A Practical Use Case Modeling Approach to Specify Crosscutting Concerns: Industrial Applications

Tao Yue[1, 2], Huihui Zhang[3], Shaukat Ali[1], Chao Liu[3]

[1]Simula Research Laboratory, Oslo, Norway
{Tao, Shuakt}@simula.no
[2]University of Oslo, Oslo, Norway
[3]Beihang University, Beijing, China
{zhhui, liuchao}@buaa.edu.cn

Abstract. Use case diagrams together with use case specifications are commonly used to specify system requirements. To reduce imprecision, ambiguity, and incompleteness in use case specifications, an approach with template and restriction rules is often recommended to achieve better understandability of use cases and improves the quality of derived analysis models. However, when crosscutting concerns are modeled together with non-crosscutting concerns as use case models, resulting use case models often result in cluttered diagrams and redundant information in use case specifications. Therefore, the overall reusability of the use case models is usually low. To cope with these, we extend a general use case approach, named as RUCM, for modeling crosscutting concerns, along with a weaver to automatically weave aspect use case models into their corresponding base model to facilitate, e.g., automated requirements analysis. The extended RUCM approach has been evaluated with three industrial applications from communication, maritime and energy domains and aviation. We also compared the modeling effort required to model three sets of crosscutting concerns from the industrial applications, when using and not using the extended RUCM approach. Results show that more than 80% of modeling effort can be saved.

Keywords: Use case modeling, Reuse, crosscutting concern, aspect-orientation.

1 Introduction

Use case models (UCMods) are a widely used means for specifying functional requirements of systems, which are generally text-based and contain ambiguity. To decrease such ambiguity, previously we proposed the Restricted Use Case Modeling (RUCM) methodology [18]. RUCM contains standard UML use case diagram notations, a use case template and a set of restriction rules for textual Use Case Specifications (UCSs).

Use case modeling of communication and control systems poses special requirements such as specifying the communication medium and its various properties (e.g., packet loss). Behaviors related to such properties are often redundant across use cases and if modeled directly with them can result in cluttered use case diagrams and redundant UCS fragments, thus making them difficult to comprehend and reuse. However, such behaviors are essential for specifying use cases, e.g., for robustness testing (the

need of one of our industrial partners) [5]. Therefore, it is required to capture sufficient information in the UCMod such that this kind of analysis/testing can be facilitated. One possible way of facilitating such analysis/testing is to transform UCMods specified using our approach into other software artifacts (e.g., standard UML state machines or even aspect state machines [5]).

The motivation of specifying crosscutting concerns was driven by our existing project at Cisco Norway[1] to develop tools and techniques to support model-based robustness testing. The need of the current work arose due to the difficulty of constructing required UML state machines for them, which are used for supporting model-based robustness testing. However, they are more familiar with writing textual use cases. Such difficulty is by no means specific to Cisco Norway and is commonly observed in industry. Therefore, specifying crosscutting concerns at the requirements level and facilitating (automated) transition to required UML state machines or other artifacts are considered an appropriate solution in practice.

Inspired by Aspect-Oriented Requirements Engineering (AORE) [12] and also driven by needs of our industrial partners to deal with specifying crosscutting concerns, we extend our RUCM approach to support modeling crosscutting concerns, named as AspectRUCM. The AspectRUCM methodology comprises of the AspectRUCM profile (extending UML use case diagram notations) and a set of guidelines (formalized as a UML activity diagram) for applying the profile for specifying crosscutting behaviors.

Eliciting and identifying crosscutting behaviors or applying the AspectRUCM methodology to support other requirements engineering activities (e.g., requirements verification and validation) is not the focus of the paper. However, as the first step towards supporting automated analysis or generation (e.g., test cases), we need a formalization mechanism to model textual UCMods. We have already developed such a formalization mechanism: a use case metamodel referred to as UCMeta, in our previous work [19]. Based on it, we present a weaver to automatically weave aspect UCMods specified using AspectRUCM to their corresponding base UCMod.

Our work is evaluated with three industrial applications and results demonstrate that AspectRUCM is applicable for real, industrial applications. We also evaluated modeling effort required when using AspectRUCM and not using AspectRUCM to model three sets of crosscutting concerns of the three industrial applications. Results show that more than 80% of modeling effort can be saved when using AspectRUCM.

The rest of the paper is organized as follows. In Section 2, we briefly discuss RUCM, UCMeta, and the running example used to illustrate our approach. The AspectRUCM methodology is discussed in Section 3. Section 4 presents the evaluation. Related work is presented in Section 5. The paper is concluded in Section 6.

2 Background

We present the running example in Section 2.1. In Section 2.2, we introduce RUCM. The metamodel of formalizing RUCM and AspectRUCM is presented in Section 2.3.

[1] http://www.cisco.com/web/NO/index.html

2.1 Running Example

We used a subsystem of a Video Conferencing System (VCS) developed by Cisco, Norway that has been previously used in our previous works [5]. Fig. 1 shows that the VCS is responsible for sending/receiving multimedia streams, i.e., audio and video to a number of other VCS (Endpoints). The core functionality of a typical VCS includes: establishing/disconnecting audio/videoconferences and starting/stopping presentations in addition to audio/videoconferences. The other Endpoints have the similar functionality.

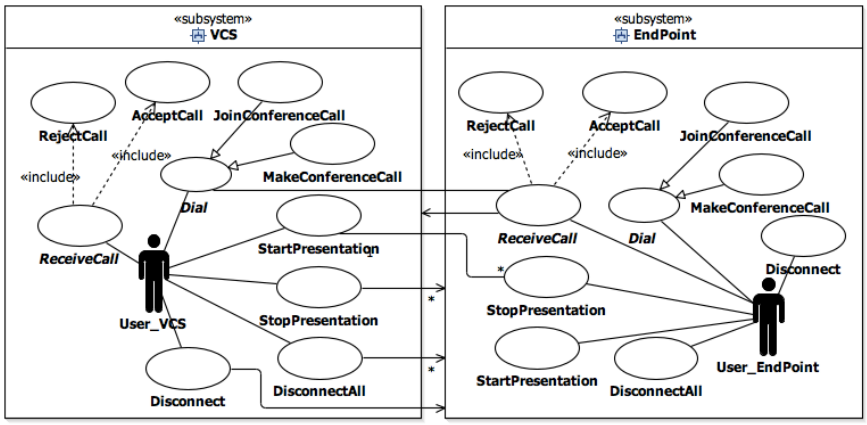

Fig. 1. Use Case Diagram of VCS

We group use cases into two packages corresponding to VCS or EndPoint. Both have the same set of use cases as they are equivalent communication end points. However, their implementations might be different, which form different products with the same functionalities. In Fig. 1, we have defined associations between use cases, e.g., *StartPresentation* of VCS with *StopPresentation* of Endpoint with cardinality 0 to many (*) on the *StopPresentation* side. This means when presentation is started on VCS, it stops the presentation of any other endpoint, which is currently presenting.

