Chapter 3

REVIEW OF ASPECT-ORIENTED REQUIREMENTS ENGINEERING (AORE) APPROACHES

"We can send a message around the world seven times in one second, but it still takes 20 years for that message to go through a quarter inch of human skull."

-- Stephen C. Johnson

The success of a software system depends on how well it fits the needs of its users and its environment [NUS 00]. Software requirements comprise these needs, and requirements engineering (RE) is the process by which the requirements are determined [PAR 99]. Successful RE involves understanding the needs of users, customers, and other stakeholders; understanding the contexts in which the to-be-developed software will be used; modelling, analysing, negotiating, and documenting the stakeholders' requirements; validating that the documented requirements match the negotiated requirements; and managing requirements evolution [BET 09]. Existing requirements engineering approaches, such as use cases [JAC 92], viewpoints [SOM 97b], and goals [VAN 01] provide good support for identification and treatment of some kinds of requirements. However, these requirements approaches do not explicitly support well broadly-scoped requirements, such as crosscutting concerns, and do not explicitly support their composition. Moreover, they all suffer from the "tyranny of the dominant decomposition" [BRU 05]. AORE therefore, complements these approaches by providing systematic means for handling such crosscutting concerns.

This chapter consists of four sections. The first section presents a literature review of AORE approaches. The following section, Section 3.2, presents a critical analysis of AORE approaches discussed in Section 3.1. Section 3.3, presents a roadmap to the research work proposed in next chapter. Finally, the chapter finishes with a summary.
3.1 A Review of AORE Approaches

Over the last few years, several research efforts have been devoted in this area. We briefly discuss some of them here.

3.1.1 Aspect-Oriented Component Requirements Engineering (AOCRE)

Grundy has proposed an approach called Aspect-Oriented Component Requirements Engineering (AOCRE) [GRU 99]. This approach focuses on identifying and specifying the functional and non-functional requirements relating to key “aspects” of a system each component provides or requires. Aspects may be decomposed into aspect “details”, for example the data transfer, event broadcasting and version management provides/requires aspect details for collaborative work support. Some components may have many aspects and others a few. Unlike traditional object-oriented analysis object services, aspects may share component services, required aspects are as important to characterise as provided aspects, and often more than one other component may provide or require a component’s aspects. These “overlapping” aspects are a natural consequence of high-level categorisation of the systemic properties of components, and help requirements engineers gain understanding of related component characteristics. Basic notion of component aspects is depicted as figure 3.1.

Candidate components are found from OOA diagrams, by reverse engineering software components, or bottom-up consideration of individual, reusable components. They have found "perfect" identification of components is not essential during AOCRE, with requirements-level "components" acting as groupings of related services and aspects. These can be split, merged or otherwise refined at design-time, in a similar way to OOA objects being refined into classes. For each component, we identify (using possible stakeholder requirements and object services) aspects for which the component provides services or requires services from other components.

The basic AOCRE process begins after analysing general application requirements or individual or groups of components requirements. This allows iterative top-down and bottom-up requirements refinement. Engineers characterise a component’s aspects, aspect details, provided and required details, functional and non-functional properties,
and reason about interrelated components’ aggregate aspects. Components and aspects identified are refined into detailed component designs.

After identifying a component’s provided and required aspects, related components and aspects can be reasoned about. Inter-component relationships inferred by provided and required aspects allow engineers to reason about the validity relationships and aspects specified.

Figure 3.1: Basic Notions of Component Aspects, taken from [GRU 99]

Aspect-oriented component requirements assist when designing and implementing components. They provide a focused set of functional and non-functional constraints a design can be refined from, and provide a specification that an implementation can be tested against. Requirements-level components can be refined directly to matching design-level software components, or can be split, merged or otherwise revised, as
can requirements level component aspects. They also allow for design decisions to be influenced by weakening or strengthening aspect-level constraints.

AOCRE exhibit improved reusability and extensibility, and systems built with these components exhibit improved allocation of responsibility for data and behaviour among both reused and application-specific components.

3.1.2 Early Aspects: A Model for Aspect-Oriented Requirements Engineering

Rashid et al. proposed Early Aspects: a Model for Aspect-Oriented Requirements Engineering [RAS 02]. The model is composed of six activities as shown in figure 3.2. This is based on treating PREView concerns as adaptations of the AOP notion of aspects.

![Figure 3.2: Early Aspects: A Model for Aspect-Oriented Requirements Engineering, taken from [RAS 02]](image)

The first activity is to identify concerns which may be followed by discovering requirements by identifying viewpoints. The order in which these two activities are accomplished is dependent on the dynamics of the interaction between requirements engineers and the stakeholders. The next activity is to specify concerns in more detail. After identifying concerns and viewpoints, it is useful to relate concerns to
requirements using the templates of JPREView. The next activity is to identify
candidate aspects where a candidate aspect means a concern crosscuts several
requirements (i.e. if a concern may influence or constrain more than one viewpoint).
The next activity is to specify candidate aspects in details which includes refine
aspects, making them more concrete, and to identify interactions and conflicts
between them. In order to resolve conflicts among aspects a prioritisation approach is
used. The last activity in the model is identification of the dimensions of an aspect.
Aspects at early stages of software development can be described in terms of two
dimensions mapping and influence.

