Adapt Cases: Extending Use Cases for Adaptive Systems

Markus Luckey, Benjamin Nagel*, Christian Gerth, Gregor Engels*
*s-lab - Software Quality Lab
University of Paderborn
Warburger Straße 100
33098 Paderborn, Germany
{luckey, gerth, engels}@upb.de, bnagel@s-lab.upb.de

ABSTRACT

Adaptivity is prevalent in today's software. Mobile devices self-adapt to available network connections, washing machines adapt to the amount of laundry, etc. Current approaches for engineering such systems facilitate the specification of adaptivity in the analysis and the technical design. However, the modeling of platform independent models for adaptivity in the logical design phase remains rather neglected causing a gap between the analysis and the technical design phase.

To overcome this situation, we propose an approach called *Adapt Cases*. Adapt Cases allow the explicit modeling of adaptivity with domain-specific means, enabling adaptivity to gather attention early in the software engineering process. Since our approach is based on the concept of use cases it is easy adoptable in new and even running projects that use the UML as a specification language, and additionally, can be easily incorporated into model-based development environments.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques —*Object-oriented design methods*

General Terms

Design, Languages

Keywords

Adaptive systems, requirements, model-based, adapt case, use case

1. INTRODUCTION

Today's software engineering implies new challenges due to increasing complexity, increasing size and unpredictable environmental changes. For instance, software that is critical concerning availability requires a fast maintenance and adaptation to the changing environment. While addressing these challenges, it becomes necessary to develop so-called *adaptive systems* that are able to react to a changing environment.

Recent approaches such as the autonomic computing paradigm [12] implement adaptation logic in external mechanisms to simplify maintenance and enable reusability. Whereas technical concepts provide a clear distinction among functionality and adaptivity, engineering processes and methodologies still lack the explicit separation of these concerns. As a consequence, the aspect of adaptivity is not sufficiently considered in the design process hindering the implementation of external adaptation mechanisms.

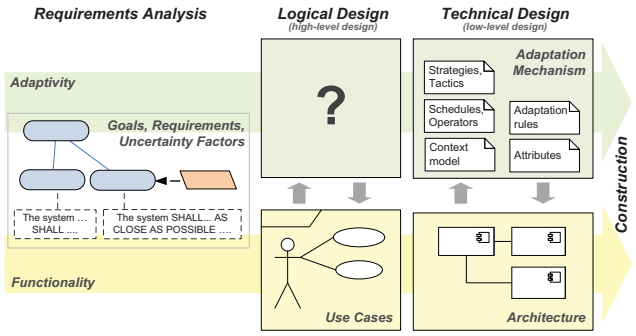

Figure 1: Design Process—Separation of Concerns

We investigated standard software development processes as shown in Figure 1[1]. During the analysis phase of those processes, high-level goals and requirements are identified. These requirements are refined into more concrete logical use cases where logical means platform independent (cf. MDA [16]). The subsequent phase is called technical design, where the platform specific model is created—typically the concrete software architecture. Logical use cases are either mapped to components in the platform specific model and/or are refined to more technical use cases [8]. Finally, the technical design model is implemented in the construction phase using frameworks, libraries, etc. Of course, the lines between the phases blur depending on the concrete project context. For example, the models created in the technical design phase may be used to directly generate code or to be interpreted by special frameworks. Having the OMG MDA initiative in mind, we consider the resulting models of the two phases analysis and logical design as platform indepen-

[1]Names of phases vary in the different process models.

SEAMS'11, May 23–24, 2011, Waikiki, Honolulu, HI, USA

dent models (PIM) and the results of technical design and construction as platform specific models (PSM).

While specifying a software system, the functional aspects are logically modeled using use cases in the logical design phase. Besides, UML use cases offer different means to model adaptivity, such as decision nodes and exception in attached activity diagrams. However, we claim that these means fail in maintaining the separation of concerns principle, i.e. adaptivity and functional aspects are mixed in the model. For this reason, designers might miss important adaptivity patterns, or even induce errors because of the immense complexity. Furthermore—MDA in mind—model-based analysis and development are aggravated since it is burdensome to extract adaptivity aspects from the mixed models. Hence, we argue that having a dedicated modeling language for adaptivity aspects would support better software quality. Regarding adaptivity, current software engineering methods reveal a gap between the analysis and technical design phase, which we aim to fill using Adapt Cases.

In this paper, we propose Adapt Cases, a domain-specific modeling language for adaptive systems based on use cases. Adapt Cases use and refine the concept of UML use cases for the explicit and formal specification of adaptivity. Adapt Cases help sharpen the awareness for adaptation aspects early in the design process and thereby support the designer to recognize common adaptation patterns and even design flaws.