2.2 RUCM

Though there exist many requirements specification techniques, these techniques are either fully formal or fully informal [16]. Use case modeling is one of the most widely applied and structured specification techniques, which nicely combines diagrammatic and textual descriptions and offers an intuitive and precise foundation for requirements specification. We have previously devised a methodology named as Restricted Use Case Modeling (RUCM) [18] to reduce ambiguity and improve understanding of requirements, and facilitate automated analysis. Table 1 is an example of UCS documented with an editor implemented the RUCM methodology. Use case *Disconnect* contains one basic flow and two specific alternative flows. The two specific alternative flows are used to branch from the basic flow under specific conditions. RUCM specifies three different types of alternative flows. Specific and bounded flows indicate from

Table 1. Use Case Disconnect (specified in the RUCM editor)

Use Case Name	Disconnect
Brief Description	User disconnects an Endpoint participating in a conference call.
Precondition	The system is in a conference call.
Primary Actor	User_VCS
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 User_VCS sends a message to the system to disconnect an Endpoint.
	2 The system VALIDATES THAT Endpoint to be disconnected is in the conference call.
	3 The system sends a disconnection notification to Endpoint via Ethernet network.
	4 Endpoint sends an acknowledgement message back to the system via Ethernet network.
	5 The system VALIDATES THAT The conference call has only one EndPoint.
	6 The system disconnects Endpoint.
	Postcondition The system is idle.

Specific Alternative Flow "alt1" ▼	RFS 2
	1 The system sends a failure message to User_VCS.
	2 ABORT
	Postcondition The system is in a conference all.

Specific Alternative Flow "alt2" ▼	RFS 5
	1 The system disconnects Endpoint.
	2 ABORT
	Postcondition The system is in a conference call.

which step in which flow of reference they branch whereas a global flow can branch from any step. For instance, the specific alternative flow in Table 1 branches from Reference Flow Step (RFS) 2 in the basic flow, and the condition for branching is the negation of step 2 of the basic flow. Restrictions to natural language take the form of keywords, such as VALIDATES THAT [18].

2.3 UCMeta

UCMeta is the intermediate model in aToucan [19], used to bridge the gap between a textual UCMod and a UML analysis model (e.g., class and sequence diagrams). As a result, we have two transformations: from the textual UCMod to the intermediate model, and from the intermediate model to the analysis model. UCMeta can also be considered as a way to formalize textual UCMods and therefore the formalized UCMods can be used for automated analysis or test generation. Metamodel UCMeta also complies with the restrictions and use case template of RUCM.

UCMeta is hierarchical and contains five packages: *UML::UseCases*, *UCSTemplate*, *SentencePatterns*, *SentenceSemantics*, and *SentenceStructure*. *UML::UseCases* is a package of UML 2 superstructure [2], which defines the key concepts used for modeling use cases such as actors and use cases. Package *UCSTemplate* not only models the concepts of the use case template but also specifies three kinds of sentences: *SimpleSentence*, *ComplexSentence*, and *SpecialSentence*. In linguistics, a *SimpleSentence* has one

independent clause and no dependent clauses [8]: one *Subject* and one *Predicate*. UCMeta has four types of *ComplexSentences*: *ConditionCheckSentence*, *ConditionalSentence*, *IterativeSentence*, and *ParallelSentence*, which correspond to four keywords (i.e., VALIDATES THAT, IF-THEN-ELSE-ELSEIF-ENDIF, DO-UNTIL, and MEANWHILE) that are specified in RUCM to model conditions, iterations, concurrency, and validations in UCS sentences. UCMeta also has four types of special sentences to specify how flows in a use case or between use cases relate to each other. They correspond to keywords RESUME STEP, ABORT, INCLUDE USE CASE, and EXTENDED BY USE CASE.

3 The AspectRUCM Approach

This section presents our AspectRUCM approach. Section 3.1 presents the domain model capturing main aspect concepts, Section 3.2 discusses the profile, Section 3.3 defines weaving directive interaction overview diagram, and modelling guidelines are presented in Section 3.4. Our weaver is presented in Section 3.5. We used example of *AdaptCallRate* (Table 2) to explain the concepts.

Table 2. Use *Case AdaptCallRate* (specified in the RUCM editor)

Use Case Name	AdaptCallRate
Brief Description	The system adjusts the call rate based on the quality of services of the network.
Precondition	Network connection is established.
Primary Actor	Timer
Secondary Actors	None
Dependency	EXTENDED BY USE CASE SelectedUseCases
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 The system VALIDATES THAT the network is experiencing packet loss.
	2 The system gradually decreases conference call rate.
	Postcondition: The system is in a degraded mode.

Specific Alternative Flow (Untitled) ▼	RFS 1
	1 ABORT
	Postcondition: The system is in a normal operation mode.

3.1 Domain Model

A domain model for AspectRUCM is shown in Fig. 2. An aspect describes a crosscutting concern, which in our context is a set of system requirements, which crosscuts another set of system requirements describing the main functionalities of the system. A joinpoint is a model element, which corresponds to a pointcut where an advice (e.g., use cases and actors in a use case diagram, and additional steps of flows of events, preconditions and postconditions in UCSs) might be applied. Theoretically, all model elements in UML use case diagrams and constructs of UCSs are possible joinpoints. However, we only define five types of joinpoints in AspectRUCM: actors, use cases, preconditions, postconditions, and steps of flows of events, which are sufficient based on our experience of evaluating AspectRUCM with three industrial case studies (Section 4.1). A pointcut selects

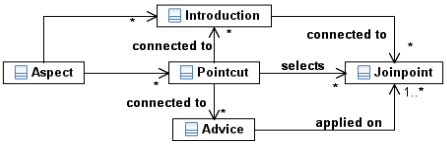

Fig. 2. Domain Model

one or more joinpoints with similar properties. A model element (e.g., actor) can be introduced in two different ways. It can be introduced to an aspect UCMod without being connected to any pointcut and it can also be connected to a pointcut through another model element.