The model supports separation of crosscutting properties from early stages of the
development and identification of their mapping and influence on later development
stages. Now, it is possible to identify conflicts, establish possible trade-offs, and
promotes traceability throughout the system development and its evolution. The early
separation of crosscutting concerns improves modularisation and hence, it is possible
to build flexible and adaptable systems that meet the needs of volatile domains such
as banking, telecommunications and e-commerce. But it lacks on validation of
aspects, their composition with other requirements and resolution of possible conflicts
resulting from the composition process. It also lacks a notation to describe aspects,
their interactions and composition relationships at the requirements level.

3.1.3 Theme: An Approach for Aspect-Oriented Analysis and Design

Baniassad and Clarke proposed Theme: an Approach for Aspect-Oriented Analysis
and Design [BAN 04] that provides support for aspect-oriented software development
at two different stages. Theme/Doc, which is used for viewing and analyzing the
requirements at requirements phase; and Theme/UML, which allows a developer to
model features and aspects of a system, and specifies how they should be combined at
design phase. In this approach, a theme is an element of design, which is a collection
of structures and behaviors that represent one feature. Multiple themes can be
combined or integrated to form a system. Themes are further classified as base
themes, which may share some structure and behavior with other base themes, and
crosscutting themes which have behavior that overlay the functionality of the base
themes. Crosscutting themes are known as aspects.
Using the Theme/Doc tool, a developer can view the relationship between behaviours described in requirements documentation, and determine which behaviours are base, and which are crosscutting. Action view is used to identify crosscutting behaviors. To create an action view, two inputs are needed: a list of key actions identified by the developer by looking at the requirements document and picking out sensible verbs, and the requirements as written in the original document. Theme/Doc then performs lexical analysis of the text and generates the action view. An example of Theme/Doc is depicted as figure 3.3.

![Theme/Doc Theme View: logged, taken from [BAN 04]](image)

This approach involves three main activities as finding themes, modeling and composing themes, and checking Themes/UML. The process begins with finding themes. Here, identify actions and generate an action view to examine their relationships. After analysis of the view, determine that all of these actions will not be modeled as separate themes. Instead, determine the relationships between the actions to decide how to group the actions into larger themes. Here, we also determine which themes are crosscutting and which are base. The next step in theme process is modeling and composing themes. Here, the theme view is used to drive the modeling and composition semantics for design using Theme/UML. An example of Theme/UML is depicted as figure 3.4.

51
To ensure that the developer carefully considers the order in which crosscutting themes are composed with base themes, Theme/UML allows only one crosscutting theme per composition. Therefore it always needed to inspect the crosscutting relationships to determine the order of binding. For this use the clipped action views. In this view, the themes are positioned hierarchically, based on whether they crosscut one another. Finally, check the validity of the design choices.

![Diagram of Theme/UML Theme View: logged, taken from [BAN 04]](image)

This approach supports effective aspect identification, requirements coverage, traceability, and scalability of action views. However, this approach is only applicable for structured requirements document. As for the developers, they must possess the domain knowledge. Hence they must go through the whole requirements source document to identify the crosscutting concerns. They have to manually map the relationship between the themes and requirements. It is costly and time consuming to handle large amount of requirement sources.

### 3.1.4 Aspect-Oriented Software Development with Use Cases

Jacobson et al. proposed an approach called Aspect-Oriented Software Development with Use Cases [JAC 04] to handle crosscutting concerns. This approach is an extension to the traditional Use Case approach proposed by the same author and introduced new concepts like use case slices, extension use case, and pointcut. Here, a system is built use case by use case. The process begins with identifying use cases.
Further, we need to specify each use case, to analyze it, and to design use cases in terms of use case slices and use case modules. Use case slices are used to employ aspects. Use case modules are used to contain the specifics of a use case over all system models. Extension use cases are the special kind of use cases that contain additional functionality of the use case. We implement and compose them using a composition mechanism to weave them at loading time, compilation time, or run-time to form a complete system. As each use case progresses through the various models, the corresponding use case slice is updated in the respective models. These slices are shown shaded in Figure 3.5.

Figure 3.5: Use Case Slices, taken from [JAC04]

<table>
<thead>
<tr>
<th>Use case Specification</th>
<th>Use case design</th>
<th>Component design &amp; Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Withdraw Cash</td>
<td>Interface Cash</td>
<td>Cash Withdrawal</td>
</tr>
<tr>
<td>Transfer Funds</td>
<td>Interface Transfer Funds</td>
<td></td>
</tr>
<tr>
<td>Deposit Funds</td>
<td>Interface Deposit Funds</td>
<td>Cash</td>
</tr>
</tbody>
</table>

Figure 3.6: UML Elements for each Use Case Slice, taken from [JAC 04]
Figure 3.6 shows an example of use case progress through the UML models. For example, Withdraw Cash use case has Interface, Cash Withdrawal and Cash components in the Use Case design model and Component design & implementation model. Furthermore, the Interface and Cash from the Component design & implementation model are also components of Transfer Funds and Deposit Funds use cases.

