this task.

Identify Responsibilities. Responsibilities are knowledge or proprieties the concern must maintain and offer. A concern without responsibilities indicates that either the system requirements are missing or that the concern is not needed in the analysis.

Identify Contributions. A concern may contribute positively (‘+’) or negatively (‘-’) to another concern, depending if it helps or damages (Wiegers, 2003). Contributions may be unidirectional, meaning that if a concern has a negative contribution to another concern, the inverse is not mandatory. If the contribution from one concern to another one is ambiguous, the concern shall be specialized or decomposed in two or more sub-concerns to describe each of the situations, and contributions shall be set from these concerns to the target concern.

Identify Required Concerns. This step identifies concern dependencies for functional concerns. For example, if the "Update" concern requires the "Persistence" concern, it is not possible to achieve "Update" without "Persistence". Two concerns requiring each other indicate an analysis error, or that perhaps they should be merged together.

Identify Stakeholder Priorities. Different stakeholders may allocate different priorities to the same concerns: *Very Important*, *Important*, *Medium*, *Low*, *Very Low*, or *Don't Care* – the default value.

Decompose Concerns. Functional concerns can be decomposed, as in object-oriented or component-based software development. Non-functional concerns, on the other hand, can be decomposed using a *softgoal* dependency graph (Chung et al., 2000) with a set of softgoals or operationalizations. When performing the system analysis, the choice of which softgoals and operationalizations are to be implemented is based on the graph analysis.

Build Concern Models. Concern models can be built using UML 2.0. A simple set of rules help to generate a use case diagram: (i) for each stakeholder, map to an actor; (ii) for each functional concern, map to an use case and link to the actors (stakeholders) that have an interest on it; (iii) for each "required" relationship, create an <<include>>, <<extend>>, or <<constrain>> (for non-functional concerns) relationship in the diagram; (iv) for each concern decomposition, create a <<part of>> relationship between the concern and its decomposed concerns.

3.5 Analyse Match Points

The purpose of this task is to compose the concerns to allow the identification and resolution of conflicting situations. This task is supported by automatically generated documentation from the analysis specification.

Review Stakeholders. This review is made through a table composed of three columns: *Name*, *Description*, and *Role*, where the stakeholder related information is presented.

Review Concerns. This review is accomplished through the use of two tables, showing functional and non-functional concerns. Both tables share three columns: *Name*, *Description*, and *Stakeholders*. The non-functional concerns table has an extra column, Classification according to Stapleton (1997). Concerns are grouped according to their hierarchy. For concerns with a parent concern, the parent(s) path of specialization is depicted, where "Security > Performance", for example, means that the concern "Security" is specialized by the concern "Performance", following the same convention as OCL.

Review Concern Contributions. A table "concerns vs. concerns" is built to provide a global view of the concerns contributions. This table can be simplified by pruning the unused rows and columns, but even doing that, in some cases it may not be a scalable visualization. For these cases, it is preferable to view this information using lists. An example is shown in Figure 2 for a test case.

Review Required Concerns. To achieve this, a table "concerns vs. required concerns" is proposed. Again, a scalability problem may occur, so it is also proposed to use lists as an auxiliary visualization of this content. A cell with a checked mark ("√") means that the concern in the row requires the concern in the column.

Identify Crosscutting Concerns. A concern is crosscutting if it is required by more than one concern. Crosscutting concerns, or candidate aspects, represent functionalities and constraints that are scattered among other concerns. These can be mapped into architectural design choices, functions or implementation aspects (Rashid, Moreira & Araújo, 2003).

Identify Match Points. A *match point* is a set of concerns that need to be composed together to accomplish certain functionalities. In a match point, one of the concerns plays the role of base concern to which the behaviour of the remaining concerns needs to be weaved. Match points are specified based on the required relationship between concerns in the Specify Concerns step. This task is performed by (i) produce a list of match points and their respective concerns; (ii) analyse match points table with concerns and stakeholder priorities.

Concerns vs. Concerns	Access Security	Accessibility	Availability	Client Portability	Computation in Parallel	Concept Integrity	Concept Removal	Concept Retrieval	Concept Update	Configurability	Cost	Data Format Interoperability	Data Security	Documentation	Error Notification	Execution	Extensibility	Fault Tolerance
Access Security		–									–							
Accessibility	–												–					
Client Portability		+										+						
Computation in Parallel						–	–	+	–							+		–
Concept Integrity																		
Concept Navigation																	–	
Concept Persistency														+				
Concept Validity			+											+				+
Configurability	+	+		+	+			+					+		+		+	
Cost	–		–										–					–
Data Format Interoperability		+		+														
Data Security		–									–							
Documentation				+						+							+	
Error Notification					+													
Fault Tolerance					+													

Figure 2: Concern contributions for a test case.

