Aspect-Oriented Requirements Engineering with Problem Frames

Stephan Faßbender, Maritta Heisel and Rene Meis
paluno - The Ruhr Institute for Software Technology University of Duisburg-Essen, Germany
{*firstname.lastname*}*@uni-due.de*

Keywords: Early Aspect, Problem Frames, Requirements Engineering

Abstract: Nowadays, the requirements of various stakeholders for a system do not only increase the complexity of the system-to-be, but also contain different cross-cutting concerns. In such a situation, requirements engineers are really challenged to master the complexity and to deliver a coherent and complete description of the system-to-be. Hence, they are in need for methods which reduce the complexity, handle functional and quality requirements, check completeness and reveal interactions, and are tool supported to lower the effort. One possible option to handle the complexity of a system-to-be is the separation of concerns. Both, aspect-oriented requirements engineering and the problem frames approach implement this principle. Therefore, we propose a combination of both, the AORE4PF (Aspect-Oriented Requirements Engineering for Problem Frames) method. AORE4PF 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. AORE4PF provides tool support for most activities. We exemplify our method using a smart grid case obtained from the NESSoS project. For validation, the results of a small experiment in the field of crisis management systems are presented.

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 (Hofmann and Lehner, 2001). 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 (Cavusoglu et al., 2004; Khansa et al., 2012), 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 (Boehm and Papaccio, 1988; Willis, 1998). 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 (Firesmith, 2003), even more when considering complex systems.

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* (Parnas, 1972). 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 (Parnas, 1972).

Both, aspect-oriented requirements engineering and the problem frame approach implement this principle, but for different reasons. The approach of *AORE (aspect-oriented requirements engineering)*, 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* (Jackson, 2001) 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 Section 3, we briefly 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 Section 4. A small experiment for validation is presented in Section 5. Work related to this paper is discussed in Section 6. Finally, Section 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" (Kreutzmann et al., 2011) provided by the German Federal Office for Information Security[2], and "Requirements of AMI (Advanced Multi-metering Infrastructure") (OPEN meter project, 2009) 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).

For the rest of the paper, we have chosen a small selection of requirements to exemplify 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 smart meter 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 (Jackson, 2001), 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 (Hatebur and Heisel, 2010). 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 2 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., ≪causalDomain≫ 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 Figure 2 the notation LMN!{forwardData} means that the phenomenon in the set {forwardData} is controlled by the domain LMN and observable by the machine domain SMGReceiver, which is connected to it. These interfaces are represented as associations, and the name of the associations contain the phenomena and the domains controlling the phenomena.

In Figure 2, the lexical domain TemporaryMeterData is constrained and the SmartMeter is referred to, because the machine SMGReceiver has the role to request meter data from the SmartMeter and store the response as TemporaryMeterData for satisfying requirement R1. These relationships are modeled using dependencies that are annotated with the corresponding stereotypes.

The full description for Figure 2 is as follows: The causal domain SmartMeter controls the sendData command, which is forwarded by the LMN and finally observed by the machine domain SMGReceiver. The SMGReceiver controls the phenomenon writeTemporaryData, which stores the received information in the lexical domain TemporaryMeterData. Additionally, the SMGReceiver can requestData which is forwarded by the LMN to the SmartMeter. The connection domain LMN forwards the data and requests it observes. The requirement R1 constrains the TemporaryMeterData and refers to the *SmartMeter*.

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 instances for its domains, phenomena, and interfaces. The UML4PF framework provides tool support for this approach. A more detailed description can be found in (Côté et al., 2011).

4 Method

An illustration of our method is given in Figure 1. 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 Section 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*

