Aspect Oriented Requirements Engineering

Ahmed Yakout A. Mohamed
Arab Academy for Science, Technology & Maritime Transport
College of Computing and Information Technology, Egypt
Tel: 002-011-8606-888 E-mail: systems.analyst@hotmail.com

Prof Dr. Abd El Fatah .A. Hegazy
Arab Academy for Science, Technology & Maritime Transport
College of Computing and Information Technology, Egypt
E-mail: ahegazy@aast.edu

Dr. Ahmed R.Dawood
Chairman, MIS Department
Arab Academy for Science, Technology & Maritime Transport, Egypt
E-mail: ahmedredadawoad@yahoo.com

Abstract

Requirements engineering techniques that explicitly recognize the importance of clearly identifying and treating crosscutting concerns are called Aspect-oriented Requirements Engineering Approaches (AORE approaches). The emergence of aspect-oriented programming languages has raised the explicit need to identify crosscutting concerns already during the analysis phase. Besides this observation, the modular representation of crosscutting requirements is a first step to ensure traceability of crosscutting concerns through all other artifacts of the software lifecycle (architecture, design and implementation).

Aspect-oriented requirements engineering approaches improve existing requirements engineering approaches through an explicit representation (and modularization) of concerns that were otherwise spread throughout other requirements artifacts (such as use cases, goal models, viewpoints, etc.).

AORE approaches adopt the principle of separation of concerns at the analysis phase (the early separation of concerns). In other words, AORE approaches provide a representation of crosscutting concerns in requirements artifacts.

Keywords: Requirements engineering, Aspect-oriented requirements engineering

1. Introduction

Requirements are always derived from some business problem whether it is, for example, processing passport applications, improving automotive safety systems or adding features to cell phones (P. Sawyer). Projects may develop new products or they may be concerned with evolving existing or legacy systems. In all cases, the software system will be embedded in an operational context so it will have interfaces to human users, business process elements or other software or hardware systems (Figure 1).

A requirement defines a property or capability that must be exhibited by a system in order for it to solve the business problem for which it was conceived.

The classic way to categorize requirements is according to whether they are functional or non-functional. Functional requirements describe features that the software must provide (for example, "activate the siren when a sensor is tripped"). Non-functional requirements, on the other hand, describe qualities of a system. The most important class of non-functional requirements addresses how well the system operates within its

environment. Essentially, they specify the quality of delivery of the functional requirements.

Requirements are usually specified at several points on a spectrum that ranges from those with a business focus to those with a technical focus. The technically-focused requirements exist only to make it possible to satisfy the business-focused requirements.

For convenience, we will treat these as levels where the highest level requirements are those with a business focus. Such high-level requirements are the goals of the system that set out in very broad strategic terms what is needed to solve some business problem.

The next level of requirements defines the properties required by the people who will use the system or imposed by other people, organizations or systems within the environment. These are often called the user requirements.

One of the main tasks of requirements analysis is to elaborate the user requirements to discover more about the implications of satisfying them. This involves deriving new, lower-level, requirements (derived requirements) that focus more on detailed technical issues.

Transforming a requirement into software is a complex process. The deeper into the process, the more design and implementation strategies become committed to satisfying the requirement. Consequently, the costs of rectifying errors in the requirements increase dramatically as development proceeds. An effective RE process which minimizes the occurrence of requirements errors and mitigates the impact of requirements change is therefore critical to the success of any development project.

When such allocation is possible, the resulting software is well modularized, the modules have good communication with each other and all requirements are clearly separated. However, there are many classes of requirements for which this property of allocation into modules is not possible using ‘traditional’ software paradigms. Many non functional requirements fall into this category (A. Rashid, P. Sawyer, A. Moreira, & J. Araujo 2002).

If some requirements are not effectively modularized, it is not possible to reason about their effect on the system or on each other. Furthermore, the lack of modularization of such properties can result in a large effect on other requirements and their corresponding components upon evolution. These requirements are called crosscutting (aspectual) requirements. Examples of such properties include security, mobility, availability and real-time constraints. From all information gathered from previous researches in RE, Initiatives began to work on AORE approaches which complements the other requirements engineering models e.g. viewpoints, use cases and goals by providing systematic means for handling the cross cutting concerns.

Therefore, AORE approaches adopt the principle of separation of concerns at the analysis phase (the early separation of concerns). In other words, AORE approaches provide a representation of crosscutting concerns in requirements artifacts. For example, AORE Arcade provides the notion of crosscutting concern next to viewpoints in a requirements artifact. Besides this fact, these approaches also focus on the composition principle: it should be possible to compose each concern/requirement with the rest of the concerns/requirements of the system under construction to understand interactions and trade-offs among concerns. This composability of requirements is a central notion of AORE. It means that AORE techniques should have well-defined join point models and composition semantics. Join point models of AORE approaches identify points through which requirements can be composed. For example, join points can be individual requirements expressed in a viewpoint.

To draw an analogy with the aspect-oriented concepts introduced before: In the above composition specification, the green parts correspond to pointcuts, the underlined elements are the kind of advice and the red elements are the advice behavior. In this example, the requirements aspect itself is separated from its composition specification (which is provided above).

The composability of requirements allows not only reviewing the requirements in their entirety, but also detection of potential conflicts very early on in order to either take corrective measures or appropriate decisions for the next development step. Last but not least, the mapping of the concerns at the requirement level to concerns in later lifecycle stages will reveal wether the concern maps to a crosscutting artifact or wether it becomes absorbed into other artifacts.

Summarizing, AORE approaches focus on systematically and modularly treating, reasoning about, composing and subsequently tracing crosscutting concerns via suitable abstraction, representation and composition mechanisms tailored to the requirements engineering domain. Arcade, ARGM, Aspectual Use Cases, Cosmos, AOREC, etc. are such aspect-oriented requirements engineering approaches that are extensively described in (Ruzanna Chitchyan, & Awais Rashid 2005).

The rest of this paper is structured as follows. Section 2 defines what is AORE and points out its key objectives and benefits. Section 3 highlights some of the major RE aspect and non aspect RE approaches. In section 4, we list the criteria on which the RE approaches are evaluated, followed by a comparison between both types of approaches. Finally we place the conclusion in section 5.

2. Aspect Oriented Requirements Engineering

2.1 What is AORE?

The approach is called "aspect-oriented requirements engineering" as it is one of the first approaches to introduce aspect-oriented concepts at the requirements level. In fact (J. Araujo, A. Moreira, I. Brito, & A. Rashid 2002) was the first paper to introduce the term "Early Aspects" which is now a defector term for work on aspect-oriented requirements engineering and architecture design. In order to avoid confusion with the general topic of aspect-oriented requirements engineering (AORE) we will refer to this approach as "AORE with Arcade" from this point onwards, Arcade being the tool that supports the AORE approach in question.

AORE is a technique that has been showing encouraging results at handling crosscutting concerns by proposing means to their systematic identification, modularization and composition. Such crosscutting concerns are those whose implementation is scattered among several implementation modules, producing tangled systems that are hard to understand, difficult to maintain and hard to evolve.

In order to maximize its benefits, Aspect Oriented Software Development (AOSD) should be used from the early stages of software development such as domain analysis and requirements engineering. Identifying aspects at an early stage helps to achieve separation of concerns in the initial system analysis, instead of deferring such decisions to later stages of design and code, and thus, having to perform costly solutions.

2.2 AORE Method.

The Aspect-Oriented RE approach initially outlined in (J. Araujo, A. Moreira, I. Brito, & A. Rashid 2002) and developed in (A. Rashid, A. Moreira 2003) proposes a technique for separating aspectual and non-aspectual requirements, as well as their composition rules. This work has developed a general RE process model (demonstrated in Figure 2) which can be instantiated with any requirements engineering techniques.

The approach balances the need to formalize the concerns, viewpoints, requirements, and composition operators and rules to ease representation and proof generation with the need to keep the artifacts simple enough to be understandable for the end users (who will be involved in conflict resolution with these artifacts). Consequently, the approach uses XML for artifact representation, keeping them structured, semi formalized, yet simple and understandable. An additional benefit of using XML-based composition rules and operators is the ease of their change and extension as required for an individual project.

AORE with Arcade also assists in preparing the identified requirements for the next stage of the development lifecycle by providing directions for their mapping. The requirements document, and conflict resolution decisions produced via AORE with Arcade method are then provided as inputs to the PROBE framework (S. Katz and A. Rashid 2004). PROBE was developed to establish links between aspectual requirements and their later lifecycle-stage artifacts. It generates proof obligations for AORE artifacts and supports tracing these from requirements through architecture, design, and implementation stages.