The approach includes processes like identifying, specifying, analyzing, designing, and implementing use cases. The approach strongly related to UML; but lacks in conflicts handling.

3.1.5 Concern-Oriented Requirements Engineering (CORE)

Moreira et al. proposed a model called Concern-Oriented Requirements Engineering (CORE) model [MOR 05a], which supports multi-dimensional separation of concerns at the requirements level. Here, a concern is any coherent collection of requirements. This multi-dimensionality is achieved through a uniform treatment of both functional and non-functional properties. They also not classified concerns into viewpoints, use cases or aspects though their concerns encapsulate the coherent sets of functional and non-functional requirements. Concern space at the requirements level is represented as a hypercube as depicted as figure 3.7. Each face of the hypercube represents a particular concern of interest.

![Figure 3.7: Concern Space Represented as a Hypercube, taken from [MOR 05a]](image)

The process model is shown in figure 3.8. The process begins with identifying and specifying concerns using existing requirements elicitation mechanisms such as such
as viewpoints, use cases, and goals. The identified concerns are specified using well-defined templates. The second step is to identify coarse-grained relationships among concerns by relating concerns to each other through a matrix. These relationships are identified using techniques such as domain analysis, ethnography, and natural language processing. The third step is to specify the possible projections of each concern on other concerns, which is achieved through composition rules. The fourth step is to identify and resolve conflicts (if any) among the concerns. This is achieved by building a contribution matrix, where each concern may contribute positively or negatively to the others. Prioritization mechanism is used to solve conflicts and helping negotiation and decision-making. The last activity is to identify the dimensions of concerns. There are two dimensions of a concern at requirements level, which are mapping and influence.

![Diagram](image)

**Figure 3.8: Concern-Oriented Requirements Engineering, taken from [MOR 05a]**

The model supports multi-dimensional separation of concerns at the requirements level, which is achieved through a uniform treatment of both functional and non-functional properties. This is in direct contrast with existing RE approaches which typically focus on identifying the effects of non-functional requirements with reference to the functional requirements. Consequently, any broadly scoped influence of functional properties is not effectively dealt with.
This approach supports multi-dimensional separation of concerns, which treats each concern uniformly, hence, avoiding the dominant decomposing. It also establishes early trade-offs and solve conflicts that help negotiation and decision-making. But, it does not focus on the exact kind of relationships between two concerns, validation of proposed model with more case studies, and setting the concern specific actions and operators.

3.1.6 Aspect-Oriented Agile Requirements Approach

Araujo et al. proposed an approach called Aspect-Oriented Agile Requirements approach [ARA 05] that incorporates aspect-oriented concepts into agile software development at requirements level. Agile software development aims at fast communication and incremental delivering of software artifacts. The aspect-oriented agile requirements approach focuses on defining and modeling initial crosscutting requirements as scenarios. Scenarios are descriptions of desired or existing system behavior. Scenarios are commonly used in requirements engineering because they are easily understood by all stakeholders. Scenarios may crosscut other scenarios. Crosscutting scenarios are called aspectual scenarios. The process model is illustrated in Figure 3.9.

![Figure 3.9: Process Model, taken from [ARA 05]](image_url)
First, begin with identifying main functionalities and refine those using scenarios. Secondly, identify aspectual scenarios by analyzing all scenarios and observing some behavior that crosscut several scenarios. Third, it needs to compose aspectual and non-aspectual scenario definitions. Compositions are specified through simple rules. Finally, by analyzing the composition rules there may be some ambiguities, errors omissions and conflicts in these scenarios. This analysis can be realized through inspections. Also, the participation of the stakeholders is crucial. Conflict identification can be accomplished by adapting the existing mechanism. The final set of aspectual and non-aspectual scenarios plus composition rules are used to implement users' functionalities. The approach lacks on complete composition mechanism, conflict identification mechanism, validation, and tool support to guarantee that the approach will be used in an agile fashion.

3.1.7 Integrated Approach for Aspectual Requirements

Brito et al. proposed an Integrated Approach for Aspectual Requirements [BRI 06] to handle separation, modularization, representation and composition of concerns. The approach defines three main tasks, each one divided in several subtasks (see figure 3.10).

![AORE Process Model](image)

**Figure 3.10: AORE Process Model, taken from [BRI 06]**
The first task is identifying concerns, which aims at identifying the concerns of a system. A concern can be defined as a set of coherent requirements that the future system must have. This can be accomplished by analyzing the initial requirements, transcripts of stakeholders' interviews, etc. The second task is specifying concerns, which consists of many subtasks such as collecting information about concerns, specifying them using a template, and to design models like UML use case, interaction and class diagrams. The final task is composing concerns incrementally until the whole system is obtained. Each composition takes place in a match point in the form of a composition rule. A match point tells us which concerns (crosscutting or non-crosscutting) should be composed together. A composition rule shows how a set of concerns can be weaved together by means of some pre-defined operators. In order to accomplish this, we need to identify crosscutting concerns. The approach defined the main concepts as an extension of the UML metamodel, which allows a developer to better capture, analyze and understand the approach. The tool facilitates the specification of concerns, identification of crosscutting concerns, generation of the match point table and definition of composition rules. The approach does not define any method and tool with a reference model to support forward and backward traceability.