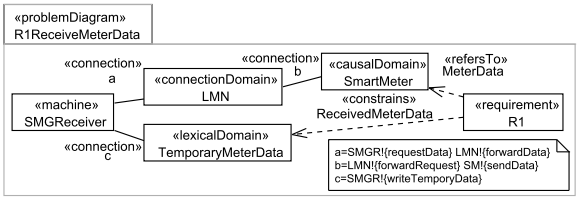

Figure 2: Problem Diagram R1: Receive meter data

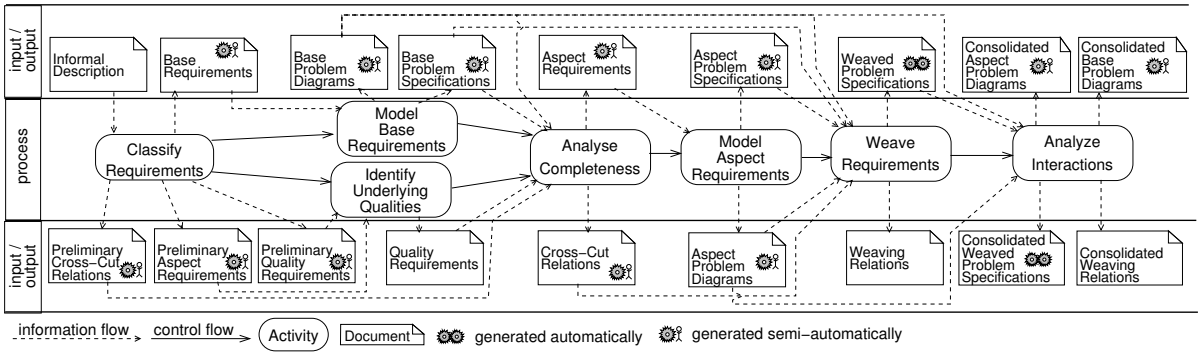

Figure 1: The AORE4PF method

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 (Moser et al., 2011) 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 should provide functionality to prevent eavesdropping. [...]

(`R29A`) Privacy [...] 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.

(`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 (Sampaio et al., 2007), Theme/Doc (Baniassad and Clarke, 2004) or REAs-

sistant[4] can aid the requirements engineer. This way we identify the following two aspects:

(R28B) 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.

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 smart meter 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*. If a requirement is separated into a functional requirement (base or aspect) and a quality, then we add a cross in the corresponding cell of the table. In Table 1, we documented that the aspect R28B is related to the quality R28A. If functional requirements are separated into base and aspect requirements, then we also add respective crosses. In Table 1, we documented that the aspect R30 cross-cuts the base requirements R17A and R18A. Note that everything given in **bold** is discovered later on in the annotated step (^{x}).

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

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

Table 1: Requirements (Cross-Cut) Relation Table for the Smart Grid Scenario

		Quality				Aspect	
		R28A	R29A	R29B	**R31**	R28B	R30 (R17B, R18B)
Base	R1	**X**[4]	**X**[4]	**X**[4]	**X**[4]	**X**[4]	**X**[4]
	R17A	**X**[4]			**X**[3]	**X**[4]	*X*
	R18A				**X**[3]		*X*
Quality	R28A		**I**[7]	**I**[7]	**I**[7]	*X*	
	R29A	**I**[7]		**I**[7]	**I**[7]	**X**[4]	
	R29B	**I**[7]	**I**[7]		**I**[7]	**X**[4]	**X**[4]
	R31	**I**[7]	**I**[7]	**I**[7]			**X**[3]

by the problem frames approach introduced in Section 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 R1 is shown in Figure 2 and explained in Section 3. Figure 3 shows the problem diagram for R17A. The biddable domain AuthorizedExternalEntity controls the updateFirmware command, which is observed by the connection domain WAN. The SMGFirmwareStorage controls the phenomenon storeNewFirmware, which is observed by the lexical domain FirmwareUpdate. The WAN forwards the firmware update it observes. The requirement R17A (for a textual description see Section 2) constrains the FirmwareUpdate and refers to the AuthorizedExternalEntity.

For every problem diagram, we have to provide a reasoning, called *frame concern* (Jackson, 2001), why the *specification* of the submachine together with the knowledge 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 (Jackson and Zave, 1995) can be used for this task. Figure 4 shows the sequence diagram for the sub-problem New firmware. The interaction is started by an AuthorizedExternalEntity causing the phenomenon updateFirmware (requirement). This request is forwarded via the WAN to

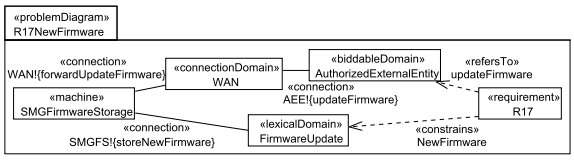

Figure 3: Problem Diagram R17A : New firmware

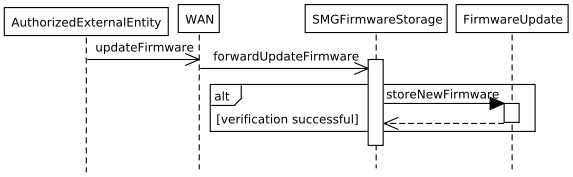

Figure 4: Sequence diagram for R17A

the sub-machine SMGFirmwareStorage (domain knowledge). The sub-machine then performs a verification on the received firmware update (specification). In the case of a successful verification, the new firmware is stored in the lexical domain FirmwareUpdate (specification). Hence, the gateway accepts new firmware from authorized external entities (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 (Rago et al., 2013) 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.