2.3 AORE Process.

The process of AORE with Arcade usage (as described in (A. Rashid, A. Moreira, & J. Araujo 2003)) is depicted in Figure 2. The process starts by identifying stakeholders' requirements. When using the PREviewbased instantiation, this implies identification of viewpoints and collection of their related requirements. The identified requirements then are specified through provided XML templates.

The next step focuses on identification and specification of concerns. Unlike PREview, where concerns are discovered first and viewpoint requirements collected with these concerns in mind, AORE discovers concerns by analyzing the initial requirements and also allows recursive concern and requirement identification. This is followed by identification of coarse-grained relationships between concerns and viewpoints. For this a matrix is built to mark which concerns affect which viewpoints. If a concern affects

several viewpoints in the matrix it is considered a candidate aspect.

After composition the conflict handling is supported at two levels. First, due to the ability to compose aspectual and non-aspectual requirements at fine granularity, the need for analyzing possible conflicts is eliminated when different concerns affect different requirements within the same viewpoint. Secondly, a mechanism for conflict assessment and resolution is provided when concerns do affect the same requirement. In this case a contribution matrix is to be built, with only negative contributions presenting a real problem. Then weight attribution based on fuzzy intervals is used to prioritize the conflicting requirements. Only if equal weights are assigned to conflicting requirements, there is a need for negotiation between stakeholders.

2.4 Identification and Treatment of Crosscutting Concerns with AORE.

This approach has a generic mechanism for concern handling which can easily suit both functional and non-functional concerns. However, it is not clear how the crosscutting functional concerns should be identified. Identification of crosscutting non-functional concerns is also not completely satisfactory, as there is no support (e.g., guidelines, tools, etc.) for the requirements engineer in detecting these from the requirements collected from the stakeholders.

On the other hand, the very recent work (A. Sampaio, N. Loughran, A. Rashid, & P. Rayson 2005) on this approach turns to application of semantic linguistic analysis of natural language and requirements documents for aspect identification. Although this work is at an early stage at the time of preparation of this report, it already has a sound well structured mechanism for identifying both functional and non-functional crosscutting concerns.

2.5 Key Objectives and benefits

AORE aims at:

- Providing support for separation of crosscutting functional and nonfunctional properties during requirements engineering hence offering a better means to identify and manage conflicts arising due to tangled representations (A.Rashid, & R. Chitchyan 2002).
- Identifying the influence of requirements level aspects on artifacts at later development stages hence establishing critical trade-offs before the architecture is derived.

It also offers a number of benefits such as:

- Ensuring consistency of stakeholders' concerns with global requirements and constraints.
- Tackling tangled requirements representations that are difficult to understand and maintain.
- Better support for modularization hence reducing the development, maintenance and evolution costs.
- Promoting traceability of requirements and constraints throughout system development, maintenance and evolution.
- Support for development of systems tailored to unanticipated changes hence meeting the adaptability needs of volatile domains such as banking, telecommunications and e-commerce.

3. Overview of RE approaches.

There are several RE approaches. We will present some of them (R. Chitchyan 2005), (R. Chitchyan, A.Rashid, & P.Sawyer 2008).

3.1 Non AO Approaches

3.1.1 PREview

PREview is a viewpoint-oriented approach which complements the standard notion of viewpoints with that of organizational concerns. A PREview concern is a generalization of the notion of goal; it includes both organizational goals and constraints that restrict the system or process to be analyzed. PREview approach was developed to capture such broad requirements as response time, safety, and security.

PREview is considers the problem related information from different agents (e.g. users of the software system) which can have different perspectives on the problem. The combination of the agent and the view that the agent holds is termed a viewpoint.

It starts with establishing high level concerns gathered from discussions with senior management. These concerns are elaborated and presented as external requirements. After this elaboration, the viewpoints have to be identified from analysis of the applications. This is followed by requirement discovery for each identified viewpoint. At this stage it is essential to apply the questions and requirements derived from the concerns to each viewpoint. After requirement discovery viewpoint interactions have to be analyzed using decision tables. Once identified, viewpoint inconsistencies need to be resolved through requirement negotiation and some trade-off mechanism, however this stage is not directly supported by the PREview process.

The PREview approach is structured around recognition of the importance of the impact of non-functional crosscutting concerns. These concerns are identified using interviews with the stakeholders at the very offset of the development and are modularized in concern templates. The concern impact on other concerns, however, is not modularized: answers to concern-related questions and external requirements are spread across each affected viewpoint.

The PREview concern identification and treatment for non-functional concerns is appropriate only when a small stable set of well defined non-functional requirements are involved. This indeed is the type of development that PREview was intended for. The approach will not be suited to volatile domains.

Also, though PREview suggests that Functionality may be a concern, in the same way as the non-functional concerns, there is no demonstration of how such concerns are identified and treated. Consequently, we conclude that the method does not support identification and treatment of crosscutting characteristics for functional concerns.

Some of the outstanding problems in PREview are:

- Only a small number of concerns (no more then 5) can be effectively addressed in each project. Larger number of concerns makes the amount of generated information unmanageable.
- Similarly only a small number of viewpoints (under 6) should be used.
- Absence of clear guidelines for concern decomposition.
- Absence of a mechanism for inconsistency management, trade-off analysis and decision support.
- Though Functionality may be a concern, no examples of functional concerns are provided, raising the questions: can this approach really identify functional crosscutting concerns?

3.1.2 Non Functional Requirements Framework (NFRF)

The NFRF Framework is intended for representing and analyzing the non functional requirements. It starts with acquiring knowledge about the domain and the system to be constructed and then the Non Functional Requirements (NFRs) of the system can be identified. Once identified, the high level non functional requirements become the top level soft goals which are then decomposed using the NFRF soft goal type and refinement catalogues. Through this process abstract soft goals, such as security, get refined into more specific non functional requirements, such as confidentiality, availability, etc. For each sub soft goal suitable operationalizations are then identified providing alternative solutions for the sub-goals.

The NFR framework consists of 5 major components: softgoals, interdependencies, evaluation procedure, methods, and correlations.

Softgoals are used for representing non-functional requirements. There are 3 types of softgoals: *NFR softgoals*, *operationalising softgoals*, and *claim softgoals.* The *NFR softgoals* act as overall constraints on the system. They are satisfied via *operationalising softgoals* that represent more concrete design or implementation solutions (e.g., operations, processes, data representations, etc.) obtained as a result of decisions made for the NFR softgoal. Operationalizations provide the design alternatives available for the given NFR solution. *Claim softgoals* provide rationale for design and development decisions. Through *claims* domain characteristics can be reflected in the decision making process, particular choices explained, support for prioritizing certain softgoals over others provided, etc.

When softgoals are refined *offspring* softgoals are created which relate to their parents through an *relationship*. The offspring's also *contribute* to their parents either positively, or negatively. In the case of positive contribution, satisfying the offspring leads to satisfaction of the parent too, while in the case of negative contribution it leads to dissatisfying the parent. The framework provides a set of refinement methods for different types of softgoals: *decomposition methods* contain the knowledge of how to break softgoals into more detailed ones; *operationalization methods* assist in operationalising a softgoal, and *augmentation methods* help to represent additional information about a softgoal.

The NFR softgoals, their refinement, and their operationalizations, complemented with claims, together build the Softgoal Interdependency Graph (SIG). The *evaluation procedure* is applied to the SIG to determine the degree to which the initial NFR softgoal is satisfied with the given set of decisions. After NFR requirements have been decomposed, operationalized and evaluated, the NFR SIG is related to the appropriate functional requirement via *design decision* links and the operationalizations are related to the specifications (design decisions) via *operationalization* links. Thus, when turning to the design stage, the functional requirements have clear links to their related nonfunctional ones.

The *refinement methods* component of the NFRF provides a set of generic procedures for refining a softgoal or interdependency into one of more offspring's. These methods are simply patterns (or templates) and guidelines for decomposing softgoals and interdependencies into sub-softgoals and sub-dependencies, based on requirements engineers' past experience and domain knowledge.

The non-functional requirements can conflict (e.g., cost and quality) or support (e.g., availability and dependability) each other. These relationships between non-functional requirements (as well as the requirements and operationalizations, and between two operationalizations) are called *correlations*. *Correlations* are used to examine the cross impact of the softgoals during trade-off analysis

3.1.3 I* Method

I* approach is centered on agent and agent-relationship modeling. Agents are characterized in terms of their relationships with other agents. An agent that depends on another agent for a goal, task, or resource is called depender, that goal/task/resource is called dependum, and the agent dependent upon is the dependee.

