

# The DISCERN Method: Dealing Separately with Crosscutting Concerns

Lars Rosenhainer

Department of Computer Science, Chemnitz University of Technology, 09107 Chemnitz, Germany  
lars.rosenhainer@informatik.tu-chemnitz.de

## Abstract

*Nonfunctional requirements (NFRs) of software systems are typically crosscutting in nature. In this paper, a method is described how to identify NFRs of certain types in given documents, derive aspects from them and compose them with use cases. Further, an improved technique for specifying scenario-based pointcuts is presented. A webshop case study validates the approach's applicability.*

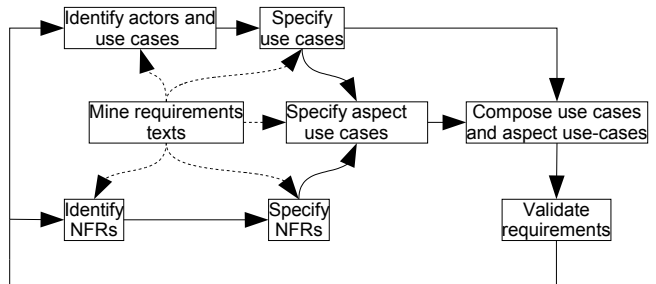


Figure 1. Overview of the DISCERN method

## 1 Introduction

Handling crosscutting concerns during requirements analysis is critical. It improves requirements traceability, facilitates easier assessment of change impact, supports requirements evolution, enables to apply aspect-orientation from the very start of the software lifecycle, and prevents easy overlooking of crosscutting influences. It has been shown (see e.g. [17, 13]) that many types of nonfunctional requirements (NFRs), such as response time and security, are crosscutting in nature. This paper will concentrate on them. In sec. 2, a method is described how to identify NFRs of certain types in existing requirements documents, derive use-case related aspects, and compose them with use cases. The method builds on some ideas stated in [13]. Sec. 3 presents a case study where the method is applied. In sec. 4, related work is discussed. Finally, sec. 5 contains some conclusions and remarks on future work.

## 2 The DISCERN method

To address crosscutting concerns separately during the requirements process, the DISCERN method (**D**ealing **S**eparately with **C**rosscutting **C**oncerns) as depicted in Fig. 1 is suggested. It is use-case based, iterative and resembles those proposed by e.g. [10, 19] but differs from or extends them at several places. The main contributions made include the following ones: support for mining crosscutting NFRs in requirements-related documents, support for collecting information from requirements documents to refine

use case descriptions so as to allow unambiguous definition of pointcuts, and more elaborate specification of pointcuts in use-case based aspects. The method steps are described in the following sections.

### 2.1 Functional requirements

#### 2.1.1 Identify actors and use cases

As many other methods, DISCERN captures a system's functional requirements by means of actors and their related use cases. Since there are many established methods for use case development, we will not go into detail here.

#### 2.1.2 Specify use cases

Use cases are specified in the form of scenarios as known from the literature (e.g. [7]). There is typically one main success scenario and a number of alternative scenarios which are called extensions here. As will be seen in the case study, specifying NFRs will often lead to a refinement of use cases since they must be unambiguous and explicit enough to allow a definition of pointcuts. Further, changes to use cases may have an impact on aspects.

### 2.2 Nonfunctional requirements

Most early-aspect approaches do not address the topic of mining aspects from textual documents such as require-

ments descriptions, although they belong to the most important information sources for system development and evolution. In contrast, DISCERN explicitly includes aspect mining from texts, which is mainly based on the application of NLP (Natural Language Processing) techniques for locating NFRs, from which aspects are derived.

### 2.2.1 Identify NFRs

In a text, it is in many cases trivial to locate statements representing, implying or eventually leading to NFRs. For example, if some statement clearly demands that ‘all system responses shall occur within 10 seconds’, it is obvious that it represents a response time (RT) requirement. In many other cases, however, it is not trivial. Domain knowledge and much experience as an analyst is necessary to identify NFRs not easily locatable on the text surface. So, for example, some knowledge in the area of toll collect systems is needed to conclude that a statement like ‘When an authorised vehicle passes through a green lane, a green light is turned on ...’ [6] implies an RT requirement since the statement does not include clear temporal references.

**Dealing with incomplete information.** Thus requirements analysis addresses two connected questions. First, *what information is necessary but currently incomplete to identify requirements?* Second, *where can be found additional information to identify requirements?* Locating vague or ambiguous statements in a requirements document will help in answering the first question. For example, if a sentence reads ‘changes to profiles must be reflected immediately’, an experienced analyst will ask: What does ‘immediately’ mean? Is there a hidden user request with respect to RT? Perhaps, this question may be answered from the analyst’s own domain and problem knowledge, or from finding more information on this topic in the text. If not, the client must be asked. Obviously, a tool that can highlight the ambiguous adverb ‘immediately’, relate it to the RT concern, and collect statements regarded as similar to the topic ‘profile change’, will help the less experienced analyst or the experienced one who overlooks something. Thus, an analyst is assisted in locating statements which might either give rise to new questions necessary for collecting missing information (which helps in answering the first question) or which contain part or all of the details needed for identifying NFRs (which helps in answering the second question).