3.2 AspectRUCM Profile

The profile diagram of AspectRUCM is provided in Fig. 3. An aspect describes a crosscutting concern and we specify stereotype «Aspect», which extends UML *Package*. «Aspect» has two attributes: *baseUCM* specifying the comma separated name(s) of the base UCMod(s), on which an aspect UCMod will be weaved, and the name of the aspect itself. We use a package to group model elements including use cases and actors to specify a crosscutting concern. For example, as shown in Fig. 4, package *NetworkDegradation* stereotyped with «Aspect» contains use case *AdaptCallRate*, actor *Timer*, etc. Another example is provided in Fig. 5, where the crosscutting concern *Standby* is mod-

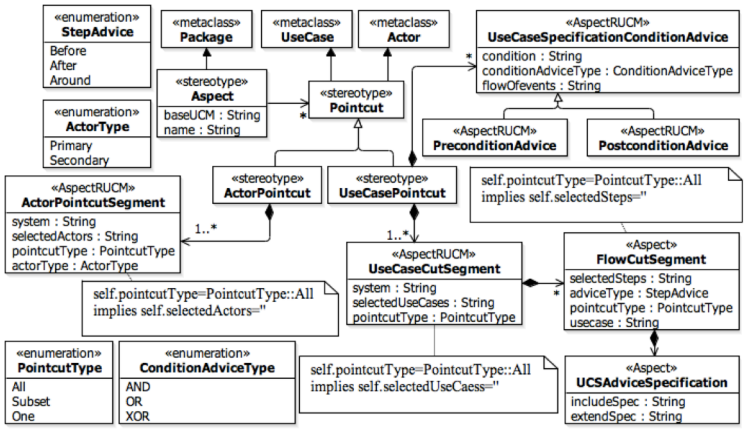

Fig. 3. Profile diagram of AspectRUCM

eled as an aspect UCMod. The *Standby* behavior of the VCS becomes active when it is idle for 5 minutes (a property of *Timer*). When any activity is performed by any actor of the system while it is in *Standby*, the system becomes active. One benefit of using a package to group model elements of an aspect UCMod, is that the model elements contained in the package and without stereotypes (from the AspectRUCM profile) applied are by default considered as elements newly introduced to the base UCMod. By not explicitly stereotyping model elements in an aspect UCMod reduces modeling effort (in terms of the reduced number of elements that could have stereotypes applied instead) and therefore results in less cluttered use case diagrams.

As shown in Fig. 3 an aspect UCMod might have one or more pointcuts. We specify two types of pointcuts in our profile: «UseCasePointcut» and «ActorPointcut», which specialize the abstract stereotype «Pointcut».

Use Case Pointcut. A use case pointcut selects one or more use cases of a system and the flows of events of the UCSs of the selected use cases. This is realized via the composition association between stereotype «UseCasePointcut» and class *UseCaseCutSegment*, which is further associated to class *FlowCutSegment* (Fig. 3). Class *UseCaseCutSegment* specifies the system where a selected use case belongs to (attribute system: String), a set of selected use cases (*selectedUseCases : String*), and the type of the pointcut (enumeration *PointcutType* and attribute *poincutType: PointcutType*), which can be selecting *All*, *Subset*, or *One* use case(s) of the system. As shown in Fig. 4, we apply «UseCasePointcut» to use case *SelectedUseCases*. The values of the attributes of the stereotype show that we select all the use cases of the VCS and EndPoint systems. *AdaptCallRate* extends all the selected use cases and is triggered by *Timer* periodically. The example in Fig. 5 shows that this aspect introduces two new use cases (i.e., *Standby* and *Exit-Standby*) by extending all the use cases of the two systems as indicated by the values of the attributes of «UseCasePointcut». Use case *Standby* is triggered by *Timer* and any actor of the two systems can trigger use case *Exit-Standby* (via actor pointcut *SelectedActors*, Section 3.2).

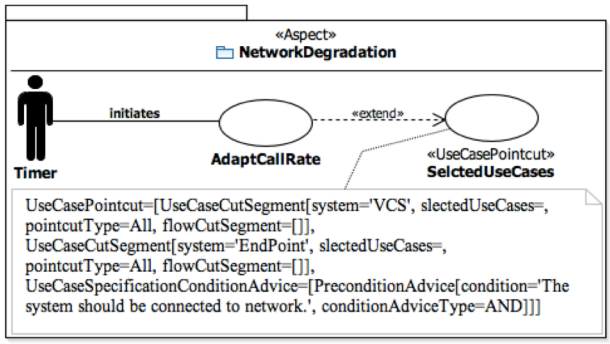

Fig. 4. Aspect Use Case Diagram of *Network Degradation*

As shown in Fig. 3, *UseCasePointcut* should have at least one *UseCaseCutSegment*. A *UseCaseCutSegment* is composed of zero to many *FlowCutSegment*s, which specify the selected steps of the flows of events of a selected use case (*selectedSteps : String*), where *Before*, *After*, or *Around* advice (*adviceType : StepAdvice*) should be applied. This part of the pointcut should also indicate the type of the pointcut: selecting *All*, *Subset*, or *One* step of a UCS, and the step sentence to be introduced through advice (*adviceSpec : UCSAdviceSpecification*). Note that it is possible that a *UseCaseCutSegment* does not contain any *FlowCutSegment* when there is no need to get into the UCS level. When the pointcut type of a *UseCaseCutSegment* or *FlowCutSegment* is *PointcutType::All*, then there is no need to specify attribute *selectedUseCases* of metaclass *UseCaseCutSegment* or attribute *selectedSteps* of class *FlowCutSegment*. This constraint is formalized as the OCL expression attached to metaclass *UseCaseCutSegment* (Fig. 3). For example, as shown in Fig. 5, the use case pointcut *SelectedUseCases* consists of two *UseCaseCutSegment*s: one is to select all the use cases of the VCS system and the other is to select all the use cases of the EndPoint system. In these two *UseCaseCutSegment*s, no *FlowCutSegment* is specified.

