A study on different approaches towards Aspect Oriented Requirements Engineering

Suchetha Vijay

Asst. Professor
PG Dept of Information Technology
AIMIT, St. Aloysius College
Mangalore

Annapoorna Shetty

Asst. Professor
PG Dept of Information Technology
AIMIT, St. Aloysius College
Mangalore

Abstract

Requirements and their quality are very important for the success and on time delivery of a project. However, Requirement Elicitation is not an easy and small task to begin with. Conflicts arise in various ways such as way of looking at things and analyzing them, differences in expectation, mental model and many more. In many cases, clients and users are not completely sure about their actual needs in the beginning. Hence the actual solution for all such problems and conflicts is a step by step process called Requirements Engineering (RE). RE aids us in identifying, analyzing and documenting system requirements. This is an important task because it helps to eradicate mistakes and conflicts, if any, in the very early stage of development. In any approach of Software Development, RE is a very important step. Aspect Oriented Software Development is not an exception to this. Aspect Oriented Requirements Engineering (AORE) helps in providing better solution by offering new methods and approaches for separation of requirements. Several AORE approaches have emerged over the years. In this paper, we try to summarise the various approaches of AORE with a brief explanation to each of these approaches.

Keywords: Aspects, Requirements Engineering, Concerns

1. Introduction

Requirement Elicitation (RE) is one of the important and primary activity in Software Development Process as it helps in identifying, analyzing, specifying, verifying and managing system requirements. The main aim of RE is to very clearly say about the stakeholders' expectations thus enabling the software engineers to get an indepth understanding about the functionalities, restrictions, constraints and properties of the system to be developed, environment in which the system will operate, people who are going to use the system. Many AORE approaches have been stated over the years whose main aim is to separate out concerns at this level. These separated out concerns are called as early aspects. If not separated as early as this, the aspects get scattered in various forms during the remaining stages of development, in which case it becomes difficult to meet user expectations. In this paper we make an attempt to define an aspect and study various approaches to AORE. In Section 2 we try to define the term aspect and also say what is the lifetime of aspects that are identified. Section 3 explains the asymmetric AORE approaches using asymmetric languages and tools. In Section 4 we explain symmetric AORE approaches along with the tools available for it. Section 5 lists couple of other tools available to mine aspects. We conclude in Section 6 and also propose future work in Section 7.

2. Aspect And Its Lifetime

Conceptually an aspect is defined as an element of functionality which is woven and entangled throughout other system behaviours. They are present in two different forms in two different levels of Software development. At the source level, aspects are entangled and scattered. At the requirements level, aspects are still entangled but appear as scattered descriptions of functionalities.(Fig 1) Hence delaying aspect requirements to a later level without separating them at the requirements level may result in potential conflicts later with the original set of

requirements. It might also affect the operational elements of the design. The overall result of this would be an unhappy client. In addition, this might sound like contracting one application and developing a different one. Function requirements show what the system should perform and the quality aspects show how it performs. For example, in a lift system, safety aspects can restrict the movement of the cabinet if the door is not closed. Thus, the specification of the functions and the restrictions enforced by qualitative aspects should be better specified at the same level. So, not only should the aspects not die, as in at the implementation stages, destroying traceability, but aspects should also be alive from the beginning, at the user needs and system requirements stage. Aspects can also be traced down from a high-level goal to the terminal goal level, where they can be retranslated into early design form.

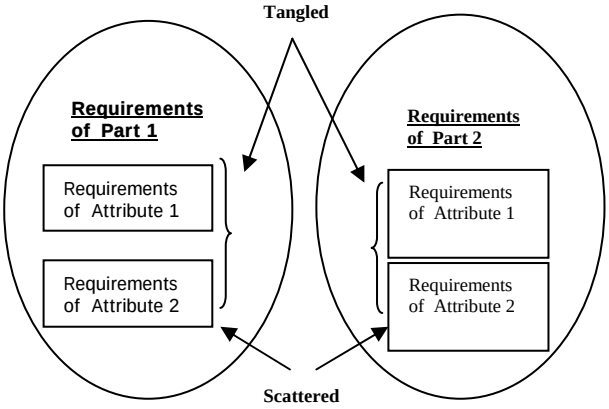

Fig 1: Requirements entangled and scattered in the early stage of development

3. Asymmetric AORE Approaches Using Asymmetric Languages And Notations

AORE approaches provide a clear separation of what are the base and cross cutting concerns. Here aspects crosscut the base and interactions with the base or amongst aspects are studied from the perspective of aspects.