Match Points	Match Point Concerns	Administrator	Client Application	Developer
MPClient Portability	Client Portability	Medium	Important	Very Important
	Data Format Interoperability	Important	Very Important	Very Important
MPConcept Integrity	Concept Integrity	Very Important	Important	Important
	Concept Persistency	Important	Important	Important
	Concept Validity	Very Important	Important	Important
MPConcept Navigation	Access Security	Important	Important	Important
	Concept Integrity	Very Important	Important	Important
	Concept Navigation	Low	Medium	Important
	Concept Persistency	Important	Important	Important
	Concept Visualization	Medium	Important	Very Important
	Instance Persistency	Important	Important	Important
	Navigation Response Time	Low	Medium	Low

Figure 3: Match points identified for a test case.

Identify Conflicts. A conflicting situation is detected whenever two or more concerns that contribute negatively to each other need to be composed into the same match point. Conflicts can be classified according to their severity: simple resolution conflicts (if their stakeholder priorities are different) and requiring negotiation conflicts (if the stakeholder priorities for the concern are the same). For conflicts requiring negotiation, the stakeholders must agree on the dominant concern. This is analysed at two levels: concern and stakeholder. As such, a list of match points and respective conflicts is produced, at both levels of analysis. Additionally, the match point identification table is used to identify the conflicts, by highlighting the match points and concerns that are involved in conflicts (yellow for conflicts of simple resolution and red for conflicts requiring negotiation). An example is shown in Figure 3.

List unused Concerns. This is an analysis validation step to identify concerns that do not participate in any relation.

3.6 Analyse Requirements Traceability

With this document an analyst can perform the following.

Review Stakeholders. This is analogous to the Analyse Match point step.

Review Stakeholder Requirements. This step produces a table showing the stakeholder requirements in the system.

Review system requirements. Build a table that shows the system requirements in the system.

Map Concerns. Produce a list of concerns to system requirements mappings. Concerns that are not mapped into any system requirement are highlighted in yellow. The analyst should take those into account, as they may be unnecessary for the system.

Map Stakeholder Requirements. Build a list with the mappings from stakeholder requirement to system requirements. Stakeholder requirements that are not mapped into system requirements are highlighted. The analyst should take this into account, as those are an indication that the stakeholders may not be fully satisfied with the resulting system.

4 PROPOSED ARCHITECTURE

The Metadata Repository provides a framework where the knowledge of the analysis can be stored, queried and validated (Ferreira et al, 2005; Ferreira & Moura-Pires, 2007). This framework facilitates the construction of tools to support the approach described in the previous section, providing a user interface for structuring knowledge, detecting inconsistencies, validating the user input, and generating documentation. By using XML internally, the proposed tools can store all the analysis specification in the Metadata Repository, guaranteeing its validation, and taking advantage of the automatic document generation capabilities of the repository (as exemplified in sections 3.5 and 3.6).

A high-level architecture of the system consists into three main components: the Metadata Repository, the Client tools, and a Concern Catalogue (Chung et al., 2000). Future tool or services' extensions can be developed through the use of External Applications.

4.1 Metadata Repository

Information Model and Technologies. The Metadata Repository stores information of various types, each one classified according to a certain terminology. Each type of information is considered as a different layer, as shown in
Figure 4.

The lowest abstraction level (zero level) refers to the source objects in a certain reality, being either physical or non-physical (e.g., persons, or information stored in a database). These objects are considered external, therefore not being represented in the Metadata Repository. In this work, the objects are the systems to be analysed according to the proposed approach.

The first level defines models for the previous level. A model is a finite description of a source object for a specific purpose. In the Metadata Repository context, models are called instances and are represented by XML documents, where relations can be established to other instances by using a specific syntax. Examples can be Stakeholders (e.g.

"Owner", "Developer"), concerns (e.g. "Performance", "Security") or systems.

The second level defines metamodels for the various types of models in the first level. In the Metadata Repository, metamodels are called concepts and are defined using XML Schema and SchemaTron. Therefore, a concept is the definition of a structured language that describes a certain type of instances, as presented in Section 2.

The third level defines meta-metamodels that describe style rules and common structure definitions for all metamodels in the second level. In the Metadata Repository these are called rules and include styling and predefined structures to be used in every concept, for concept structure consistency and standardization.

