A Problem-, Quality-, and Aspect-Oriented Requirements Engineering Method

Stephan Faßbender, Maritta Heisel, and Rene Meis(✉)

Paluno - The Ruhr Institute for Software Technology,
University of Duisburg-Essen, Essen, Germany
{stephan.fabbender,maritta.heisel,rene.meis}@uni-due.de

Abstract. Requirements engineers not only have to cope with the requirements of various stakeholders for complex software systems, they also have to consider several software qualities (e.g., performance, maintainability, security, and privacy) that the system-to-be shall address. In such a situation, it is challenging for requirements engineers to develop a complete and coherent set of requirements for the system-to-be. Separation of concerns has shown to be one option to handle the complexity of systems. The problem frames approach address this principle by decomposing the problem of building the system-to-be into simpler subproblems. Aspect-orientation aims at separating cross-cutting functionalities into separate functionalities, called aspects. We propose a method called AORE4PF, which shows that aspect-orientation can be integrated into the problem frames approach to increase the separation of concerns and to benefit from several methods that exist on problem frames to develop a complete and coherent set of requirements. We validated our method with a small experiment in the field of crisis management.

Keywords: Early aspects · Problem frames · Requirements engineering

1 Introduction

Keeping an eye on good and sufficient requirements engineering is a long-known success factor for software projects and the resulting software products [1]. Nonetheless, larger software incidents are regularly reported, which originate in careless dealing with, for example, security requirements. Beside reputation damage, loss of market value and share, and costs for legal infringement [2,3], fixing defects that caused the incident is costly. Fixing a defect when it is already fielded is reported to be up to eighty times more expensive than fixing the corresponding requirements defects early on [4,5]. Therefore, it is crucial for requirements engineers to identify, analyze, and describe all requirements and related quality concerns. But eliciting good requirements is not an easy task [6], even more when considering complex systems.

Part of this work is funded by the German Research Foundation (DFG) under grant number HE3322/4-2.

A. Holzinger et al. (Eds.): ICSOFT 2014, CCIS 555, pp. 291–310, 2015.
DOI: 10.1007/978-3-319-25579-8_17

Nowadays, for almost every software system, various stakeholders with diverse interests exist. These interests give rise to different sets of requirements. These diverse requirements not only increase the complexity of the system-to-be, but also contain different cross-cutting concerns, such as qualities, which are desired by the stakeholders. In such a situation, the requirements engineer is really challenged to master the complexity and to deliver a coherent and complete description of the system-to-be.

One possible option to handle the complexity of a system-to-be is the concept of *separation of concerns* [7]. In its most general form, the separation of concerns principle refers to the ability to focus on, and analyze or change only those parts of a system which are relevant for one specific problem. The main benefits of this principle are a reduced complexity, improved comprehensibility, and improved reusability [7].

Both, *AORE (aspect-oriented requirements engineering)* and the problem frame approach implement this principle, but for different reasons. The approach of AORE, which originates from aspect-oriented programming, is to separate each cross-cutting requirement into an *aspect*. Instead of integrating and solving the cross-cutting requirement for all requirements it cross-cuts, the aspect is solved in isolation. Hence, aspect-orientation leads to a clear separation of concerns. To combine an aspect with a requirement, an aspect defines a pointcut (set of join points), which describes how the aspect and a requirement can be combined. The *problem frames approach* [8] generally also follows the separation of concerns principle. It decomposes the overall problem of building the system-to-be into small sub-problems that fit to a problem frame. Each sub-problem is solved by a machine, which has to be specified using the given domain knowledge. All machines have to be composed to form the overall machine. We will show that aspect-orientation gives guidance for the process of decomposing the overall problem and especially for the composition of the machines. As both ways of separating concerns seem to be complementary, it is promising to combine both. Hence, we propose the AORE4PF (Aspect-Oriented Requirements Engineering for Problem Frames) method that provides guidance for classifying requirements, separating the different concerns, modeling requirements for documentation and application of completeness and interaction analyses, and weaving the reusable parts to a complete and coherent system. Furthermore, AORE4PF provides tool support for most activities.

The rest of the paper is structured as follows. Section 2 introduces a smart grid scenario, which is used as a case study. In Sect. 3, we introduce the problem frames approach and UML4PF as background of this paper. Our method for the integration of AORE into the problem frames approach is presented in Sect. 4. A small experiment for validation is presented in Sect. 5. Work related to this paper is discussed in Sect. 6. Finally, Sect. 7 concludes the paper and presents possible future work.

2 Case Study

To illustrate the application of the AORE4PF method, we use the real-life case study of smart grids. As sources for real functional requirements, we consider

diverse documents such as "Application Case Study: Smart Grid" provided by the industrial partners of the EU project NESSoS[1], the "Protection Profile for the Gateway of a Smart Metering System" [9] provided by the German Federal Office for Information Security[2], and "Requirements of AMI (Advanced Multi-metering Infrastructure") [10] provided by the EU project OPEN meter[3].

We define the terms specific to the smart grid domain and our use case in the following. The *smart meter gateway* represents the central communication unit in a *smart metering system*. It is responsible for collecting, processing, storing, and communicating *meter data*. The *meter data* refers to readings measured by smart meters regarding consumption or production of a certain commodity. A *smart meter* represents the device that measures the consumption or production of a certain commodity and sends it to the gateway. An *authorized external entity* can be a human or an IT unit that communicates with the gateway from outside the gateway boundaries through a *wide area network (WAN)*. The *WAN* provides the communication network that interconnects the gateway with the outside world. The *LMN (local metrological network)* provides the communication network between the meter and the gateway. The *HAN (home area network)* provides the communication network between the consumer and the gateway. The term *consumer* refers to end users of commodities (e.g., electricity).

We have chosen a small selection of requirements to illustrate our method. These requirements are part of the 13 minimum use cases defined for a smart meter gateway given in the documents of NESSoS and the open meter project. The considered use cases are concerned with gathering, processing, and storing meter readings from smart meters for the billing process. The requirements are described as follows:

(R1) **Receive Meter Data.** The gateway shall receive meter data from smart meters.

(R17) **New Firmware.** The gateway should accept a new firmware from authorized external entities. The gate shall log the event of successful verification of a new version of the firmware.

(R18) **Activate New Firmware.** On a predetermined date the gateway executes the firmware update. The gateway shall log the event of deploying a new version of the firmware.

(R28) **Prevent Eavesdropping.** The Gateway should provide functionality to prevent eavesdropping. The gateway must be capable of encrypting communications and data by the safest and best encryption mechanisms possible.