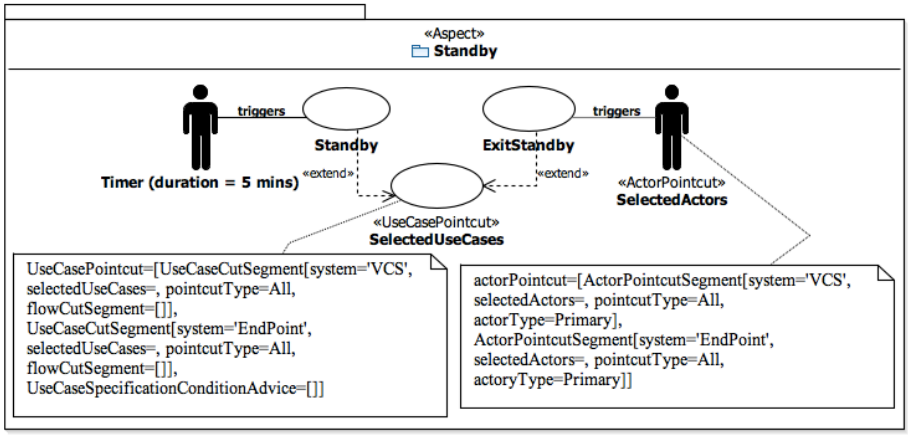

Fig. 5. Aspect use case diagram of *Standby*

If there is no UCS specified for a newly introduced use case in the aspect UCMod or if the aspect UCMod does not need to get into the level of UCSs, the use case is weaved into the base use case diagram through three different types of relationships of use case diagrams: *Extend*, *Include*, *Generalization*, which are explicitly captured in the use case diagram of the aspect UCMod. However, the use case pointcut should also specify the steps of the selected use cases (via use case pointcut) where the newly introduced use case should extend or be included. This is realized by *FlowCutSegment* and *UCSAdviceSpecification* of *UseCasePointcut*. Attributes *includeSpec* and *extendSpec* of class *UCSAdviceSpecification* specify two sentences: *INCLUDE USE CASE <name of the newly introduced use case>* and *EXTENDED BY USE CASE <name of the newly introduced use case>*. During weaving, these two sentences should be added before, after the selected steps of *FlowCutSegment*, or replace existing ones, through *Before*, *After* or *Around* advice. For the cases when the selected use cases extend or are included by a newly introduced use case in the aspect UCMod, the inclusion and extension points are however specified in the newly introduced use case and therefore no extra information is required in the point cut specification.

We specify a special type of advice *UseCaseSpecificationConditionAdvice*, with two sub-types: *PreconditonAdvice* and *PostconditionAdvice*, to introduce precondition and postcondition sentences to the selected UCSs. The introduced sentences can be weaved with the ones of the base UCSs in three different ways: *AND*, *OR* and *XOR*, which are defined as the enumeration *ConditionAdviceType* as shown in Fig. 3. As shown in Fig. 4, the use case pointcut has one *PreconditionAdvice* with condition "The system should be connected to network". This precondition sentence should be weaved to the preconditions of the UCSs of the use cases selected by the use case pointcut, via a conjunction, which is indicated by assigning "AND" to attribute *conditionAdviceType: ConditionAdviceType* of *UseCaseSpecificationConditionAdvice*.

Actor Pointcut. An actor pointcut selects one or more actors and consists of one or more *ActorPointcutSegment*s, which specify the system that the actor belongs to, the selected actors, and the pointcut type. In Fig. 5, actor *SelectedActor*s is stereotyped with

«ActorPointcut». The values of its attributes show that all the actors of the two systems are selected. Same as for *UseCaseSegment*, if an *ActorPointcutSegment* has *poincut-Type* as *PointcutType::All*, there is no need to specify *selectedActors*. Note that, there are two types of actors: *Primary* and *Secondary*, as shown in enumeration *ActorType*, which makes it easier to specify actor pointcut expressions. For instance, the actor pointcut selects all the primary actors of the base UCMod (Fig. 5).

3.3 Definition of Weaving Directive Specification

Each crosscutting concern is specified as a separate aspect UCMod. Aspect UCMods of multiple crosscutting concerns should be weaved into their corresponding base UCMod in a specific order to ensure that the woven UCMod is correct. To achieve this, an ordering must be defined and provided to the weaver as an input. However, UML use case diagram does not provide such a capability. We therefore choose to use the UML interaction overview diagram notations to specify such orderings, denoted as weaving-directive interaction overview diagrams. One example is given in Fig. 6.

UML interaction overview diagrams define interactions through a variant of Activity Diagrams, in a way that promotes overview of the control flow [2]. Weaving-directive interaction overview diagrams contain interaction uses representing and referencing to all aspect UCMods, ordered using UML activity diagram's flow control features such as decision, join, and fork. Of course, UML activity diagrams can equally perform the same functionality. Choosing UML interaction overview diagram notations instead of activity diagram notations is simply because the former is simpler than the later since interaction overview diagrams abstract away *Message*s and *Lifeline*s and therefore the approach would be easier to be accepted in practice.

A weaving-directive interaction overview diagram contains the following model elements: 1) An initial activity node; 2) A set of interaction uses, each of which refers to an aspect UCMod; and 3) A set of control flow edges that can be of any of the following two types: a control flow edge from the initial activity to an interaction use representing the first aspect UCMod to weave, and a set of control flow edges connecting interaction uses (e.g., decision, join and fork) to show the order in which the interaction uses (aspect UCMods) will be weaved into the base UCMod.

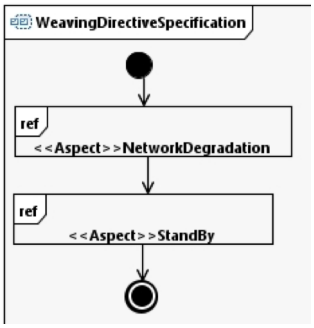

Fig. 6. Example of Weaving-directive Interaction Overview Diagram

3.4 Modeling guidelines

The AspectRUCM profile (Section 3.2) provides a notation to specify crosscutting concerns as aspect UCMods (Section 2.3). Before applying AspectRUCM, crosscutting concerns have to be first identified at the requirements level (activity *A1*), as shown in

Fig. 7. Different approaches (e.g., [15]) can be used for this purpose. Our AspectRUCM approach can be used in conjunction with these existing works. However, we do not discuss this further in this paper as it is out of the scope of this paper.

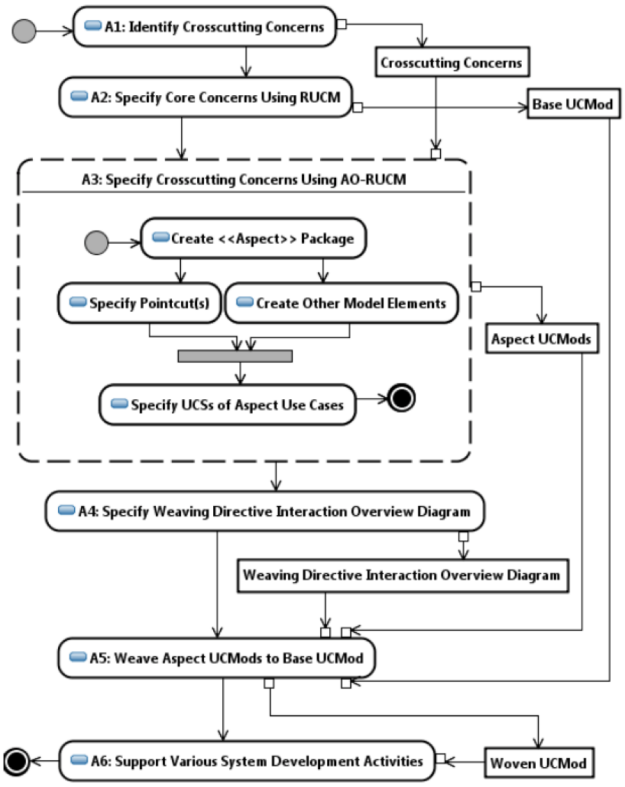

Fig. 7. Guidelines for Applying AspectRUCM

Core concerns of the system are specified using RUCM (activity *A2*), leading to the creation of *Base UCMod*. Followed by *A2*, *A3* specifies crosscutting concerns using the AspectRUCM profile, which includes sub-activities of creating a UML package stereotyped with «Aspect», then specifying pointcut(s) and creating other model elements of the use case diagram (e.g., actors, use cases), and finally specifying UCSs of the introduced use cases in the aspect UCMod using the RUCM template. The output of this activity is a set of *Aspect UCMods* created for each identified crosscutting concern. Activity *A4* specifies the weaving ordering and outputs the weaving directive interaction overview diagram.