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 (Beckers et al., 2014), the law (identification) pattern method (Faßbender and Heisel, 2013) provides guidance for compliance, security is covered by the PresSuRE method (Faßbender et al., 2014), 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. The already existing aspect requirements are sufficient to cover the newly found relations. This information is used to update the cross-cut relation table (see Table 1).

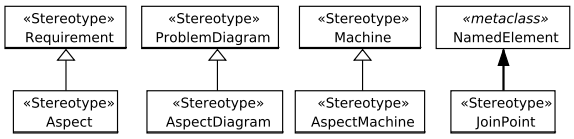

Figure 5: AORE4PF UML Profile

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. The AORE4PF UML profile is shown in Figure 5. 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 metaclass NamedElement. During the weaving, join points are instantiated with domains of the problem diagrams the aspect cross-cuts.

To create an aspect diagram, we have to identify

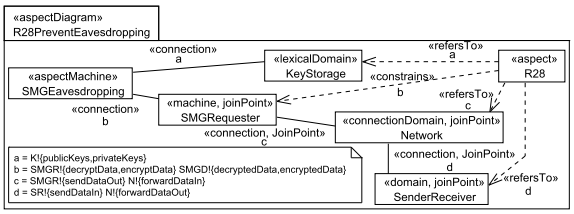

Figure 6: Aspect diagram for aspect requirement R28

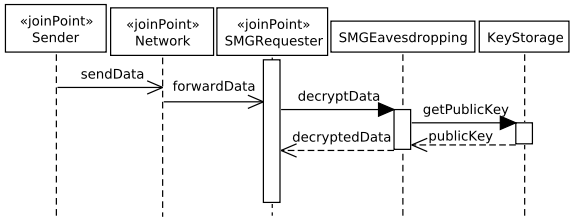

Figure 7: Sequence diagram for aspect requirement R28

the join points which are necessary to combine the aspect with the base 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 diagram 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. We have to define appropriate interfaces between the aspect machine and the identified join points, so that the aspect can be combined with all base requirements in a uniform way. 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 a sequence diagram for each aspect diagram.

For reasons of space, we will only discuss the aspect requirement R28 in detail. R28 covers the prevention against eavesdropping attacks in the smart grid scenario by encrypting all communication. The corresponding aspect diagram is presented in Figure 6. It contains the aspect machine SMGEavesdropping, which has access to the public and private keys for the encryption and decryption. The keys are stored in the KeyStorage. Furthermore, the aspect diagram contains three domains as join points. The machine SMGRequester will be instantiated by a problem machine and the interface b describes that the problem machine is able to send a request to SMGEavesdropping in order to decryptData and encryptData. SMGEavesdropping returns decryptedData and encryptedData, respectively. The join points Network and SenderReceiver are added to refer to the network an attacker may eavesdrop and to the sender or receiver of the data to be transmitted. Hence, we can have two different scenarios for the aspect of preventing eavesdropping. The first scenario is concerned with the decryption of data a problem machine received from a sender, and the second scenario is concerned with the encryption of data a problem machine shall send to a receiver. The sequence diagram for the first scenario is shown in Figure 7. The sequence diagram shows that when the machine SMGRequester receives data from some sender over a network, it asks the aspect machine SMGEavesdropping to decrypt the received data (aspect requirement). Then the aspect machine retrieves the needed public key from the KeyStorage to decrypt the data in order to send it back to the machine SMGRequester (specification). The sequence diagram for the second scenario is built in a similar way, but left out due to space limitations.

Note that the modularization of the prevention against eavesdropping into an aspect allows us to easily change the encryption mechanism if a better and newer mechanism is available, so that we are able to encrypt communications and data by the safest and best encryption mechanisms possible.

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 relation. 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).

The cross-cut relation (see Table 1) is 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. We create a table for each base requirement that defines how the aspects are integrated into the base problem. A row in the table consists of a message of the sequence diagram of the base requirement, a keyword, and the aspect that shall be weaved into the requirement. For this paper, we use the keywords *before* and *after*, but we plan to extend this list, to provide more possibilities to integrate aspects into problems. The *keyword* describes whether the *aspect* has to be integrated before or after the *message*. If multiple aspects shall be integrated before or after a message, we have to define the order in which the aspects are integrated. Table 2 shows the refined cross-cut relation for base requirement R17A.