3.1.8 Aspect-Oriented Requirements Modeling

Z. Jingjun et al. proposed Aspect-Oriented Requirements Modeling [ZHA 07] aiming to apply AOP paradigm at requirements engineering stages of software development. This approach supports separation of concerns both functional and non-functional, and modeling them in UML with class diagrams and state-chart diagrams respectively. The process includes five activities as follow: identifying and specifying concerns, analyzing concerns, composing concerns, weaving concerns, and simulating and validating requirements. First, identify both functional and non-functional concerns from system requirements, and then specify them in UML as OOP class and aspect class. Second, analyze the relation among concerns by detecting and removing the conflicts if any. Third, compose concerns by describing the static structure of the system. Next, during weaving concerns, the whole state-chart diagram of the system is given, and then finishes the weave process. Finally, simulate the system function with the whole state chart, and validate the function whether it meets the system requirements or not. If not, return to the first activity, identify and specify concerns.
again. Or, complete the model process. This model supports separation and modeling of concerns. It also supports an effective method to solve the mismatch among the aspects, which reduces the complexity of the system and increases software’s reusability and maintainability. It uses terms functional and non-functional concerns as core and crosscutting concerns respectively. But, a crosscutting concern may be functional as well as non-functional. So, this method does not clearly identify and specify crosscutting concerns which are functional.

3.1.9 Requirements Description Language (RDL)

Chitchyan et al. proposed Requirements Description Language (RDL) [CHI 07], which is a symmetric AORE approach. It modularizes the requirements in symmetric fashion and represents them using the same abstraction, i.e., a Concern, to represent both crosscutting and non-crosscutting elements. A concern may be simple i.e. containing only requirements or composite i.e. containing requirements and other concerns. Both concern and requirement can be described as multi-sentence elements; where an element can be a subject or an object or relationship. A subject is described as an entity that undertakes actions and in RDL; it corresponds to the grammatical subject in the clause. An object is described as an entity that is affected by the actions undertaken by the subject of the sentence. Here, it corresponds to the grammatical object in the clause. A relationship is described as the action performed by the subject on or with regards to its object(s) and can be expressed by any the verbs or verb phrases in natural language. The main semantic load is carried by subject-relationship-object structure. The subject and object denote the entities of significance in it, whereas, the interactions between these entities are reflected by relationships. In this approach the relationship denotes the most central function, as it defines the functionality and/or properties that the subjects and objects provide. When specifying requirements stakeholders often qualify how important or significant a specific functionality or property is to them. In RDL such qualifications are represented by the degree element. Degree element depicts the strength of the relationship between the subject and object. The RDL elements discussed above are used for requirements description. The elements constituting our RDL are presented in the RDL metamodel in Fig. 3.11.
The next activity is element composition. The assembling of separately defined requirements modules aiming to ensure their desired interactions or addressing undesired ones is termed as composition. Here, three sub-elements of a composition element are constraint, base, and outcome. A Constraint element specifies checks and restrictions applied on a set of requirements, and the action taken in imposing these constraints. Base element provides a query for selecting the set of requirements that are affected by some constraints; and the temporal or conditional dependency between these requirements and the constraints. The Outcome element defines how to treat the imposition of constraints upon the base sets of requirements. Composition specifications are written based on these semantics rather than requirements syntax, hence providing improved means for expressing the intentionality of the composition, in turn facilitating semantics-based reasoning about aspect influences and trade-offs.

However, the approach requires a complete and precise requirements document, which can’t be expected before requirements elicitation and analysis. Also, to validate this approach, it is still required to refine requirements, identify and resolve conflicts and trade-offs.

3.1.10 Aspect-Oriented Approach to Handle Crosscutting Concerns in Activity Modeling

J. Zhang et al. proposed an aspect-oriented approach to handle crosscutting concerns in activity modelling [ZHA 08]. Activity modelling is about specifying the behavioural aspect of a system. It is typically used to define a computational process in terms of the control flow and data flow among its constituent actions.
Figure 3.12: Simplified Activity Metamodel, taken from [ZHA 08]
Figure 3.12 shows the simplified activity metamodel in the Meta-Object Facility (MOF) [MOF 06] specification. It consists of many kinds of nodes and edges. The sequencing of actions is controlled by control flow and object flow edges. An activity node can be an action, an object node or a control node. An object node holds data that flow through the activity model. Control nodes are responsible for routing control and data flows in an activity. Activities can be divided into different partitions that represent different kinds of activity groups for identifying actions that have some characteristics in common. Activity specifications grow with increasing complexity of the system and require lifecycle maintenance for the concerns that crosscut different activity modules.

