Aspect Frames – Describing Cross-Cutting Concerns in Aspect-Oriented Requirements Engineering

RENE MEIS and MARITTA HEISEL, paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Duisburg, Germany

Cross-cutting concerns often arise when non-functional requirements are operationalized, because non-functional requirements are themselves cross-cutting. In the field of aspect-oriented requirements engineering (AORE), functional requirements that cross-cut multiple other functional requirements are called aspects. An aspect describes in most cases a solution for a non-functional requirement and how this solution can be integrated into the realization of the functional requirements it cross-cuts. Examples for cross-cutting concerns are logging, encryption, and access control. We observed that aspects often share a basic structure, behavior, and the way of how they have to be integrated into the realization of the functional requirements they cross-cut. We propose in this paper aspect frames. An aspect frame is a kind of pattern for aspects that share a common concern, behavior, and way how they are integrated into the realization of the functional requirements they cross-cut. These aspect frames support requirements engineers to describe concrete aspects that fit to an aspect frame.

CCS Concepts: •**Software and its engineering** → **Patterns;** *Designing software;*

Additional Key Words and Phrases: Patterns, Aspect-Oriented Requirements Engineering, Problem Frames

ACM Reference Format:

Rene Meis and Maritta Heisel. 2017. Aspect Frames – Describing Cross-Cutting Concerns in Aspect-Oriented Requirements Engineering. *EuroPLoP* (July 2017), 28 pages.
DOI: 3147704.3147732

1. INTRODUCTION

The concept of aspect-orientation has its roots in aspect-oriented programming [Kiczales et al. 1997]. It was introduced as a means to decouple often reoccurring code that *cross-cuts* several other functionalities into so called *aspects*. Prominent examples for aspects are the logging of events, encryption and decryption, and access control. All these functionalities have to be considered during multiple other operations. An aspect consists of the reoccurring code and a definition of a pointcut (that consists of join points). The pointcut describes under which circumstances the aspect's code shall be called during the execution of the software. A pointcut can, e.g., define that an aspect shall be executed before or after a specific operation call (join point) at run-time. The interpreter or compiler of the aspect-oriented programming language automatically integrates an aspect's code if a code part is executed that matches the aspect's defined pointcut. This process is called *weaving*.

Aspect-orientation was also transferred into the requirements engineering domain and is called there aspect-oriented requirements engineering (AORE) [Moreira et al. 2013]. In AORE, aspects are in most cases cross-cutting requirements. That is, aspects are requirements that are part of several

EuroPLoP '17, July 12-16, 2017, Irsee, Germany

ACM ISBN 978-1-4503-4848-5/17/07...$15.00
https://doi.org/10.1145/3147704.3147732

other requirements or related to several other requirements. The first step in AORE is to separate the requirements into *base requirements* and *aspects*. Additionally, we need to document which base requirements are cross-cut by which aspects to ensure that these are later correctly weaved.

In this paper, we introduce aspect frames as a means to support the description of cross-cutting concerns in AORE. An aspect frame describes a class of aspects that share a common concern and behavior. For example, encryption, anonymization, filtering, compression, and correction mechanisms have in common that these often need to be applied on data before these are transmitted to a specific receiver. From these examples, we derived the Transform Before Transmission Aspect frame (introduced in Section 4.3) that represents the class of all aspects that perform a transformation on data before these can be transmitted to a specific receiver. We observed the aspect frames during our research on the presentation of privacy enhancing technologies (PETs) as aspects [Meis and Heisel 2017].

We propose a pattern format to uniformly represent aspect frames as patterns. The idea of aspect frames is similar to Jackson's problem frames [Jackson 2001]. Namely, the definition of a class of software development problems that share a common structure and behavior. This is the reason why we use our extension of the problem frames approach for aspect-orientation [Faßbender et al. 2015] to describe aspect frames. Note that we have observed the aspect frames only in our own research on the presentation of PETs as aspects and not yet in other work. Hence, we do not claim that aspect frames are patterns. They are rather pattern candidates.

The rest of the paper is structured as follows. Section 2 presents Jackson's problem frames and aspect-oriented requirements engineering for problem frames as background. The pattern format for aspect frames is introduced in Section 3. Section 4 presents the four aspect frames Decision Aspect (Section 4.1), Transform Received Data Aspect (Section 4.2), Transform Before Transmission Aspect (Section 4.3), and Independent Behavior Aspect (Section 4.4). Guidance for the creation and usage of aspect frames is given in Section 5. Section 6 discusses related work, while Section 7 concludes the paper.

2. BACKGROUND

2.1 Problem Frames

The problem frames approach proposed by Jackson [2001] is a methodology for the analysis of software requirements. Jackson states that a problem frame is a kind of pattern. A problem frame describes a basic problem that exists in an environment described by a functional requirement Ⓡ that the machine ▯ (the system-to-be) shall address. The environment is structured in domains. A domain can either be biddable Ⓑ (a human), causal Ⓒ (a technical devices with a predictable behavior), or lexical Ⓧ (a physical representation of data).

Figure 1 shows the simple workpieces problem frame. It shows the basic elements of which a problem frame consists. On the right side, we see the functional requirement Command effects (oval) that describes the desired behavior of the environment. On the left, we see the machine Editing Tool (rectangle) that shall be implemented in a way that the desired behavior described by the functional requirement Command effects is guaranteed. In the middle, we see the relevant domains (rectangles) of the environment. For the simple workpieces problem, these are the biddable domain User and the lexical domain Workpieces. The solid lines in the problem frame show the interfaces that exist between the domains. An interface contains *phenomena* (e.g., events, commands, or data) shared between the involved domains. Each phenomenon is controlled by exactly one of the domains. This is indicated, e.g., by the notion U!E3, where U is the abbreviation of the domain User that controls (!) the phenomena of the set E3. E3 contains the commands that the user can issue and the editing tool observes, e.g., commands to request that the editing tool adds, deletes, or modifies a workpiece. Similarly, E1 contains the com-

mands issued by the Editing Tool and observed by the Workpieces. Y2 contains symbolic phenomena, e.g, data and states, of the Workpiece that are observable by the Editing Tool. A dashed line between a requirement and a domain expresses that the requirement references phenomena of that domain. If the dashed line has an arrow head, this means that the desired behavior of the environment described by the requirement constrains the behavior or state (phenomena) of that domain. The requirement Command effects refers to the phenomena E3 controlled by the User and constrains the symbolic phenomena Y4 of the Workpieces. Note that the sets of phenomena constrained by the requirement (Y4) and the symbolic phenomena observable by the machine (Y2) may differ from each other. That is, there can be a gap between the phenomena a requirement refers to or constrains and the phenomena the machine is actually able to observe and control.

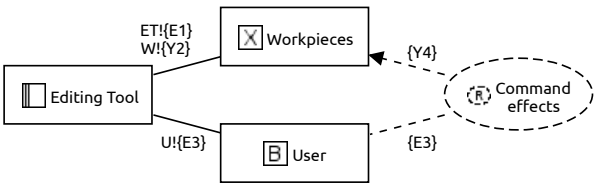

Fig. 1. Simple workpieces frame (based on [Jackson 2001])

In Jackson's problem frames approach the overall problem of building the machine is decomposed into simpler subproblems until these subproblems fit to a problem frame. An instance of a problem frame is called problem diagram.

2.2 Aspect-Oriented Requirements Engineering for Problem Frames

We described a method to elicit cross-cutting concerns and to systematically integrate them into base requirements of a system-to-be in previous work [Faßbender et al. 2014; Faßbender et al. 2015]. The base requirements are modeled in problem diagrams and additionally, we create for each problem diagram a sequence diagram that describes how the machine of the diagram shall lead to the satisfaction of the corresponding requirement.

The cross-cutting functional requirements (also called aspects) are modeled in aspect diagrams. An aspect diagram is similar to a problem diagram, but instead of a (base) functional requirement it contains a cross-cutting functional requirement, also called aspect. Additionally, an aspect diagram contains at least one join point. Join points are placeholders for domains and interfaces of the base problems they can be integrated into. For each aspect diagram, we also create sequence diagrams that describe how the aspect's machine satisfies its cross-cutting requirement.

The join points are instantiated with domains and interfaces of the base problem when the aspect is integrated into the base problem. This integration is also called *weaving*. The weaving of the aspect's sequence diagram into the base problem's sequence diagram can be performed systematically [Faßbender et al. 2015].

An example for a cross-cutting concern is the authorization of users that want to access specific resources. This authorization may be necessary in multiple base functionalities and is hence cross-cutting. Figure 2 shows a typical solution for the non-functional requirement authorization. This solution is access control based on an Access Policy. The machine Access Control provides an interface to Base Machines to check whether Users are allowed to request a resource. Base Machine, User, and the interface between them are the join points of the aspect. We indicate the join points using gray background for domains and thick lines for interfaces. The join points indicate that this aspect can

be integrated into all subproblems in which the Base Machine receives requests of a User and has to decide whether the User is authorized to perform the request. This decision is delegated to the aspect's machine Access Control that decides whether the User is authorized or not based on the Access Policy that is available to the machine. The cross-cutting functional requirement Check Access Rights refers to the request of the User and the Access Policy, and constrains the Base Machine to only provide the requested resource to the User if the Access Policy allows this.