(R29) **Privacy and Legislation.** Many countries protect customers' and people's rights by laws, to ensure that personal and confidential information will not be disclosed easily within communicating systems. Grid systems shall not be a way to reveal information.

[1] http://www.nessos-project.eu/.
[2] www.bsi.bund.de.
[3] http://www.openmeter.com/.

3 UML-Based Problem Frames

Problem frames are a means to describe software development problems. They were proposed by Jackson [8], who describes them as follows: *"A problem frame is a kind of pattern. It defines an intuitively identifiable problem class in terms of its context and the characteristics of its domains, interfaces and requirement."* It is described by a *frame diagram*, which consists of domains, interfaces between domains, and a requirement. We describe problem frames using UML class diagrams extended by stereotypes as proposed by Hatebur and Heisel [11]. All elements of a problem frame diagram act as placeholders, which must be instantiated to represent concrete problems. Doing so, one obtains a *problem diagram* that belongs to a specific class of problems.

Figure 1 shows a problem diagram in UML notation. The class with the stereotype ≪machine≫ represents the thing to be developed (e.g., the software). The classes with some domain stereotypes, e.g., ≪biddableDomain≫ or ≪lexicalDomain≫ represent *problem domains* that already exist in the application environment. Jackson distinguishes the domain types *causal domains* that comply with some physical laws, *lexical domains* that are data representations, *biddable domains* that are usually people, and *connection domains* that mediate between domains.

Domains are connected by interfaces consisting of shared phenomena. Shared phenomena may be events, operation calls, messages, and the like. They are observable by all connected domains, but controlled by only one domain, as indicated by an exclamation mark. For example, in Fig. 1 the annotation WAN!{forwardUpdateFirmware} means that the phenomenon in the set {forwardUpdateFirmware} is controlled by the domain WAN and observable by the machine domain SMGFirmwareStorage, which is connected to it. These interfaces are represented as associations with the stereotype ≪connection≫, and the name of the associations contain the phenomena and the domains controlling the phenomena.

In Fig. 1, the lexical domain FirmwareUpdate is constrained and the AuthorizedExternalEntity is referred to, because the machine SMGFirmwareStorage has the role to store new FirmwareUpdates from AuthorizedExternalEntity for satisfying requirement R17. These relationships are modeled using dependencies that are annotated with the corresponding stereotypes.

The full description for Fig. 1 is as follows: The biddable domain AuthorizedExternalEntity controls the updateFirmware command, which is forwarded by the WAN and finally observed by the machine domain SMGFirmwareStorage. The SMGFirmwareStorage controls the phenomenon storeNewFirmware, which stores the received information in the lexical domain FirmwareUpdate.

Software development with problem frames proceeds as follows: first, the environment in which the machine will operate is represented by a *context diagram*. Like a problem diagram, a context diagram consists of domains and interfaces. However, a context diagram contains no requirements. Then, the problem is decomposed into subproblems. If ever possible, the decomposition is done in such a way that the subproblems fit to given problem frames. To fit a subproblem to a problem frame, one must instantiate its frame diagram, i.e., provide

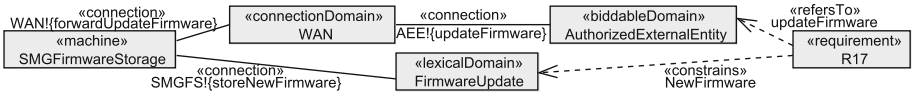

Fig. 1. Problem diagram R17: new firmware.

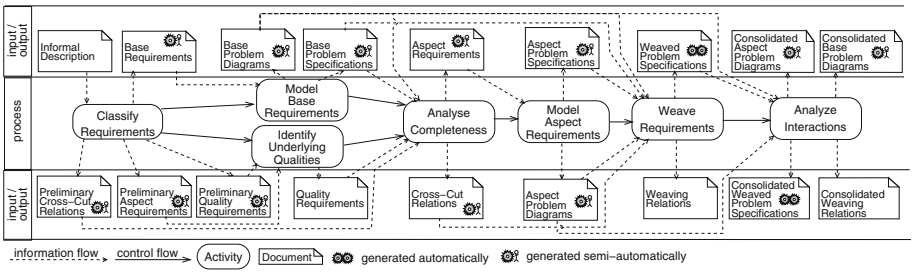

Fig. 2. The AORE4PF method.

instances for its domains, phenomena, and interfaces. The UML4PF framework provides tool support for this approach. A more detailed description can be found in [12].

4 Method

An illustration of our method is given in Fig. 2. The initial input for our method is a textual *informal description* of the requirements the system-to-be shall fulfill. These requirements are *classified* into *preliminary aspect requirements* (or short *aspects*), which are functional and cross-cutting, *preliminary quality requirements* (or short *qualities*), which are non-functional and cross-cutting, and *base requirements* (or short *bases*), which are not cross-cutting. Additionally, the relations between requirements and aspects or qualities are documented as preliminary cross-cut relations. Then all identified *base requirements* are *modeled* following the problem frames approach introduced in Sect. 3, such that for each *base requirement* a *base problem diagram* is created. Additionally, we create a sequence diagram for each problem diagram. The sequence diagrams serve as a *base problem specification*. To prepare the completeness analysis, we *identify* for all *preliminary aspect requirements* the underlying qualities they address. The already known *preliminary quality requirements* can aid the identification. As a result, we get a set of *quality requirements*. Based on the identified *quality and base requirements*, we can analyze whether there is a cross-cut relation between a quality requirement and a base requirement not discovered yet. Thus, we *analyze the completeness* of the *preliminary cross-cut relations* and update them if necessary. The results are a set of *cross-cut relations* and also updated *aspect requirements*. Next, the *aspect requirements* are *modeled* in a similar way as requirements using specialized problem diagrams, called *aspect problem diagrams*. Again, we specify the machine behavior using sequence diagrams, which

results in *aspect problem specifications*. For the next step, *weave requirements*, the *base problem specifications* and *aspect problem specifications* are weaved to fulfill the base and aspect requirements as defined by the *base problem diagrams* and *aspect problem diagrams*. For the weaving, we have to accomplish two activities. First, we define the *weaving relations*. These relations refine the *cross-cut relations*. Then, we can automatically generate for each requirement a *weaved problem specification* representing the weaved system behavior. Last, we have to *analyze* the *base* and *aspect problem diagrams* for unwanted *interactions*, such as conflicts. The *weaving relations* and the *weaved problem specifications* can support this activity. The results of this step are *consolidated base and aspect problem diagrams* as well as *consolidated weaving relations and problem specifications*. We will discuss all steps of our method in detail in the following sections.