The I* process begins by identifying the agents involved in a given process or system. The agents can be identified by considering who or what in the system has the intentionality characteristics. Once agents are identified, the Strategic Dependency diagrams are constructed for them, representing their top level goals and dependencies on other agents. These Strategic Dependencies (SD) diagrams are then elaborated into Strategic Rational diagrams by "looking inside" each agent. Here the structures of goal decomposition developed with NFRF framework are used to analyze how an agent internally evaluates his/her goals and think about the procedures for their achievement.

I* approach is centered on agent and agent-relationship modeling. Agents are characterized in terms of their relationships (dependencies) with other agents. An agent that depends on another agent for a goal, task, or resource is called *depender*, that goal/task/resource is called *dependum*, and the agent dependent upon is the *dependee.* These dependencies allow an agent to achieve goals that he/she could not have achieved alone. On the other hand, these dependencies make the depender vulnerable if the dependee does not deliver the dependum agreed upon by both sides.

I* consists of 2 major parts: SD part and the Strategic Rational (SR) part. The SD part studies actors and their dependencies, thus encouraging a deeper understanding of the business process. It helps to understand what is at stake in case of each particular dependency, who and how will be affected if a dependency fails.

In this framework an agent is *intentional*, i.e. possesses the qualities of (E. Yu 2001):

- Intentionality: agents want to fulfill certain goals or commitments, i.e. they have intentions. The intentionality can also be attributed to an agent (e.g., non-human agents) externally, by the modeler, as this provides a useful way of analyzing an agent. Thus, an agent can be thought of as a locality for intentionality (E. Yu 2001), (E. Yu 2005). Agents can also relate to each other at intentional level.
- Autonomy: agents are independent (i.e. has free will) and can act towards their goals, thus, they are not completely knowable or controllable by the modeler. Yet, the behavior of an agent can be partially characterized using his/her known intentions.
- Sociality: agents exist within the network of relationships with other agents and environment. These relationships are multi-lateral, can be conflicting, and restrict an agent's behavior.
- Identity and Boundaries: define who the agents are and what their responsibilities are. Not only physical entities, but also abstract ones (e.g., teams, roles) can be agents. The boundaries of an agent can change when a responsibility is re-assigned from it to another agent.
- Strategic Reflectivity: agents can reflect upon their actions and operation and reevaluate these.
- Rational Self-interest: agents strive to achieve their goals, yet an agent may choose to constrain its strategies (in its own interest) due to social dependencies. However the rationality is partial and bounded due to the partial knowability of an agent and the limited resources available to the modeler.

The SR part investigates different alternative configurations of actors and their dependencies. This allows to systematically exploring the space of possible new process designs. Each alternative may have different implications for the agents and the system as a whole.

3.1.4 Problem Frames (PF)

The key point of this approach (R. Laney, L. Barroca, M. Jackson, & B. Nuseibeh 2004) is the decomposition of complex problems into familiar sub problems, because if one has solved a similar problem in the past it will be easier for one to deal with the present problem. Thus, the approach sets out to identify the common simple problems which can be used as patterns onto which the complex problems should be decomposed. Development with PF starts with focusing on a problem, locating it in the real world and clarifying its boundaries by establishing what the software problem is and which real world domains it interacts with. This results in creation of the context diagram for the problem. The context diagram is used to identify sub-problems and structure the problem as a collection of interacting sub-problems.

Once sub-problems are identified, the developer should analyze each sub-problem and decide on the problem frame suitable for it. Thus, the problem frame should be adjusted correspondingly and the frame concern will need to be adapted to fit the sub-problem in hand. After each sub-problem has a fitting problem frame, these should be composed to produce the composite frame. Finally, the frame concern of the composite frame should be checked to ensure that the initial problem is satisfactorily resolved.

The broad characteristics of these identified problem classes, along with descriptions of interaction of the real world problem domains with the intended computer system of the respective problem classes, are extracted and catalogued as problem frames. An example of such a problem frame is presented in Figure 3. This frame addresses the issue of building a software system that imposes certain behavior on a part of the

physical world.

Development with PF starts with focusing on a problem, locating it in the real world and clarifying its boundaries by establishing what the software problem is and which real world domains it interacts with. This results in creation of the context diagram for the problem.

The context diagram is used to identify sub-problems and structure the problem as a collection of interacting sub-problems. The intention of this decomposition is to try and map the problem onto familiar simple sub-problems. During this activity, for each sub problem a problem diagram is constructed. The diagram consists of relevant domains (selected from the domains of the context diagram), the projection of the machine (software) from the context diagram, and the sub-problem requirement.

Once sub-problems are identified, the developer should analyze each sub-problem and decide on the problem frame suitable for it. When the problem at hand fits a problem frame, it will satisfy, and so be solvable through, its frame concern. On the other hand, if the problem frame for a given problem is selected inappropriately, it will be reflected by the frame concern: either the frame concern will require some descriptions which will not make sense for the problem, or some necessary descriptions for the problem will be unavailable in the frame concern.

However, in most cases the problem frame selected for a sub-problem will not fit perfectly: the problem is likely to be a variant or adaptation of the original frame. Thus, the problem frame should be adjusted correspondingly and the frame concern will need to be adapted to fit the sub-problem in hand.

After each sub-problem has a fitting problem frame, these should be composed to produce the composite frame, where composition concern is another issue to be addressed. Composition concerns between pairs of sub-problems arise mainly if they use different projections of the same problem domain. Here composition is considered in terms of:

- Consistency between domain descriptions.
- Precedence between inconsistent domain descriptions.
- Interference between interactions with a domain.
- Scheduling of machines that interact with a common domain.

All these have to be addressed on a case-by-case basis for each individual problem. Finally, the frame concern of the composite frame should be checked to ensure that the initial problem is satisfactorily resolved.

3.2 AO Approaches

3.2.1 Aspect Oriented Requirements Engineering for Component Based Software Systems (AOREC).

AOREC focuses on identifying and specifying the component requirements relating to key aspects of the system. Such aspects are a way to take multiple, systemic perspectives onto components to better understand and reason about the data, functionality, constraints, and interrelationships of components. For instance, they can be helpful in reasoning about how different components interact via providing and requiring aspect details or what will be required when reusing a specific component, or if a given configuration of components will be valid with respect to a given aspect of the system etc.

The process as shown in figure 4 starts with analyzing general application requirements. The system requirements are used to identify candidate components, as shown in step 1in figure 4. The requirements for identified components are then elaborated. Aspects for each component are identified (step 2) and refined to determine the provided and required aspect details (step 3). Aspects are used to reason about component composition and configuration. The refined aspects for groups of components, or the whole system, if appropriate, are analyzed for aggregate aspects (step 4). The aggregate aspects can then be identified as new components, thus also causing change in previously analyzed components and initiating revision cycles (step 3.b).Once all system requirements are allocated per components; aspects for components and component groups are identified and aggregated, the produced components and aspects are verified against the system requirements (step 5). If the requirements are satisfactorily met by the produced component and aspect requirement model, the design phase starts.

Identification and Treatment of Crosscutting Concerns with AOREC:

- AOREC does not provide any general support for crosscutting concern identification. The aspects and aspect details are identified on a case-by case basis by the requirements engineer.
- For a small subset of aspects (namely User Interface, Collaboration, Persistence, Distribution, and Configuration) used in case studies for publications on AOREC an initial reusable breakdown of aspects to aspect details is suggested.
- This approach is well suited for treating component contributions to non-functional concerns, as many (all) components will contribute to any given non-functional requirement. Examples of such non-functional aspects are provided in the AOREC work. On the other hand, it is difficult to perceive how a functional requirement could be crosscutting in a component-based system, where components are intended to modularize functionality. Presently there are no examples of functional concerns for AOREC. Yet, the general approach of using provided/required aspect details for an aspect is well suited for both non-functional and functional aspects.

3.2.2 Concern-Oriented Requirements Engineering (CORE).

CORE strives to decompose requirements in a uniform fashion regardless of their functional or non functional nature. This makes it possible to project any particular set of requirements on a range of other requirements, hence allowing the flexibility to choose "base" and "crosscutting" concerns as desired, rather than having to follow "functional base and non-functional crosscutting" tradition.