Because of the aspect requirement R28 all commu-

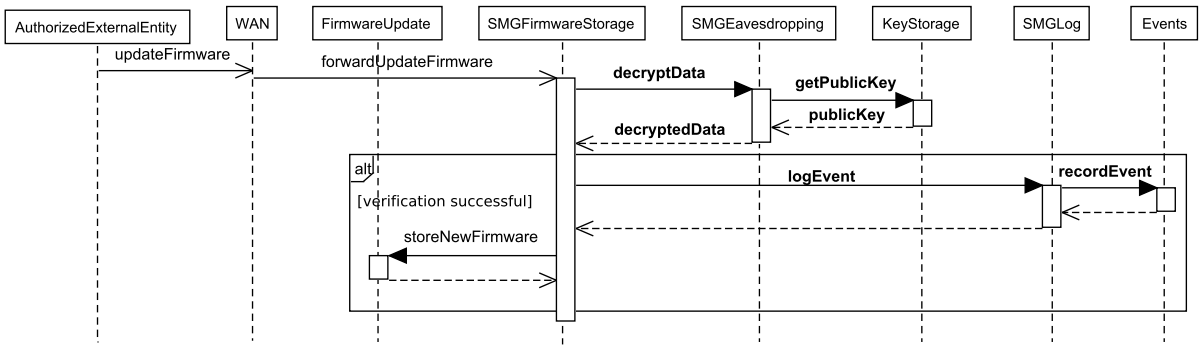

Figure 8: Weaved sequence diagram for R17A

Table 2: Refined Cross-cut relation table for R17A

Message	Key	Aspect Sequence Diagram
forward-Update-Firmware	after	Eavesdropping[WAN/Network, Authorized-ExternalEntity/Sender, SMGFirmware-Storage/SMGRequester]
storeNew-Firmware	before	Logging[SMGFirmwareStorage/SMG-Requester]

nications have to be encrypted to prevent eavesdropping attacks. This implies that all external messages that a sub-machine receives have to be decrypted. For R28 we express this in the first row of Table 2. The row defines that *after* the message forwardUpdate-Firmware was received by the sub-machine the sequence diagram Eavesdropping of aspect requirement R28 shall be integrated. Additionally, it is defined that the join points Network, Sender, and SMGRequester are instantiated with the domains WAN, AuthorizedExternalEntity, and SMGFirmwareStorage. In this way, we addressed the concern that external messages have to be decrypted in the base requirement R28.

The refined cross-cut relations are used to automatically generate the weaving sequence diagrams from the sequence diagrams of the problem and aspect diagrams. These automatically generated sequence diagrams have then to be adjusted, such that the overall behavior satisfies the weaving requirement. The generated sequence diagram is shown in Figure 8. For the sake of readability, we highlighted the messages from the integrated aspect specifications using a bold font. The other messages are the messages from the problem specification, which form the basis of the weaving sequence diagram. In accordance with Table 2, the sequence diagrams Eavesdropping is integrated after the message forwardUpdate-Firmware and the sequence diagrams Logging before the message storeNewFirmwareBase. The latter addresses the concern that the event of a successful firmware verification shall be logged (aspect requirement R17B). For the presented example, we do not need further adjustments because it already addresses the combination of the base requirement with its aspect requirements adequately.

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 (Alebrahim et al., 2014b) functional requirements are treated, and (Alebrahim et al., 2014a) describes how to analyze quality requirements for interactions. Both works use the smart grid as a case study. Hence, we re-used the methods and results also for this work. The results are documented in Table 1 using bold **I**.

5 Validation

To validate our method, we applied it to the crisis management system (CMS) (Kienzle et al., 2010) 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 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 hours 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 (Moser et al., 2011) or EA-Miner (Sampaio et al., 2007) require some additional input like training documents or an existing ontology. But unfortunately, such inputs

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

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

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%**

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 spend on the modeling steps are unavoidable even when using another method or notation. And 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 (Alebrahim et al., 2014b) 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. (Landuyt et al., 2010) report an effort spent for their method of 170 hours for the requirements engineering related activities,