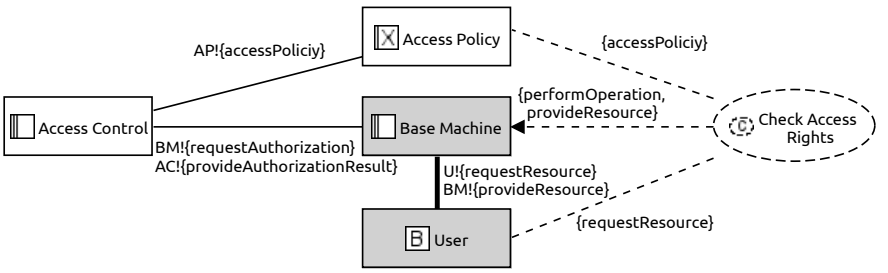

Fig. 2. Aspect diagram for an access control mechanism

In this paper, we introduce aspect frames. These aspect frames are patterns for functional cross-cutting concerns (aspects), similar to Jackson's problem frames that distill the essence of basic re-occurring problems during software development. The access control aspect shown in Fig. 2 is an instance of the Decision Aspect frame introduced in Section 4.1.

3. THE PATTERN FORMAT FOR ASPECT FRAMES

In this section, we describe the pattern format that we use to describe aspect frames. We base our pattern format on the suggestions of Wellhausen and Fießer [2011]. For the sections Context and Solution, we introduce subsections that are tailored to present aspect frames. In the following, we show and discuss the sections of the pattern format. We refer to instances of aspect frames as aspects.

Name. The name of the aspect frame

Context.

The context of an aspect frame describes the *base problems* into which it shall be integrated. We propose to present on the one hand a structural view on the base problem and the relevant behavior of the aspect.

Structure.

The structure of a base problem is described by a (possibly incomplete) problem frame. This problem frame has to contain the core domains of a problem into which instances of the aspect frame might be integrated into. We permit that the domain types in the context description is left open. That means that the context is not limited to base problems with domains of a specific type. Additionally, it is allowed that a base problem contains more domains than described by the provided kind of problem frame. This is, the provided problem frame does not need to be complete.

Behavior.

The relevant behavior of the base problem is described in a sequence diagram. This sequence diagram describes in most cases the behavior that needs or leads to the integration of an aspect.

Problem.
A short phrase that describes the problem of the base requirement that is addressed by aspects fitting to the aspect frame.

Forces.
A list of possibly conflicting properties that make it hard to address the aforementioned problem existing in the base problem.

Solution.
The solution contains the actual aspect frame in the sense of Jackson's problem frames. Similar to the context, we divide the description of the solution into subsection that explain its structure, behavior, and how it is integrated into the base problem.

Structure.

We describe the basic structure of the aspect frame as a high-level aspect diagram that contains as join points the parts of the base problem that are relevant for the functionality of the aspect and the domains that the aspect newly introduces to address the problem. We allow that domain types are left out in the aspect frames. This leads in most cases to the definition of variants of the aspect frame (see *Variants.*).

Behavior.

The behavior part shows when and how the machine of the base problem will integrate the aspect and the aspect's behavior to achieve the needed solution of the problem using a sequence diagram.

Integration.

This part describes how the aspect's behavior shall be integrated into the base problem, by combining the sequence diagrams of the base problem and the aspect frame.

Variants.

This section is optional. In some cases, domains of the aspect frame are left out and can have different types while the underlying concern of the aspect frame remains the same. In these cases, we permit to describe variants that assign specific domain types to the general domains of the aspect frame. These variants can then be referenced in the consequences to be more precise on the benefits and liabilities of the different variants.

Formally, the variants are candidates for separate aspect frames, but we decided to allow the descriptions of these in one pattern if they share the same context, problem, and solution.

Consequences.
The positive and negative impact of the integration of the aspect frame into the base problem shall be discussed in the consequences section. The consequences shall be discussed based on the previously defined forces for all described variants of the aspect frame.

Benefits.

The positive consequences of instances of the aspect frame.

Liabilities.

The negative consequences of instances of the aspect frame.

Examples.
Known aspects that are instances of this aspect frame. If variants are defined, then examples for all variants shall be given.

4. THE ASPECT FRAMES

4.1 Decision Aspect Frame

Name. Decision Aspect

Context.

Structure.

You have base problems whose behavior shall depend on specific properties of the received request. The base problem's essential structure is shown in Fig. 3. It consists of a Base Machine that is concerned with handling requests of a domain Requester concerning a Resource that the Base Machine manages.

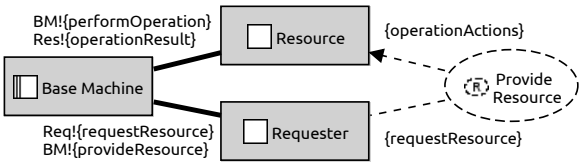

Fig. 3. The base problem of Decision Aspects

Behavior. Figure 4 shows the relevant behavioral view on the base problem. It consists of a *before* behavior, the request of the Requester to the Base Machine, and an *after* behavior.

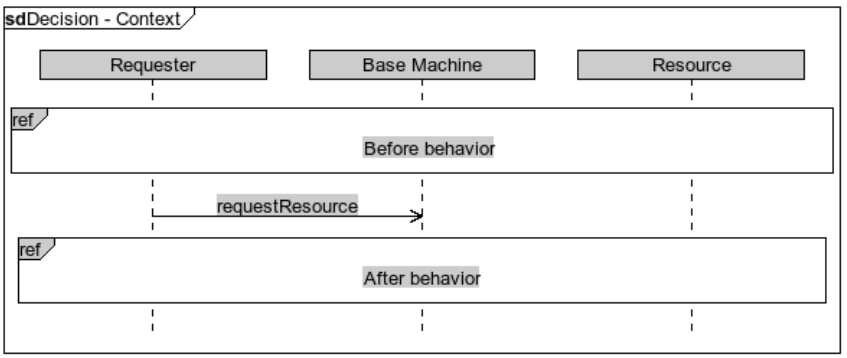

Fig. 4. Relevant behavior of the base problem of Decision Aspects

Problem.

You want to integrate a mechanism that decides based on the Requester's request whether the after behavior shall occur or not.

Forces.

—*Complexity of the implementation:* The complexity to implement the mechanism should be kept to the minimum needed.

—*Integration into the base problem:* It shall be possible to integrate the decision mechanism in a way that the base problem is only affected in the desired manner.

—*Response time of decision:* The selected mechanism shall respond in a timely manner depending on the needs of the base problem.

—*Reliability of decision:* The selected mechanism shall provide a reliable decision.

—*Security of decision process:* It shall be as hard as possible to by-pass the decision mechanism by malicious requesters.

Solution.

Structure.

A mechanism to decide whether a base machine shall initiate the after behavior or not has the basic structure shown in Fig. 5. Figure 5 shows the Requester and the Base Machine as the relevant parts of the base problem (join points) for the Decision Aspect. The Decision Machine is the domain that provides a decision based on additional information provided by an Information Source to the Base Machine. The Base Machine then uses this decision to determine whether it shall proceed with its behavior or not. The cross-cutting requirement Provide Decision refers to the request of the Requester and the used Information Source as basis to derive the decision, and it constrains the Base Machine to consider the provided decision. Note that the provided decision does not need to be Boolean. However, it shall be in a form that is easily accessable by the Base Machine without much further (computational) effort.

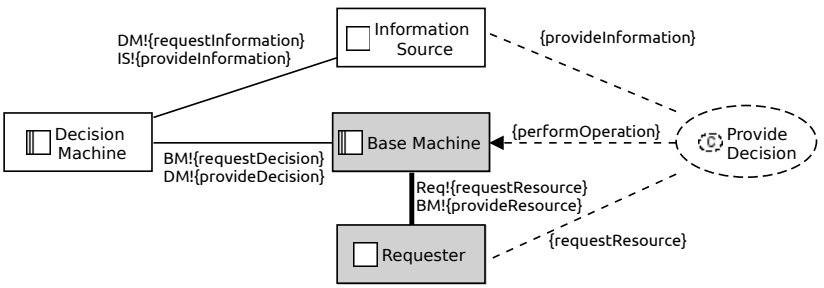

Fig. 5. The structure of Decision Aspects

Behavior.

The behavioral part of the solution is shown in Fig. 6. The interaction is started by the join point Requester that requests a resource from the join point Base Machine. The Base Machine then requests a decision for this request from the aspect's machine Decision Machine. The Decision Machine provides a decision based on additional information that it retrieves from an Information Source beforehand.