The remainder of this paper is structured as follows: In Section 2 we introduce Adapt Cases and illustrate them by example. We apply our approach in terms of a case study in Section 3. Next, we discuss related work in Section 4 and, finally, conclude the paper in Section 5 and give an overview about future work.

2. ADAPT CASES

In Section 1, we identified a gap between the analysis and the technical design of self-adaptive systems. In the following, we describe Adapt Cases, the modeling language, which is intended to bridge the described gap. First, we discuss the requirements for this language. Next, we introduce the Adapt Cases with their underlying meta-model, and finally, we describe the language's semantics and implementation.

We identify the following requirements for our modeling language:

1. Explicit description of adaptivity on logical design level
2. Expressiveness of use case modeling method
3. Integratability with standard software modeling techniques (i. e. UML)

That is, we search for a language and method, which support the explicit description of adaptivity without being hidden behind the scenes or being mixed with different aspects (e. g. functional). The language shall be as expressive as use cases and concerning adaptivity even more expressive via dedicated domain-specific modeling means. This is because use cases showed to be sufficient in modeling refined requirements over the last years and perfectly suit to support communication with customers. The last requirement aims at integration with UML use cases. Thereby, adaptation may be modeled as specific aspect, which impacts the functionality described by use cases. Furthermore, the language can be integrated in every software development process using the UML.

We designed Adapt Cases to address these requirements. Adapt Cases extend use cases [8] to allow the explicit description of adaptivity on a higher logical level. Thereby, concerns may be separated such that use cases describe the standard application logic while Adapt Cases describe the reaction to errors or different events with subsequent adaptation, i. e. the adaptation logic. Having adaptivity extracted from use cases, a dedicated analysis and translation into domain specific framework code (e. g. Rainbow) is supported. Figure 2 sketches the Adapt Case approach on a very high level. The use case UC1 describes some core functionality such as delivering a web site. The Adapt Case AC1 describes which system characteristics are monitored and what action is performed if the characteristics behave abnormal. For instance, AC1 might describe the change of the amount of multimedia in the delivered website as reaction to a peak load. We say the Adapt Case AC1 adapts the use case UC1.

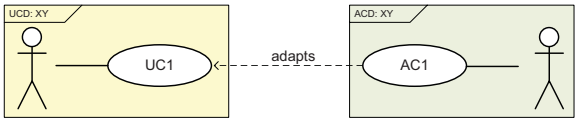

Figure 2: Adapt Case Approach

Similar to use cases, additional information such as conditions, scenarios, etc. may be modeled. To this end, Adapt Cases introduce domain-specific model concepts. We describe all these concepts of Adapt Cases and the underlying meta-model in the following section and give their semantics using OCL expressions in Section 2.2.

2.1 Syntax of Adapt Cases

An exhaustive specification language for adaptivity on a logical level needs to consider all aspects of adaptive behavior. To this end, we analyzed different adaptivity specification and implementation methods, which are used during the engineering process. We approached the identified gap in a top-down (analysis phase towards construction phase) and a bottom-up fashion (construction towards analysis phase). While approaching the gap from the top, we investigated several requirements approaches, the most important being RELAX [6], goal-based approaches [3], and Awareness Requirements [14]. Thereby, we realized that requirements approaches do not put any restriction on the methods used in the logical design. However, we determined that it would be useful to support different standard concepts of use case modeling such as the definition of pre- and postcondition, actors, scenarios, etc. Given the generic nature of requirements and goals, using a control-loop approach for Adapt Cases is valid. Following the bottom-up approach to creating the meta-model, we investigated common well-known frameworks and models such as Rainbow [7], StarMX [2] and the 3-Layer model [13]. Most of the models we were investigating were either using or at least compatible with control-loops.

The well-known control-loop model MAPE-K proposed by IBM [12] provides a process-oriented reference model for realizing adaptive behavior. The MAPE-K control loop model is shown on the right hand side of Figure 3. The model consist of a monitor that observes the system and the environment,