**Locate statements on NFRs by textual patterns.** To identify statements related to certain NFR types in a text, DISCERN proposes the application of a reusable and extendable repository of textual patterns. Textual patterns may include regular expressions for keywords or key phrases

(e.g. /immediate/), certain grammatical structures in combination with keywords (e.g. a noun phrase containing both a cardinal number and a time unit) or morphological properties (e.g. certain endings). The usage of textual patterns based on grammatical structures requires a pre-processing step during which certain NLP techniques are applied to a document, which include part-of-speech (POS) tagging, sentence boundary disambiguation, lemmatizing, and parsing for deriving syntax trees. Except of lemmatizing for which WordNet [8] is currently employed, all these techniques follow a statistical approach [1, 18].

**Usage of textual patterns.** Textual patterns might, for example, be used in two different modes: first, in a running text, by configurably highlighting matches possibly related to certain NFR types, which can help in focusing the analyst’s attention and therefore in avoiding overlooking of certain text signals; second, as a filtering mechanism, which enables to just display the matches (and their contexts in a configurable way) that seem to be related to an NFR type of interest. Of course, the degree as to how definite these patterns point to a certain NFR type varies. Moreover, textual patterns may be related to more than one type. A tool’s certainty with respect to the relation of a matching text passage to certain types might, however, be visually signaled in terms of color or numbers on a certain scale if the patterns have appropriately been qualified. This classification can statistically be derived from experiments that assess the patterns’ recall and precision.

### 2.2.2 Specify NFRs

If aspects are to be derived from NFRs, DISCERN demands the NFRs to state unambiguously and in a testable way what constraint or behavior they require and which functional requirements they impact. This might for example be done by using templates as suggested by [10].

**The *What?* and the *Where?* of NFRs.** Interestingly, NFRs of certain types such as security, response time or auditability are typically written in a way not unsimilar to aspect orientation since they ‘encapsulate’ statements as to the *What?* (what is the required constraint or behavior) and *Where?* (which functional requirements are impacted). Consider for instance the RT requirement from the example in section 3: ‘Creating orders < 5 seconds’. It says *what* (< 5 seconds) applies *where* (creating orders). Thus it is reasonable that aspect derivation starts with specifying an NFR’s *What?* and clarifying its *Where?* (i.e. collecting the impacted requirements). However, there is a difficulty. Since we must deal with natural language, the requirements impacted by an NFR are often not easily locatable. Even if there are statements on the *Where?*, they may

be more or less specific or general. And even if we find the impacted requirements, they may not be stated exactly or detailed enough so as to serve as ‘joinpoints’ where the *What?* applies. In our example, the *What?* should apply to certain steps in the *Purchase pets* use case. However, as will be seen later, the requirements document is not clear about what is actually meant by ‘creating orders’.

**Clarifying the *Where?* of NFRs.** Thus it is desirable to get support in locating information on the behavior impacted by an NFR. Support may for example be offered by searching for statements containing one or more keywords drawn from partial knowledge on an NFR’s *Where?* and *What?* or similar words (synonyms, hypernyms, hyperonyms or other semantically related words) by means of thesauri. To this end, DISCERN currently provides for the application of WordNet [8]. Locating relevant information will eventually help in identification and, possibly, necessary refinement of functional requirement descriptions so as to derive unambiguous pointcuts from them. Since DISCERN is use-case based, pointcuts capture scenario steps or chunks of scenario steps.

### 2.3 Specify aspect use cases

**When to create an aspect?** Several authors (e.g. [10, 19]) suggest that an NFR be classified as crosscutting or a candidate aspect when it has an impact on more than one functional requirement (possibly embodied in the form of e.g. a use case). Such classification, however, may become obsolete when requirements are added or modified. Moreover, many, if not most, NFR types are known to be of a crosscutting character. A decision on whether to create aspects or not should therefore not primarily rely on the number of influenced requirements but on available knowledge on crosscutting NFR types, which may be as rudimentary as a simple list of evaluated NFR types or as elaborate as a body of reusable knowledge, possibly in a similar way as exemplified by the NFR framework [5].

**Specification of aspects.** An *aspect use case* is a use case with the particularity of functioning as an aspect. It encapsulates a scenario graph specifying behavior and crosscutting the scenario graphs of one or more non-aspect and/or aspect use cases. It is derived from specified NFRs<sup>1</sup> by operationalizing them. DISCERN uses a terminology similar to AspectJ. A *joinpoint* is an identifiable point in use case specifications. It may be a scenario step or a chunk of consecutive scenario steps with particular properties. A *pointcut* formalizes an NFR’s *Where?* part by capturing a set of

<sup>1</sup>It is also possible to derive aspect use cases from crosscutting functional requirements. In this paper, this is not considered though.

joinpoints. Finally, a *composition rule* is similar to an advice in AspectJ. It formally defines how a joinpoint is composed with a scenario graph derived from an NFR’s *What?* part. The way how elements are composed is controlled by composition rule operators as similarly suggested by [10], which enable the insertion of a scenario graph (a) before or after a joinpoint (*before* and *after*), meaning that the behavior  $B_G$  specified by the graph is inserted before resp. after the behavior  $B_J$  specified by the joinpoint, (b) instead of a joinpoint (*override*), meaning that  $B_G$  replaces  $B_J$ , and (c) around a joinpoint (*wrap*), meaning that  $B_G$  wraps  $B_J$ .