Aspect UCMods (from *A3*) are weaved into their corresponding *Base UCMod* (from *A2*), based on the weaving ordering specified in the interaction overview diagram (from *A4*), to automatically generate a woven UCMod (A5), which can be used to facilitate automated analyses (*A6*) such as requirements analyses, automated creation of analysis and design models, and automated derivation of test cases. It is sometimes more effective to perform various requirements analyses (e.g., identifying and managing conflicts and tradeoffs among concerns [9]) based on the same woven UCMod, instead of separate aspect UCMods and the base UCMod. Based on [19], automated transition from the woven UCMod to different UML diagrams can be supported. If the derivation or generation of downstream artifacts (e.g., test cases) relies on the transformation from an AOM approach at the requirements level (e.g., AspectRUCM) to another AOM approach at the design or testing level (e.g., AspectSM [5]), there is no point to perform

weaving at the requirements level and hence activities *A4*, *A5* and *A6* are unnecessary.

3.5 Weaver

Aspect UCMods are weaved into their base UCMod by a weaver, which reads the base and aspect UCMods and the weaving-directive interaction overview diagram, and produces a woven UCMod. The pseudo code of the weaving algorithm is provided in Appendix A, Fig. 8. We developed a weaver for AspectRUCM using Java to weave one or more aspect UCMods into a base UCMod. Aspect UCMods are specified in AspectRUCM (Section 3.2). A base UCMod is modeled using RUCM.

Due to the reason that UCSs of both the base and the aspect UCMods are textual, all of them have to be formalized such that weaving can be performed. Therefore, the weaver has a formalization engine, which contains a set of transformation rules transforming textual UCMods into instances of UCMeta (Section 2.3). The aspect UCMods are formalized into instances of extended UCMeta with AspectRUCM while the base UCMod is transformed into an instance of UCMeta. UCMeta and its extension are implemented as an Ecore model, using Eclipse EMF [1]. We also use the Stanford Parser [3] as a NL parser for the transformation of textual sentences in UCSs to instances of UCMeta. The parser is written in Java and generates a syntactic parse tree for a sentence and the sentence's grammatical dependencies (e.g., subject, direct object). It is important to notice that it is not necessary to have the transformation from UML use case diagrams (either with or without the AspectRUCM profile applied) to instances of the *UML::UseCases* package of UCMeta (Section 2.3), as UCMeta directly imports the *UML::UseCases* package and the AspectRUCM profile.

The weaver takes the formalized aspect and base UCMods and the weaving-directive interaction overview diagram as inputs and generates a woven UCMod, which is an instance of UCMeta. The weaver is fully automated. The automatically generated woven UCMod can be used as an input for further analysis (e.g., automated requirements verification and validation) or generation (e.g., generating UML analysis models). Currently our approach and its weaver do not support modeling and weaving interactions that may occur between different aspects and will be investigated in the future.

4 Evaluation

Section 4.1 presents the three case studies, Section 4.2 discusses how AspectRUCM reduces modeling effort, and Section 4.3 summarizes evaluation results.

4.1 Industrial case studies

We used three case studies from the telecommunication domain, the maritime and energy sector and the aviation domain: VCS, Subsea Oil Production System (SOPS) and Navigation System (NAS). Table 3 presents the characteristics of their UCMods.

Table 3. Characteristics of Base and Aspect UCMods

System	# of Base Use Case	Total # of UCSs	# of Aspect UCMods	# of Actors
VCS	40	10	8	5
SOPS	65	12	6	9
NAS	46	11	7	9

VCS. VCS contains four systems/endpoints, which are of the same functionalities. These functionalities are modeled as the same set of use cases. Each endpoint has 10 use cases; in total the whole system contains 40 use cases. The core functionality of the systems manages the sending and receiving of multimedia streams. Audio and video signals are sent through separate channels and there is also a possibility of transmitting presentations in parallel with audio and video. Presentations can be sent by only one conference participant at a time and all others receive it. Each of the video conference system endpoint is operated by a human actor. A timer is needed to periodically initiate the adaption of call rate. Based on our previous work with Cisco [5], the following eight crosscutting concerns are specified using our aspect use case modeling approach:

1. *Network Degradation:* This crosscutting behavior is invoked whenever communication medium is experiencing different faulty situations in it such as duplicate and corrupt packets in communication.
2. *Standby:* The Standby behavior of a VCS becomes active when it is idle for a specific amount of time. When any activity is performed on the VCS while it is in Standby mode, it becomes active.
3. *Media Quality Recovery*: An important robustness behavior of a VCS is to recover from media (audio and video) quality loss. Whenever a VCS is in a video conference, it checks audio/video quality after every certain time. If the quality is within the threshold it continues the normal operation, otherwise it tries to recover audio/video quality. If it successfully recovered the audio quality it continues its normal operation, otherwise it restarts the VCS.
4. *Do Not Disturb*: Whenever the *Do Not Disturb* feature is on, a VCS ignores all incoming calls. If VCS is already in a call, it will remain in the call, but ignores any new incoming calls.
5. *Synchronization Mismatch*: Whenever there is a mismatch between audio and video, the proprietary algorithms are invoked to reduce the mismatch between audio and video.
6. *Intelligent Packet Loss Recovery:* Whenever, there is a packet loss in network during a videoconference, an algorithm is invoked to deal with packet loss.
7. *Echo Reduction:* Whenever, there is an echo in an audio during a video conference or an audio call, a VCS invokes echo reduction algorithms to deal with it.
8. *Noise Cancellation:* During a videoconference, a VCS may experience arbitrary noise, which is cancelled by invoking certain algorithms.

SOPS. SOPSs are large-scale, integrated, distributed, and highly configurable systems of systems for managing exploitation of oil and gas production fields, with various field layouts ranging from single satellite wells to large multiple sites (more than 50 wells). SOPS has four different types of systems, three of which are located above the sea level and the other is located in subsea. These systems have distinct functionalities and are connected through different types of communication media. Due to the reason that we had no access to all the requirements of these systems, we were not able to specify the UCSs of all the systems. Only 12 out of 65 representative use cases were

specified. We modeled the following six crosscutting concerns using AO-RUCM.