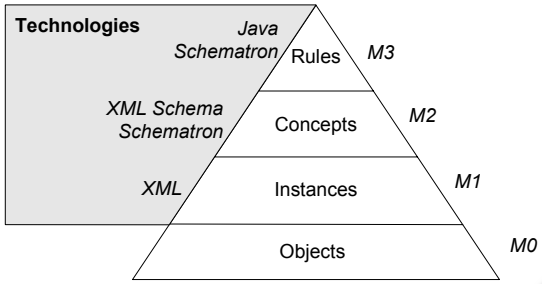

Figure 4: Metadata Repository technologies and abstraction levels.

The Metadata Repository supports a set of basic functionalities for storing and querying metadata information, and a simple interface for easy integration among external systems. Additional features, like instance relations and versioning, provide added functionality and capabilities to the repository.

Storage. The Metadata Repository stores both instances and concepts. As these are represented as XML and XML Schema documents, they are stored in eXist, a native XML database (http://exist.sourceforge.net).

Validation. Validation is an important task for maintaining coherence in the repository. Concepts and instances are validated according to their syntax and structure, and reference integrity is checked.

Instance Relations. The Metadata Repository supports relations among instance versions, keeping and ensuring integral referencing, i.e. the target instance version of a relation always exists.

Querying. Once metadata is stored in the repository, querying mechanisms are provided to retrieve metadata in various formats. By using XQuery, it is possible to write queries to be performed over the stored instances and/or concepts by the definition of an output document, being possible to return XML or non-XML results, as shown in Figure 5. If the query results are XML, it is possible to transform them into other formats by using either a single XSLT (Transform), or a sequence of XSLT (Transform Pipeline). These are commonly used to generate HTML documentation from the query results.

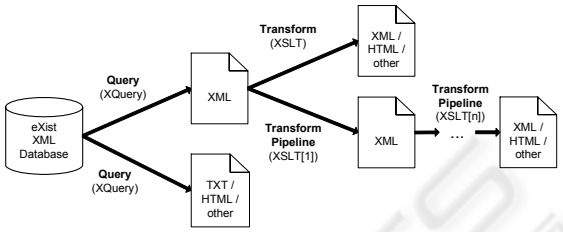

Figure 5: Metadata Repository querying capabilities.

Web Service. The Metadata Repository interface is a web service, providing a set of management methods, invoked by client applications.

4.2 Client Tools

For user interface, a set of client tools is developed, the Aspect Development Assistant Tools (ADAT). The client tools architecture is based on previous work (Ferreira et al. 2005) and include: (i) Stakeholder Library, for maintaining and managing the stakeholders to be used by the systems. (ii) Concern Library, for maintaining and managing the concerns to be supported by the systems. With this application, concerns are defined system independently so that they can be reused by several systems (Chung et al., 2000). (iii) Decomposition Library, for maintaining and managing the concern decompositions, by creating reusable decomposition graphs. (iv) System Editor, the main ADAT application (screenshot in Figure 7). It maintains and manages systems to be analysed by the proposed approach, allowing the specification of stakeholders and concerns from the Stakeholder Library and Concern Library applications and defining Stakeholder Requirements, and System Requirements. This application also outputs analysis documentation, as previously described.

The client tools share a similar user interface, as shown in Figure 6. Two panes compose it: the navigation pane and the content pane, marked as 1 and 2 respectively. The navigation pane (1) contains two tabs: the tree tab and the search tab. The tree tab allows navigating through the grouper and instance nodes, allowing node expansion and access to the available features for the corresponding instance type. The search tab allows finding instances by their descriptive fields. The selected item in the

navigation pane is visualized in the content pane (2). This pane allows several tabs, depending on the current application and node type. Although usually the tabs correspond to edition forms, they can also have source XML and analysis output.

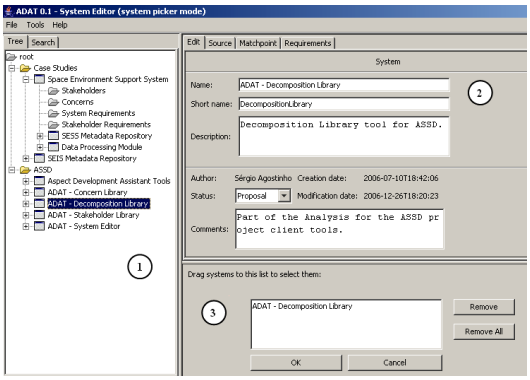

Figure 6: ADAT layout.