The process in figure 5 starts with concern identification which can be carried out with any mix of RE approaches, such as viewpoints or use case based approaches. The identified concerns are related to each other through a matrix. The relationships between concerns are identified using domain analysis, ethnographic studies, natural language processing or alike. Having established the relations between concerns, the more specific kind of influence between them is defined through composition rules. The composition rules apply at the requirements-level granularity of the concerns and are similar to those of the AORE approach. After specifying the compositions between the concerns and their requirements, conflict identification and resolution begins with building of a contribution matrix. Once the conflicts are resolved, the mapping of the concerns onto the later stages of development is identified in the same way as for AORE.

Additionally, the influence of concerns (I. Brito, F. Vieira, A. Moreira, R. Ribeiro 2007) on the development lifecycle is specified: for instance, an availability concern may influence system architecture, while mobility may influence all: architecture, design, and implementation. The trade-off and interaction analysis conducted at the requirements level helps to tailor these choices to ensure that the architecture has an optimal pull in the various directions and meets the stakeholders' requirements effectively.

Identification and Treatment of Crosscutting Concerns with CORE:

- Similar to AORE, the CORE approach has a generic mechanism for concern handling suitable for both functional and non-functional concerns. Moreover, CORE does not make any particular distinction between these types. The approach also suggests a number of methods that can be used for concern identification as well as concern relationship identification.
- The meta concern space based concern identification approach can be complemented by the semantic natural language processing-base concern identification tool which is presently under construction. Such an integrated technique can provide a powerful concern identification mechanism well suited for both AORE approaches based on an aspect base separation and those employing a multi-dimensional perspective for requirements analysis.

3.2.3 Aspects in Requirements Goal Models (ARGM).

It is a goal based aspect oriented approach in which the decomposition begins with a procedure called Aspect Finder, which takes as input a set of goal and soft goal nodes, iteratively breaks them into sub-goals and sub-soft goals using the Decompose procedure, correlates these goals and their decompositions using the Correlate procedure, and resolves conflicts between goals/sub goals through the Resolve Conflicts (I. Brito, F. Vieira, A. Moreira, & R. Ribeiro 2007) procedure. The Correlate procedure establishes an initial relationship between root functional goals and soft goals. The relationship is represented by one or more correlation links. The sub-goals propagate their contributions up to the parent goal satisfaction, over the correlation links.

The Decompose procedure is used to input new sub-goal nodes, defined from the parent goals and soft goals. If any of the sub-goals provides a negative contribution to the parent, the Resolve Conflicts procedure is invoked, which removes the link between the offending sub-goal and its parent goal. The refinement process must be monotonic: no goal/soft goal must become less satisfied due to decomposition.

Identification and Treatment of Crosscutting Concerns with ARGM:

- With ARGM the goals (functionality) and softgoals (non-functional concerns) are recursively decomposed until they can be reduced to a specific task.
- The crosscutting functional and non-functional concerns can then be identified as the tasks that contribute to several goals and softgoals. However, due to the decomposition mechanism which does not allow decomposition to sub-goals that negatively contribute to the parent goal; it is not evident that the user requirements are adequately mapped to sub-goals and tasks.

3.2.4 Aspect-Oriented Software Development with Use Cases (AOSD/UC)

It suggests that use cases are crosscutting (A. Moreira, J. Araujo, J. Whittle 2006), (I. Brito, F. Vieira, A. Moreira, R. Ribeiro 2007) concerns, since the realization of each use case affects several classes. The AOSD/UC process is very similar to that of the traditional Use Case process. The only significant difference is the inclusion of separate use cases for non-functional requirements. These should be identified, defined and recorded using the infrastructure use cases along with identification and processing of functional use cases. Another minor difference is the packaging of each use case and its related elements separately into a use case slice.

The approach distinguishes two types of use cases: peer and extension. Peer use cases are distinct and independent of each other, each can be used separately with no reference to the other; they are the base requirements.

When peer use cases are composed, their full operations are composed without intervention in their execution. However, composition of peer use cases needs to take into consideration the overlapping behavior and conflicts between classes used for realization of different use cases.

Extensions are additional features on top of the base use cases. Though extensions can be defined independently of the base cases, they should be used along with the base. When extensions are composed with the base cases, their operations usually interfere with the execution of the operations of the base use case. The AOSD/UC also encourages the capture of non functional requirements as use cases, using a construct called infrastructure use case which refers to the activities that the system infrastructure needs to perform to meet user requirements.

The newly introduced notions of *use case slice* and *use case module* in AOSD/UC are both used to localize use case related and complementary artifacts respectively at a specific lifecycle stage and across all stages. Some of the elements that can be contained in a requirements level use case slice are demonstrated in Figure 6.

Identification and Treatment of Crosscutting Concerns with AOSD/UC:

- Identification and treatment of crosscutting functional concerns by AOSD/UC at the requirements engineering stage is quite similar to that of the Use Case approach, with minor differences of using classifier representation with an Extension Pointcut section for use case representation.
- In addition, AOSD/UC advocates applicability of use cases for non-functional requirement treatment: “so long as a requirement requires some observable response to be programmed into the system you can apply use cases” and “as long as you can define a test, you can define a use case for it” says.
- However, the argument for the non-functional use case definition seems to disagree with the actual definition of a use case as what the user does and what the system does in response. A user does not do anything for the non-functional requirements, s/he simply requires them to be present. So there is no action involved from the user’s side in most cases, for instance, when a user enquires about the weather over the internet, the user does not act with an intention to get a response within 2 seconds, or to synchronize his/her query with that of the weather update system – these are by-products of asking about the weather. Thus, the non-functional use case identification in this case should traverse in the following order: (a) what the user wants, (b) how it can be tested, (c) design test, (d) convert it into a use case, (e) extend the functionality with new use case.

- Another uncertain area is that of aspect definition in AOSD/UC. As mentioned earlier, it is suggested that a use case is an aspect as it crosscuts many classes. This, however, assigns a characteristic to the requirements level concern on the basis of its design representation. Such a concern will certainly be crosscutting at the design level (with OO classes), but this might not be sufficient for it to be crosscutting at the requirements level as well.

3.2.5 Aspectual Use Case Driven Approach (AUCDA).

The Aspectual Use Case Driven Approach is similar to AOSD/UC in that it separates the crosscutting functionality into inclusion and extension use cases and also suggests to model quality attributes as use cases. However, here quality attributes do not have to be attached to an infrastructure use case. Also this approach provides a structured way, a template - for identification of crosscutting concerns.

The aspectual use case driven approach is concerned with extending the use case model to integrate non functional requirements and identify the crosscutting functional use cases. The separated use cases for all concerns should also be integrated into a complete model using the provided composition mechanism.

The process as shown in figure 7 starts with actors and use case identification, as per the common Use Cases approach. This is followed by refinement of the identified use cases to externalize the functionalities that are spread across several use cases; these functionalities are encapsulated in include or extend use cases. Having modularized the functional concerns, the process addresses the identification of the non functional ones by analyzing the already elicited requirements as well as obtaining additional information from the system stakeholders. These NFRs are represented in a template. At the next step, the identified NFRs are integrated into the use case model derived in the previous steps. The integration is achieved by extending the use case model with the new stereotyped relationships which link the functional use cases with the new NFR use cases for each of the identified non functional concern. The extended set of relationships (i.e. extend, include, inherit, collaborate, damage, constrain), is used. Finally, the process summarizes the candidate aspect use cases: if a use case is related to more than one other use case, then that use case is a good candidate for an aspect.

Identification and Treatment of Crosscutting Concerns with AUCDA:

- In this approach the functional requirements are identified through use cases. The use cases are then analyzed for repeated behavior and the crosscutting functionality is separated from all use cases into included or extending use cases. Also, a single standing functionality can be identified as potentially crosscutting if it has relationships with many other use cases. Treatment of functional requirements is supported via the use cases approach, as well as can be assisted with the newly introduced additional relationships (damage, collaborate, constrain).
- Identification of crosscutting NFRs, on the other hand, is not supported with a clear procedure. The NFRs are identified via analysis of other requirements and additional information. Once identified, the NFRs receive a systematic treatment: by being represented via templates which help to record their inter-relationships, via use case diagrams with quality attributes and generalized use case pattern and action specifications.

3.2.6 Theme/Doc.

The core of the approach (E. Baniassad, & S. Clarke 2004), (E. Baniassad, & S. Clarke 2005) is the concept of a theme which represents a meaningful unit of cohesive functionality. The themes are loosely similar to functionalities identified by use cases. The method is centered on graph-based representation of the potential themes and this representation assisted analysis. It starts with the requirements engineer identifying action words from the requirements document and providing these, along with the requirements document as inputs to the Theme/Doc tool. Using the provided inputs the tool generates action views. This view demonstrates the links between requirement sentences and the action words.