Arcade is a tool which supports Asymmetric AORE approach. In Arcade, the main requirement decomposition units are viewpoints and aspects. Here aspect encapsulates the requirements that crosscut the viewpoint decomposition. Each of the viewpoint and aspect have a unique name while every requirement has an unique identification number.

Probe Framework provides ways to link Arcade specification of viewpoints and aspects into UML design models. The requirement specification in the form of XML and UML designs are the initial inputs to Probe framework. At a later stage user created mappings mad code are provided. The various modules of Probe are:

- ***Temporal Logic Generator***, which parses the XML representation of the aspectual requirements, viewpoint requirements and composition rules in the ARCADE specification, and uses it to generate individual temporal formulas which, in turn, are grouped into aspect proof obligations.
- ***UML Analyzer***, which determines separate proof obligations by analyzing the aspect-oriented UML design of the system
- ***Conflict Analysis*** module, which generates proof obligations from the aspect trade-offs and conflicts from the ARCADE specification
- ***Ontology***, which provides predefined generic temporal formulas or state functions, and records predicate names used to connect to the implementation
- ***Integration*** module, which integrates the outputs of the previous modules to generate groupings of temporal logic assertions, in terms of predicate names from the ontology
- ***Instantiation*** module, which generates the final concrete proof obligations for the implementation.

Theme/Doc is a tool which provides a graphical notation to model the requirements which are later mapped to theme/UML designs. This tool is based on the notion of a theme, which is nothing but a feature of the system. Many such themes can be combined to form one functional unit. There exist two kinds of themes – base themes and crosscutting themes. The tool operates on an assumption that if two behaviors are found in the same

requirement, then they are related to one another. Behaviors can be related in any of the three ways – they can be erroneously or coincidentally related or they can be hierarchically related or can be related by crosscutting. Be it any type of relationship, Theme/Doc tool exposes the behaviors that are found in the same requirement. The advantage of this is that the developer can decide what kind of relationships exists between behaviors.

Scenario modeling with aspects uses interaction pattern specifications to model crosscutting scenarios which are subsequently translated to executable state machine specifications. Scenarios will be modeled, in this paper, as UML interactions (in particular, sequence diagrams) UML interactions, when used to model requirements, show the required behavior of several system components communicating towards a common goal. Interactions are a good way of modeling early requirements because they show global exchanges between system components and are very natural and intuitive to write down. Interactions give a global view of the requirements, but to simulate the requirements, a local view of each system component is necessary. Finite state machines (FSMs) can be used to model the local, internal behavior of system components

AO use cases extend the classical use case model with AspectJ-style intertype declarations and advice to support use case driven development across the lifecycle. It provides clean modularization of crosscutting concerns, such as error checking and handling, synchronization, context-sensitive behavior, performance optimizations, monitoring and logging, debugging support, and multi-object protocols.

4. Symmetric AORE Approaches Using Symmetric Languages And Notations

In contrast, symmetric AORE approaches treat all types of concerns uniformly. This provides a more fluid model whereby any set of concerns can be selected as a base to study the dependencies and interactions with any other set. In other words, the crosscutting effect can be studied from multiple perspectives providing a more powerful reasoning mechanism.

The Requirements Description Language (RDL) utilises the semantics of the natural language as a basis for concern composition in order to alleviate composition fragility. Its composition semantics are described in and a mapping from the natural language composition patterns to architectural descriptions. We use the richness of the natural language to support expressive semantics-based pointcuts within a requirements description language (RDL). We utilize the fact that the natural language itself has a clearly defined set of syntactic rules and semantic elements, precise enough to support definition of a flexible composition mechanism for requirements analysis. Though a more formal semantics would undoubtedly be more precise, the natural language semantics capture the stakeholders' needs as expressed in the elicited requirements and hence are more suited to aspect-oriented requirements analysis. Our RDL annotates the syntactic elements of the natural language and exploits the fact that each syntactic element has a designated semantic role – these semantic roles form the basis of expressive pointcut expressions in our RDL. The RDL is an XML-based language – the XML annotations help automate the analysis.