The application of aspect-oriented approaches to activity modelling provides a solution to support this kind of modularization by encapsulating crosscutting concerns in a specialized unit called aspect. Following the AOP terminologies, two fundamental constructs are involved in an aspect model. First, we need to specify “where” (i.e., the locations, or join points) in the models the crosscutting behaviour emerges. Based on the activity metamodel definition, which defines an activity as being composed of a sequence of actions, join points refer to various kinds of actions that are allowed in activity modelling. A group of particular join points are represented in a special construct called pointcut, which defines a pattern to identify matching join points. Second, we need to specify “what” (i.e., the behavior) makes up the crosscutting concern. In activity modelling, the concern behavior is implemented using an activity model referenced by a special action called advice. An advice may contain a proceed operation action that refers to the current join point action. An advice can also obtain the join point information through a set of predefined reflective APIs.

These aspect-activity modelling concepts are defined upon a light-weight extension of UML through profiles and stereotypes [UML 03]. As shown in Figure 3.13, an aspect is a special activity that encapsulates a crosscutting concern. Pointcuts and advice are denoted as special actions. An aspect-activity model contains a binding diagram that defines which advice is bound to which pointcuts. Those bindings are realized by a stereotype named binds. Aspects are deployed to the base activity models through a special association stereotyped by the name crosscuts. The thisJoinPoint class defines
a set of APIs that are used to retrieve the reflective information of the matched join points (e.g., the signature of an operation action, or the kind of the join point action).

3.1.11 SLAI Methodology: an Aspect-Oriented Requirement Identification Process

Budwell and Mitropoulos proposed a methodology called The SLAI Methodology: an Aspect-Oriented Requirement Identification Process [BUD 08]. The SLAI (Structured Lexicon for Aspectual Identification) methodology is for systematically identifying potential aspects, both from functional and non-functional requirements, and defining their crosscutting effects to other functional requirements using a use case framework. The methodology is based upon the Language Extended Lexicon (LEL) for capturing requirements. LEL is used for vocabulary acquisition or understanding problem language. LEL consists of three elements: signs, notions, and behavioral responses. Signs represent any word or phrase that has a special meaning within the Universe of Discourse (UoD). Notions and behavioral responses help to define the meaning of each sign. Notions define the signs in the context of the UoD, whereas behavioral responses define how the sign is used. The principle of circularity and minimal vocabulary is used to define signs.
This methodology is divided into two flows, functional requirements flow and non-functional requirements flow. These flows are independent of each other and can be done simultaneously.

In functional requirements flow, actors of the system are identified first. Next, use cases are identified and detailed them into use case steps. These use cases have the form of verb and object of the verb. Each verb part of the use case is entered into the SLAI as a sign along with each object part of the use case. The use case phrase is also entered as a sign and as a verb phrase. The verb phrase repository is used to search for verb phrases so that before a verb phrase can be entered into the SLAI, there is a check to see if that verb phrase already exists or one similar to it exists, highlighting the fact that a potential aspect may exist. The verb phrase repository contains classifiers for the verb phrase such as use case, operationalization, aspect or potential aspect.

In non-functional requirements flow, SIG graph is used where the roots of the SIG graph represent non-functional requirements and leaf nodes for each path represent operationalization, which are either operations or design constraints. Since design constraints tend to be considerations that need to be addressed in the design phase, they are not included in the SLAI. The use case information is not enough to classify crosscuts use cases as aspects. Thus, for each crosscuts use case, a table is generated that lists each use case with which it interfaces. For each of these use cases listed, the condition of the extension, the composition rule operator, and the affected point are defined.

The process begins with clearly identifying and defining aspects at the requirements phase of software development. An aspect is a crosscutting concern or matter of interest in a software system. A concern must be derived from either functional or non-functional requirements. The next step is to capture crosscutting requirements (functional or non-functional) as aspects. Functional requirements can be identified as aspects in the form of UML use cases. A functional aspect is defined as a crosscuts use case. This is a new type of use case in UML after modifying extends and includes use cases. Non-functional requirements are converted into operationalization and then into use cases.
The approach supports the identification of aspectual or crosscutting use cases from both functional and non-functional requirements as well as systematically identifying both aspects and potential aspects by the use of a limited set of vocabulary for the terms defining the requirements of the system. From the study, it is observed that the methodology lacks on aspect composition as well as conflict resolution.

3.1.12 Crosscutting Concern Identification Using NLP

Ali and Kashirun developed a tool for crosscutting concern identification using NLP [ALI 08]. The model is based on the proposed approach which is adapted from Theme/Doc [BAN 04a] and EA-Miner [SAM 05]. The model is fully automated and involves non-collaborative processes unlike conventional requirements engineering processes, which are generally collaborative and iterative in nature.