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 pub-

Table 5: Aspects identified

	Landuyt et al., 2010	Mussbacher et al.,2010
Found and separated	83%	75%
Found and not separated	17%	25%
Not Found	0%	0%
Not Mentioned	38%	25%

lications considering requirements engineering using the same scenario (Landuyt et al., 2010; Mussbacher et al., 2010). The result of the comparison is shown in Table 5. Overall, our results are very similar. The result of applying our method includes all requirements which are treated as aspects in the other works. And most of these found requirements are also separated as aspects by our method (Row "Found and separated"). Only a minor number was not treated as aspect (Row "Found and not separated"), but a detailed investigation showed that both views on these requirements are reasonable. The indicator, "Found and separated", computes as ((requirement present as aspect in both documents) / (requirements present as aspect in the publication at hand)). In respect, "Found and not separated", computes as ((requirement present in both documents but not as aspect in our results) / (requirements present as aspect in the publication at hand)). Some of the aspects our method found were not mentioned in the other works (Row "Not Mentioned"). This indicator computes as ((requirements only present in our result) / (all requirements present in our results)). 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 (Kienzle et al., 2010). 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 (Rashid, 2008; Yu et al., 2004; Jacobson and Ng, 2004; Whittle and Araujo, 2004; Sutton and Rouvellou, 2002; Moreira et al., 2005; Grundy, 1999). 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 (Alrajeh et al., 2009). Recently, there were papers which reported a successful integration of goal- and problem-oriented methods (Mohammadi et al., 2013; Beckers et al., 2013). Hence, one might benefit from integrating goal-models in our method.

Conejero et al. (Conejero et al., 2010) 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. (Lencastre et al., 2008) 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 and the problem frames approach. 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.

For future work, we plan to improve the tool support and provide an integrated tool chain. Additionally, we plan to integrate our new model elements into more already existing methods and analyses based on UML4PF to cover a broader spectrum of qualities.

Acknowledgment

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

References

Alebrahim, A., Choppy, C., Faßbender, S., and Heisel, M. (2014a). Optimizing functional and quality requirements according to stakeholders' goals. In *System Quality and Software Architecture*. Elsevier. to appear.

Alebrahim, A., Faßbender, S., Heisel, M., and Meis, R. (2014b). Problem-based requirements interaction analysis. In *Requirements Engineering: Foundation for Software Quality*, volume 8396 of *LNCS*, pages 200–215. Springer.

Alrajeh, D., Kramer, J., Russo, A., and Uchitel, S. (2009). Learning operational requirements from goal models. In *IEEE 31st International Conference on Software Engineering*, pages 265–275. IEEE Computer Society.

Baniassad, E. and Clarke, S. (2004). Finding aspects in requirements with Theme/Doc. In *Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*, pages 15–22. http://trese.cs.utwente.nl/workshops/early-aspects-2004/workshop_papers.htm.

Beckers, K., Faßbender, S., Heisel, M., and Meis, R. (2014). A problem-based approach for computer aided privacy threat identification. In *Privacy Technologies and Policy*, volume 8319 of *LNCS*, pages 1–16. Springer.

Beckers, K., Faßbender, S., Heisel, M., and Paci, F. (2013). Combining goal-oriented and problem-oriented requirements engineering methods. In *Availability, Reliability, and Security in Information Systems and HCI*, volume 8127 of *LNCS*, pages 178–194. Springer.

Boehm, B. W. and Papaccio, P. N. (1988). Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, 14(10):1462–1477.

Cavusoglu, H., Mishra, B., and Raghunathan, S. (2004). The effect of internet security breach announcements on market value: Capital market reactions for breached firms and internet security developers. *International Journal of Electronic Commerce*, 9(1):70–104.

Conejero, J. M., Hernandez, J., Jurado, E., and van den Berg, K. (2010). Mining early aspects based on syntactical and dependency analyses. *Science of Computer Programming*, 75(11).

Côté, I., Hatebur, D., Heisel, M., and Schmidt, H. (2011). UML4PF - a tool for problem-oriented requirements analysis. In *Proceedings of the 19th IEEE International Requirements Engineering Conference*, pages 349–350. IEEE Computer Society.

Faßbender, S. and Heisel, M. (2013). From problems to laws in requirements engineering using model-transformation. In *ICSOFT '13*, pages 447–458. SciTePress.

Faßbender, S., Heisel, M., and Meis, R. (2014). Functional requirements under security pressure. In *ICSOFT '14*. SciTePress. to appear.

Firesmith, D. (2003). Specifying good requirements. *Journal of Object Technology*, 2(4):77–87. http://www.jot.fm/issues/issue_2003_07/column7.

Grundy, J. C. (1999). Aspect-oriented requirements engineering for component-based software systems. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 84–91, Washington, DC, USA. IEEE Computer Society.