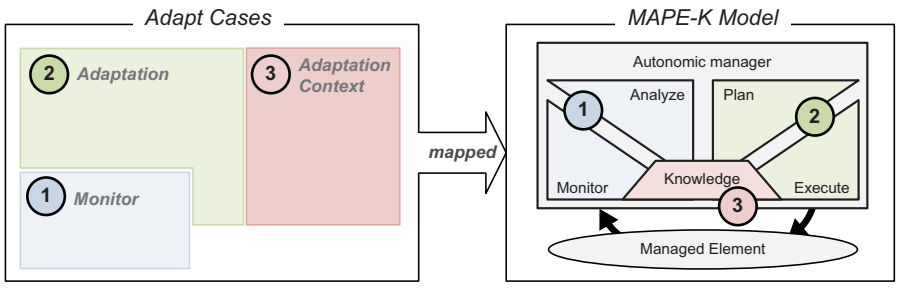

Figure 3: Adapt Cases structured according to MAPE-K

an analyzer that analyzes the monitoring results, a planning component that decides for an appropriate adaptation action, and an execution component that executes the chosen adaptation action. Taking up these concepts, the language of Adapt Cases is divided into three packages, which can be mapped to the MAPE-K model as shown in Figure 3. The monitor package includes the MAPE-K phases *monitor* and *analyze*. The adaptation package includes the two MAPE-K phases *plan* and *execute* and finally, the MAPE-K *knowledge* maps to the adaptation context package.

To support an elaborate modeling of adaptation rules, we further refine the MAPE-K concept of monitors by corridors that are used to specify a system's valid states. If the system leaves the corridor, the adaptation is effected. Modeling Adapt Cases using these concepts allows the software engineer to easily distinguish between cause for a demanded adaptation and the concrete adaptation action. In requirements engineering nomenclature, this maps to the distinction of *problem* and *solution*.

To clarify the chosen concepts of Adapt Cases, we present them along the exemplary specification of a web shop, before presenting the metamodel. The web shop shall autonomously adapt its product pages in the case that a product runs out of stock. The system model of the web shop is visualized in Figure 4. The contents of the context package correspond to the context or domain model that is usually created during requirements engineering. Extending the traditional understanding [1] of the term *context*, the context includes any entity in the system or its environment that is relevant for the monitoring or the adaptation of the system. Both the monitor and the adaptation package contain the model elements that must be modeled in addition to traditional use cases. The monitor definition defines a corridor with lower and upper bound and a pulse that considers the product quantity in the storage area. The storage area is part of the system context and holds the stored products that shall be monitored. Whenever the product is running out of availability, the event *Product not available* with the attached product ID is fired. The Adapt Case further defines an adaptation action that listens to events named *Product not available*. The adaptation action is implemented with two alternatives. The first one disables the buy button of the respective product. The second disables the whole product page. Both button and product page are components of the web shop—the system under consideration.

Next, we describe the concepts used in the example model in more detail. All these concepts will be elements of the metamodel. We group the concepts into three packages being *monitor*, *adaptation*, and *adaptation context*.

Monitor.

The monitor package contains elements to describe what context elements are observed, what states of those elements are valid and invalid, how invalid states are translated into events, and which information is collected for adaptation.

Monitor Definition The monitor definition is an abstract representation of the monitoring concept related to the Adapt Case. The monitor definition monitors the behavior of the adaptation context. A monitor definition consists of a corridor and a pulse and fires events if necessary.

Corridor The corridor defines both the lower and upper bound that are set for a specific system characteristic the monitor shall observe. The corridor contains valid values and states of adaptation context elements.

Pulse The Pulse represents the value or status of a monitored element in the adaptation context. The pulse can be used to aggregate multiple values or map discrete values to concrete statuses.

Event An event is thrown by the monitor definition to indicate that a change in the adaptation context has occurred. An event is fired whenever the pulse leaves the corridor.

Adaptation.

The adaptation package contains elements to describe the adaptation action, which is performed to adapt the system. Further, the package enables the description of alternatives and respective conditions used during adaptation planning.

Actor The actor performs or is involved in the Adapt Case. The actor can be the system itself (autonomous adaptation) or even any human actor who is semi-automatically performing the Adapt Case.

Precondition The precondition defines a set of conditions that must be fulfilled before the Adapt Case can be performed. The precondition can be related to any attribute (and even a combination of different attributes) in the adaptation context.

Postcondition The postcondition defines a condition that must be fulfilled after the Adapt Case has been performed. The postcondition can be related to any attribute in the adaptation context, too.

Invariant A critical condition that must hold before, after, and throughout the Adapt Case performance.