4.1 Classify Requirements

As a first step, we have to identify and analyze the requirements contained in the *informal description*. We have to separate and classify these requirements as they will be treated differently afterwards. A requirement can be (1) a base, which is functional and not cross-cutting, (2) an aspect, which is functional and cross-cutting, and (3) a quality, which is non-functional and cross-cutting. Note that we see quality requirements as requirements, which are not operationalized to an aspect right now. Hence, there is a clear relation between qualities and aspects, and we will later on refine qualities to aspects. Normally, statements in an informal description are not given that clear-cut as given by the three discussed classes of requirements. Hence, one can find requirements mixing different classes, for example, aspects are already combined with the corresponding bases or qualities are mentioned in the according bases. In consequence, identifying statements which constitute requirements is only half of the job, but also a separation of mixed requirements has to be performed.

First, we separate functional and quality requirements. A tool like OntRep [13] can support the requirements engineer in this step. This way we identify `R29` as requirement containing two quality requirements (`R29A` and `R29B`) and `R28` containing one quality (`R28A`) and one functional requirement (`R28B`):

`(R28A)` **Security.** The Gateway shall be protected against external attacks.

`(R29A)` **Privacy.** [...] personal and confidential information will not be disclosed easily within communicating systems. Grid systems shall not be a way to reveal information.

`(R29B)` **Compliance.** Many countries protect customers' and people's rights by laws.

Thus, we have identified and separated the *preliminary quality requirements*.

Second, we have to analyze the functional requirements for aspects and separate them. For this activity tools like EA-Miner [14], Theme/Doc [15] or REAssistant[4] can aid the requirements engineer. This way we identify the following two aspects:

[4] https://code.google.com/p/reassistant/.

(R28B) **Network Encryption.** [...] The gateway must be capable of encrypting communications and data by the safest and best encryption mechanisms possible.

(R30) **Logging.** The gate shall log the occurring important events.

Note that while eavesdropping is already formulated as separate aspect, logging is introduced as a new aspect that is extracted from R17 and R18 which both contain the logging aspect:

(R17B) **New Firmware: Logging.** The gate shall log the event of successful verification of a new version of the firmware.

(R18B) **Activate New Firmware: Logging.** The gateway shall log the event of deploying a new version of the firmware.

These two requirements describe how the aspect R30 has to be integrated into the corresponding base requirements. This information is used later on during the weaving process. Thus, we have identified and separated the *preliminary aspect requirements*.

The remaining functional requirements form the *base requirements* for our system:

(R1) **Receive Meter Data.** The gateway shall receive meter data from smart meters.

(R17A) **New Firmware.** The gateway should accept a new firmware from authorized external entities.

(R18A) **Activate New Firmware.** On a predetermined date the gateway executes the firmware update.

We document the relations between the separated functional, quality, and aspect requirements in a *preliminary cross-cut relation table*. These relations are given in Table 1 with crosses in *italic* in the regions (Base,Quality), (Base,Aspect), and (Quality,Aspect). Note that everything given in **bold** is discovered later on in the annotated step (^{x}). Furthermore, the regions (Aspect,Quality) and (Aspect,Aspect) are considered in step 4, and (Quality,Quality) in step 7. If a requirement is separated into a functional requirement (base or aspect) and a quality, then we add a cross in the region (Base,Quality) of the table if the functional requirement is a base requirement, representing that the quality has to be taken into account for the base requirement, and in the region (Quality,Aspect) if it is an aspect requirement, representing that the aspect requirement addresses the software quality. In Table 1, we documented that the aspect R28B is related to the quality R28A. This kind of mapping will later on be used to provide guidance for the selections of mechanisms to address the quality requirements. If functional requirements are separated into base and aspect requirements, then we also add respective crosses in the upper right quadrant. In Table 1, we documented that the aspect R30 cross-cuts the base requirements R17A and R18A.

Table 1. Requirements (Cross-cut) relation table for the smart grid scenario.

		Quality				Aspect		
		R28A	R29A	R29B	R31[3]	R28B	R30(R17B, R18B)	R32[4]
Base	R1	$\mathbf{X}^4$	$\mathbf{X}^4$	$\mathbf{X}^4$	$\mathbf{X}^4$	$\mathbf{X}^4$	$\mathbf{X}^4$	
	R17A	$\mathbf{X}^4$			$\mathbf{X}^3$	$\mathbf{X}^4$	X	
	R18A				$\mathbf{X}^3$		X	
Aspect	R28B							
	R30	$\mathbf{X}^4$						$\mathbf{X}^4$
	R32[4]							
Quality	R28A		$\mathbf{I}^7$	$\mathbf{I}^7$	$\mathbf{I}^7$	X		$\mathbf{X}^4$
	R29A	$\mathbf{I}^7$		$\mathbf{I}^7$	$\mathbf{I}^7$	$\mathbf{X}^4$		$\mathbf{X}^4$
	R29B	$\mathbf{I}^7$	$\mathbf{I}^7$		$\mathbf{I}^7$	$\mathbf{X}^4$	$\mathbf{X}^4$	$\mathbf{X}^4$
	R31[3]	$\mathbf{I}^7$	$\mathbf{I}^7$	$\mathbf{I}^7$			$\mathbf{X}^3$	

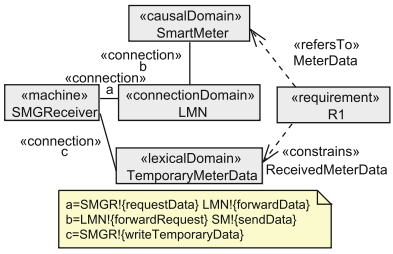

Fig. 3. Problem diagram for R1.

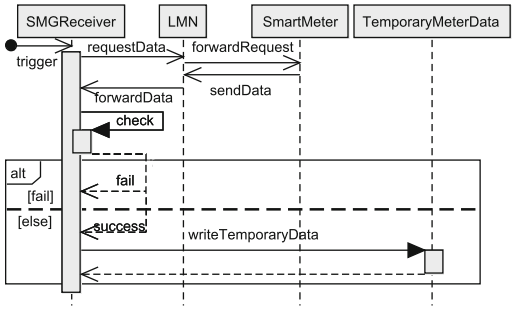

Fig. 4. Sequence diagram for R1.