In this paper, the syntax of pointcuts will only be explained by examples. The following pointcut is named ‘onlyLibrarian’ and captures all the scenario steps (step joinpoints) starting with scenario step 1 and following it until step 5 in the use cases ‘borrow book’ and ‘return book’:

<b>pointcut</b>	onlyLibrarian:
<b>steps</b>	<b>from step 1 to step 5</b> <b>in use cases</b> (borrow book, return book)

Scenario steps may either be referenced by their number or an ID. Use cases and agents may either be referenced by names, regular expressions or IDs. An agent is the initiator of a scenario step. It may either be an entity external to and interacting with the system (i.e. an actor) or the system (or a part of it) itself. It is possible to distinguish between actor and system actions by identifying an agent’s type (actor or system (part)). Often, the type will be clear from the agents’ naming. If not, it may for example be derived from a use-case diagram (see e.g. Fig. 2).

It is observable that many scenario descriptions contain a set of several consecutive steps with a number of identical properties (e.g. the agent of all these steps is the system to be developed). Since we may not want to insert some aspect’s scenario graph, say, before or after each of these steps but only before the first resp. after the last of them, so-called *chunk joinpoints* are introduced. In the following example, the ‘searchings’ pointcut captures all chunks of scenario steps (chunk joinpoints) in the use case ‘search book’ where ‘System’ is the agent:

<b>pointcut</b>	searchings:
<b>chunks</b>	<b>in use cases</b> (search book) <b>with agents</b> (System)

A chunk joinpoint is formed by one or more consecutive steps that have a number of identical properties (e.g. identical agent). Thus it is possible to refer to this aggregation of steps as a single joinpoint.

So far, pointcuts may also be combined by conjunction (**and**) or disjunction (**or**) operators thus forming more complex pointcut definitions.

### 2.4 Compose and validate

Non-aspect and aspect use cases may be composed as defined by the composition rules in order to validate the speci-

fication. Before composition, conflicts must be detected and resolved. DISCERN enables automatic recognition of conflicts between different aspects. Two aspects are in conflict to each other, iff a composition rule of one aspect affects at least one joinpoint whose (a) first step is also the first of a joinpoint affected by a rule of the other aspect and the rules' operators are either *before*, *wrap* or *override*, or (b) whose last step is also the last of a joinpoint affected by a rule of the other aspect and the rules' operators are either *after*, *wrap* or *override*. Conflicts may be resolved by assigning priorities to aspects. This can visually be depicted in a use case diagram using a new relationship called 'precede' (which is inspired by AspectJ's aspect precedence rules). Aspects with higher priority precede those with lower priority. The higher the priority, the later the aspect is applied during the composition process.

## 2.5 Tool support

A prototype is being developed, which can support the following tasks at the time of this writing: preprocessing texts by certain NLP techniques (POS tagging, sentence boundary disambiguation, lemmatizing, parsing for deriving syntax trees); collecting textual patterns and allocating them to specific concerns, which is organized as a repository; searching for textual patterns in a text, displaying the text matches, showing their contexts in a configurable way; classifying text segments by assigning them to certain concerns; and searching for text segments containing similar words, which is based on the application of WordNet [8].

## 3 Case study: The Pet Store application

Suggested as common example for the workshop, the Pet Store webshop application serves to validate the approach's applicability. Basically, customers may browse the webshop's product selection and purchase pets by placing corresponding orders. Our analysis is based on the Pet Store requirements document (PSRD, v. 1.1.3).<sup>2</sup>

### 3.1 Functional requirements

**Identify actors and use cases.** We can identify two primary actors, which are *Customer* and *Customer service representative (rep)* as well as one secondary actor, namely *Credit Authorizer*. They make use of the following use cases (see Fig. 2):

*Browse products:* A *customer* may browse the webshop's selection of products, which are pets.

*Purchase pets:* A *customer* may purchase one or more

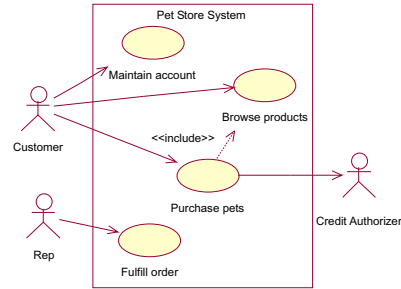


Figure 2. Pet Store's actors and use cases

pets via the webshop, which needs to contact the *credit authorizer* for placing a customer's order.