The execution of processes in this model is sequential where each process requires output of the previous process as input for execution. The model is depicted in Figure 3.14. The model consists of processes like structuring requirements, removing redundancy, Part-of-Speech (PoS) analysis, semantic analysis, filter verbs identification, map relationship view, refining the relationship view, identifying dominating verb, and modeling crosscutting influences. Structuring requirements task involves numbering all the requirements agreed by the stakeholders to identify and manipulate each requirement uniquely in the next stages. Sometimes, same requirements are specified many times by different stakeholders. The redundant requirements are eliminated during removing redundancy process.

Verbs are extracted from each requirement during PoS analysis and they will be used for modeling the relation with the requirements and interdependency among other verbs. Semantic analysis task utilizes semantic tagger to analyze the context of the phrase in which the verb is used. This information is used to identify verbs used to describe similar requirements. Based on the semantic analysis performed, duplication of the verbs in terms of the context is discarded during filter verbs identification process. Next, we map the requirements using a matrix during map relationship view process. The requirements shared by more than one verb and the scattered verbs are identified based on the relationship view. Finally, identify the dominating verb in the requirement, which are the candidate aspect and model them to identify the crosscutting concern using Action View Model as used in Theme/Doc approach.
Figure 3.14: Crosscutting Concern Identification Model, taken from [ALI 08]

This tool provides automated support for crosscutting concern identification at the requirements level. The tool utilizes natural language processing technique to reason about properties of the concerns and model their structure and relationship. But, this
model lacks on conflict resolution and implementation and validation of the tool and tests it with case studies.

3.1.13 Aspect-oriented User Requirements Notation (AoURN)

G. Mussbacher proposed an approach Aspect-oriented User Requirements Notation (AoURN) that extends the User Requirements Notation (URN) with aspects [MUS 08]. This approach extends the URN by defining a joinpoint model for the Goal-oriented Requirements Language (GRL) and Use Case Maps (UCMs). GRL is a visual modeling notation and supports reasoning about goals and Non-Functional Requirements (NFRs). UCM is a visual scenario notation that supports the definition of scenarios. A scenario describes a specific path through the UCM model where only one alternative at any choice point is taken. Given a scenario definition, a traversal mechanism can highlight the scenario path or transform the scenario into a Message Sequence Chart (MSC).

All nodes of GRL graphs or UCMs optionally residing within the boundary of an actor or component are deemed to be joinpoints. Joinpoints can be matched by pointcut expressions. Therefore, pointcut expressions for AoURN models can identify any URN node which in turn can be transformed by the aspect. Joinpoints matched by a pointcut expression are indicated with small, filled diamonds called aspect markers, identifying the insertion points for aspectual behaviour in the base model.

Pointcut expressions are defined on pointcut diagrams, which are standard URN diagrams, allowing the requirements engineer to continue working with familiar models. Aspects, goals, and behaviour are defined on advice diagram, which are loosely coupled to pointcut diagrams, allowing advice diagrams and pointcut diagrams to be reused independently from each other. Further, Flexible composition rules are defined with URN itself and are therefore as expressive as URN and not restricted by the capabilities of any particular pointcut language.

Aspect-oriented GRL (AoGRL) [MUS,07] adds support for aspect-oriented modelling to GRL. Advice graphs are very similar to the notion of GRL catalogues if the catalogue describes the goal model of only one concern. It adds the ability to easily include GRL catalogues multiple times into a GRL model by visually specifying a pointcut expression on a pointcut graph. All nodes and links in a pointcut expression
are identified by *pointcut markers*. Pointcut graphs contain not just the pointcut expression, but also other elements not identified with a pointcut marker. Basic Elements of AoGRL Notation are depicted in figure 3.15.

Aspect-oriented Use Case Maps (AoUCM) [MUS 06] extends UCMs with the ability to specify *pointcut stubs*, thus enabling aspect-oriented modelling. Pointcut stubs (Fig. 4) are structurally the same as dynamic stubs but have a slightly different semantic meaning. While dynamic stubs contain plug-in maps that further describe the structure and behavior of a system, pointcut stubs contain zero or more pointcut maps that visually describe pointcut expressions. This is the only semantic change to traditional UCMs required in order to model aspects with AoUCM. Basic Elements of AoUCM Notation are depicted in figure 3.16.

This approach extends URN with aspects and thus unifies goal-oriented, scenario-based, and aspect-oriented concepts in one framework. Minimal changes to URN ensure that requirements engineers can continue working with goal and scenario models expressed in a familiar notation. At the same time, concerns in goal and scenario models, regardless of whether these concerns crosscut or not, can be managed across model types. But, it uses flexible composition rules that are only limited by the expressiveness of URN itself.
3.1.14 Representing Aspects in Design

Iqbal and Allen suggest an AOSD model [IQB 09] as depicted in figure 3.17 that represents aspect from the initialization of software to its implementation. It suggests the identification of aspects in the Use Case Model and Sequence Diagrams of the system. Use cases which involve multiple use cases like included or extended use cases may be considered as candidate aspects since they have the probability of crosscutting representation in design as well as in implementation. Similarly, the objects which have communication with multiple objects and which are represented in multiple sequence diagrams may also be regarded as candidate aspects. Proper specification of the candidate aspects can help identifying actual aspects.