4.2 Model Base Problems

In this step, we model the functional requirements identified in the previous step. For each functional requirement, we create a problem diagram as proposed by the problem frames approach introduced in Sect. 3. For reasons of space, we only show the problem diagrams for the requirements R1 and R17A, but these two problem diagrams are sufficient to understand the rest of the paper, even though we use the five selected requirements for exemplifying our method. The problem diagram for R17A is shown in Fig. 1 and explained in Sect. 3. Figure 3 shows the problem diagram for R1. The problem described in this diagram is that the machine SMGReceiver shall requestData via the LMN from the SmartMeter. In response, the SmartMeter will sendData that was requested via the LMN back to the machine. The machine does then writeTemporaryData received from the smart meter in the lexical domain TemporaryMeterData.

For every problem diagram, we have to provide a reasoning, called *frame concern* [8], why the *specification* of the submachine together with the knowl-

edge about the environment (*domain knowledge*) leads to the satisfaction of the *requirement*. To visualize how frame concern is addressed in the specific problems, we create at least one sequence diagram for each problem diagram. These sequence diagrams describe the specification (behavior of the machine) and the domain knowledge (behavior of the domains) which is necessary to satisfy the requirement. How to systematically create the sequence diagrams is out of scope of this paper, but the approach presented by Jackson and Zave [16] can be used for this task. Figure 4 shows the sequence diagram for the sub-problem Receive Meter. The interaction is started the sub-machine SMGReceiver causing the phenomenon requestData (specification). This request is forwarded via the LMN to the SmartMeter (domain knowledge). The smart meter then answers the request and sends the meter data (requirement) using the phenomenon sendData (domain knowledge). The data is forwarded via the LMN to the sub-machine (domain knowledge). In the case of a successful check of the received data, the received data is stored in the lexical domain TemporaryMeterData (specification). Hence, the gateway stores the meter data received from smart meters (requirement).

4.3 Identify Underlying Qualities

In order to check whether the cross-cut relation is complete, we identify for all aspects the software qualities they address. Note that the relationship between aspects and qualities is many-to-many. That is, an aspect can address multiple software qualities. For example, the logging of system events possibly addresses the software qualities accountability, transparency, maintainability, performance, and traceability. On the other hand, a software quality can be addressed by multiple aspects, for example, the software quality confidentiality could be addressed by the following aspects: encryption, authentication and authorization, and data minimization. For the identification of underlying qualities tools such as QAMiner [17] can be used. This way we discover that in our case the aspect R30 has the underlying quality maintainability:

(R31) **Maintainability.** All events which are useful to trace a malfunction of the gateway shall be logged.

We document the relation between the aspect and the identified underlying quality in cross-cut relation table. In Table 1, we added the bold cross $\mathbf{X}^3$ in the lower right quadrant. Furthermore, we add the relations between the identified quality to the base requirements which are implied by the relations of the corresponding aspect. For our smart grid scenario, we added the bold crosses $\mathbf{X}^3$ in the upper left quadrant of Table 1. The consideration of the underlying qualities allows requirements engineers to access whether the selected mechanisms (aspects) sufficiently address the respective quality.

4.4 Analyze Completeness

Based on the identified qualities, we can re-use quality-dependent analysis techniques on problem frames to check the completeness of the cross-cut relation. For

example, for privacy one can use the ProPAn method [18], the law (identification) pattern method [19] provides guidance for compliance, security is covered by the PresSuRE method [20], and so forth. These analysis techniques identify for a given problem frames model and the respective quality in which functional requirements the quality has to be considered. At this point of our method, we have all inputs that the analysis techniques need. Using the results of the analysis techniques, we can update the cross-cut relation and check whether the selected aspects together with the defined cross-cut relation guarantee the intended software qualities.

In this way, we identify that, for example, several qualities are relevant for `R1`. Privacy (`R29A`) is relevant as the consumption data metered by the smart meters enables one to analyze what the persons in the household are currently doing. Hence, the consumption data is an asset which has to be protected. As result, the security analysis also shows that the consumption data has to be protected against eavesdropping (`R28A`). Maintainability (`R31`) is also relevant for `R1`, as a malfunction can also occur while receiving consumption data. The compliance analysis (`R29B`) reveals and strengthens the importance of privacy because of different data protection acts. Additionally, the logging mechanism is not only relevant for maintainability but also for compliance as several laws require the fulfillment of accountability requirements whenever there is a contractual relation between different parties. This information is used to update the cross-cut relation table (see bold crosses $\mathbf{X}^4$ in Table 1). The already existing aspect requirements are sufficient to cover the newly found relations.

Furthermore, we have to check whether a software quality that was identified as relevant for a base requirement is also relevant for an aspect requirement that cross-cuts the base requirement. E.g., we have to check whether the logs written for the base requirements `R1` and `R17B` contain confidential information that has to be protected against an external attacker. For presentation purpose, we assume that such an attacker has to be considered in the smart grid scenario and add an aspect requirement for the encryption of persistent data that cross-cuts the logging aspect.

`(R32)` **Data Encryption.** Persistent data shall be stored encrypted on the gateway.

We update the regions (Aspect,Quality) and (Aspect,Aspect) of the cross-cut relation table (see Table 1) to document that the quality `R28A` has to be taken into account for the aspect `R30` (cross in region (Aspect,Quality)), and that the aspect `R3` is cross-cut by the newly introduced aspect `R32` (cross in region (Aspect,Aspect)).

4.5 Model Aspect Requirements

To model aspect requirements in a similar way as base requirements, we extended the UML profile of the UML4PF tool with aspect-oriented concepts. To differentiate aspect requirements, the machines that address them, and the diagram they are represented in, from base requirements and their machines

and diagrams, we introduce the new stereotypes ≪Aspect≫, ≪AspectMachine≫, and ≪AspectDiagram≫ as specialization of the stereotypes ≪Requirement≫, ≪ProblemDiagram≫, and ≪Machine≫, respectively. In addition to problem diagrams, an aspect diagram has to contain a set of join points, which together form a pointcut. These join points can be domains and interfaces. Hence, we introduced the new stereotype ≪JoinPoint≫, which can be applied to all specializations of the UML meta-class NamedElement. During the weaving, join points are instantiated with domains of the diagrams the aspect cross-cuts.