*Fulfill order:* A *rep* fulfills a customer's order by shipping the requested pets to the destination given in it.

*Maintain account:* A *customer* must create an account to be able to place orders. The account information may also be updated.

**Specify use cases.** Use cases are specified by scenarios. Tables 1 and 3 contain two exemplary scenario extracts, derived from PSRD use case descriptions and refined during NFR specification (see sections 3.2 and 3.3).

### 3.2 Dealing with response time

**Identify NFRs.** The PSRD is already well structured with respect to response time (RT) requirements. Thus it is not difficult to find them in the document. If necessary, a repository of textual patterns related to RT could be employed. For example, a search for matches of the pattern 'cardinal number followed by *second(s)* or *minute(s)*'<sup>3</sup> would easily lead us to the RT requirements on page 36 of the document. Further, a search for sentences including adjectives or adverbs such as 'quick(ly)' or 'immediate(ly)', which might point to some vaguely expressed RT requirements or goals, returns one match, which is not related to RT though. In the following, we will concentrate on one of the identified RT requirements: *Creating orders < 5 seconds*.

**Specify NFRs.** Specifying an NFR will transform it into a testable one. With respect to RT, this means that the requirement will become quantifiable. This helps to determine its satisfaction during runtime, by measuring the response times of implemented system functions constrained by it. The NFR *Creating orders < 5 seconds* is already written in a quantifiable form. For deriving an aspect, the NFR's *What?* and *Where?* must be stated. The result can be seen in the following table:

<sup>2</sup><http://www-128.ibm.com/developerworks/rational/library/1072.html>

<sup>3</sup>For simplicity, patterns are paraphrased here by natural language.

What?	Where?
< 5 seconds	Creating orders

Pointcuts are derived from the NFR’s *Where?* part. To be highly specific, the constrained scenario steps must first be identified. Since they embody the relevant joinpoints, they must later unambiguously be referenced by a pointcut. At the beginning, we constrain the search to the use case descriptions (PSRD section 3.2) since we assume that the NFR refers to some behavior described there. However, we do not find any sentence containing both ‘create’ and ‘order’<sup>4</sup>. As we do not find exact matches of the *creating orders* functionality in the use case descriptions, two things must be done:

- Extend the search space to the whole text to locate relevant information on the functionality. If the found information is incomplete (i.e. it is not enough to arrive at a clear decision), the search query may be modified, possibly supported by a tool (e.g. search for similar words based on WordNet [8]). If this still is not enough, the text must be read or other sources (e.g. customer, other documents) must be consulted until an answer has been found.
- Refine the use case descriptions so as to be usable for the derivation of pointcuts.

Reading the contexts of sentences containing both ‘create’ and ‘order’, we get clearer understanding of what is meant by the *creating orders* phrase. Apparently, it is part of the *Purchase pets* use case, namely the process starting from the point where a customer indicates his wish to order the goods in his cart till the one where he receives confirmation of his order being accepted. However, PSRD section 3.3 may suggest that the phrase very specifically only applies to some successive steps in this process, namely the ones in which an authorization code for the credit card transaction is obtained and a new order actually is created.

Anyway, we are now able to refine the *Purchase pets* use case by including main and alternative scenarios. For example, we may add steps to the main scenario as shown in Tab. 1. However, since it is still impossible to tell whether the RT requirement constrains every step in the interactive process leading to a new order or just some of them, using WordNet we modify our search so as to employ similar words of ‘order’ and ‘create’ that occur together in a sentence. An analysis of the search results, however, shows that they cannot answer the question. This, being the worst case, necessitates studying the whole document, which does not yield a satisfactory result either. Not being able to ask a customer, we simply assume here that the RT requirement constrains all steps in the process. Further, we cannot find

<sup>4</sup>Searching for the search words’ inflected forms is not necessary since the text’s words are lemmatized in a pre-processing step.

...
3. <i>Customer</i> indicates the wish to purchase the items in the customer’s cart. (SS-002000)
4. <i>System</i> presents a view to enter credit card information and billing address.
5. <i>Customer</i> enters credit card information and billing address.
...
8. <i>System</i> obtains transaction authorization code from credit authorizer for amount of the order.
9. <i>System</i> creates order and appends it to the unfulfilled order work queue.
10. <i>System</i> confirms created order to the customer.

**Table 1. Exemplary extract of main success scenario of refined *Purchase pets* use case**

out what to do in case the requirement is violated during runtime. This should definitely be stated in specifications for real-time systems. For the Pet Store application, however, a violation is not critical. So we simply log it.