The process of Theme/Doc is shown in Figure 8. It commences with the requirements engineer identifying action words from the requirements document and providing these, along with the requirements document

itself, as inputs to the Theme/Doc tool. It should be noted that synonyms referring to the same action can be grouped under one action word, and so can “minor” actions which do not have strong enough themes of their own.

In case of many links from a requirement to action words, the predominant action for the requirement should be defined (and thus selected as base), and the secondary action(s) should be clipped (using the tool), which indicates that the secondary action will crosscut the base behavior. Finally, the theme view is created by identifying entities, to be used at design stage, from the requirements document and providing these as additional input along with the action words and the requirements document. The theme view can be used for the design stage to produce separate themes for mapping onto Theme/UML. This view also helps to design generic views of the crosscutting themes.

Currently the work on Theme/Doc is focused on addressing the scalability issue of the approach. It is also looking at other possible clues for detecting crosscutting, in addition to the current one of having several actions/concerns mentioned in the same requirement.

Identification and Treatment of Crosscutting Concerns with Theme/Doc:

- As discussed above, Theme/Doc approach has based aspect identification on using action words, which makes it well suited to identify crosscutting functional requirements. On the other hand, non-functional requirements often do not have any action associated with them. The approach suggests that in such cases the requirements can be re-written to include action words.
- However this assumes that such requirements can be identified by the requirements engineer which in most cases is precisely the problem that the identification needs to address. Besides, the issue of how a particular non-functional requirement is related to other requirements is still unresolved (e.g., how security affects response time of mark allocation).

4. Comparison between RE Approaches.

Since AOSD techniques are fairly new, more so at the requirements, architecture and design levels, there are no established metrics or characteristics to compare AOSD approaches and the support they should provide to software engineers. However, a number of such desirable characteristics can be derived from existing best practice in software engineering. Our general comparison criteria (A.Rashid, R. Chitchyan 2002) are, therefore, based on a number of qualities that should be facilitated by any analysis and design approach from which: traceability, composability, evolvability and scalability. The definition of each will be demonstrated in the following section.

4.1 Comparison Criteria.

4.1.1 Identification and handling of functional and nonfunctional crosscutting concerns.

It is closely related to requirement discovery and understanding. It indicates if a given RE approach supports recognition of both types of crosscutting concerns. This is simply because if a crosscutting concern is not detected no further treatment can be applied to it. The handling part of the criterion stipulates that merely detecting a concern is not sufficient, a process should be provided to help in treating the given concern.

4.1.2 Composability.

It is the support for combining individual requirements into coarser grained requirements. Using the AO terminology, this support should include well defined join point model and composition semantics. The join point model exposes structured points through which requirements can be composed. The composition semantics (A. Rashid, A. Moreira, & J. Araujo 2003) provide systematic meaning to the composition. It is related to understanding, checking and management activities. It is a highly desirable property for an RE approach, allowing not only reviewing the requirements in their entirety, but also detection of potential conflicts very early on in order to either take corrective measures or appropriate decisions for the next development step.

4.1.3 Trade-off analysis and decision support.

This criterion indicates if an RE approach can facilitate the identification, analysis and resolution of conflicts that can arise both from crosscutting and non-crosscutting requirements, thus relating to under-standing, checking, management, and communication activities.

4.1.4 Traceability throughout lifecycle.

It is the preservation of traceability between the artifacts of the software lifecycle is a necessary quality for understandable, maintainable, and manageable recorded software. This criterion could be broken into two counterparts:

- Traceability of requirements and change to their sources of origin.
- Traceability between lifecycle artifacts.

4.1.5 Support for mapping.

Most current RE approaches recognize that not all identified requirements/concerns will progress into formal design artifacts: some will map onto decisions, trade-offs, or similar. This criterion indicates if an RE approach provides support for decisions on mapping (especially for crosscutting concerns) that facilitate efficient solution choice. It is related to recording, communication, and management activities.

4.1.6 Evolvability.

It is the ease of adapting requirements artifacts due to change in the requirements or removal/addition of a requirement. Ideally, effects due to change should be localized and easy to introduce to make it viable to keep the recorded requirements artifacts up to date with the changing requirements. This supports understandability and management.

4.1.7 Scalability.

Scalability is a desirable characteristic because a viable approach should be equally well suited for small and large projects, in particular because projects that start as small ones may grow with time. In some cases the scalability issue can be reduced to tool support. But there are cases where tool support is not sufficient.

4.2 Comparison Table.

The following tables shows some of the previous RE approaches discussed through our research and evaluated based on the criteria just mentioned above.

Table 1 shows the comparison of the features that support composability of requirements for all the non-AO and AO approaches.

Table 2 shows the comparison of the features that support trade-off and decision support for all the non-AO and AO approaches.

Table 3 shows the comparison of the features that support traceability of requirements and change to their sources of origin for all the non-AO and AO approaches.

Table 4 shows the comparison of the features that support traceability of requirements between lifecycle artifact representations for all the non-AO and AO approaches.

Table 5 shows the comparison of the features that support mapping of requirements to types of artifacts (e.g., decisions, structures, procedures, etc.) of later stages of lifecycle for all the non-AO and AO approaches.

Table 6 shows the comparison of the features that support evolvability of requirements and change to their sources of origin for all the non-AO and AO approaches.

Table 7 shows the comparison of the features that support scalability of requirements and change to their sources of origin for all the non-AO and AO approaches.

5. Conclusion

Requirements engineers and developers feel most comfortable when a requirement for a system property can confidently be allocated to a particular software component that assumes responsibility for satisfying the requirement. When such allocation is possible, the resulting software is well modularized, the modules have clear interfaces with each other, and all requirements are cleanly separated.

However, there are many classes of requirements for which clear allocation into modules is not possible using 'traditional' software paradigms. Many non-functional requirements come into this category. For example, performance is a factor of the system architecture and its operational environment; one cannot develop a performance module independent of other parts of a software system. Similarly, it could be hard to allocate responsibility for providing certain kinds of functionality in a cohesive, loosely coupled fashion.

Contemporary (non-aspect-oriented) requirements engineering approaches have been developed to primarily deal with one type of concerns. Some approaches have underlined the importance of non-functional concerns and proposed means to ensure their fulfillment in a system. Other approaches have focused on ensuring the required functionality of a system. In contrast, aspect-oriented approaches such as Cosmos and CORE explicitly acknowledge that all concerns are equally important and should be treated consistently.

Furthermore, AORE recognizes that all kinds of requirements (both functional and nonfunctional) can have a crosscutting influence on other requirements. For example, in traditional viewpoint-based analysis, security concerns are present in many viewpoints but the security requirement has no modular representation of its own. Furthermore, security and response-time requirements can influence each other.

The most important contributions of this approach are its ability to separate and then compose crosscutting and non-crosscutting requirements thorough simple yet powerful and flexible composition support; thorough conflict identification and resolution support, and the possibility of validating and tracing the requirements throughout the whole software development lifecycle.

Aspect oriented requirements engineering not only aims to provide improved support for separation of crosscutting functional and non-functional properties during requirements engineering, but also to provide a better means to identify and manage conflicts arising due to tangled representations of crosscutting requirements. Such means of early conflict resolution help to establish critical trade-offs even before the architecture is derived.

In our research, we have presented a range of representative non AO approaches to analysis and design and a comprehensive set of the significant AO approaches. In many cases, the AO approaches have built on the strength of the non AO approaches, but also aimed to address the previously overlooked issues of modularizing crosscutting concerns. We have discussed each presented approach based on a set of general criteria that reflects desirable properties of any software engineering approach, as: traceability, composability, evolvability and scalability.

From this we can conclude that generally traceability is improved due to modular representation of crosscutting concerns, and so are evolvability and scalability. The composability criterion, on the other hand, requires additional composition operators and procedures (e.g., design artifacts) when used with modularized crosscutting concerns, but also brings to light previously unexplored issues e.g., composition of requirements-level and architectural concerns and conflict detection through composition. From the discussion of non AO and AO approaches we can recognize the main contributions of the AO paradigm in the area of requirements engineering.

References

A.Rashid,R. Chitchyan. (2002). ”Aspect-Oriented Requirements Engineering: A Roadmap”, Computing Department Lancaster University, Lancaster, LA1 4WA, UK,2002.