To create an aspect diagram, we have to identify the join points which are necessary to combine the aspect with the problems it cross-cuts and to understand the problem of building the aspect machine. In most cases, we have a machine, besides the aspect machine, as join point in an aspect diagram. This machine will be instantiated during the weaving with the machine of the problem that the aspect is weaved into. The interface between this join point and the aspect machine describes how a problem machine can utilize an aspect and which context information is needed by the aspect machine. We have to derive the join points important for the problem described by the aspect from its description and the requirements it cross-cuts. Besides the specialized stereotypes for the machine and the requirement, and the definition of join points for the later weaving, the process of building an aspect diagram is similar to the process of building problem diagrams. As for problem diagrams, we also create sequence diagrams for each aspect. The sequence diagrams contain two kinds of information. First, the messages annotated with the stereotype ≪JoinPoint≫ describe the pointcut scenario. I.e., these messages describe when during the behavior necessary to accomplish the cross-cut requirement the behavior of the aspect can be integrated. Note that we can represent the common pointcut definitions used, e.g., in AspectJ, such as before, after and around, by a sequence diagram with the behavior description for the aspect before, after, or around the pointcut scenario, respectively. Second, all other messages describe the internal behavior necessary to accomplish the aspect requirement.

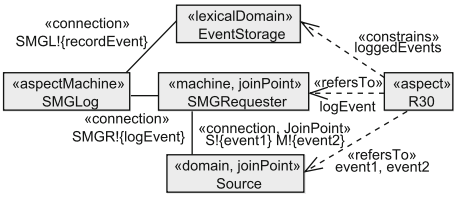

Fig. 5. Aspect diagram for aspect R30.

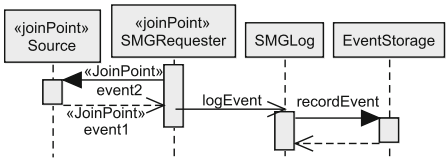

Fig. 6. Sequence diagram for aspect R30.

For reasons of space, we will only discuss the aspect requirement R30 in detail. The aspect R28B and the sequence diagram for the decryption of received data is described in [21]. R30 covers the logging of important events in the system. The corresponding aspect diagram is presented in Fig. 5. It contains the

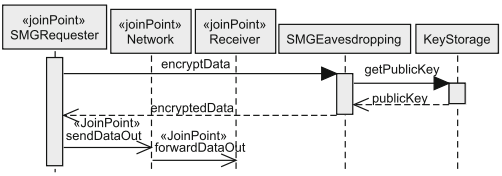

Fig. 7. Sequence diagram for aspect R28B.

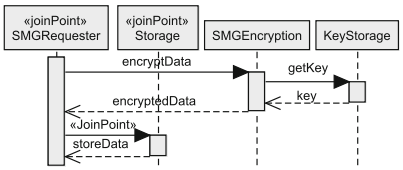

Fig. 8. Sequence diagram for aspect R32.

aspect machine SMGLog, which is able to record events in the EventStorage. Furthermore, the aspect diagram contains two domains as join points. The machine SMGRequester will be instantiated by a problem machine and the domain Source by the origin of the event to be logged. The machine SMGRequester observes the phenomenon event1 of Source and is able to issue the phenomenon event2. These phenomena represent the events that shall be logged and need to be instantiated during the weaving. If an event that has to be logged is observed, then SMGRequester instructs the aspect machine SMGLog to log that event (logEvent). In general, we have to distinguish four cases for the event to be logged. The event could be issued using a synchronous or asynchronous message of the Source, or a synchronous or asynchronous message from the machine SMGRequester to the Source. For the sake of simplicity, we only consider the case shown in the sequence diagram in Fig. 6. This sequence diagram shows the case that SMGRequester sends a synchronous message to Source and receives a result (requirement). Then SMGRequester asks SMGLog to log the observed event (requirement). The machine SMGLog then records the event (specification). Hence, the observed event is logged (requirement). Figures 7 and 8 show the sequence diagrams for the behavior of aspect R28B for sending encrypted data via a network and aspect R32 for encrypting data that shall be stored persistently.

4.6 Weave Requirements

For each base requirement, we now create a sequence diagram that describes how the aspect requirements have to be weaved into it to address the cross-cut relations. The basis for the weaving sequence diagram is the sequence diagram of the requirement. The behavior of the sub-machine is extended with the invocation of the aspects given by the row of the base requirement in the cross-cut relation table (see Table 1). Furthermore, we have to consider whether the base requirement is cross-cut by an aspect a_1 that is itself cross-cut by another aspect a_2. If this is the case, we have to weave the aspect a_2 into the base requirement after the aspect a_1 was weaved into it.

The cross-cut relations are not sufficient to weave the aspect requirements into the base requirement. The reason is that the cross-cut relation does not define how and when an aspect has to be integrated into the base problem. Nevertheless, we can identify the situations during the dynamics of the base problem where an aspect could be integrated using the pointcut scenarios described in the sequence diagrams of the aspect. For each base requirement, we create a

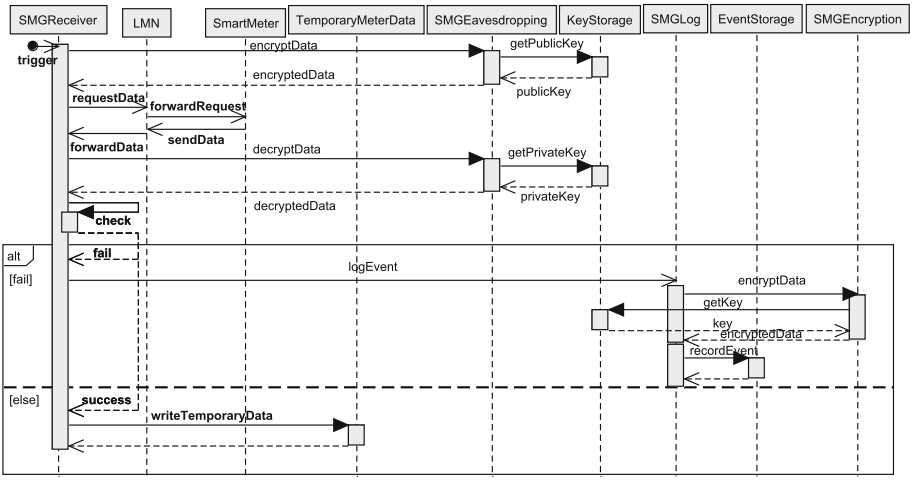

Fig. 9. Weaved sequence diagram for R17A.

table that defines the weaving relations, i.e., how and in which order the aspects have to be integrated into the base problem. A row in the table consists of the aspect sequence diagram that shall be weaved into the requirement, and the instantiation of the join points of the aspect with the domains and messages of the base sequence diagram. An instantiation of a join point j by a domain or message b of the base problem is denoted by b/j. The instantiated messages uniquely describe how and when the aspect is integrated into the base sequence diagram. Table 2 shows the weaving relations for base requirement R1.