**Derive aspect use cases.** Now, we are in a position where it is possible to unambiguously specify pointcuts and finally derive an aspect. Since RT requirements belong to NFR types which cannot directly be translated into functionality, we must find another way to operationalize them. One way is the creation of a monitor which can detect and handle violations to RT requirements during runtime, or, by defining appropriate time thresholds, detect imminent violations in sufficient time to apply adequate remedies. The result of creating an aspect called *Monitor RT of order creation* is depicted in Tab. 2. The pointcut *order creation* captures chunk joinpoints starting with scenario step SS-002000 and following it in any scenario of the *Purchase pets* use case, and for which the agent *System* is responsible. The wrap composition rule related to the *order creation* pointcut operationalizes the RT requirement (by creating a monitor) and defines what to do in case it is violated. The numbering of scenario steps must be interpreted as relative. After complete scenarios have been composed, the steps can be renumbered so as to provide for the desired step sequence.

### 3.3 Dealing with security

**Identify NFRs.** The PSRD is also well structured with respect to security requirements. So we can easily find them in the document. Much the same as above may be said of the support we could get from using a repository of textual patterns. A search for matches of the pattern ‘sentences containing *sign* followed anywhere by *in*’, for example, returns 11 occurrences in the text and would lead us to the following requirement in PSRD section 5.5.1.2, which we

<b>Aspect UC Monitor RT of order creation</b>
<b>pointcut</b> order creation: <b>chunks from step with id SS-002000 in use cases (Purchase pets) with agents (System)</b>
<b>wrap</b> order creation: 1. <i>System</i> measures current time and saves it as $t_{start}$ — wrapped — 2. <i>System</i> measures current time and saves it as $t_{end}$ 3. <i>System</i> determines that constraint $\{t_{end} - t_{start} < 5s\}$ is satisfied <b>Extensions:</b> 3a. System detects constraint violation: 3a1. <i>System</i> logs the constraint violation.

**Table 2. Monitor RT of order creation aspect**

...
<b>Main success scenario:</b> 1. <i>Customer</i> selects “register account”. ...
<b>Extensions:</b> 1a. <i>Customer</i> selects “update account”. 1a1. <i>System</i> presents a view with the customer’s address, preferences and ... (SS-002500) ...

**Table 3. Exemplary extract of refined Maintain account use case**

will concentrate on in this example: *MaintainOrderBilling*, *MaintainOrderShipping* ... for signed-in users only.

**Specify NFRs.** Again, the *For signed-in users only* requirement is already specified in a testable form, so first let us state its *What?* and *Where?*

What?	Where?
for signed-in users only	MaintainOrderBilling, MaintainOrderShipping, ConfirmAddresses, OrderConfirmed, SignInWelcome, MaintainAccount, ConfirmAccount-Saved

We again constrain the search to the use case descriptions (PSRD section 3.2) and basically apply the same procedure for identifying joinpoints as with respect to RT. This time, however, it is not necessary to modify the main scenario of the *Purchase pets* use case. Changing joinpoints referenced by existing pointcuts would of course necessitate checking whether an adaptation of those pointcuts is necessary. In our example, it is not. Although ‘MaintainAccount’ nearly perfectly matches a use case’s title, the statements available there are not sufficient for deriving a pointcut. Thus, analogously as described above, further information must be collected and, based on it, the use case must be refined. Tab. 3 shows an exemplary extract from it.

<b>Aspect UC Sign-In</b>
<b>pointcut</b> for signed-in users only: <b>chunks from step with id SS-002000 in use cases (Purchase pets) with agents (System)</b> <b>or</b> <b>chunks from step with id SS-002500 in use cases (Maintain account) with agents (System)</b>
<b>before</b> for signed-in users only: 1. <i>System</i> validates that customer is signed in. <b>Extensions:</b> 1a. Customer is not signed in: 1a1. <i>System</i> presents a view to sign in or register. 1a2. <i>Customer</i> signs in with Email ID and password. 1a2a. Customer wants to register: 1a2a1. <i>Customer</i> registers an account (UC-010000 Maintain Account). 1a3. Continue at step 1.

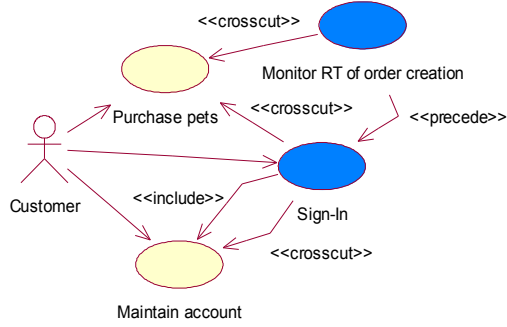
**Table 4. Sign-In aspect (simplified)**

**Derive aspect use cases.** We may now create aspect *Sign-In*, which is shown in Tab. 4. The pointcut *for signed-in users only* captures chunks of scenario steps in pretty the same way as the *order creation* pointcut does, with the difference that it additionally captures chunks from the *Maintain account* use case. The corresponding before composition rule adds the behavior of checking whether a customer has signed in. The case that s/he has not signed in is addressed by an extension scenario.