A. Rashid, A. Moreira, and J. Araujo. (2003). "Modularization and Composition of Aspectual Requirements," presented at 2nd International Conference on Aspect Oriented Software Development (AOSD), Boston, USA, 2003.

A. Rashid, P. Sawyer, A. Moreira, and J. Araujo. (2002). "Early Aspects: a Model for Aspect-Oriented Requirements Engineering," presented at International Conference on Requirements Engineering (RE), Essen, Germany, 2002.

A. Rashid, A. Moreira, and J. Araujo. (2003). "Modularisation and Composition of Aspectual Requirements," presented at 2nd International Conference on Aspect Oriented Software Development (AOSD), Boston, USA, 2003.

A. Moreira, J. Araujo, J. Whittle. (2006). "Modeling Volatile Concerns as Aspects", Conference on Advanced Information Systems Engineer-ing (CAiSE), 2006.

A. Sampaio, N. Loughran, A. Rashid, and P. Rayson. (2005). "Mining Aspects in Requirements," presented at Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (held with AOSD 2005), Chicago, Illinois, USA, 2005.

E. Baniassad and S. Clarke. (2004). "Finding Aspects in Requirements with Theme/Doc," presented at Workshop on Early Aspects (held with AOSD 2004), Lancaster, UK, 2004.

E. Yu. (2001). "Agent Orientation as a Modelling Paradigm," Wirtschaftsinformatik, vol. 43, pp. 123-132, 2001.

E. Yu. (2005). "Strategic Actor Modeling for Requirements Engineering," Modelling Your System Goals - The I* Approach. London, UK: British Computer Society -Requirements Engineering Special Interest Group, 2005.

I. Brito, F. Vieira, A. Moreira, R. Ribeiro. (2007). “Handling Conflicts in Aspectual Requirements Compositions”, Transactions on AOSD, 2007, to appear.

J. Araujo, A. Moreira, I. Brito, and A. Rashid. (2002). "Aspect-Oriented Requirements with UML," presented at Workshop on Aspect-Oriented Modelling with UML (held in conjunction with the International Conference on Unified Modelling Language UML 2002), 2002.

P. Sawyer. "Software Requirements," in Software Engineering, vol. 1, R. Thayer and M. Dorfman, Eds., 3 ed: IEEE Computer Society Press, to appear.

R. Chitchyan, et al. (2005). Survey of analysis and design approaches, May2005, pp. 1–82.

R. Chitchyan, A.Rashid, P.Sawyer. (2008). ”Comparing Requirements Engineering Approaches for Handling Crosscutting Concerns”, Computing Department, Lancaster University, Lancaster LA1 4WA, UK, 2008.

R. Laney, L. Barroca, M. Jackson, and B. Nuseibeh. (2004). "Composing Requirements Using Problem Frames," presented at Requirements Engineering Conference (RE 2004), Kyoto, Japan, 2004.

S. Clarke, E. Baniassad. (2005). Aspect-Oriented Analysis and Design: The Theme Approach: Addison-Wesley, 2005.

“Survey of Aspect-oriented Analysis and Design Approaches”, Report of the EU Network of Excellence on AOSD by Ruzanna Chitchyan, Awais Rashid, Pete Sawyer, Allesandro Garcia, Monica Pinto Alarcon, Jethro Bakker, Bedir Tekinerdogan, Siobhan Clarke, Andrew Jackson, 2005.

S. Katz and A. Rashid. (2004). "From Aspectual Requirements to Proof Obligations for Aspect-Oriented Systems," presented at International Conference on Requirements Engineering (RE), Kyoto, Japan, 2004.

S. Katz and A. Rashid. (2004). "PROBE: From Requirements and Design to Proof Obligations for Aspect-Oriented Systems," Computing Department, Lancaster University, Lancaster COMP-002-2004, 2004.

Table 1. Comparison of the features that support composability of requirements

Approach	**Features That Support Composability**
AORE	Flexible and extensible composition rules and operators, unique ids for viewpoints, requirements and sub-requirements
PREview	not considered
NFRF	indirectly: SIG
I*	agents, dependency relationships, indirectly: SIG
PF	common domains
AOREC	per-component provided/required aspect details, provides/requires links
CORE	Flexible and extensible composition rules and operators, unique ids for viewpoints, requirements and sub-requirements, concern projections and compositional intersections
ARGM	matching topics of goals and task
AOSD/UC	*extension pointcut* ; *extend* and *include* relationships an use cases
AUCDA	new use case relationships: *collaborate, damage,* and *constrain* as well as standard *extend* and *include*; role parameters in UCPS and APS; binding specifications
Theme/Doc	order for theme composition in *clipped action view*

Table 2. Comparison of the features that support trade-off and decision support

Approach	**Features That Support Trade-Off Analysis And Decisions**
AORE	conflict detection through composition, contribution tables, temporal logic assertions of PROBE framework; weights assignment; stakeholder negotiations
PREview	Rule “organizational concerns take precedence over viewpoint requirements”
NFRF	correlation catalogues; priority assignment to softgoals; SIG augmented with claims about past decisions and contribution types
I*	Strategic Dependency and Strategic Rationale, use of NFR features
PF	partial support for assigning priorities to machines and events
AOREC	aspect details
CORE	conflict detection through composition, contribution tables, weights assignment; stakeholder negotiations, use of cumulative effects through folded tables and compositional intersection
ARGM	Conflict resolution process with Rule: “remove negative contribution link“
AOSD/UC	*before, after,* and *around* keywords for extension use cases
AUCDA	not considered
Theme/Doc	not considered

Table 3. Comparison of the features that support traceability of requirements and change to their sources of origin

Approach	Features That Support Traceability Of Requirements And Change To Their Sources Of Origin Criterion
AORE	viewpoint of requirements
PREview	source and change history sections in template; use of viewpoint focus
NFRF	SIG for sub-goals from the softgoals, indirectly: softgoal topic
I*	SD and SR diagrams, within SR diagrams the SIGs for goal and softgoals from the softgoals; implied intentionality
PF	Indirectly: context and problem diagrams and problem frames.
AOREC	not considered
CORE	abstract concerns in the meta concern space
ARGM	SIG for sub-goals from the softgoals, indirectly: softgoal topic
AOSD/UC	actors in use case diagram
AUCDA	actors in use case diagram, source in templates for NFR
Theme/Doc	not considered

Table 4. Comparison of the features that support traceability of requirements between lifecycle artifact representations

Approach	Features That Support Traceability Between Lifecycle Artifacts Criterion
AORE	Records of mapping decisions; PROBE framework.
PREview	Unique identifiers link requirements to viewpoints.
NFRF	Design decision and operationalization and correlation links, claims and augmentations in SIG.
I*	Design decision and operationalization and correlation links, claims and augmentations in goal and softgoal interdependency graph.
PF	Problem diagrams and annotations, frame concerns.
AOREC	Design-level aspects.
CORE	Records of mapping decisions and influence of concern on architectural choices.
ARGM	design decision and operationalization and correlation links, claims and augmentations in SIG.
AOSD/UC	Collaboration diagrams; use case slices and modules.
AUCDA	Collaboration diagrams.
Theme/Doc	Direct match between its requirements engineering and design models; major action and theme views.

Table 5. Comparison of the features that support mapping of requirements to types of artifacts (e.g., decisions, structures, procedures, etc.) of later stages of lifecycle

Approach	**Features That Support Mapping**
AORE	Guidelines
PREview	templates for concern decomposition
NFRF	type and decomposition catalogues; operationalizations – functional requirement links; contribution records
I*	actors, use of NFR framework (type and decomposition catalogues; contribution records) and goal and softgoal operationalizations
PF	not considered
AOREC	use of same aspects and details at both requirements and design levels.
CORE	Guidelines
ARGM	type and decomposition catalogues; operationalizations – functional requirement links; contribution records.
AOSD/UC	collaboration diagrams, some guidelines; pointcuts and aspects
AUCDA	collaboration diagrams, some guidelines
Theme/Doc	closeness of requirements and design models; theme view of the Theme/Doc tool

Table 6. Comparison of the features that support evolvability of requirements

Approach	**Features That Support Evolvability**
AORE	separation of concerns and composition concerns ; cross-reference tables
PREview	not considered
NFRF	correlation catalogue; claims and augmentation on SIG, sub-goal decomposition; link of softgoal to functional goal
I*	understanding reasons for requirements, use of NFR facilities
PF	sub-problem decomposition
AOREC	use of aspect details
CORE	separation of concerns and composition concerns ; cross-reference tables; absence of base/aspect distinction
ARGM	correlation catalogue; claims and augmentation on SIG, sub-goal decomposition; link between functional goal-goal-task
AOSD/UC	use case decomposition; minimalist notation for use case diagrams; use case slices and modules
AUCDA	NFRs templates, generic role elements, bindings
Theme/Doc	automation provided by the tool