Because of the aspect requirement R28B all communications have to be encrypted to prevent eavesdropping attacks. This implies that all external messages that a sub-machine sends have to be encrypted and the ones it receives have to be decrypted. Hence, we have to integrate the aspect R28B twice into the base requirement R1. The pointcut scenarios in the two sequence diagrams R28B (Out) (shown in Fig. 7) and R28B (In) can only be instantiated in one way, because in the sequence diagram for R1 (see Fig. 4) there is only one communication from the machine via a network (LMN) to a receiver (SmartMeter) and one back from the sender (SmartMeter) via the network (LMN). The first two lines of Table 2 describe these integrations. The pointcut scenario of the aspect R30 matches for all synchronous message calls with a reply (see Fig. 6). Hence, we have two possible situations in the sequence diagram for R1 where the aspect could be integrated. The event to be logged is a failed check of the received meter data and hence, we integrate aspect R30 as described by the third line in Table 2. Finally, we have to integrate aspect R32 that cross-cuts aspect R30. The pointcut scenario for R32 (see Fig. 8) has to be instantiated with the recording of the event (see Fig. 6) as described in line four of Table 2.

The weaving relations are used to generate the weaving sequence diagrams from the sequence diagrams of the problem and aspect diagrams. These auto-

Table 2. Weaving relations for base requirement R1.

Aspect	Domain Instantiations	Message Instantiations
R28B (Out)	SMGReceive/SMGRequester, LMN/Network, SmartMeter/Receiver	requestData/sendDataOut, forwardRequest/forwardDataOut
R28B (In)	SMGReceive/SMGRequester, LMN/Network, SmartMeter/Sender	sendData/sendDataIn, forwardData/forwardDataIn
R30	SMGReceive/SMGRequester, SMGReceive/Source	check/event2, fail/event1
R32	SMGLog/SMGRequester, EventStorage/Storage	recordEvent/storeData

matically generated sequence diagrams have then to be adjusted, such that the overall behavior satisfies the weaving requirement. The generated sequence diagram for R1 is shown in Fig. 9. For the sake of readability, we use a bold font for messages from the original problem specification of R1. In accordance with Table 2, the date sent to the smart meter is encrypted before sending and the received data is decrypted when received. Furthermore, in the case of a failed check of the received data an encrypted log is recorded.

4.7 Analyze Interactions

For reasons of space, we do not go into detail for this step. Alebrahim et al. provide methods for interaction analysis using problem frames. In [22] functional requirements are treated, and [23] describes how to analyze quality requirements for interactions. Both works use the smart grid as a case study. Hence, we reused the methods and results also for this work. The results are documented in Table 1 using bold **I**.

Table 3. Effort spent (in person-hours/minutes) for conducting the method.

	Classify Requirements	Model Base Requirements	Identify Underlying Qualities	Analyze Completeness	Model Aspect Requirements	Weave Requirements	Analyze Interactions
Number of items	27 requirements	10 base requirements	6 aspect requirements	10 base requirements	5 aspect requirements	10 base requirements	16 functional requirements
Ø per item	11min	36min	7min	7min	34min	23min	6min
Total	5h 00min	6h 3min	45min	1h 15min	2h 51min	3h 53min	1h 45min

5 Validation

To validate our method, we applied it to the crisis management system (CMS) [24] that Kienzle et al. proposed as a case study for aspect-oriented modeling. We derived an informal scenario description and the textual use case descriptions from the original as input for our method[5]. The method was executed by a requirements expert, who did not know the case beforehand. From the

[5] For the inputs and the results see http://imperia.uni-due.de/imperia/md/content/swe/aore4pf_cms_report.pdf.

information provided to the requirements analyst, he identified 13 base requirements that he modeled using 10 problem diagrams, 8 aspect requirements that he modeled using 5 aspect diagrams, and 6 quality requirements.

The effort spent for conducting our method on the CMS is summarized in Table 3. It took 5 h to classify the requirements. Note that for the case study this step was done manually. The reason was that tools such as, for example, OntRep [13] or EA-Miner [14] require some additional input like training documents or an existing ontology. But unfortunately, such inputs were not available. Hence, the first step can be sped up significantly using these tools. Another big block of effort is the modeling of base and aspect requirements. Here the tool support already helps to speed up the modeling, but is subject for further improvement. Note that the modeling steps do not only include the modeling itself, but also the analysis and improvement of the original requirements, which make the requirements more precise and unambiguous. Therefore, parts of the effort spent on the modeling steps are unavoidable even when using another method or notation. The modeling itself pays off as it allows the usage of the broad spectrum of methods and tools which need problem frame models as input. For example, the analysis of completeness uses these models and takes about an hour for different kinds of qualities. The weaving of aspects is quite time consuming right now. Here the tool support is on an experimental level, but the observations taken during the case study imply that a full fledged tool support will significantly drop the effort. The interaction analysis takes round about two hours, which is significantly below the effort of doing such an analysis without a problem frame model (see [22] for further information). All the effort spent sums up to 21,5 person hours, which is significant but reasonable with regards to the results one gets. And compared to efforts other authors report, the effort spent for our method seems to be even low. For example, Landuyt et al. [25] report an effort spent of 170 h for the requirements engineering related activities.

Table 4. Requirements identified.

		1) Functional	2) Availability	3) Reliability	4) Persistence	5) Real-Time	6) Security	7) Mobility	8) Statistic Logging	9) Multi-Access	10) Safety	11) Adaptability	12) Accuracy	13) Maintainability	14) Performance	15) Scalability	Sum
Same Class	Identified	100%	33%	50%	0%	0%	67%	0%	0%	0%	75%	0%	0%				30%
	Partly	0%	0%	0%	0%	0%	33%	0%	0%	0%	0%	0%	0%				3%
	Not Identified	0%	0%	0%	0%	0%	0%	33%	0%	0%	25%	0%	75%				13%
Other Class	Identified As		13)	2)	1)	14)		1)	1)	15)		1)	1)				
	Identified	0%	67%	50%	100%	67%	0%	67%	100%	100%	0%	25%	25%				45%
	Partly	0%	0%	0%	0%	33%	0%	0%	0%	0%	0%	75%	0%				10%
Aggregated	Identified	100%	100%	100%	100%	67%	67%	67%	100%	100%	75%	25%	25%				75%
	Partly	0%	0%	0%	0%	33%	33%	0%	0%	0%	0%	75%	0%				13%
	Not Identified	0%	0%	0%	0%	0%	0%	33%	0%	0%	25%	0%	75%				13%

To asses the sufficiency of the method and the used tools, the requirements and qualities found within our method were compared to the original document as described by Kienzle et al. Table 4 shows the comparison. Overall, the results are satisfying as most requirements were found and classified in the correct class (30 %) or in another, also correct, class (45 %). The high amount of requirements classified differently are due to specific classes given in the original documents. For example, persistence and statistical logging were completely described as functional requirements in the documents but treated as qualities. For such requirements it is a more general discussion if they are quality requirements or not. Hence, we accepted both views as correct. For some specific qualities, such as mobility or accuracy, the overall observation cannot be acknowledged. The reasons are subject to further investigations.