### 3.4 Composition

Our example reveals a conflict between the aspects *Monitor RT of order creation* and *Sign-In*, since there is at least one scenario step (e.g. step 4 in Tab. 1) for which one of the necessary conflict conditions from Sec. 2.4 holds. The *Sign-In* aspect adds necessary behavior to that of *order creation*. Thus the RT aspect must have higher priority since we do not want to measure the response time of incomplete but of the complete behavior. Fig. 3 shows the relationships among non-aspect and aspect use cases by means of a use case diagram. The new ‘crosscut’ relationship is inspired by [16]. It specifies that a use case’s behavior is crosscut by that of an aspect. It is justified in more detail in sec. 5. The ‘precede’ relationship defines the RT aspect’s higher priority.

Figures 4a-e informally illustrate the composition process by means of activity diagrams representing partial or complete scenario graphs. An activity represents a scenario step. Its name is formed according to the following schema: <mnemonic for use case>.<C|S><number of scenario step>, where C and S represent actions of the Customer and the System, respectively. Further, decision nodes represent branchings of the scenario graph. Except



**Figure 3. Relationships among non-aspect use cases (light) and aspect use cases (dark)**

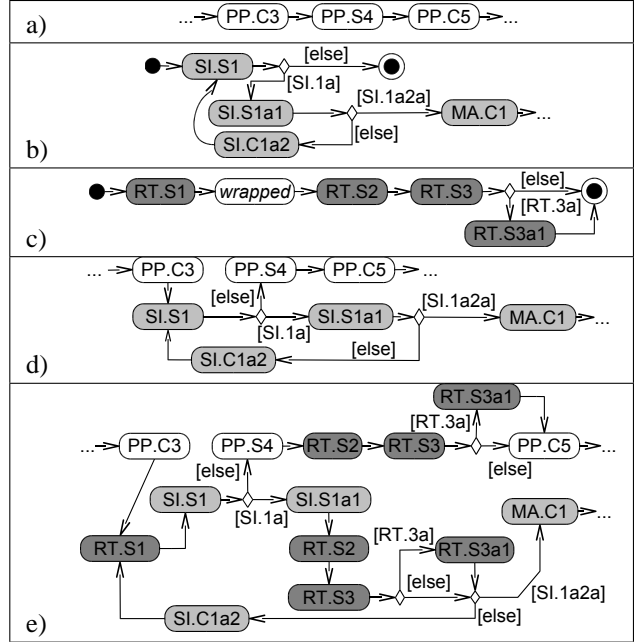
of the C/S distinction, their guards are named similarly as activities. An extract of the scenario graph of the *Purchase pets* (PP) use case (see Tab. 1) is shown in Fig. 4a. Fig. 4b contains the scenario graph of the *Sign-In* (SI) aspect use case (see Tab. 4). SI includes the *Maintain account* (MA) use case, which is, however, just hinted to by activity MA.C1 in this example. The scenario graph of aspect use case *Monitor RT of order creation* (RT, see Tab. 2) is contained in Fig. 4c. The diagram includes an additional activity<sup>5</sup> called *wrapped* which serves as a placeholder for the joinpoints to be wrapped. Fig. 4d presents the result of the composition of PP with SI. Since PP.S4 is a joinpoint, SI’s scenario graph is inserted before it. The final result is shown in Fig. 4e, which contains the composition of the intermediate product (PP and SI) with RT. The two step sequences SI.S1 and PP.S4 as well as SI.S1 and SI.S1a1 of the intermediate product each form a chunk joinpoint which are wrapped by RT’s scenario graph.

Composition enables validation of the specification. For example, it may now be determined where to draw the line: Does the RT requirement also include scenario steps of the *Maintain account* use case since registering an account may be needed for order creation? In case they are included, does this lead to conflicts with other RT requirements? Modifying aspects and successive composition thus offer a simple way to evaluate different design alternatives and, possibly, to further refine NFRs.

## 4 Related work

Several methods (e.g. [12, 10, 19, 4, 9]) have been suggested for addressing ‘early aspects’. As to the relation between use cases and aspects, there are a number of approaches that are similar to the DISCERN method. Differences to this work lie, for example, in the way how points (‘joinpoints’) are selected where composition rules apply:

<sup>5</sup>Idea inspired by [20].



**Figure 4. Informal visualization of composition process. For explanation see sec. 3.4.**

identification of sequence diagram elements [10, 19]; selection of single scenario steps by number [16]; definition of extension points within use cases so as to prepare the insertion of additional behavior [9]. An extension point is similar to the usage of scenario step identifiers by DISCERN. However, DISCERN also provides for referencing joinpoints by other means (step numbers, involved agents). To the best of the author’s knowledge, pointcut specifications for selecting joinpoints by involved agents and for aggregating consecutive scenario steps (with a number of identical properties) into a single joinpoint, is a novel contribution to the way scenario-based pointcuts can be defined.