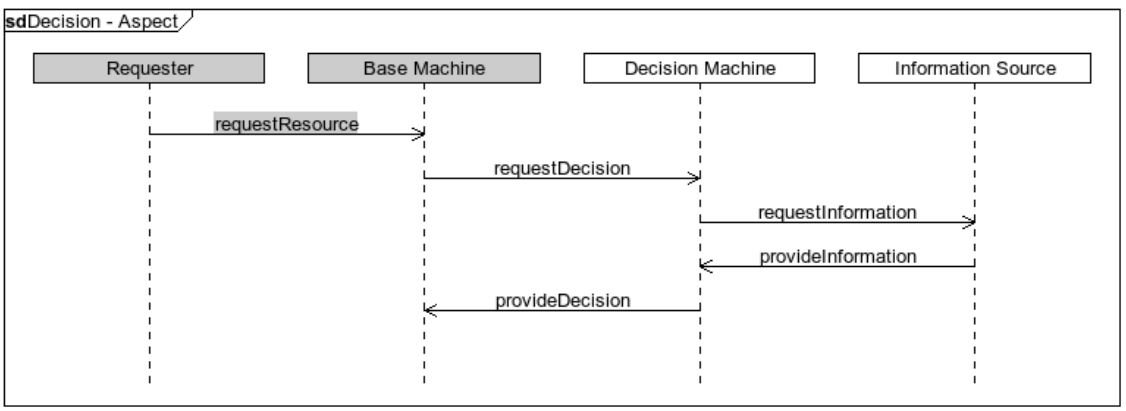

Fig. 6. Behavior of Decision Aspects

Integration.
How a Decision Aspect is integrated into a base problem is shown in Fig. 7. First, the Before behavior of the base problem (cf. Fig. 4) happens. In reaction to the Requester's request, the aspect's behavior is integrated (cf. Fig. 6). Based on the provided decision the base machine has now to decide whether the base problem's After behavior is executed or not. Optionally, the base problem could also be extended to perform an Error behavior. Note that the message provideDecision of the Decision Machine could provide helpful information for the error behavior, for example, the information why the decision was not positive.

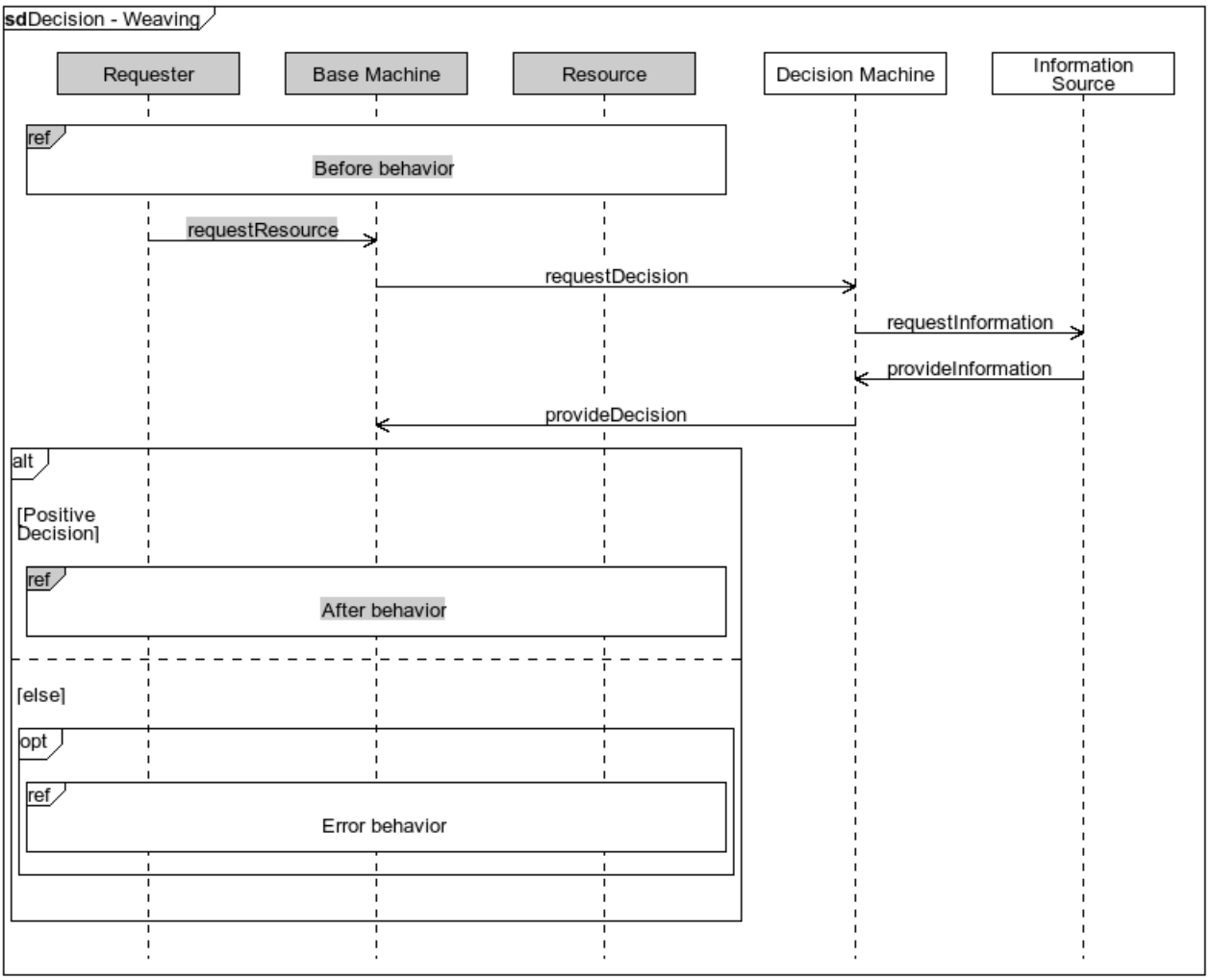

Fig. 7. Integration of Decision Aspects

Variants.
We can distinguish three variants of Decision Aspects depending on the type of the information source.

Data-based decision	*Service-based decision*	*Human-based decision*
If the information source is lexical, this means that the decision machine has to compute the decision based on the data contained in this lexical domain.	If the information source is causal, this means that an external service is used that provides necessary information to derive the decision.	If the information source is biddable, this means that the decision machine needs input of a human operator to provide a decision.

Consequences.

Benefits.

—*Complexity of the implementation:*

Data-based decision	*Service-based decision*	*Human-based decision*
-	As the decision making is outsourced, the implementation is only concerned with the correct usage of the service's API.	In general, we do not expect complex algorithms to be implemented if the decision making is based on humans.

—*Integration into the base problem:*

All: -

—*Response time of decision:*

Data-based decision	*Service-based decision*	*Human-based decision*
The response time of data-based decisions benefits from the local availability and automated evaluation of the decision.	The computation of the decision is delegated to an external service. This can be an advantage if the computational capabilities of the machine are limited and it is not reasonable for it to manage the decision itself.	-

—*Reliability of decision:*

Data-based decision	*Service-based decision*	*Human-based decision*
The decision is under the control of the machine. Hence, its reliability is controllable by it.	A certified external service that is specialized on providing the needed decision can be considered as reliable.	Trained humans are in several situations the only option for reliable decisions. Especially in cases where the decision is based on requests whose semantics are hardly machine processable.

—*Security of decision process:*

Data-based decision	*Service-based decision*	*Human-based decision*
The decision is under the control of the machine. Hence, its security is controllable by it.	A certified external service that is specialized on providing the needed decision can provide the needed security.	Well trained personnel may be harder to be by-passed than automatic processes.

Liabilities.

—*Complexity of the implementation:*

Data-based decision	*Service-based decision*	*Human-based decision*
We have to manage the lexical information source that, e.g, can be a database. The management of the lexical information source can introduce additional base problems that are concerned with adding, changing, or deleting information from the source. Depending on the mechanism, complex algorithms have to be implemented to derive the decision.	It has to be ensured that a reliable connection to the service exists and that its API is suitable for the needed purposes and correctly used.	A user interface has to be created for the humans that act as information source. This interface has to provide the humans all information that they need to take the decision and shall be usable.

—*Integration into the base problem:*

All: The phenomenon requestResource of the base problem has in most cases to be enhanced in a way that it also transmits the information that is needed to decide whether the base machine shall perform the *after* behavior or not. The base machine has then to consider the provided decision. Optionally, an additional error behavior has to be implemented in the case of a negative decision.

—*Response time of decision:*

Data-based decision	*Service-based decision*	*Human-based decision*
The decision machine needs to have sufficient computational resources to compute the decision based on the information source in a timely manner.	To provide a timely decision, the external service has to have an appropriate response time and has to be available.	The response time depends on the availability of the human resources, their efficiency, and the number of requests that they have to handle. In general, we have to expect significant higher response times than for completely automatic decisions.

—*Reliability of decision:*

Data-based decision	*Service-based decision*	*Human-based decision*
The reliability of a data-based decision depends on the correctness of the information source. Hence, the machine has to ensure that it is accurate and up-to-date.	The external service has to be trusted to provide reliable information and decisions.	The humans involved in the decision process have to be trained to be able to provide correct decisions. Depending on the base problem it can be reasonable to involve more than one human in the decision process to improve the reliability of the made decisions.