To asses the aspects identified, we compared the results of our method to the results given in other publications considering aspect-oriented requirements engineering using the same scenario [25,26]. The set of requirements identified with our method includes all requirements which are treated as aspects in the other works. 83 % of the aspects found and separated in [25] and 75 % of those in [26] were also separated as aspects by our method. The other 17 % of aspects in [25] and 25 % in [26] were identified as base requirements by our method. A detailed investigation showed that both views on these requirements are reasonable. Some of the aspects our method found were not mentioned in the other works. 38 % and 25 % of the requirements identified by our method where not mentioned in [25,26], respectively. Reasons for the missing requirements might be that they were not reported due to lack of space or that they were not found.

We could not asses our completeness analysis quantitatively as the other works using the scenario stick to the original requirements. But the qualitative investigation of the completeness analysis showed reasonable results. This observation is also true for the cross cut relations. We also compared the weaved specification with sequence diagrams or state machines given by the original document and works in [24]. Here we observed that the specifications produced by our method were at least as good as the chosen assessment artifacts. Again, the interaction analysis could not be assessed quantitatively due to missing benchmarks. But the found interactions seemed to be real problems which have to be resolved in a real case.

6 Related Work

There are many works considering early aspects [27–33]. Most of these approaches deal with goal-oriented approaches and use-case models. But goal or use-case models are of a higher level of abstraction than problem frames. Additionally, goal and use-case models are stakeholder-centric, while problem frames are system-centric. Therefore, refining functional requirements taking into account more detail of the system-to-be and analyzing the system-to-be described by the functional requirements is reported to be difficult for such

methods [34]. Recently, there were papers which reported a successful integration of goal- and problem-oriented methods [35,36]. Hence, one might benefit from integrating goal-models in our method.

Conejero et al. [37] present a framework alike the method presented in this paper. Their process also starts with unstructured textual requirements. Then different tools and modeling notations are used along the frame work to identify and handle aspects. In difference to our process, they do not consider a completeness or interaction analysis and especially for the modeling of aspects they lack tool support.

Only few approaches consider the integration of early aspects in the problem frames approach. Lencastre et al. [38] also investigated how early aspects can be integrated into problem frames. Their method to model aspects in the problem frames approach differs from ours. For an aspect, the authors first select a problem frame as *PF Pointcut Scenario*. This pointcut scenario defines into which problems the aspect can be integrated. The pointcut scenario is then extended to the *PF Aspectual Scenario*, which is similar to our aspect diagrams, with the difference that the pointcut always has to be a problem frame. This reduces flexibility, because an aspect (e.g., logging of all system events) may have to be integrated into different problem diagrams.

7 Conclusions

In this paper, we presented the AORE4PF method which integrates aspect-orientation into the problem frames approach and utilizes many quality analysis method based on problem frames to be a problem-, quality-, and aspect-oriented requirements engineering method. We extended the UML4PF profile with stereotypes that allow us to create aspect diagrams. We further introduced a structured methodology to separate aspects from requirements, to model aspects, and to weave aspects and requirements together. We considered both the static and the behavioral view on the requirements, aspects, and their weaving. We exemplified our method using a smart grid scenario from the NESSoS project as case study and validated it using a crisis management system.

The contributions of this work are (1) the integration of aspects into the problem frames approach, (2) a structured way of separating base, quality and aspect requirements, starting from a textual description, (3) the detection of implicit qualities given by aspects, (4) identification of all base requirements relevant for a quality and the related aspects, (5) a structured method to weave base and aspect requirements, and (6) the integration of an interactions analysis between the resulting requirements. The AORE4PF method is (7) tool-supported in most steps. The resulting requirements model not necessarily leads to an aspect-oriented implementation of the software. The identified aspects can also help to define the structure of a component-based implementation.

For future work, we plan to improve the tool support. More steps of our method, such as the instantiation of pointcut scenarios during the weaving, can be automated to a higher degree and we want to provide an integrated tool chain

for the requirements separation. Additionally, we will investigate how architectures can be derived from the aspect-oriented requirements model.

References

1. Hofmann, H., Lehner, F.: Requirements engineering as a success factor in software projects. IEEE Softw. **18**, 58–66 (2001)
2. Cavusoglu, H., Mishra, B., Raghunathan, S.: The effect of internet security breach announcements on market value: capital market reactions for breached firms and internet security developers. Int. J. Electron. Commer. **9**, 70–104 (2004)
3. Khansa, L., Cook, D.F., James, T., Bruyaka, O.: Impact of HIPAA provisions on the stock market value of healthcare institutions, and information security and other information technology firms. Comput. Secur. **31**, 750–770 (2012)
4. Boehm, B.W., Papaccio, P.N.: Understanding and controlling software costs. IEEE Trans. Softw. Eng. **14**, 1462–1477 (1988)
5. Willis, R.: Hughes aircraft's widespread deployment of a continuously improving software process. AD-a358 993. Carnegie-Mellon University (1998)
6. Firesmith, D.: Specifying good requirements. J. Object Technol. **2**, 77–87 (2003). http://www.jot.fm/issues/issue_2003_07/column7
7. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM **15**, 1053–1058 (1972)
8. Jackson, M.: Problem Frames. Analyzing and structuring software development problems. Addison-Wesley, New York (2001)
9. Kreutzmann, H., Vollmer, S., Tekampe, N., Abromeit, A.: Protection profile for the gateway of a smart metering system. Technical report, BSI (2011)
10. OPEN meter project: requirements of AMI. Technical report, OPEN meter project (2009)
11. Hatebur, D., Heisel, M.: A UML profile for requirements analysis of dependable software. In: Schoitsch, E. (ed.) SAFECOMP 2010. LNCS, vol. 6351, pp. 317–331. Springer, Heidelberg (2010)
12. Côté, I., Hatebur, D., Heisel, M., Schmidt, H.: UML4PF - a tool for problem-oriented requirements analysis. In: Proceedings of the 19th IEEE International Requirements Engineering Conference, pp. 349–350. IEEE Computer Society (2011)
13. Moser, T., Winkler, D., Heindl, M., Biffl, S.: Requirements management with semantic technology: an empirical study on automated requirements categorization and conflict analysis. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 3–17. Springer, Heidelberg (2011)
14. Sampaio, A., Rashid, A., Chitchyan, R., Rayson, P.: EA-Miner: towards automation in aspect-oriented requirements engineering. In: Rashid, A., Akşit, M. (eds.) Transactions on AOSD III. LNCS, vol. 4620, pp. 4–39. Springer, Heidelberg (2007)
15. Baniassad, E., Clarke, S.: Finding aspects in requirements with Theme/Doc. In: Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, pp. 15–22 (2004). http://trese.cs.utwente.nl/workshops/early-aspects-2004/workshop_papers.htm
16. Jackson, M., Zave, P.: Deriving specifications from requirements: an example. In: ICSE, pp. 15–24. ACM Press, USA (1995)
17. Rago, A., Marcos, C., Diaz-Pace, J.A.: Uncovering quality-attribute concerns in use case specifications via early aspect mining. Requirements Eng. **18**, 67–84 (2013)