With the exception of the well-known Theme/Doc [4] method, none of the above approaches supports aspect mining from textual documents, which is an important ingredient of DISCERN. In Theme/Doc, a keyword list must manually be set up, which is then used for identifying relationships among ‘actions’ (behaviors in requirements) and deriving a model from it. Candidate aspects are then determined based on the number of relationships actions have to each other. Another similar approach is suggested in [14], namely a method for aspect mining from textual documents, which is based on WMATRIX [15], a framework for the application of several probabilistic NLP techniques (POS tagging, regular expressions, concordance creation, and semantic analyses based on a normative corpus). Supported by it, documents are analyzed to identify concerns and viewpoints. From these intermediate results, a mining

tool assisted by WMATRIX derives a model representing relationships between requirements, and determines candidate aspects based on the degree of dispersion among the requirements. In contrast to both methods, DISCERN uses a textual pattern repository for locating NFRs and available knowledge on NFR types for determining their aspectual character. Moreover, searches for relationships among requirements are additionally supported by WordNet.

## 5 Conclusions and future work

The DISCERN method has been suggested, which combines aspect mining from requirements documents with the ensuing creation of aspect use cases and their composition with non-aspect use cases. A novel way for specifying pointcuts has been presented, which allows to capture scenario joinpoints in a more elaborate way than before. Further, the application of the method to a case study basically confirms its usefulness. There are a number of open issues which we anticipate to address in the future: extending the textual patterns repository and assessing the effectiveness of textual patterns in terms of precision and recall, supporting the identification of NFRs by checklists (such as questions to be answered), and mapping of aspect use cases to analysis models such as activity diagrams. All this should of course be accompanied by further evaluation and, possibly, refinement of the DISCERN method, based on applications to other case studies and by improving its tool support.

Finally, we will discuss some points raised with respect to the method.

**Why an extra *crosscut* relationship beside *include* and *extend*?** According to UML 2.0 [11], an *include* relationship between use cases implies that ‘the behavior of the included use case is inserted into the behavior of the including use case.’ Also, an ‘included use case is not optional, and is always required for the including use case to execute correctly.’ Thus it is obvious that an including use case must know the included use case. If we want use cases to be oblivious to aspect use cases, it is therefore clear that we cannot use *include* relationships to introduce behavior of the latter into that of the former. An *extend* relationship ‘specifies that the behavior of a use case may be extended by the behavior of another . . . use case’ [11]. Here, in contrast, the extended (base) use case does not know the extending use case. The *extend* relationship thus seems to suggest itself to be used for crosscutting behavior, which is also recommended by e.g. [9]. However, an extension is always bound to one or more extension points pre-defined in the extended use case [11]. This is a problem for example pointed to by Cockburn [7] when talking about *extend* relations as a way to extend use cases in locked requirements documents. If new extension points are necessary, the use case to be

extended must be changed, which is not permitted though. Further, a pointcut in an aspect use case is able to capture scenario steps not only by means of IDs (which are very similar to extension points since they also identify locations in the base use case and are defined there) but by means of step properties (e.g. who/what is the agent) as well. Obviously, the *extend* relationship is limited in its usefulness with respect to previously unknown behavior crosscutting that of a base use case. Consequently, a more general relationship between base and aspect use cases is needed, one which does not necessarily demand extension points to be present. It is called *crosscut* and may be thought as subsuming the *extend* relation.

**Are the pointcut definitions too formal for the level of use case specifications?** Basically, pointcut definitions refer to use cases, agents and scenario steps and are therefore on the same abstraction level. It is likely, however, that a client does not understand which scenario steps are selected by a pointcut. If understanding is desired by the client, s/he should be assisted by the analyst. Moreover, it is not even necessary for a client to see and understand the pointcut definitions. Instead, it should be the analysts’ and developers’ duty alone. For requirements validation and elicitation, it suffices if the client sees the composed use cases and scenarios. But, to facilitate precise compositions and thus detailed scenarios, it is not enough just to say that an aspect impacts *most* or *some* scenario steps but to say exactly which steps are impacted. This can of course only be bought at the price of a certain degree of formality which, however, does not seem to be too high.

**Is it reasonable to use IDs for reference purposes in pointcuts?** For reference purposes, pointcuts make use of IDs, which may either be automatically assigned to all scenario steps etc. or inserted by an analyst when an aspect is developed. The usage of IDs is common practice for establishing traceability in requirements documents as it avoids the usage of text such as names for reference purposes. Texts are often changed, so referencing them will effectively lead to maintenance problems. The case of minor changes to referenced scenario step descriptions, for example, might necessitate an otherwise avoidable modification of pointcuts.

**Evolvability.** Here, we only consider the particularities related to aspect use cases and leave aside the respective standard issues of use case approaches (e.g. with respect to *include* and *extend* relations). When a non-aspect use case is added, it becomes necessary to check all aspect use cases whether and how they affect the new one and to adapt the affective ones by extending their composition rules’ scopes to the new use case. When an aspect use case is added, it

is additionally needed to find out whether and how it affects all other non-aspect and aspect use cases. The affected ones only need refining if they do not yet allow the definition of unambiguous pointcuts (which, however, is unlikely when they have sufficiently been detailed). Further, conflicts with aspect use cases impacting the same use cases may arise and must be handled. Similarly to additions, when a non-aspect or aspect use case is changed, all aspect use cases crosscutting it must be dealt with. As a benefit from aspect orientation, it is very unlikely that changes to aspect use cases entail changes to impacted use cases. However, renewed conflict resolution among aspects impacting the same use cases may be needed. Altogether, these measures appear to be natural when dealing with the interactions among non-crosscutting and crosscutting requirements.