—*Security of decision process:*

Data-based decision	*Service-based decision*	*Human-based decision*
The information source has to be sufficiently secured against unintended modification.	We have to trust the security claims of the used external service and have to ensure that the communication with the service is secure.	The humans involved in the decision process need to be trained in security to prevent, e.g., social engineering attacks and insecure behavior.

Examples.

Data-based decision	*Service-based decision*	*Human-based decision*
Access control mechanisms, e.g., role-based access control (RBAC) and attribute-based access control (ABAC), are examples for data-based decisions.	Authentication or authorization based on an external service is an example for a service-based decisions. Examples for these mechanisms are “Login with Facebook” or “Pay with Amazon”.	Examples for human-based decisions are moderated internet forums or a report functionality of posts on social media sites. In moderated internet forums a moderator serves as information source and decides whether a post is published or not. For the report functionality of posts on social media sites, one or several humans have to decide whether the report is grounded and which action shall be performed.

4.2 Transform Received Data Aspect Frame

Name. Transform Received Data Aspect

Context.

Structure.

You have base problems in which the Base Machine needs to transform data received or retrieved from a domain Sender to proceed to achieve its requirement (see Fig. 8).

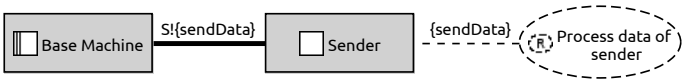

Fig. 8. The base problem of Transform Received Data Aspects

Behavior. Figure 9 shows the relevant behavioral view on the base problem. It consists of a *before* behavior, the send event of the Sender to the Base Machine, and an *after* behavior.

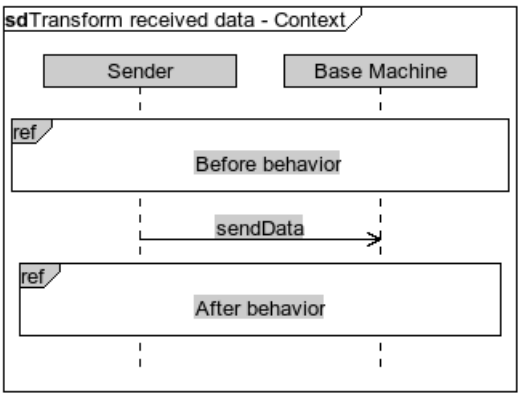

Fig. 9. Relevant behavior of the base problem of Transform Received Data Aspects

Problem.

You want to integrate a mechanism that transforms the data received from the Sender in order to proceed with the processing of that data (after behavior).

Forces.

—*Complexity of the implementation:* The complexity to implement the mechanism should be kept to the minimum needed.

—*Integration into the base problem:* It shall be possible to integrate the transformation mechanism in a way that the base problem is only affected in the desired manner.

—*Response time of transformation:* The selected mechanism shall respond in a timely manner depending on the needs of the base problem.

—*Reliability of transformation:* The selected mechanism shall perform the transformation reliably.

Solution.

Structure.

A mechanism to transform data received from a sender to allow a further processing of this data has the basic structure shown in Fig. 10. Figure 10 shows the Sender and the Base Machine as the relevant parts of the base problem (join points) for the Transform Received Data Aspect. The Transformation Machine is the domain that performs the transformation of the received data using a Transformation Resource (optional) and provides the transformed data to the Base Machine. The Base Machine can then further process the transformed data. The crosscutting requirement Transform received data refers to the sending event of the Sender and the Transformation Resource that is used to perform the transformation, and it constrains the Base Machine to further process the transformed data.

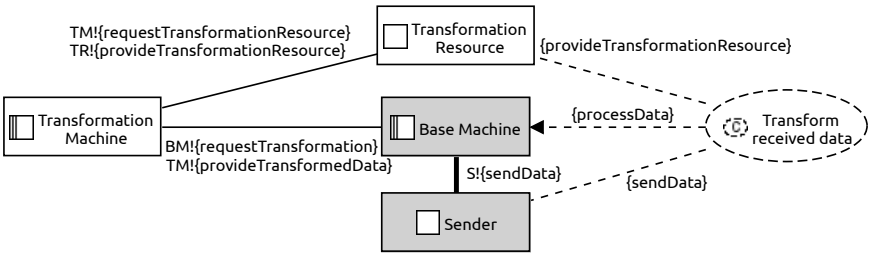

Fig. 10. The structure of Transform Received Data Aspects

Behavior.
The behavioral part of the solution is shown in Fig. 11. The interaction is started by the join point Sender that sends data to the join point Base Machine. The Base Machine then requests the transformation of the received data from the aspect machine Transformation Machine. The Transformation Machine then provides the transformed data that it computed using the Transformation Resource.

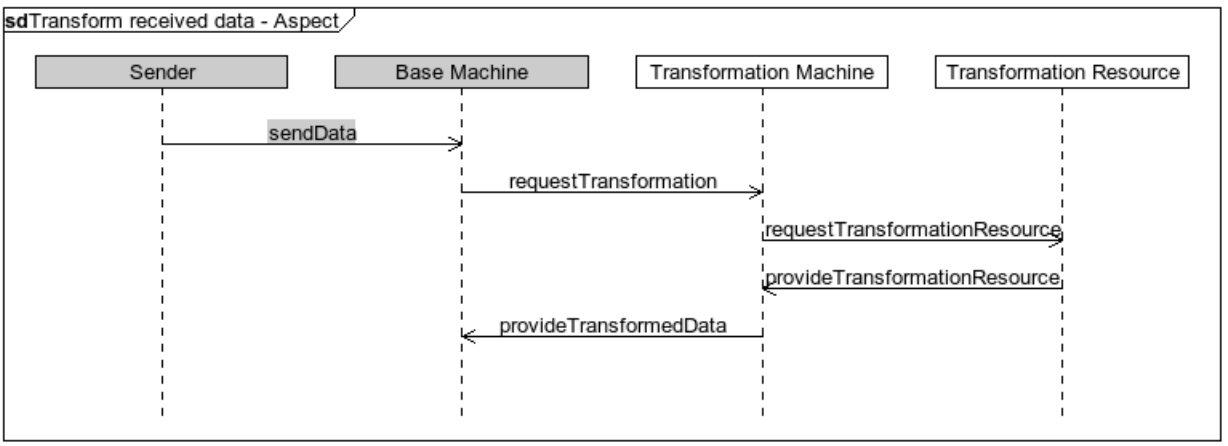

Fig. 11. Behavior of Transform Received Data Aspects

Integration.
How a Transform Received Data Aspect is integrated into a base problem is shown in Fig. 12. First, the Before behavior of the base problem (cf. Fig. 9) happens. After the Sender sends data, the aspects behavior is integrated (cf. Fig. 11). Using the transformed data, the base machine executes the After behavior.

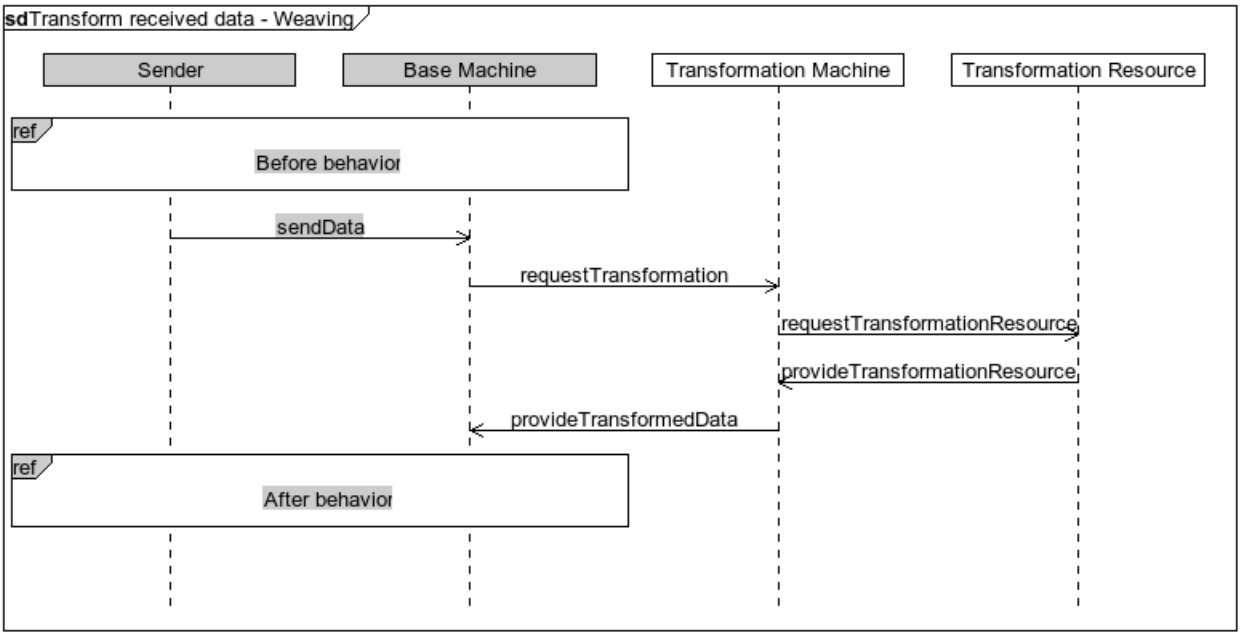

Fig. 12. Integration of Transform Received Data Aspects

Variants.