1. *Operation Mode Exchange*: It is possible to change the operation mode of a system from *Normal* to *Emulate*. A system has different, but might overlapped, set of functionalities for each operation mode.
2. *Backup Communication*: For safety critical systems such as SOPSs, when optical fibers are installed, an electrical communication system between the topside and the subsea shall be considered for an additional backup communication system. In principal, all functionalities shall be available in the backup communication mode. However in the situation with very limited bandwidth, some of these functionalities will not be usable in the backup communication mode and they should be disabled.
3. *Communication Timeout*: Topside control systems frequently wait for responses from subsea control systems. If the timeout expires, the request from the topside control systems to the subsea control systems is removed from the queue of unanswered requests.
4. *Runtime Configuration*: It is possible to configure some attributes of the control software deployed to systems when they are configured and running.
5. *Communication Bandwidth Limiting*: Bandwidth limitation could be important if bandwidth hungry and external Ethernet equipment is connected to control systems. The necessity of bandwidth limitations depends on the bandwidth of the communication infrastructure and whether slower secondary/backup communication lines are used. Such a communication bandwidth limiting has directly impact on various functionalities of multiple systems.
6. *Data Update Mechanism Switch*: Two different schemes for transferring data from subsea systems to topside systems can be either dynamically switched from one to the other according to the traffic status of communication links between topside and subsea, or manually switched by operators through a topside control device. The switching from one scheme to the other has impacts on various functionalities of the systems.

NAS. NAS [20] controls and guides an aircraft, based on control law computation that takes data sampled from sensors as input and sends commands to actuators. NAS has two operating modes: Auto mode and Manual mode and a pilot can switch the modes during flight. To ensure safe operation, NAS is fault tolerant with a redundant design. At the end of each clock cycle, redundant inputs from sensors are given to the autopilot system and multiple computation methods produce redundant outputs to be voted. There are many periodical tasks in the system and the system period is set as 20 milliseconds—the minimal one among the periods of all the tasks (with periods as 20ms, 40ms or 60ms). We specified the following seven crosscutting concerns using AspectRUCM: *System Synchronization*, *Flight Mode Exchange*, *Periodical Action*, *Data Monitoring*, *Data Voting*, *Fault Handling*, and *Communication Timeout*.

The first one is *Start System* and the second is *Power-up Built-in Test (PUBIT)* responsible for the built-in test when the system is powered on. Use case *Handle Faults* is designed to tackle any unexpected error during the operation of the system. *Sample Data* is responsible for obtaining all needed data via sensors.

1. *System Synchronization*: To achieve high reliability, NAS synchronizes two redundant systems at each period to conduct specific taks.
2. *Flight Mode Exchange*: It is possible to change the operation mode of a system from *Auto* to *Manual*. A system has different, but might overlapped, set of functionalities for each operation mode.

3. *Periodical Action*: There are many periodical tasks are conducted in each period, and some of them have the same extra requirements, for example, memory resource, data accuracy.
4. *Data Monitoring*: Redundant inputs from sensors are given to the system, therefore the system need to validate the data before using them.
5. *Data Voting*: The system is a redundant system and it means for each input/output data the system have different copies, therefore they system should vote the best one to use.
6. *Fault Handling*: It is responsible for capturing different exceptional events during the normal execution.
7. *Communication Timeout*: As two redundant systems need to conduct synchronous tasks, they should make sure the communication is conducted timely.

Notice that both VCS and SOPS have eight common network abnormal use cases since both of these systems employ the same type of Ethernet communication medium. VCS has two extra abnormal use cases, which are specific to video conferencing protocols, i.e., H323 and SIP.

4.2 AspectRUCM Evaluation

One way of evaluating if AspectRUCM reduces modeling effort is to estimate modeling effort through a surrogate measure such as the number of modeling elements required to be modeled. This number can then be compared in aspect UCMods and RUCM UCMods when modeling crosscutting concerns. Table 4 summarizes the modeling tasks involved when using and not using AspectRUCM for modeling three sets of crosscutting concerns from the three case studies. Note that we do not count modeling effort required to specify UCSs and the effort evaluation is at the level of use case diagrams.

For the VCS case study, we have eight crosscutting concerns, which are described in Section 4.1. When we used AspectRUCM to model these eight crosscutting concerns, we significantly reduced modeling effort for modeling relationships between use cases (95% (=420/440) on average, see Table 4). In other words, for all the eight crosscutting concerns together, we modeled 20 relationships when using AspectRUCM, whereas we need to model 440 relationships without using AspectRUCM. In terms of actors, using AspectRUCM we modeled 10 actors in all eight crosscutting concerns, whereas we modeled 8 actors without using AspectRUCM. For use cases, we modeled 19 use cases using AspectRUCM for all eight crosscutting concerns together, whereas we modeled 11 use cases when not using AspectRUCM. Considering that modeling effort for an actor, use case, and a relationship is roughly equal, for all eight crosscutting concerns together, we modeled 459 modeling elements without using AspectRUCM, whereas with AspectRUCM we modeled only 49 modeling elements. This means on average we saved 89% of modeling effort in our industrial case study.

Using AspectRUCM, we also needed to model pointcuts for all crosscutting concerns. In total, we modeled 10 pointcuts (Table 4) for VCS and modeling these pointcuts is the additional modeling effort required in AspectRUCM. In conclusion, modeling 10 pointcuts can save us modeling 410 modeling elements. We assume that the modeling effort of 10 pointcuts is less than modeling 410 modeling elements and

thus modeling effort using AspectRUCM can be reduced. For SOPS/NAS, we modeled 6/7 crosscutting concerns. Similar to VCS, as one can observe from Table 4, using AspectRUCM significantly reduced the modeling effort equivalent to 80%/82%.