**Scalability.** With respect to composition visualization, we again confine ourselves to the particularities related to aspect use cases. Since aspect use cases may crosscut many other use cases, inclusion of all their *crosscut* relationships into a use-case diagram (UCD) will lead to cluttering of the diagram and thus reduce the scalability of visualizations. Still, it is possible to alleviate this problem by only displaying single use cases and their context (i.e. related actors, non-aspect and aspect use cases) in a UCD. Much the same can be said about diagrams visualizing a composed scenario graph (e.g. activity diagrams). Often, compositions with one non-trivial aspect or just a few trivial ones will already lead to graph explosion. As a remedy, only paths through the scenario graph (i.e. scenarios) might be displayed (which may even be done step by step during scenario validation).

Further, in the respect of analyzing requirements documents, DISCERN seems to foster scalability. For example, textual patterns related to certain concerns help in identifying the latter which otherwise might be overlooked in voluminous documents.

## Acknowledgements

Thanks to the anonymous reviewers for their helpful comments on a previous version of this paper. For example, the points discussed in Sec. 5 have been extracted from them.

## References

- [1] OpenNLP Maxent project. <http://maxent.sourceforge.net/>.
- [2] *Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design, Wsh., Boston, MA, USA*, 17 Mar. 2003.
- [3] J. Araújo and P. Coutinho. Identifying Aspectual Use Cases Using a Viewpoint-Oriented Requirements Method. In [2].
- [4] E. Baniassad and S. Clarke. Theme: An Approach for Aspect-Oriented Analysis and Design. In *Proc. 26th Int. Conf. on Software Engineering (ICSE'04)*, pages 158–167, 2004.
- [5] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.
- [6] R. G. Clark and A. M. D. Moreira. Constructing formal specifications from informal requirements. In *Step '97: Proc. 8th Int. Wsh. on Software Technology and Engineering Practice (STEP '97) (including case '97)*, pages 68–75, Washington, DC, USA, 1997. IEEE Computer Society.
- [7] A. Cockburn. *Writing Effective Use Cases*. The Agile Software Development Series. Addison-Wesley, Boston et.al., 2001.
- [8] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication. The MIT Press, Cambridge, Massachusetts et.al., 1998.
- [9] I. Jacobson and P.-W. Ng. *Aspect-Oriented Software Development with Use Cases*. The Addison-Wesley Object Technology Series. Addison-Wesley, Upper Saddle River, NJ et.al., 2005.
- [10] A. Moreira, J. Araújo, and I. Brito. Crosscutting Quality Attributes for Requirements Engineering. In *Proc. 14th Int. Conf. on Software Engineering and Knowledge Engineering*, pages 167–174. ACM Press, 2002.
- [11] Object Management Group. *UML Superstructure Specification, v2.0*. Aug. 2005. OMG document formal/05-07-04.
- [12] A. Rashid, P. Sawyer, A. Moreira, and J. Araújo. Early Aspects: A Model for Aspect-Oriented Requirements Engineering. In *Proc. of IEEE Joint Int. Conf. on Requirements Engineering (RE 2002)*, pages 199–202. IEEE Computer Society, 2002.
- [13] L. Rosenhainer. Identifying Crosscutting Concerns in Requirements Specifications. In *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Wsh. at OOPSLA 2004, Vancouver, Canada*, 25 Oct. 2004.
- [14] A. Sampaio, N. Loughran, A. Rashid, and P. Rayson. Mining Aspects in Requirements. In *Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design, Chicago, IL, USA*, 15 Mar. 2005.
- [15] P. Sawyer, P. Rayson, and R. Garside. REVERE: Support for Requirements Synthesis from Documents. *Information Systems Frontiers*, 4(3):343–353, 2002.
- [16] G. Sousa, S. Soares, P. Borba, and J. Castro. Separation of Crosscutting Concerns from Requirements to Design: Adapting the Use Case Driven Approach. In *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design, Lancaster, UK*, 22 Mar. 2004.
- [17] S. M. Sutton Jr. Concerns in a Requirements Model - A Small Case Study. In [2].
- [18] K. Wenzel. Maximum Entropie Modelle zur Natürlichen Spracherkennung. Unpublished. In German, 2005.
- [19] J. Whittle and J. Araujo. Scenario Modelling with Aspects. *IEE Proc. - Software Special Issue*, 151(4):157–172, 2004.
- [20] G. Zhang, H. Baumeister, N. Koch, and A. Knapp. Aspect-Oriented Modeling of Access Control in Web Applications. In *Proc. 6th Int. Wsh. on Aspect Oriented Modeling (AOM), Chicago, IL, USA*, 2005.