We can distinguish three variants of transformation aspects depending on the type of the transformation resource.

Data-based transformation	*Service-based transformation*	*Human-based transformation*
If the transformation resource is lexical, this means that the transformation machine uses this information as input for the transformation.	If the transformation resource is causal, this means that an external service exists that is used to perform the transformation.	If the transformation resource is biddable, this means that the transformation machine needs inputs from human operators to transform the received data.

Consequences.

Benefits.

—Complexity of the implementation:

Data-based transformation	*Service-based transformation*	*Human-based transformation*
-	As the transformation is outsourced, the implementation is only concerned with the correct usage of the service's API.	In general, we do not expect complex algorithms to be implemented if the transformation is performed by humans.

—*Integration into the base problem:*

All: -

—*Response time of transformation:*

Data-based transformation	*Service-based transformation*	*Human-based transformation*
The response time of a data-based transformation benefits from the local availability and automatic transformation.	The transformation is delegated to an external service. This can be an advantage if the computational capabilities of the machine are limited and it is not reasonable for it to perform the transformation itself.	-

—*Reliability of transformation:*

Data-based transformation	*Service-based transformation*	*Human-based transformation*
The transformation is under the control of the machine. Hence, its reliability is controllable by it.	A certified external service that is specialized on providing the needed transformation can be considered as reliable.	Trained humans are in several situations the only option for reliable transformations. Especially in cases where the transformation is based on requests whose semantics are hardly machine processable.

Liabilities.

—*Complexity of the implementation:*

Data-based transformation	*Service-based transformation*	*Human-based transformation*
We have to manage the lexical transformation resource that, e.g, can be a database. The management of the lexical transformation resource can introduce additional base problems that are concerned with adding, changing, or deleting information from the resource. Depending on the mechanism, complex algorithms have to be implemented to perform the transformation.	It has to be ensured that a reliable connection to the service exists and that its API is correctly used.	A user interface has to be created for the humans that act as transformation resource. This interface has to provide the humans all information that they need to perform the transformation and shall be usable.

—*Integration into the base problem:*

All: The base machine has to use the provided transformed data instead of the received data for the further processing.

—*Response time of transformation:*

Data-based transformation	*Service-based transformation*	*Human-based transformation*
The transformation machine needs to have sufficient computational resources to compute the transformation based on the transformation resource in a timely manner.	To provide the transformed data timely, the external service has to have a appropriate response time and has to be available.	The response time depends on the availability of the human resources, their efficiency, and the number of requests that they have to handle. In general, we have to expect significant higher response times than for completely automatic transformations.

—*Reliability of transformation:*

Data-based transformation	*Service-based transformation*	*Human-based transformation*
The reliability of a data-based transformation depends on the correctness of the transformation resource. Hence, the machine has to ensure that it is accurate and up-to-date.	The external service has to be trusted to provide reliable transformations.	The humans that perform the transformation have to be trained to be able to provide correct transformations. Depending on the base problem it can be reasonable to involve more than one human in the transformation process to improve the reliability of the made transformations.

Examples.

Data-based transformation	*Service-based transformation*	*Human-based transformation*
Decryption of data received by a sender is an example for a data-based transformation. In this case the transformation resource is a key storage.	An optical character recognition (OCR) or face recognition software could be used in a service-based transformation as information resource to derive information that the base machine needs for a further processing.	Examples for human-based transformations are spell-checking or formatting of provided texts.

4.3 Transform Before Transmission Aspect Frame

Name. Transform Before Transmission Aspect

Context.

Structure.

You have base problems in which the Base Machine needs to transform data (previously retrieved by the base machine) that shall be sent to or stored by a Receiver (see Fig. 13).

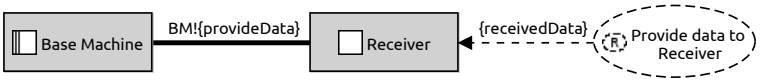

Fig. 13. The base problem of Transform Before Transmission Aspects

Behavior. Figure 14 shows the relevant behavioral view on the base problem. It consists of a before behavior, the event that the Base Machine provides data to the Receiver, and an after behavior.

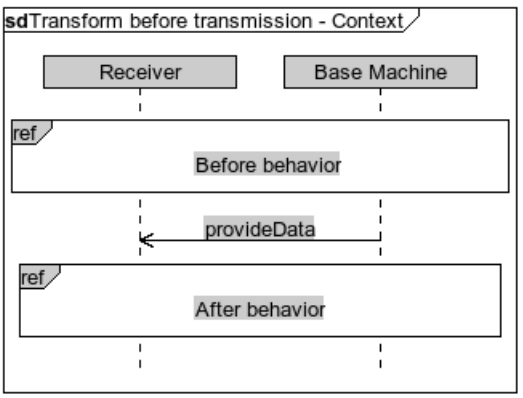

Fig. 14. Relevant behavior of the base problem of Transform Before Transmission Aspects

Problem.

You want to integrate a mechanism that transforms the data before these are provided to a Receiver.

Forces.

—*Complexity of the implementation:* The complexity to implement the mechanism should be kept to the minimum needed.

—*Integration into the base problem:* It shall be possible to integrate the transformation mechanism in a way that the base problem is only affected in the desired manner.

—*Response time of transformation:* The selected mechanism shall respond in a timely manner depending on the needs of the base problem.

—*Reliability of transformation:* The selected mechanism shall perform the transformation reliably.

Solution.

Structure.

A mechanism to transform data to provide these transformed data to a receiver has the basic structure shown in Fig. 15. Figure 15 shows the Receiver and the Base Machine as the relevant parts of the base problem (join points) for the transform before transmission aspect. The Transformation Machine is the domain that performs the transformation using a Transformation Resource (optional) and provides the transformed data to the Base Machine. The Base Machine

can then provide the transformed data to the Receiver. The cross-cutting requirement Transform before transmission refers to the Transformation Resource that is used to perform the transformation, and it constrains the Base Machine to provide this data to the Receiver and that the Receiver will receive the transformed data.

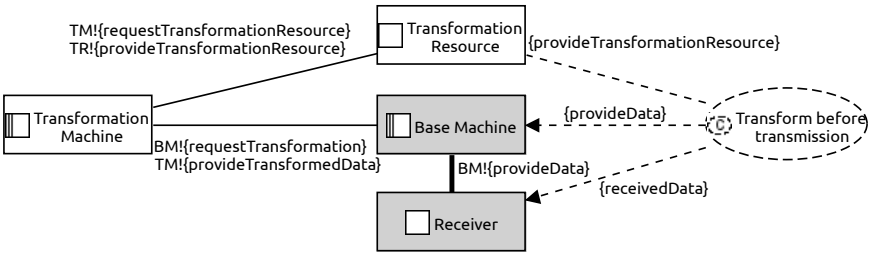

Fig. 15. The structure of Transform Before Transmission Aspects

Behavior.
The behavioral part of the solution is shown in Fig. 16. The interaction is started by the join point Base Machine that requests the transformation of the received data from the aspect's machine Transformation Machine. The Transformation Machine then provides the transformed data that it computed using the Transformation Resource. Finally, the Base Machine provides the transformed data to the Receiver.

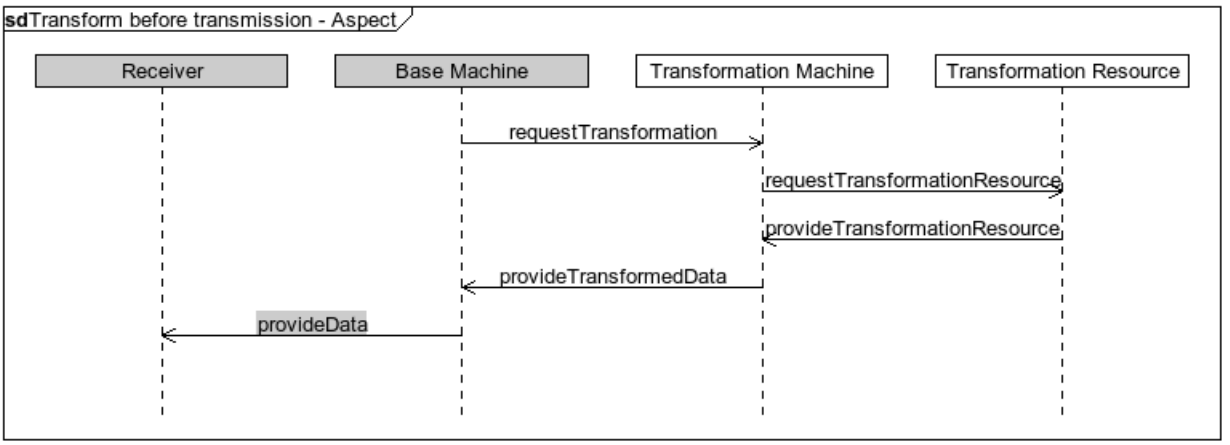

Fig. 16. Behavior of Transform Before Transmission Aspects

Integration.
How a Transform Before Transmission Aspect is integrated into a base problem is shown in Fig. 17. First, the Before behavior of the base problem (cf. Fig. 14) happens. Before the Base Machine provides data to the Receiver, the aspects behavior is integrated (cf. Fig. 16). After providing the transformed data to the Receiver, the After behavior is executed.