Table 7. Comparison of the features that support scalability of requirements

Approach	**Features That Support Scalability**
AORE	XML representation of artifacts and composition, extensibility of composition operators/actions set
PREview	not considered
NFRF	catalogues for correlations and decompositions; records of decisions via claims
I*	indirectly: NFR framework support through catalogues for correlations and decompositions; records of decisions via claims
PF	modularity of frames
AOREC	aggregate aspects , allow modularization of requirements that affect groups of components.
CORE	XML representation of artifacts and composition, extensibility of composition operators/actions set; use of XPath queries.
ARGM	catalogues for correlations and decompositions; records of decisions via claims
AOSD/UC	use of coarse-grained use cases; packaging support
AUCDA	use of coarse-grained use cases; partial use case diagrams
Theme/Doc	Coarse-grained actions, action groups

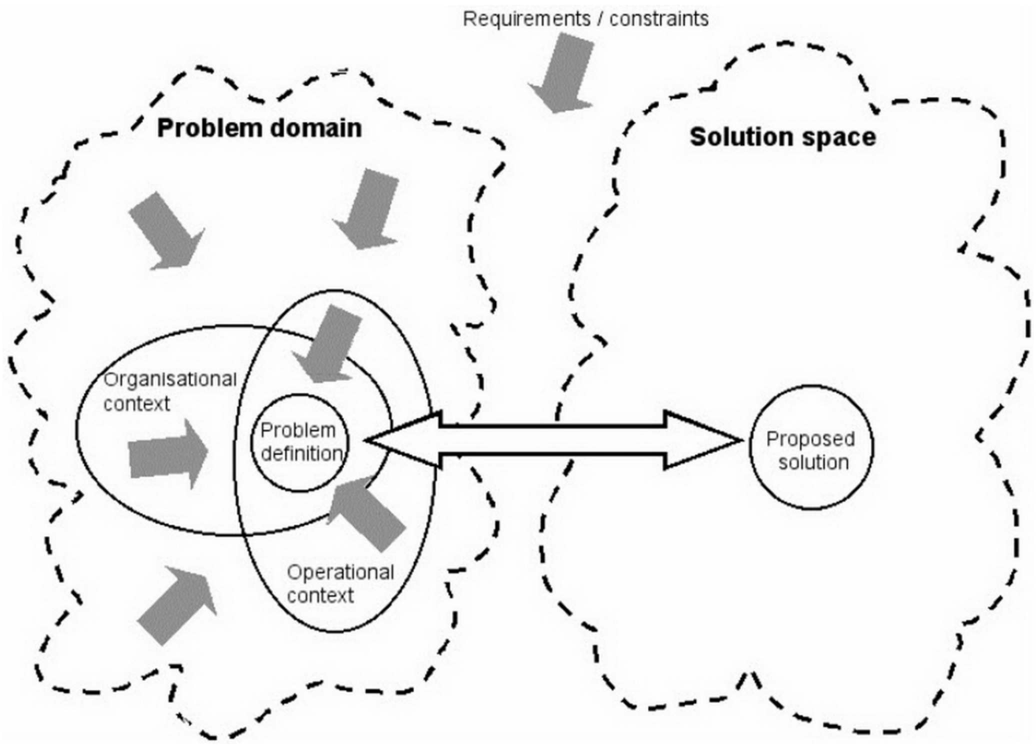

Figure 1. Requirements, Constraints, Problems and Solutions in RE

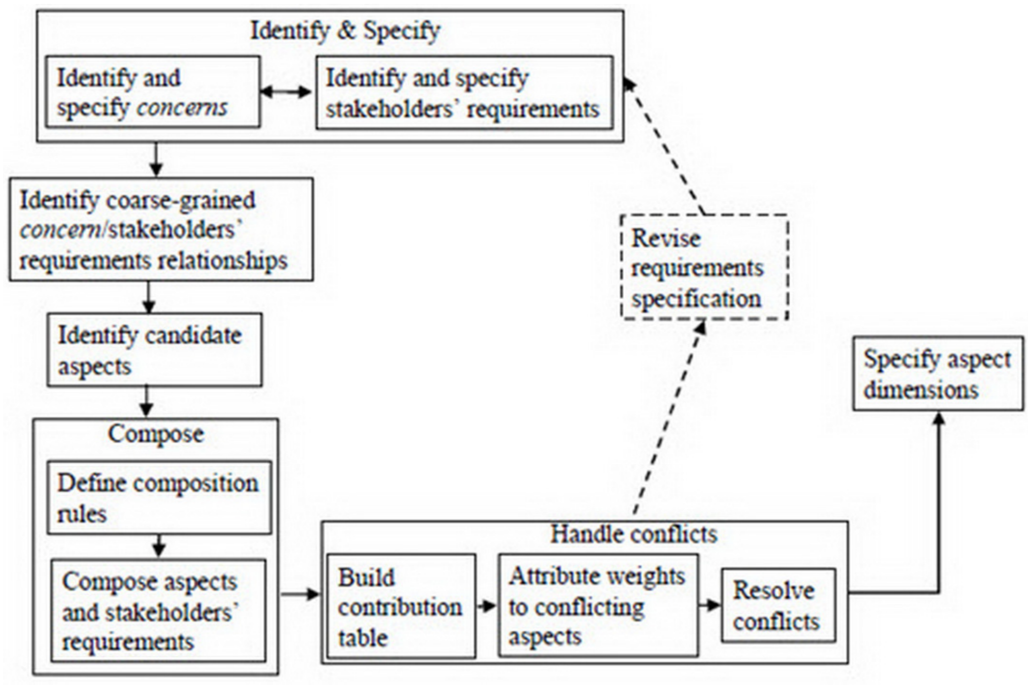
Identify & Specify
Identify and specify *concerns*
Identify and specify stakeholders' requirements
Identify coarse-grained *concern*/stakeholders' requirements relationships
Identify candidate aspects
Compose
Define composition rules
Compose aspects and stakeholders' requirements
Revise requirements specification
Specify aspect dimensions
Handle conflicts
Build contribution table
Attribute weights to conflicting aspects
Resolve conflicts