In this approach, it is not mentioned how to identify aspects. Also, it lacks on implementing the model and validation it with some case studies.
3.1.15 Candidate Aspects in Software Requirements

Hamza and Darwish proposed a new approach to identify potential aspects by analyzing the system along two dimensions: the function/non-functions requirements and propagate them to the design phase [HAM 09]. The approach needs problem definition as input and produces EBT-NFR analysis model as output. The EBT-NFR model identifies crosscutting NFR and visually shows how they are scattered across the various modules in the system. Overview of the proposed approach is depicted in figure 3.18.

The process begins with performing requirements and domain analysis. The outcomes of this step are lists of: functional requirements (FRs): non-functional requirements (NFRs), Enduring Business Themes (EBTs), Business Objects (BOs), and Industrial Objects (IOs). The next step is to develop and document main use cases in the system using identified FRs, EBTs, BOs, and IOs. After identifying the main use cases, NFR Matching is performed where each identified NFR in the system is matched with a set
of use cases. The next step is Concept Analysis step where, concepts within the problem are discovered using use cases, EBTs, BOs, and IOs. Here, classify the EBTs as concept EBTs (i.e. form a formal concept) and none-concept EBT. Next, we need to identify candidate aspects by establishing and understanding the relationships between different NFRs and EBTs in the system. This is achieved by classifying NFRs, and then developing an EBT-NFR model that shows the relationships between EBTs and NFRs. An NFR can be classified into one of three types: Localized-NFR (L-NFR), Distributed-NFR (D-NFR), or an Aspect-EBT (A-EBT). To classify a NFR, calculate Coupling Factor (CF), which measures the coupling between a NFR and each EBT in the system. Finally, develop the EBT-NFR analysis model that visualizes the interaction between the different NFRs and EBT modules in the system and can be used as a link to the design phase.

![Diagram of Proposed Approach](image)

Figure 3.18: Overview of the Proposed Approach, taken from [HAM 09]

71
The approach uses Formal Concept Analysis and EBTs that can be identified using Software Stability Model to understand the interaction between NFRs and FRs, and to identify possible aspects in early stages of the development. Further, the proposed approach is the only approach that identifies crosscutting NFRs with respect to the structural nature of the system. The approach lacks a tool to partially automate the proposed approach and applicability of the approach to several case-studies to validate the results.

3.1.16 Use case and Non-functional Scenario Template-Based Approach to Identify Aspects

Liu et al. proposed a use case and non-functional scenario template-based approach to identify aspects [ZHE 10]. This approach consists of a sequence of activities as depicted in figure 3.19. The process begins with identifying and defining actors and use cases and building an initial use case model. The next activity is to refine use case model by identifying extensions and inclusions of the use cases. After this, describe NFRs at key association points in the use case model. Association points are of many types as NFRs Association Points, which are specific points of use cases where NFRs can be associated; use case association points associate NFRs to the described functionality, actor association points specify NFRs related to external entities, actor-use case association points represent NFRs related to interaction between external entities and a functionality, and system boundary association points define NFRs that are global in nature.

A concern can be identified with a set of architectural policies, and each of these can be described using specific dimensions that specify with more details the NFRs in architectural terms in each use case. Hence, describe architectural policies at platform-independent level through Architectural Policy Scenarios. Finally, identify aspects, where an aspect is a function that influences more functions or more use cases.

This approach is based on use cases and described and map non-functional requirements into function and architectures through non-functional scenario template. It not only improves modularity in the requirements which make it possible to begin tackling the problem of tangling, scattering of the requirement as early as in requirement analysis phrase, but also improves traceability from requirement analysis.
level to implement level, so it achieves a smooth transition between the system analysis and the design. It lacks on supporting the approach with formal method and applying it in more case studies and real systems.

Figure 3.19: Activities of Use case and Non-functional Scenario Template-Based Approach

3.2 Critical Analysis of AORE Approaches

A critical analysis of existing AORE approaches discussed in literature is given in Table 3.1. Here, an attempt is made to find out strength as well as shortcomings of each approach explained in section 3.1 of this chapter.