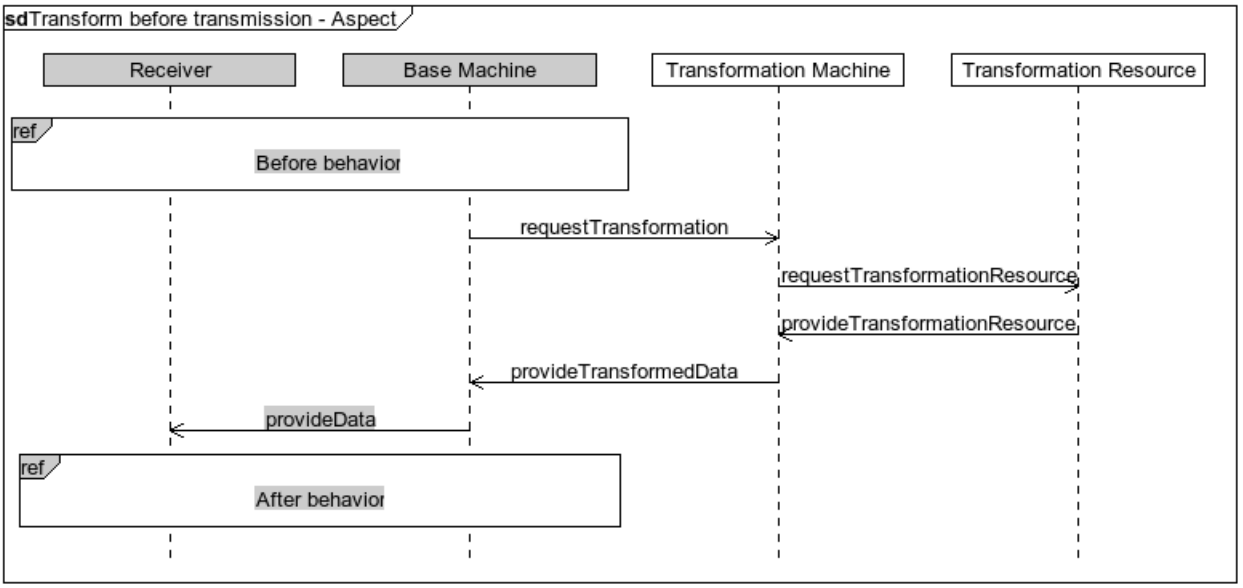

Fig. 17. Integration of Transform Before Transmission Aspects

Variants.
We can distinguish the same three variants of transformation aspects as in Section 4.2 depending on the type of the transformation resource.

Consequences.
Most consequences of the Transform Before Transmission Aspect are the same as for the Transform Received Data Aspect or apply analogously. Hence, we refer to Section 4.2 for the consequences of this aspect frame.

Examples.

For examples, we also refer to Section 4.2

4.4 Independent Behavior Aspect Frame

Name. Independent Behavior Aspect

Context.

Structure.
You have base problems into which an additional behavior shall be integrated that does not influence the behavior of the Base Machine in reaction of specific events caused by an Event Source that the Base Machine observes (see Fig. 18).

Fig. 18. The base problem of Independent Behavior Aspects

Behavior. Figure 19 shows the relevant behavioral view on the base problem. It consists of a before behavior, the event that the Base Machine receives from the Event Source, and an after behavior.

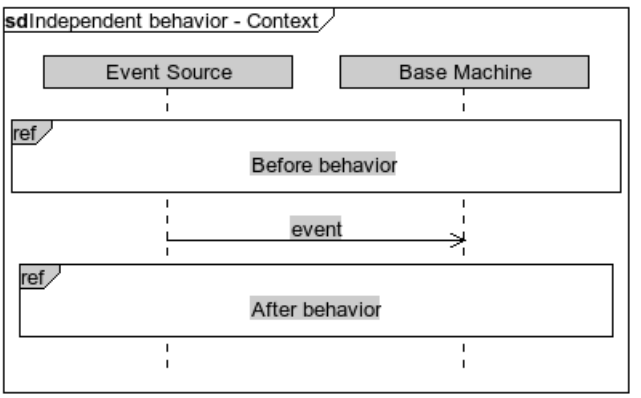

Fig. 19. Relevant behavior of the base problem of Independent Behavior Aspects

Problem.

You want to integrate a mechanism that performs a task in reaction to an event caused by an Event Source and this event shall not further influence the behavior of the Base Machine.

Forces.

—*Complexity of the implementation:* The complexity to implement the mechanism should be kept to the minimum needed.
—*Integration into the base problem:* It shall be possible to integrate the independent behavior in a way that the base problem is not affected by it.
—*Reliability of independent behavior:* The selected mechanism shall perform the independent behavior reliably.

Solution.

Structure.

A mechanism to perform an independent behavior in reaction to an event caused by an Event Source is shown in Fig. 20. Figure 20 shows the Event Source and the Base Machine as the relevant parts of the base problem (join points) for the independent behavior aspect. The Independent Machine is the domain that performs the independent behavior by influencing the Influenced Domain. The cross-cutting requirement Independent behavior refers to the Event Source that issues an event and the Base Machine that triggers the Independent Machine. The behavior of the Influenced Domain is constrained by the cross-cutting requirement.

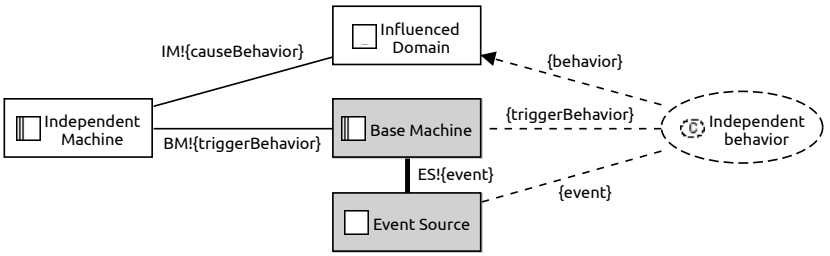

Fig. 20. The structure of Independent Behavior Aspects

Behavior.
The behavioral part of the solution is shown in Fig. 21. The interaction is started by the join point Event Source that issues an event that the join point Base Machine observes. The Base Machine then triggers the Independent Machine to cause the additional behavior that the Influenced Domain shall perform.

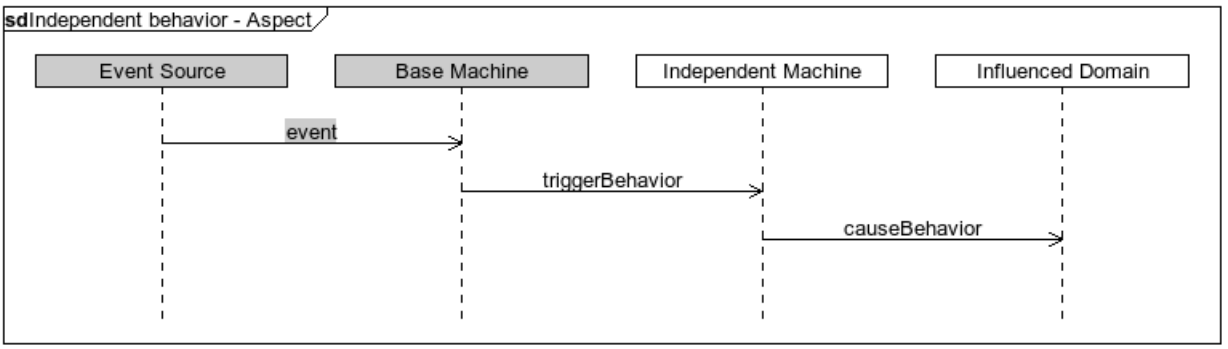

Fig. 21. Behavior of Independent Behavior Aspects

Integration.
How an Independent Behavior Aspect is integrated into a base problem is shown in Fig. 22. First, the Before behavior of the base problem (cf. Fig. 19) happens. In reaction to the Event Source's event, the aspect's behavior is integrated (cf. Fig. 6). In parallel to the aspect's behavior, the base problem's After behavior is executed.