18. Beckers, K., Faßbender, S., Heisel, M., Meis, R.: A problem-based approach for computer-aided privacy threat identification. In: Preneel, B., Ikonomou, D. (eds.) APF 2012. LNCS, vol. 8319, pp. 1–16. Springer, Heidelberg (2014)
19. Faßbender, S., Heisel, M.: From problems to laws in requirements engineering using model-transformation. In: ICSOFT 2013, pp. 447–458. SciTePress (2013)
20. Faßbender, S., Heisel, M., Meis, R.: Functional requirements under security ProbSuRE. In: ICSOFT-PT 2014 - Proceedings of the 9th International Conference on Software Paradigm Trends, pp. 5–16. SciTePress (2014)
21. Faßbender, S., Heisel, M., Meis, R.: Aspect-oriented requirements engineering with problem frames. In: ICSOFT-PT 2014 - Proceedings of the 9th International Conference on Software Paradigm Trends, pp. 145–156. SciTePress (2014)
22. Alebrahim, A., Faßbender, S., Heisel, M., Meis, R.: Problem-based requirements interaction analysis. In: Salinesi, C., van de Weerd, I. (eds.) REFSQ 2014. LNCS, vol. 8396, pp. 200–215. Springer, Heidelberg (2014)
23. Alebrahim, A., Choppy, C., Faßbender, S., Heisel, M.: Optimizing functional and quality requirements according to stakeholders' goals. In: Mistrik, I. (ed.) System Quality and Software Architecture. Elsevier, Amsterdam (2014)
24. Kienzle, J., Guelfi, N., Mustafiz, S.: Crisis management systems: a case study for aspect-oriented modeling. In: Katz, S., Mezini, M., Kienzle, J. (eds.) Transactions on Aspect-Oriented Software Development VII. LNCS, vol. 6210, pp. 1–22. Springer, Heidelberg (2010)
25. Van Landuyt, D., Truyen, E., Joosen, W.: Discovery of stable abstractions for aspect-oriented composition in the car crash management domain. In: Katz, S., Mezini, M., Kienzle, J. (eds.) Transactions on Aspect-Oriented Software Development VII. LNCS, vol. 6210, pp. 375–422. Springer, Heidelberg (2010)
26. Mussbacher, G., Amyot, D., Araújo, J., Moreira, A.: Requirements modeling with the aspect-oriented user requirements notation (AoURN): a case study. In: Katz, S., Mezini, M., Kienzle, J. (eds.) Transactions on Aspect-Oriented Software Development VII. LNCS, vol. 6210, pp. 23–68. Springer, Heidelberg (2010)
27. Rashid, A.: Aspect-oriented requirements engineering: an introduction. In: Proceedings of the 16th IEEE International Requirements Engineering Conference, pp. 306–309. IEEE Computer Society (2008)
28. Yu, Y., Cesar, J., Leite, S.P., Mylopoulos, J.: From goals to aspects: discovering aspects from requirements goal models. In: Proceedings of the 12th IEEE International Requirements Engineering Conference, pp. 38–47. IEEE Computer Society (2004)
29. Jacobson, I., Ng, P.W.: Aspect-Oriented Software Development with Use Cases. Addison-Wesley Professional, Englewood Cliffs (2004)
30. Whittle, J., Araujo, J.: Scenario modelling with aspects. IEE Proc. Softw. **151**, 157–171 (2004)
31. Sutton, Jr., S.M., Rouvellou, I.: Modeling of software concerns in cosmos. In: Proceedings of the 1st International Conference on Aspect-oriented Software Development, AOSD 2002, pp. 127–133. ACM, New York (2002)
32. Moreira, A., Araújo, J., Rashid, A.: A concern-oriented requirements engineering model. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 293–308. Springer, Heidelberg (2005)
33. Grundy, J.C.: Aspect-oriented requirements engineering for component-based software systems. In: Proceedings of the IEEE International Symposium on Requirements Engineering, pp. 84–91. IEEE Computer Society, Washington (1999)

34. Alrajeh, D., Kramer, J., Russo, A., Uchitel, S.: Learning operational requirements from goal models. In: IEEE 31st International Conference on Software Engineering, pp. 265–275. IEEE Computer Society (2009)
35. Mohammadi, N.G., Alebrahim, A., Weyer, T., Heisel, M., Pohl, K.: A framework for combining problem frames and goal models to support context analysis during requirements engineering. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CD-ARES 2013. LNCS, vol. 8127, pp. 272–288. Springer, Heidelberg (2013)
36. Beckers, K., Faßbender, S., Heisel, M., Paci, F.: Combining goal-oriented and problem-oriented requirements engineering methods. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CD-ARES 2013. LNCS, vol. 8127, pp. 178–194. Springer, Heidelberg (2013)
37. Conejero, J.M., Hernandez, J., Jurado, E., van den Berg, K.: Mining early aspects based on syntactical and dependency analyses. Sci. Comput. Program. **75**, 1113–1141 (2010)
38. Lencastre, M., Moreira, A., Araújo, J., Castro, J.: Aspects composition in problem frames. In: Proceedings of the 16th IEEE International Requirements Engineering Conference, pp. 343–344. IEEE Computer Society (2008)