5 RELATED WORK

This work presents an extension to the AORA approach (Brito & Moreira, 2003a, 2003b) and a software engineering tool to support it. Similar work has been done before with the creation of the AORA tool, for supporting an early version of the AORA methodology (Brito, Moreira & Araújo, 2006). The work described in this paper can be considered as an extension to it, since it presents a more complete approach, covering stakeholder and system requirements, and an enhanced set of tools to support it. The ASSD model differs on the AORA model regarding some extensions and changes: (i) concepts were added for requirements and tests; (ii) parent relations were added to some concepts, allowing the formation of hierarchies; (iii) the Review requirements step was added, allowing requirements traceability analysis; (iv) concern contributions are unidirectional, instead of bidirectional.

The approach proposed by Rashid et al. (2003) uses templates to represent candidate aspects and to show the impact of concerns over others and 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. The approach is supported by a tool called ARCaDe.

The Theme approach provides support for aspect-oriented development at analysis and design levels (Baniassad & Clarke, 2004). 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 the above approaches by offering a more complete concern template and a tool that help tracing the concerns from requirements to the specification, composition of concerns, management of changes of concern specifications and compositions, and a concern repository within a project. Regarding Theme, it 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. Moreover, Theme does not offer a concern repository.

6 CONCLUSIONS AND FUTURE WORK

The ASSD project presented an approach that has successfully applied to real-world case studies that were validated by DEIMOS Space (http://www.deimos-space.com/). The software architecture and client tools were validated by EADS Test & Services (http://www.ts.eads.net/). The analysis on the case studies was performed iteratively with the design of the approach, allowing its improvement and consequently obtaining interesting results in the case studies, such as the detection of some trade-offs that had to be performed. However, since the case studies were nearly complete projects, it was not possible to integrate the approach in their full development cycle; a task that would allow further validation of the approach. The use of a Metadata Repository solution for managing knowledge, ensuring its validation, consistency and providing querying mechanisms is a major advantage comparing with traditional documentation supports.

REFERENCES

Baniassad E., & Clarke S. (2004). *Theme an Approach for Aspect-Oriented Analysis and Design*. International Conference on Software Engineering, Edinburgh, Scotland.

Brito, I., & Moreira, A. (2003a). Advanced Separation of Concerns for Requirements Engineering. *VIII Jornadas de Ingeniería de Software y Bases de Datos* (JISBD), Alicante, Spain.

Brito, I., & Moreira, A. (2003b). Towards a Composition Process for Aspect-Oriented Requirements. *Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design*, AOSD 2003, Boston, USA.

Chung, L., Nixon, B., Yu, E., & Mylopoulos, J. (2000). *Non-Functional Requirements In Software Engineering*. Kluwer Academic Publishers.

Ferreira, R., Moura-Pires, J., Martins R., & Pantoquilho, M. (2005). XML based Metadata Repository for Information Systems, *12th Portuguese Conference on Artificial Intelligence (EPIA 05)*, Portugal.

Ferreira, R., Raminhos, R., & Moreira, A. (2005). Metadata Driven Aspect Specification. *ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems*, Montego Bay, Jamaica.

Ferreira, R., & Moura-Pires, J. (2007). Extensible Metadata Repository for Information Systems and Enterprise Applications. *9th International Conference on Enterprise Information Systems (ICEIS)*, Funchal, Portugal.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. Videira, Loingtier, J.-M., & Irwin, J. (1997). *Aspect-Oriented Programming. European Conference on Object-Oriented Programming (ECOOP)*, Finland.

Marques, A., Raminhos R., Ferreira, R., Ribeiro R., Agostinho S., Moreira, A., & Araújo J. (2007). Aspect-Oriented Analysis Applied To The Space Domain. *9th International Conference on Enterprise Information Systems (ICEIS)*, Madeira, Portugal.

Rashid, A., Moreira, A., & Araújo, J. (2003). Modularisation and Composition of Aspectual Requirements, *AOSD 2003*, Boston, USA.

Sommerville, I. (2006). Software Engineering (8th edition). Addison-Wesley.

Stapleton, J. (1997). *Dynamic Systems Development Method*. Addison-Wesley.

UNINOVA (2007). *ASSD – Aspects Specification for the Space Domain*. Retrieved March 8, 2008 from http://www2.uninova.pt/ca3/en/project_ASSD.htm.

Wiegers, K. (2003). *Software Requirements* (2nd edition). Microsoft Press