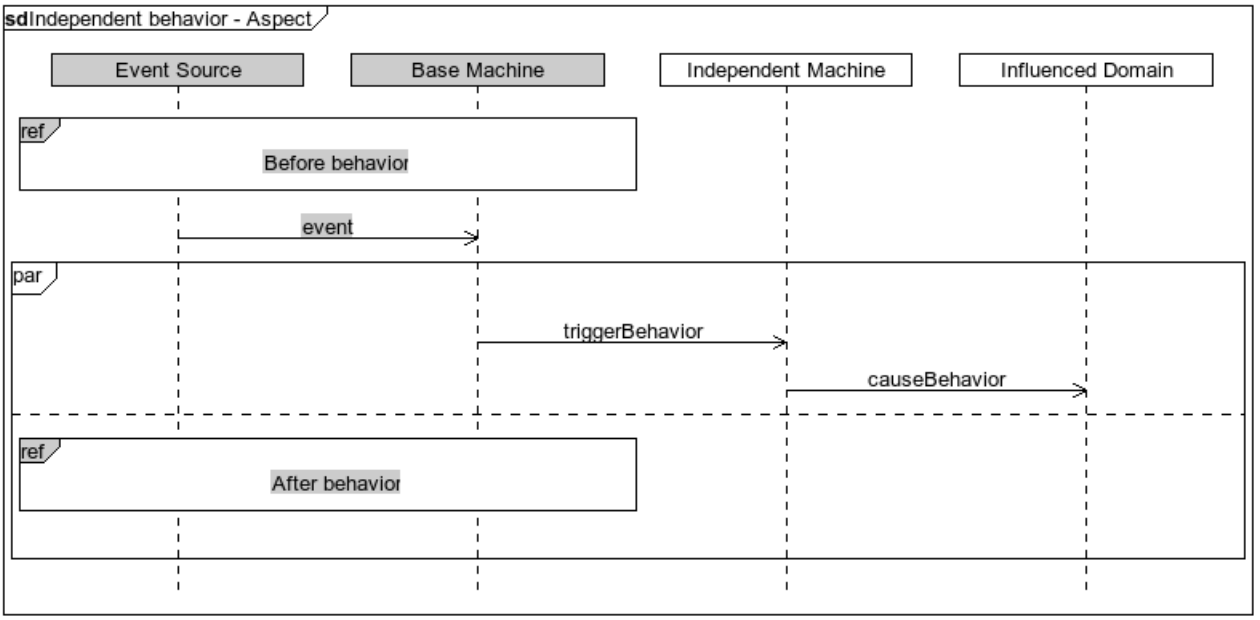

Fig. 22. Integration of Independent Behavior Aspects

Variants.

We can distinguish three variants of Independent Behavior Aspects depending on the type of the influenced domain.

Influenced data	*Influenced service*	*Influenced human*
If the influenced domain is lexical, this means that the data managed by this domain shall be modified.	If the influenced domain is causal, this means that technical device, service or the like shall be controlled.	If the influenced domain is biddable, this means that information shall be provided to humans such that they behave in a desired way.

Consequences.

Benefits.

—Complexity of the implementation:

All: -

—Integration into the base problem:

All: Outsourcing the independent behavior to the independent machine allows to parallelize the after behavior of the base machine and the independent behavior.

—Reliability of independent behavior:

All: -

Liabilities.

—*Complexity of the implementation:*

Influenced data	*Influenced service*	*Influenced human*
The complexity varies based on the complexity of the desired independent behavior. An independent behavior aspect about influencing data could only be about adding a single entry to a database, or additionally involve more complex operations like consistency checks.	It has to be ensured that a reliable connection to the service exists and that its API is correctly used.	An appropriate interface to the humans has to be chosen. This interface has to be designed in a way that the aspect machine is sufficiently be able to influence the humans in the desired way. For example, if the independent behavior is to display information to humans, then the interface could be a GUI on a computer. This GUI has to be designed in a way that the human notices the displayed information.

—*Integration into the base problem:*

All: The base machine has to provide sufficient information to allow the independent machine to perform the independent behavior. Additionally, it has to be ensured that the behavior of the independent machine and the influenced domain does not influence the base machines behavior, e.g., due to side-effects caused by the usage of shared resources.

—*Reliability of independent behavior:*

Influenced data	*Influenced service*	*Influenced human*
If data is influenced, the reliability depends on the correctness of the independent machine's implementation and the consistency of the influenced data.	The external service has to be reachable and comply to its specification.	To influence humans in the desired way, actuators (refining the interface between the independent machine and the influenced humans) have to be chosen that are noticed by the humans and cause them to behave in the desired way. In general, it is necessary to explain to the humans how, when, and why they are informed and what their reaction to this information shall be.

Examples.

Influenced data	*Influenced service*	*Influenced human*
Logging of specific events is an example for an independent behavior that influences data. In this case, the influenced domain is a log file or database, to which the new event with additional information, e.g. a timestamp, is added.	All examples of independent behavior aspects that influence humans are at some point refined to aspects that influence a service. For example, if we consider an alarm system in a hospital that informs nurses about critical vital signs of patients (event), then the interface to the nurses will be refined with, e.g., a display and/or speaker in the nurse's station. The independent machine is then only concerned with influencing this display and/or speaker in the desired way.	An example for an independent behavior that influences humans is an alarm system in a hospital that informs nurses about critical vital signs of patients (event).

5. GUIDANCE FOR CREATING AND USING ASPECT FRAMES

In this section, we provide further guidance on how aspect frames shall be created and how they can be used. We discuss the lessons that we learned during the creation of the four aspect frames presented in the previous Section.

5.1 Name

As usual, the name of a pattern should be carefully selected. An aspect frame's name shall provide a clear intuition about the kinds of problems the aspect frame's instances shall address. For example, the shown Decision Aspect is concerned with providing decisions.

5.2 Context

The specification of the aspect frame's context is not easy. The base problem description has, on one hand, to be general enough to fit to all base problems instances the aspect frame shall be integrated into. On the other hand, it should be as specific as possible to ensure that aspects of the frame are only integrated into fitting base problems and to support requirements engineers to decide whether the aspect they want to model fits to the aspect frame.

For example, we decided that the base problem of Decision Aspects consists of a base machine, a resource, and a requester (cf. Figure 3). The later two have no fixed domain type assigned. That is, they possibly be instantiated with biddable, causal, or lexical domains.

A more general description of the Decision Aspect could have left out the resources that the requester wants to access. This would not even have much impact on the other diagrams of the Decision Aspect, because the aspect's behavior is not concerned with interacting with the resource (cf. Figure 6). However, the decision is based also on the kind of resource that the requester wants to access. Hence, it is important that the base problems contain the resource that shall be accessed.

A too specific description of the Decision Aspect could, e.g., limit the join point requester to be a biddable domains and the resource to be a lexical domain. This would limit the application of Decision

Aspects to base problems in which humans want to access data which is controlled by the machine. Such a restriction would be too strong, because the types of the requester and resource are not relevant to provide the decision for most aspects that fit to the Decision Aspect.

The behavioral view on the base problem shall explicitly specify the situation in which the frame's aspects shall be integrated into it. This can be a, e.g., a single message, a specific sequence of messages, or situations described by a state predicate. In aspect-oriented programming, this situation description is called the aspect's *pointcut*. The behavior before and after the pointcut may be introduced as references (cf. Figure 4). The before and after behavior can then be used in the weaving to specify a modification of the base problem's behavior in reaction to the aspect's behavior. For Decision Aspects, the pointcut consists only of the requester's request to get access to the resource (cf. Figure 4), because a decision about the access to a resource shall be questioned if and only if the resource is requested.

5.3 Problem

The problem shall be a short statement that condenses the intention that all instances of the aspect frame share. This problem statement shall help requirements engineers to decide whether their aspect addresses the described problem and is potentially an instance of the frame.

5.4 Forces

The forces of an aspect frame summarize issues that make it difficult to solve the problem existing in the described context. The forces help to further understand the properties of instances of an aspect frame. During the development of four aspect frames, we identified two forces that are relevant for all aspect frames. These are the *complexity of the implementation* and the *integration into the base problem* that are obviously relevant for all kinds of aspects. Additionally, it may be worth it to discuss the software qualities that instances of an aspect frame should have. For example, we identified issues concerning the *response time*, *reliability*, and *security* as relevant for Decision Aspects.

5.5 Solution

Similar to the description of the base problems, it is important to describe the aspect frame general enough, but not too general. For example, if the requester would be left out in Figure 5, then the relevant interface for the decision machine to the base machine that is needed to request the decision and to provide the result of the decision is still contained, but the original request on which the base machine reacts would no longer be part of the aspect and we would miss the original trigger (pointcut) of the interaction the aspect needs to be integrated into (cf. Section 5.2). The pointcut is needed to clearly specify when the aspect's behavior shall be weaved into the base problem's behavior. Consequently, the Decision Aspect's behavior description (see Figure 6) contains the pointcut described in Figure 4, namely the message requestResource. This message is the first message in the aspect's behavior description, i.e., the aspect's behavior shall be performed after this message.

It would be reasonable to add the requested resource to Figure 5, because the decision is likely to depend on both the requester and the resource. However, the information which resource is requested is implicitly encoded in the phenomenon requestResource and the resource itself is not an actor in the aspects behavior (cf. Figure 6). Hence, we decided to leave out the resource in Figure 5 to keep the diagram as simple as possible.