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Strengths</th>
<th>Shortcomings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect-Oriented Component</td>
<td>-Supports separation of crosscutting functional and non-functional properties.</td>
<td>-Too specific for component development and there is no evidence of its use in software development in general.</td>
</tr>
<tr>
<td>Requirements Engineering</td>
<td>-Improves reusability,</td>
<td></td>
</tr>
<tr>
<td>(AOCRE)</td>
<td>extensibility, and allocates the responsibility among reused and application-specific components. - Basic tool support for aspect-oriented requirements engineering.</td>
<td>-Lacks on clearly defining the identification of crosscutting concerns. -Lacks on tool support.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Early-aspects</td>
<td>-Supports separation of crosscutting functional and non-functional properties. -Identify conflicts and establish possible tradeoffs early on in the development cycle. -Improves modularisation and traceability. -Applicable for structured and unstructured requirements document.</td>
<td>-Lacks on validation of aspects, their composition with other requirements and resolution of possible conflicts resulting from the composition process. -Lacks on a notation to describe aspects, their interactions and composition relationships at the requirements level.</td>
</tr>
<tr>
<td>Theme</td>
<td>-All the relationship between the requirements are clearly identified and mapped. -Ambiguity in the requirements can be identified. -Supports traceability from requirements to design. -Provides a tool support to create relationships between concerns and the requirements.</td>
<td>-This approach is only applicable for structured requirements document. -The developer must possess the domain knowledge. -The developer has to manually map the relationship between the themes and requirements. -It is costly and time consuming to handle large amount of requirement sources.</td>
</tr>
<tr>
<td>Aspect-Oriented</td>
<td>-Provides ability to</td>
<td>-Does not provide any</td>
</tr>
</tbody>
</table>
| Scenario Modeling | independently specify both aspectual and non-aspectual scenarios.  
- Also provides composition along with better modularization and traceability. | systematic technique for identifying aspectual scenarios and handling conflicts during composition.  
- Lacks on tool support. |
|-------------------|---------------------------------------------------------------|---------------------------------------------------------------|
| Aspect-Oriented Software Development with Use Cases | - Handles both functional and non-functional requirements in a uniform fashion.  
- Supports the identification, specification, analysis, design, and implementation of use cases. | - Does not support conflicts resolution. |
| Concern-Oriented Requirements Engineering | - Handles both functional and non-functional requirements in a uniform fashion.  
- Supports multi-dimensional separation of concerns, early trade-offs and conflicts resolution. | - Lacks on the relationships between two concerns.  
- Also lacks on implementation of proposed model with more case studies for validation. |
| Integrated Approach for Aspectual Requirements | - Support better handling of separation, modularization, representation and composition of concerns.  
- Facilitates better understanding of concepts and their analysis along with tool support. | - Lacks on providing any technique to support traceability as well as trade-offs. |
| Aspect-Oriented Requirements Modelling | - Provides mechanism to cater both functional and non-functional concerns.  
- Here, core concerns are | - Does not clearly identify and specify crosscutting concerns which are functional.  
- Also, lacks on mapping |
| Crosscutting Concern Identification using NLP | Crosscutting concerns are considered as functional and treated as non-functional. | Provides full automatic facility for identifying crosscutting concerns at early stages of software development. | Does not provide any mechanism for resolving conflicts and also not provide evidences of its validation using case studies. |
| Aspect-oriented User Requirements Notation (AoURN) | Integrates goal-based, scenario-based, and aspect-based concepts in a single framework. | Clearly, represents relationship among goal and scenario models and also provides an ability to manage crosscutting concerns among them. | It also supports composition mechanism, but only limited to User Requirements Notation (URN). |
| Use case And Non-functional Scenario Approach to Identify Aspects | Supports aspects handling at requirements level. | Improves modularity in the requirements. | Does not support any formal method and its application in real systems. |
| Identification of Crosscutting Concerns | - Supports aspects handling at requirements level. | Improves modularity in the requirements. | - Lacks on validation of aspects, their composition with other requirements and resolution of possible conflicts resulting from the composition process. |
| It is a simple method and applicable to small scale requirements | Also improves traceability from requirement analysis level to implement level. | - Lacks on tool support. | |
| It can only identify non-functional crosscutting concerns. | | | |
UML

| Aspect-Oriented Use Case Modelling | - Handles both functional and non-functional requirements. - Treats business rules as an important source of aspects when it cross-cuts more than one use cases. - Further, improves the reusability of both the base model and the representation of business rules. - Successfully, realizes the composition of aspects to the core functionalities at the requirement level. | - Does not support conflicts resolution. - Lacks on providing any technique to support traceability as well as trade-offs. - Lacks on tool support. |

3.3 A Roadmap to Research

Based on the above discussion, we present a roadmap to our research. The focus of our research work will be exactly on handling crosscutting concerns during requirements phase and to propose an integrated AORE process model for advanced separation of concerns having the following challenges:

- Prevent the tyranny of dominant decomposition symptom.
- Improve the ability to identify, specify and compose both crosscutting and non-crosscutting concerns.
- Handle conflicts that could emerge when two or more crosscutting concerns affect each other.

3.4 Summary

In this chapter, we have briefly discussed several research efforts contributed towards AORE along with their strength and flaws encountered. Some AORE approaches still suffer from the tyranny of dominant decomposition symptom because they are extensions to traditional requirements engineering approaches and result in scattering and tangling; some are unable to identify and specify both functional and non-functional crosscutting concerns because most of the approaches consider only non-
functional concerns as crosscutting; some need to manage concerns and to offer a systematic method to handle conflicts. Thus, the main motivation towards integrated AORE process models is the analysis of existing AORE approaches, incorporating their strengths into a single integrated approach. Because, evidence suggests that most of the projects failures are due to requirements. Hence, for avoiding more projects failures, we need to properly elicit requirements and concerns.

In next chapter, we have proposed such an approach that incorporates aspect-oriented concepts and supports identification, specification, and conflict resolution of all concerns.