Hatebur, D. and Heisel, M. (2010). A UML profile for requirements analysis of dependable software. In *Computer Safety, Reliability, and Security*, volume 6351 of *LNCS*, pages 317–331. Springer.

Hofmann, H. and Lehner, F. (2001). Requirements engineering as a success factor in software projects. *IEEE Software*, 18(4):58–66.

Jackson, M. (2001). *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley.

Jackson, M. and Zave, P. (1995). Deriving specifications from requirements: an example. In *ICSE, USA*, pages 15–24. ACM Press.

Jacobson, I. and Ng, P.-W. (2004). *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley Professional.

Khansa, L., Cook, D. F., James, T., and Bruyaka, O. (2012). Impact of HIPAA provisions on the stock market value of healthcare institutions, and information security and other information technology firms. *Computers & Security*, 31(6):750 – 770.

Kienzle, J., Guelfi, N., and Mustafiz, S. (2010). Crisis management systems: A case study for aspect-oriented modeling. In Katz, S., Mezini, M., and Kienzle, J., editors, *Transactions on Aspect-Oriented Software Development VII*, volume 6210 of *LNCS*, pages 1–22. Springer.

Kreutzmann, H., Vollmer, S., Tekampe, N., and Abromeit, A. (2011). Protection profile for the gateway of a smart metering system. Technical report, BSI.

Landuyt, D., Truyen, E., and Joosen, W. (2010). Discovery of stable abstractions for aspect-oriented composition in the car crash management domain. In *Transactions on Aspect-Oriented Software Development VII*, volume 6210 of *LNCS*, pages 375–422. Springer.

Lencastre, M., Moreira, A., Araújo, J., and Castro, J. (2008). Aspects composition in problem frames. In *Proceedings of the 16th IEEE International Requirements Engineering Conference*, pages 343–344. IEEE Computer Society.

Mohammadi, N. G., Alebrahim, A., Weyer, T., Heisel, M., and Pohl, K. (2013). A framework for combining problem frames and goal models to support context analysis during requirements engineering. In *Availability, Reliability, and Security in Information Systems and HCI*, volume 8127 of *LNCS*, pages 272–288. Springer.

Moreira, A., Arajo, J., and Rashid, A. (2005). A concern-oriented requirements engineering model. In Pastor, O. and Falco e Cunha, J., editors, *Advanced Information Systems Engineering*, volume 3520 of *LNCS*, pages 293–308. Springer.

Moser, T., Winkler, D., Heindl, M., and Biffl, S. (2011). Requirements management with semantic technology: An empirical study on automated requirements categorization and conflict analysis. In *Advanced Information Systems Engineering*, volume 6741 of *LNCS*, pages 3–17. Springer.

Mussbacher, G., Amyot, D., Araújo, J., and Moreira, A. (2010). Requirements modeling with the aspect-oriented user requirements notation (AoURN): A case study. In *Transactions on Aspect-Oriented Software Development VII*, volume 6210 of *LNCS*, pages 23–68. Springer.

OPEN meter project (2009). Requirements of AMI. Technical report, OPEN meter project.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058.

Rago, A., Marcos, C., and Diaz-Pace, J. A. (2013). Uncovering quality-attribute concerns in use case specifications via early aspect mining. *Requirements Engineering*, 18(1):67–84.

Rashid, A. (2008). Aspect-oriented requirements engineering: An introduction. In *Proceedings of the 16th IEEE International Requirements Engineering Conference*, pages 306–309. IEEE Computer Society.

Sampaio, A., Rashid, A., Chitchyan, R., and Rayson, P. (2007). Ea-miner: Towards automation in aspect-oriented requirements engineering. In *Transactions on Aspect-Oriented Software Development III*, volume 4620 of *LNCS*, pages 4–39. Springer.

Sutton, Jr., S. M. and Rouvellou, I. (2002). Modeling of software concerns in cosmos. In *Proceedings of the 1st International Conference on Aspect-oriented Software Development*, AOSD '02, pages 127–133, New York, NY, USA. ACM.

Whittle, J. and Araujo, J. (2004). Scenario modelling with aspects. *IEE Proceedings Software*, 151(4):157–171.

Willis, R. (1998). *Hughes Aircraft's Widespread Deployment of a Continuously Improving Software Process*. AD-a358 993. Carnegie-Mellon University.

Yu, Y., Cesar, J., Leite, S. P., and Mylopoulos, J. (2004). From goals to aspects: Discovering aspects from requirements goal models. In *Proceedings of the 12th IEEE International Requirements Engineering Conference*, pages 38–47. IEEE Computer Society.