The integration description shall explain how the aspect's behavior shall be added to the base problem's behavior. Ideally, the aspect's behavioral view contains the base problem's pointcut and the aspect can respectively be integrated into the base problem. For example, the Decision Aspect's behavioral description contains the base problem's pointcut as first message. Hence, we just need to add the aspect's behavior after this message (see Figure 7).

Additionally, the sequence diagram for the integration of aspects may describe deviations of the base problem's behavior in reaction to the aspect. For Decision Aspect, we specified that the base problem's after behavior shall only be executed if the provided decision is positive. If the decision is negative, then optionally an error behavior may be performed by the base machine. This error behavior strongly depends on the concrete context of the base problem and does itself not belong to the Decision Aspect.

We introduce the concept of *variants* to aspect frames to be able to discuss different flavors of an aspect. A variant assigns specific types to domains without specified type. Variants should be used if the aspect's structure, behavior, and integration do not depend on the type of an involved domain, but the benefits and liabilities may vary for different types of it. For example, the information source in Decision Aspects has no specific type and the structure, behavior, and integration of Decision Aspects do not depend on its type. However, the properties of the Decision Aspects, i.e., the benefits and liabilities, are different depending on the information source's concrete type. Hence, we distinguish the Decision Aspect variants data-based decision, service-based decision, and human-based decision. If a domain without specified type shall not be instantiated with a specific type, this can also be specified by declaring that this variant is not allowed or possible. Note that it is not mandatory to describe variants for all domains without specified type. For example, the requester and the resource have both no specified type in the decision aspect and they are not mentioned in the variants. This means that there are no restrictions on the type of these domains and that the benefits and liabilities are not expected to vary for different types of these.

5.6 Consequences

The positive and negative consequences should discuss all stated forces for all defined variants. If a consequence is identified that has not yet a corresponding force, this indicates that a corresponding force is missing and that it should be added.

The consequences of an aspect frame shall help requirements engineers to specify the relevant properties of the concrete mechanism they want to express as aspect. These properties can also be used to compare aspects that are instances of the same aspect frame with each other.

5.7 Examples

The examples shall help requirements engineers to understand the kind of problems that belong to the aspect frame and consequently, to decide whether the mechanism they want to express as aspect fits to the class described by the aspect frame. If variants are specified, then at least one example should be provided for each variant.

6. RELATED WORK

In this section, we provide an overview of the work related to this paper. In Section 6.1, we discuss literature that considers both aspect-orientation and patterns. Section 6.2 briefly discusses the AORE literature.

6.1 Aspect-orientation and Patterns

Clarke and Walker [2001] present an approach to model design patterns as composition patterns. They propose UML templates that capture the structure and behavior of the design patterns independently from the design elements it may cross-cut. Additionally, they show how their proposed composition patterns can be translated to the aspect-oriented programming language AspectJ. While Clarke and Walker express aspects in the design phase, we consider aspects in the requirements phase. Furthermore, we provide with the aspect frames pattern candidates that help to describe aspects that fit into the class of aspects described by the frame.

Garcia et al. [2006] performed a quantitative study that investigates the advantage of using aspects to implement 23 Gang-of-Four patterns [Gamma et al. 1995], which have shown to be cross-cutting in several cases. The authors found out that an aspect-oriented implementation of design patterns leads to a better separation of concerns, but introduces also issues like more complex operations and more lines of code in some cases.

6.2 Aspect-Oriented Requirements Engineering

There are several AORE approaches based on different requirements engineering approaches, such as use cases, viewpoints, and goals. A more detailed literature overview is provided by Singh and Gill [2011]. The work of Lencastre et al. [2008] also integrates aspect-orientation into the problem frames approach similar to our method presented in [Faßbender et al. 2015] that we use to support the description of the aspect frames. However, Lencastre et al. [2008] do not propose frames for reoccurring cross-cutting concerns. Alebrahim et al. [2012] show how role-based access control (RBAC) can be integrated into specific problem frames in an aspect-oriented way.

To the best of our knowledge, there is not yet any research on patterns to express classes of aspects in the field of AORE.

7. CONCLUSION

In this paper, we introduced aspect frames that represent classes of aspects (cross-cutting concerns) that share a common concern and behavior. The aspect frames are described in a pattern format that summarizes the shared problem and the common solution provided by aspects of the frame's class. Aspect frames are a means to support the description of aspects in AORE, because requirements engineers can use the aspect frames as blueprints for aspects and fit their cross-cutting concerns to an aspect frame. The aspect frame provides the requirements engineers with further information about forces of the problem, and the benefits and liabilities that aspects fitting to the frame might have.

We described a pattern format that can be used to describe aspect frames in a structured way. Using this pattern format, we introduced the aspect frames Decision Aspect, Transform Received Data Aspect, Transform Before Transmission Aspect, and Independent Behavior Aspect as an examples of aspect frames. Finally, we summarized our lessons learned to provide guidance on how to create and use aspect frames.

In future work, we want to further elaborate on aspect frames. For example, we want to study whether there are additional classes of cross-cutting concerns that could be represented as aspect frames and evaluate how much aspect frames help requirements engineers to specify cross-cutting concerns.

ACKNOWLEDGMENTS

We want to thank our shepherd Eden Burton for the valuable feedback that helped us to improve the quality of this paper. We also thank Uwe Zdun, Victor Sauermann, Eva Schön, Frank Frey, Olaf Zimmermann, and Eduardo Fernandez who all provided enlightening feedback and constructive improvement suggestions during the writer's workshop. This feedback and the suggestions helped us to further improve this paper.

REFERENCES

Azadeh Alebrahim, Thein Than Tun, Yijun Yu, Maritta Heisel, and Bashar Nuseibeh. 2012. An Aspect-Oriented Approach to Relating Security Requirements and Access Control. In *Proceedings of the CAiSE'12 Forum at the 24th International Conference on Advanced Information Systems Engineering (CAiSE) (CEUR Workshop Proceedings)*, Vol. 855. CEUR-WS.org, 15–22. http://ceur-ws.org/Vol-855/paper2.pdf

Siobhán Clarke and Robert J. Walker. 2001. Composition Patterns: An Approach to Designing Reusable Aspects. In *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 12-19 May 2001, Toronto, Ontario, Canada*, Hausi A. Müller, Mary Jean Harrold, and Wilhelm Schäfer (Eds.). IEEE Computer Society, 5–14. DOI:http://dx.doi.org/10.1109/ICSE.2001.919076

Stephan Faßbender, Maritta Heisel, and Rene Meis. 2014. Aspect-oriented Requirements Engineering with Problem Frames. In *ICSOFT-PT 2014 - Proc. of the 9th Int. Conf. on Software Paradigm Trends*. SciTePress, 145–156.

Stephan Faßbender, Maritta Heisel, and Rene Meis. 2015. A Problem-, Quality-, and Aspect-Oriented Requirements Engineering Method. In *Software Technologies - 9th International Joint Conference, ICSOFT 2014, Vienna, Austria, August 29-31, 2014, Revised Selected Papers (Communications in Computer and Information Science)*, Vol. 555. Springer, 291–310. DOI:http://dx.doi.org/10.1007/978-3-319-25579-8_17

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Alessandro Garcia, Cláudio Sant'Anna, Eduardo Figueiredo, Uirá Kulesza, Carlos Lucena, and Arndt von Staa. 2006. *Modularizing Design Patterns with Aspects: A Quantitative Study*. Springer Berlin Heidelberg, Berlin, Heidelberg, 36–74. DOI:http://dx.doi.org/10.1007/11687061_2

Michael Jackson. 2001. *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley.

Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-Oriented Programming. In *ECOOP*. Springer, 220–242. DOI:http://dx.doi.org/10.1007/BFb0053381

Maria Lencastre, Ana Moreira, João Araújo, and Jaelson Castro. 2008. Aspects Composition in Problem Frames. In *Proceedings of the 16th IEEE International Requirements Engineering Conference*. IEEE Computer Society, 343–344.

Rene Meis and Maritta Heisel. 2017. Pattern-based Representation of Privacy Enhancing Technologies as Early Aspects. In *Trust, Privacy, and Security in Digital Business (LNCS)*, Vol. 10442. Springer International Publishing, Cham, 49–65. DOI:http://dx.doi.org/10.1007/978-3-319-64483-7_4

Ana Moreira, Ruzanna Chitchyan, Joo Arajo, and Awais Rashid. 2013. *Aspect-Oriented Requirements Engineering*. Springer Publishing Company.

Narender Singh and Nasib Singh Gill. 2011. Aspect-Oriented Requirements Engineering for Advanced Separation of Concerns: A Review. *IJCSI International Journal of Computer Science Issues* 8, 5 (September 2011), 288–297.

Tim Wellhausen and Andreas Fießer. 2011. How to write a pattern?: a rough guide for first-time pattern authors. In *16th European Conference on Pattern Languages of Programs, EuroPLoP 2011*. ACM, 5. DOI:http://dx.doi.org/10.1145/2396716.2396721