Table 4. Evaluation Results of the Three Case Studies

Case Study	Cross-cutting con-cerns	Using AspectRUCM					Without AspectRUCM				Effort saved (%)
		UCs	Actors	Rels	Pointcut	To-tal	UCs	Ac-tors	Rels	To-tal	
VCS	1	2	1	2	1	6	1	1	40	42	86%
	2	3	2	4	2	11	2	1	80	83	87%
	3	3	1	2	1	7	2	1	80	83	92%
	4	3	2	4	1	10	2	1	80	83	88%
	5	2	1	2	1	6	1	1	40	42	86%
	6	2	1	2	1	6	1	1	40	42	86%
	7	2	1	2	1	6	1	1	40	42	86%
	8	2	1	2	1	6	1	1	40	42	86%
	Total	19	10	20	9	58	11	8	440	459	87%
SOPS	1	3	1	4	1	9	2	1	78	81	89%
	2	3	2	6	2	13	2	1	56	59	78%
	3	2	1	2	1	6	1	1	48	50	88%
	4	2	1	2	1	6	1	1	12	14	57%
	5	3	2	6	2	13	2	1	47	50	74%
	6	3	1	4	1	9	2	0	25	27	67%
	Total	16	8	24	8	56	10	5	266	281	80%
NAS	1	2	2	4	1	9	1	2	42	45	80%
	2	2	1	2	1	6	1	1	14	16	62%
	3	5	3	6	2	16	3	3	74	80	80%
	4	2	3	2	1	8	1	2	48	51	84%
	5	2	3	2	1	8	1	2	50	53	85%
	6	3	1	2	1	7	2	1	46	49	86%
	7	3	3	4	2	12	2	3	62	67	82%
	Total	19	16	22	9	66	11	14	336	361	82%

Overall, results on the three industrial case studies seem to suggest that the modeling effort can be significantly reduced when using AspectRUCM for modeling crosscutting concerns. Since using AspectRUCM requires the modeling of use case pointcuts and actor pointcuts with the «UseCasePointcut» and «ActorUseCasePoint» stereotypes, there will only be a benefit if modeling, more than 80% additional relationships on a UCMod is more time-consuming than modeling few pointcuts. Though this seems to be likely, it would need to be confirmed via controlled experiments involving human designers to determine the actual percentage of modeling effort saved when using AspectRUCM. In addition, modeling crosscutting concerns as aspect UCMods keeps the base UCMod less cluttered; hence, they are easier to read and maintain, and support reuse, as crosscutting concerns are modeled separately from the base ones.

4.3 Empirical Evaluation of RUCM

AspectRUCM extends use case diagrams, but has no extensions to the RUCM template and no new restrictions introduced. Hence, in terms of describing UCSs, AspectRUCM should be exactly the same as RUCM. In our previous work [18], we have conducted two controlled experiments to evaluate RUCM in terms of its applicability and impact on the quality of manually derived UML analysis models. Experiment results [18] show that RUCM is easy to apply and RUCM results into significant improvements over

traditional approaches in terms of the quality of derived class and sequence diagrams. These two controlled experiments particularly focus on the evaluation of the RUCM template and the restriction rules; therefore we can conclude that the evaluation results for RUCM are also applicable to AspectRUCM.

However, as discussed in Section 3.2, the AspectRUCM profile is introduced to extend the UML use case diagram notations and it should be evaluated to test its applicability and other benefits similar to other aspect-oriented modeling approaches such as enhanced separation of concerns, improved maintainability, reusability and understandability. In the future, we plan to conduct empirical studies for further evaluation.

5 Related Work

It is a common practice to follow a template to structure UCSs, thereby helping their reading and reviewing. Various templates (e.g., [6]) have been suggested to satisfy different application contexts and purposes. These templates share common fields such as: use case name, brief overall description, precondition, postcondition, basic flow, and alternative flows. The systematic review [17] we conducted to examine literature that transform textual requirements into analysis models revealed that six approaches require use cases (e.g., [14]). RUCM was built on the state of art.

An aspect-oriented use case modeling approach was proposed in [7] to connect advice use cases to base ones through a relationship stereotyped with a newly proposed stereotype «Aspect». A grammar is proposed to specify pointcut expressions based on wildcards in steps of flows of events of use cases. Four types of advices are specified: before, after, around, and concurrent. The approach does not directly introduce aspect to use case diagrams and therefore there is no graphical notation reused from use case diagrams or newly introduced. Aspect use cases are weaved with their corresponding base use cases into a petri net model, which is used as an input for further analysis. We however extend UML use case diagrams by reusing their inherent graphical notations with limited extensions via UML stereotypes.

Jacboson and Ng proposed an aspect-oriented use case modeling approach [10], by extending the meaning of extension points as joint points. With it, the base model has to be modified by inserting textual sentences of extension points directly to the UCSs of the Pointcut use cases of the base UCMod. If there are more than one Pointcut use cases (most probably the case in the context of AOM), more than one places of the UCSs of these Pointcut use cases have to be modified. This implies that this approach does not really separate aspects from their base. In addition, the approach has only one type of Advice: the extension behavior specified in an aspect use case as the whole.

Sillito et al [13] proposed a textual aspect language called AspectU, to support modularization of crosscutting concerns in UCMods. AspectU aspects are then transformed into AspectJ implementation. AspectU is purely textual and very similar to programming languages. However, AspectRUCM largely relies on the inherent graphical notations of UML use case diagrams. Therefore, in terms of usability, AspectRUCM should be easier to understand and apply for engineers, especially requirements engineers.

Mussbacher et al. [11] proposed an aspect-oriented requirements modeling approach

with use case maps. In this approach, advice and pointcut are both captured using the use case map inherit graphical notations. Several works on adding aspect concepts to goal models (e.g., [4]) have been also proposed. Other aspect oriented modeling approaches (e.g., [5]) have been proposed at different levels of abstraction of a software development lifecycle than UCMods.

6 Conclusion

Use case modeling is one of the most common practices for capturing functional requirements. However, use case specifications (UCSs) are essentially textual documents and therefore ambiguity is inevitably introduced. We proposed a general use case modeling approach, referred as RUCM [18], which has been empirically evaluated to be easy to apply. In this paper, we proposed an extension of RUCM, named as AspectRUCM, to systematically model crosscutting concerns at the level of use case models to alleviate its complexity especially for large-scale systems. AspectRUCM uses a UML profile to support the modeling of crosscutting concerns as aspects in use case diagrams and UCSs. Modeling crosscutting concerns is mandatory for networked applications since use case modeling of such systems entails the need to specify non-functional properties such as robustness behavior. Moreover, we performed and reported on three industrial case studies, which suggest that using AspectRUCM results in reducing on average more than 80% modeling effort.

Appendix A

As shown in Fig. 8, the algorithm first handles all use case pointcuts (Step 1) and then processes all actor pointcuts (Step 2) of an aspect UCMod. For each use case pointcut, the algorithm starts from identifying selected use cases specified in the pointcut expression by querying the base UCMod (Step 1-A), identifies model elements of the aspect UCMod that are newly introduced to the base UCMod (Step 1-B), and adds and connects these model elements to the selected use cases of the base UCMod (Steps 1-C and 1-D). The sub-steps of Step 1-E modify UCSs of the selected use cases according to their relationships with the newly-introduced use cases, which are clearly specified in the aspect UCMod. For each actor pointcut, the algorithm starts from identifying selected actors specified in the pointcut expression by querying the base UCMod (Step 2-A) and connects each selected actor to the newly-added use case (Step 2-B). Step 2-C handles the modification of UCSs.