Figure 2. The Process Model for AORE

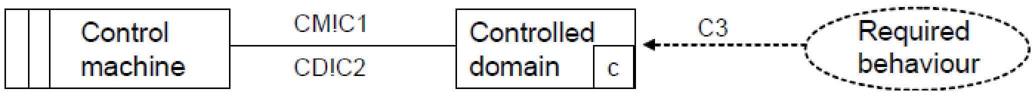

Figure 3. The Required Behavior Problem Frame

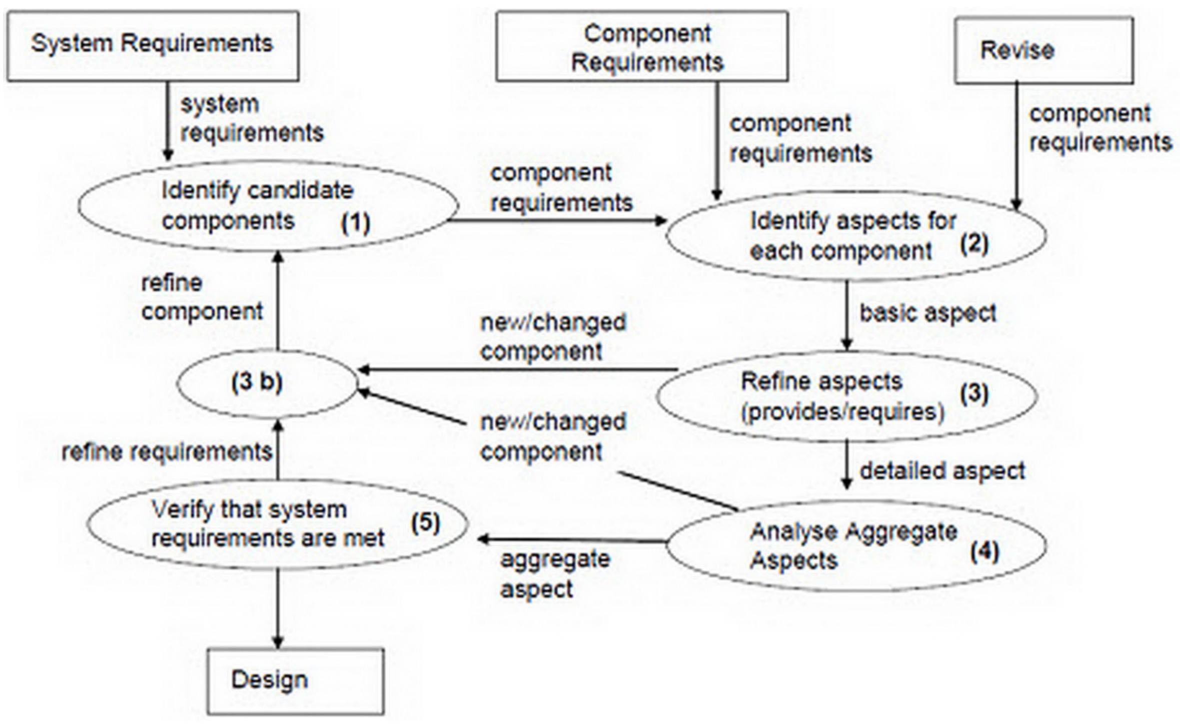

Figure 4. Basic AOREC process

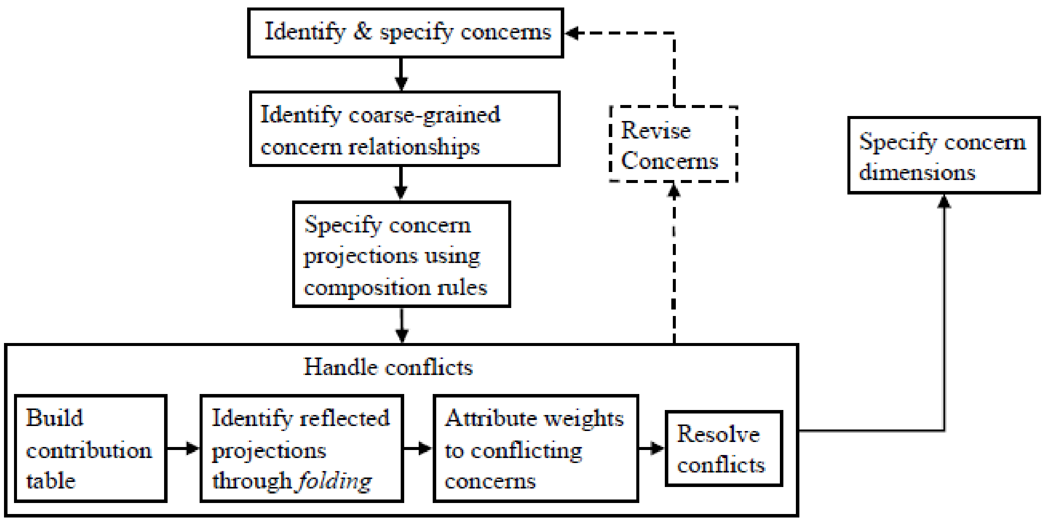
Identify & specify concerns
Identify coarse-grained concern relationships
Specify concern projections using composition rules
Revise Concerns
Specify concern dimensions
Handle conflicts
Build contribution table
Identify reflected projections through *folding*
Attribute weights to conflicting concerns
Resolve conflicts

Figure 5. The process model for CORE

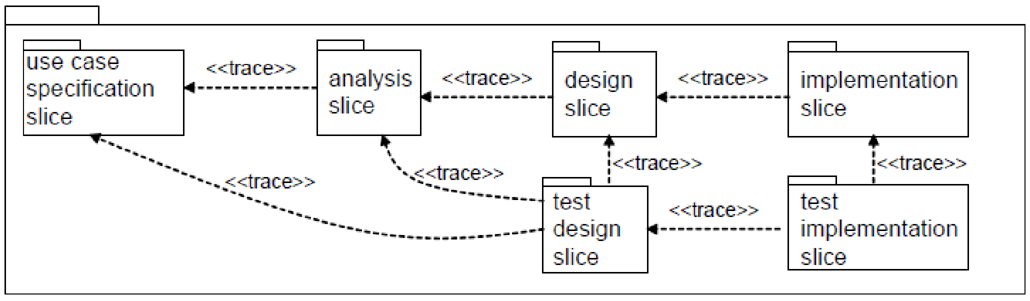

Figure 6. The Use Case Module

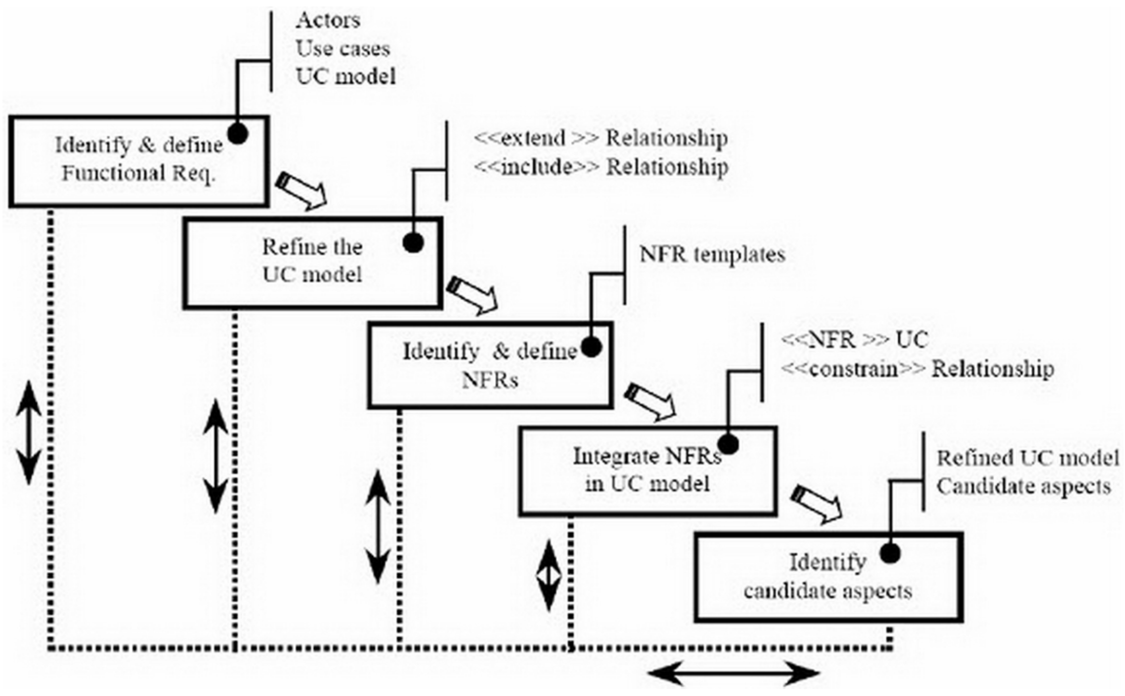

Figure 7. The process model for the Aspectual Use Case Driven Approach

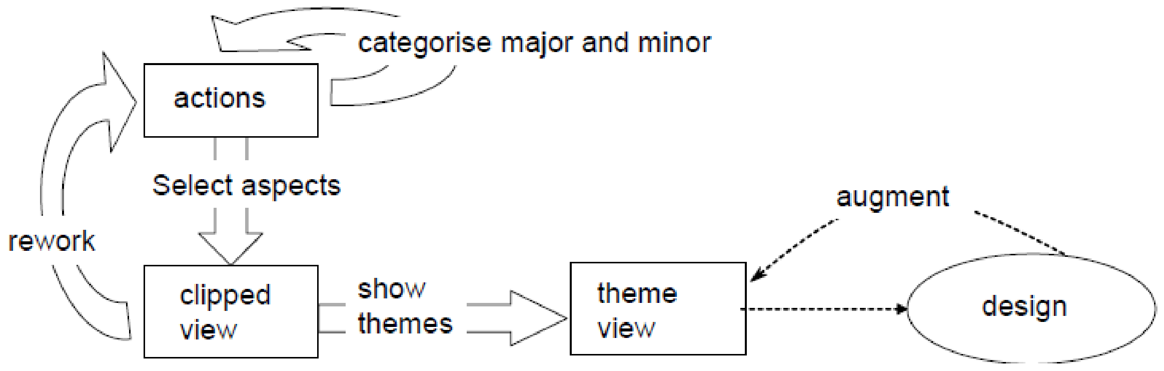

Figure 8. Theme/Doc Process