Cosmos is a concern modeling schema to represent relationships between various concerns. Cosmos divides concerns into two categories, logical and physical. Logical concerns represent concerns viewed conceptually: topics, issues, properties, problems, in general "matters for consideration". Physical concerns represent the real things of which our systems are comprised, work products, hardware, services, resources, etc. Logical concerns are further typed as kinds, instances, properties, and topics. Kinds are groups of concerns of a particular type (such as functions, behaviors, states). Instances are particular concerns of some type (e.g., particular functions, behaviors, states). Properties are characteristics of kinds and instances of concerns. Topics are (typically theme-related) groups of concerns of generally different types or kinds. Physical concerns are typed as instances or collections. Instances represent particular system elements (such as source files, design documents, workstations); collections represent groups of these.

A multi-dimensional requirement modelling extends the Arcade model to a symmetric composition approach and provides a means to compose and analyse concerns based on compositional intersections. Concerns in our multi-dimensional approach imply any coherent collection of requirements. We treat all concerns in a uniform fashion and hence, do not classify concerns into viewpoints, use cases or aspects though our concerns still encapsulate coherent sets of functional and nonfunctional requirements. We perceive the concern space at the requirements level as a hypercube. Each face of the hypercube represents a particular concern of interest. By treating all concerns as equal we can choose any set of concerns as a base to project the influence of another concern or set of concerns onto this base. The block arrows represent projections. This flexible, multidimensional view makes it possible to handle both crosscutting functional and non-functional requirements in an effective fashion.

5. Other Tools Used To Mine Aspects

EA-Miner uses natural language processing and latent semantic analysis to mine for aspects in natural language requirements and structure them into specific AORE models.

MRAT supports analysis of multi-dimensional concern models and provides various visualisation modes for reasoning about the compositions.

6. Conclusion

In this paper we have summarized different approaches towards Aspect Oriented Requirements Engineering. Detailed explanation regarding eliciting requirements is beyond the scope of this paper. Collected requirements may be converted to aspects using any of the approaches mentioned.

7. Future Work

We have summarized almost all the approaches available in AORE. A case study based comparative study on the efficiency of the approaches mentioned in this paper can be done to evaluate the efficiency of the approaches and to decide which is the best approach.

8. References

[1] Chethana Kuloor and Armin Eberlein [2003] , "Aspect-Oriented Requirements Engineering for Software Product Lines" in IEEE

[2] Américo Sampaio, Phil Greenwood, Alessandro F. Garcia and Awais Rashid, [2007] A comparative study of Aspect-Oriented Requirements Engineering Approaches in IEEE

[3] Islam A. M. El-Maddah and Tom S. E. Maibaum [2004] Tracing Aspects in Goal driven Requirements of Process Control Systems, 2004

[4] Awais Rashid Aspect-Oriented Requirements Engineering:[2008] An Introduction

[5] Shmuel Katz, Awais Rashid From Aspectual Requirements to Proof Obligations for Aspect-Oriented Systems

[6] Sampaio, R. Chitchyan, A. Rashid, P. Rayson, [2005] "EAMiner: a Tool for Automating Aspect-Oriented Requirements Identification", International Conference on Automated Software Engineering (ASE), ACM, pp. 353-355

[7] E. Baniassad, S. Clarke, [2004] "Theme: An Approach for Aspect-Oriented Analysis and Design", International Conference on Software Engineering (ICSE), IEEE CS, pp. 158-167.

[8] Rashid, A. Moreira, J. Araujo, [2003] "Modularisation and Composition of Aspectual Requirements", International Conference on Aspect-Oriented Software Development (AOSD), ACM, pp. 11-20.

[9] R. Chitchyan, A. Rashid, P. Rayson, R. W. Waters,[2007] "Semantics-based Composition for Aspect-Oriented Requirements Engineering", International Conference on Aspect-Oriented Software Development (AOSD), ACM, pp. 36-48.

[10] Elisa Baniassad and Siobh´an Clarke, Finding Aspects In Requirements with Theme/Doc

[11] João Araújo, Jon Whittle, Dae-Kyoo Kim Modeling and Composing Scenario-Based Requirements with Aspects

[12] http://www.eclipse.org/aspectj/

[13] Stanley M. Sutton Jr. and Isabelle Rouvellou Concern Space Modeling in Cosmos

[14] Ana Moreira, Awais Rashid, João Araújo Multi-Dimensional Separation of Concerns in Requirements Engineering