WeaveUseCaseModel (b : UseCaseModel, a : UseCaseModel, w : UseCaseModel)
Inputs:
b: A formalized base UCMod, which is an instance of UCMeta
a: A formalized aspect UCMod, which is an instance of UCMeta extended with the AO-RUCM profile
Output: *w*: A woven UCMod, which is an instance of UCMeta
Algorithm:

1. For each use case uc stereotyped with «UseCasePointcut» in *b*, do
   A. Query the base UCMod *b*, based on the values of the attributes of «UseCasePointcut» and obtain a collection of use cases *SelectedUCs*.
   B. Query the aspect UCMod a, obtain a collection of model elements meles (either use cases or actors) that are not stereotyped with «UseCasePoincut» or «ActorPointcut» but are directly or indirectly connected to use case *uc*.
   C. Add meles to the woven UCMod and link them to each of SelectedUCs through the associations specified in the aspect UCMod *a*.
   D. If *uc* contains an instance of UseCaseSpecificationConditionAdvice, then introduce PreconditionAdvice and/or PostconditonAdvice to each of SelectedUCs, according to the specification of the advices.
   E. For each selected use case *suc* of *SelectedUCs*, do
      a. For each use case *auc* of *meles*, do
         i. If *auc* extends *suc* as specified in the aspect UCMod *a*, update the UCS of *suc* by
            a) Adding *auc* to the field of 'Extending Use Case', and
            b) Adding a *sentence* EXTENDED BY USE CASE <name of the *auc*> Before, After, or Around the selected steps specified in FlowCutSegment contained in «UseCasePointcut».
         ii. If *suc* extends *auc*, update the UCS of *auc* by
            a) Adding *suc* to the field of 'Extending Use Case', and
            b) Adding a sentence EXTENDED BY USE CASE <name of the *suc*> Before, After, or Around the selected steps specified in FlowCutSegment contained in «UseCasePointcut».
         iii. If *auc* includes *suc*, update the UCS of *auc* by
            a) Adding *suc* to the field of 'Included Use Case' and
            b) Adding a sentence INCLUDE USE CASE <name of the *auc*> Before, After, or Around the selected steps specified in FlowCutSegment contained in «UseCasePointcut».
         iv. If *suc* includes *auc*, update the UCS of *suc* by
            a) Adding *auc* to the field of 'Included Use Case' and
            b) Adding a sentence INCLUDE USE CASE <name of the *suc*> Before, After, or Around the selected steps specified in FlowCutSegment contained in «UseCasePointcut».
         v. If *auc* specializes *suc*, update the UCS of *suc* by adding *auc* to the field of 'Specialized Use Case'.
         vi. If *suc* specializes *auc*, update the UCS of *auc* by adding *suc* to the field of 'Specialized Use Case'.

2. For each actor *acr* stereotyped with «ActorPointcut» in *b*, do
   A. Query the base UCMod *b*, based on the values of the attributes of«Actor-Pointcut» and obtain a collection of actors *SelectedActors*.
   B. Connect each selected actor to the newly-added use cases to the woven UCMod according to what is specified in the aspect use case diagram.
   C. For each *actor* of *SelectedActors*, do
      a. If the actor is a primary actor, according to the information contained in the ActorPointcutSegment of «ActorPointcut», add this actor to the field of 'Primary Actor' of the UCSs of the use cases that are connected to the *actor* as specified in the aspect use case diagram.
      b. If the actor is a secondary actor, according to the information contained in the Actor−PointcutSegment of «ActorPointcut», add this actor to the field of 'Secondary Actor' of the UCSs of the use cases that are connected to the actor as specified in the aspect use case diagram.

Fig. 8. Weaving algorithm

References

1. Eclipse EMF. https://eclipse.org/modeling/emf/
2. OMG. UML2.2. http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/
3. The Stanford Parser version 1.6. http://nlp.stanford.edu/software/lex-parser.shtml
4. Alencar, F., Moreira, A., Castro, J., Silva, C., Mylopoulos, J.: Using Aspects to Simplify iModels. In: Requirements Engineering, 14th IEEE International Conference, pp. 335-336. IEEE, Minneapolis/St. Paul, MN (2006)
5. Ali, S., Briand, L.C., Hemmati, H.: Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. Software & Systems Modeling. 11 (4), 633-670 (2012)
6. Alistair, C.: Writing effective use cases. Addison-Wesley (2001)
7. Anthonysamy, P., Somé, S.S.: Aspect-oriented use case modeling for software product lines. In: EA-AOSD'08, pp. 5. ACM (2008)
8. Brown, E.K., Brown, K., Miller, J.: Syntax: a linguistic introduction to sentence structure. Psychology Press (1991)
9. Chitchyan, R., Rashid, A., Rayson, P., Waters, R.: Semantics-based composition for aspect-oriented requirements engineering. In: Proceedings of the 6th international conference on Aspect-oriented software development, pp. 36-48. ACM (2007)
10. Jacobson, I., Ng, P.-W.: Aspect-oriented software development with use cases (addison-wesley object technology series). Addison-Wesley Professional (2004)
11. Mussbacher, G., Amyot, D., Weiss, M.: Visualizing Aspect-Oriented Requirements Scenarios with Use Case Maps. In: REV'06. IEEE (2006)
12. Sampaio, A., Rashid, A., Chitchyan, R., Rayson, P.: EA-Miner: towards automation in aspect-oriented requirements engineering Transactions on aspect-oriented software development III. pp. 4-39 Springer (2007)

13. Sillito, J., Dutchyn, C., Eisenberg, A.D., De Volder, K.: Use case level pointcuts. In: Odersky, M. (Ed.) ECOOP 2004–Object-Oriented Programming. LNCS. vol. 3086. pp. 246-268 Springer (2004)
14. Somé, S.S.: Supporting use case based requirements engineering. Information and Software Technology. 48 (1), 43-58 (2006)
15. Sousa, G., Soares, S., Borba, P., Castro, J.: Separation of crosscutting concerns from requirements to design: Adapting the use case driven approach. In: Early Aspects, pp. 93-102 (2004)
16. Van Lamsweerde, A.: Requirements engineering: from system goals to UML models to software specifications. Wiley (2009)
17. Yue, T., Briand, L.C., Labiche, Y.: A systematic review of transformation approaches between user requirements and analysis models. Requirements Engineering. 16 (2), 75-99 (2011)
18. Yue, T., Briand, L.C., Labiche, Y.: Facilitating the transition from use case models to analysis models: Approach and experiments. TOSEM. 22 (1), No. 5 (2013)
19. Yue, T., Briand, L.C., Labiche, Y.: aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models. TOSEM. 24 (3), No. 13 (2015)
20. Zhang, H., Yue, T., Ali, S., Liu, C.: Facilitating Requirements Inspection with Search-Based Selection of Diverse Use Case Scenarios. In: BICT 2015 (In Press)