Adaptation Action The adaptation action defines how a particular system component or data entity is adapted. The

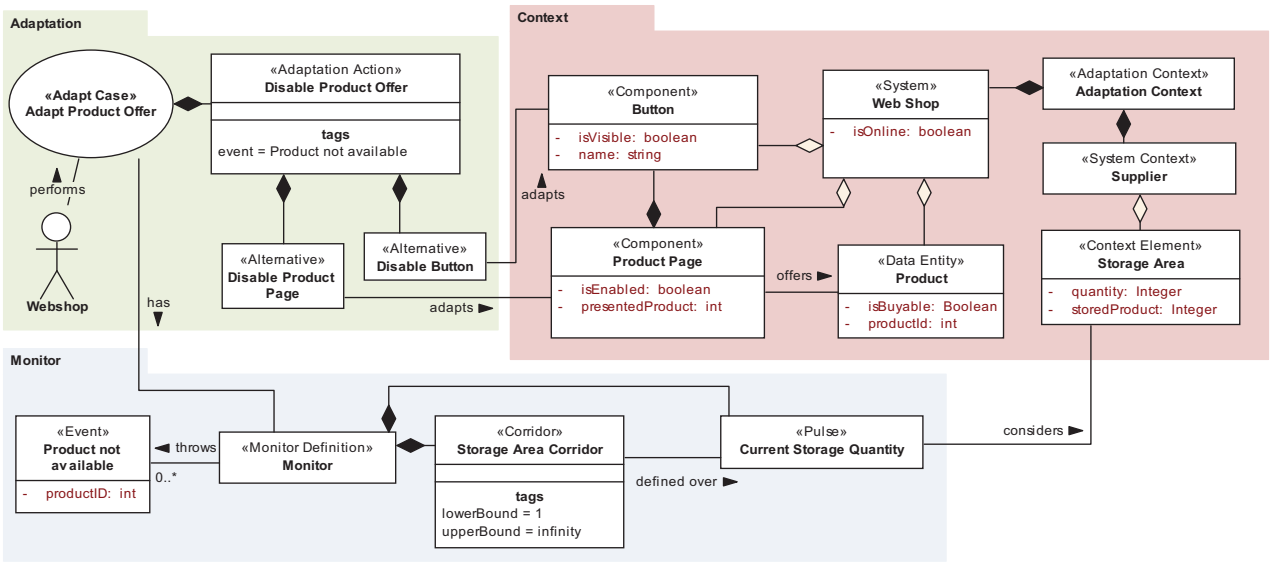

Figure 4: AC Disable Product Offer

adaptation action can describe different alternatives for the adaptation.

Alternative A description of one alternative solution for the adaptation of the system. Alternatives are assigned to and selected in the adaptation action.

Adaptation Context.

An Adapt Case needs to describe the adaptation context in sufficient detail. The system is described in terms of components and data entities. The system context is described in terms of more generic *context elements*. All three notations are composable to reflect the adaptation context's hierarchical composition. Besides the adaptation context, a Adapt Case contains a description of the actual adaptation actions.

Adaptation Context The overall context, the adaptive system is located in. The context consists of the system and the system context.

System The system describes the actual system that is adapted in the Adapt Case. The system is acting in a system context.

System Context The system context models the context the system is acting in.

Component A system is composed of components and data entities. A component is meant in the technical sense and is defined as a concrete technical unit in the system.

Data Entity A data entity is defined as a logical entity described by data stored in the system.

System Context Element A system context element is any kind of element in the system context. This element is abstracted from the distinction between a technical and logical entity.

The meta-model, containing the model elements described above, is shown in Figure 5. The example in Figure 4 was modeled by using the according UML Profile. The meta-model enables software engineers to specify an adaptation case. Having a formal meta-model, which is derived from the concepts of control loops, well integratable with UML use cases, and containing the complete set of information needed for the description of adaptation cases, we support a model-based analysis and development process for adaptive systems, aligned with the ideas of the OMG MDA initiative.

2.2 Semantics of Adapt Cases

Having defined the syntax of Adapt Cases in the previous section, we introduce its semantics in this section. To define the semantics of Adapt Cases, we have to specify generally when an event is thrown by the monitor and how a concrete Adapt Case reacts on this event. We describe the semantics of Adapt Cases using OCL constraints[2]. Therefore the classes in the meta-model are enhanced by methods whose behavior is specified using the OCL. Listing 1 specifies the behavior of the `throwEvents()` method of the monitor. An event is thrown if the corresponding pulse that monitors a particular system characteristic (here the product quantity) leaves the defined lower and upper bound of the corridor.

Figure 6 shows the call sequence throwing an event. The monitor reads the lower and upper bounds for each corridor and compares it to the current pulse value. If necessary, it creates and throws an event. The method `throwEvents()` is continuously executed.

In addition, Listing 1 defines pre- and postconditions for the execution of an Adapt Case. The `run()` method of an Adapt Case is executed if the following preconditions are met:

1. An event with given name has been fired
2. The action's and the Adapt Case's preconditions hold
3. The attached invariant is not violated during execution

After the adaptation action has been executed, the following postconditions must be satisfied:

1. The postcondition of the executed adaptation action holds

[2]For the sake of brevity we use simplified OCL syntax.

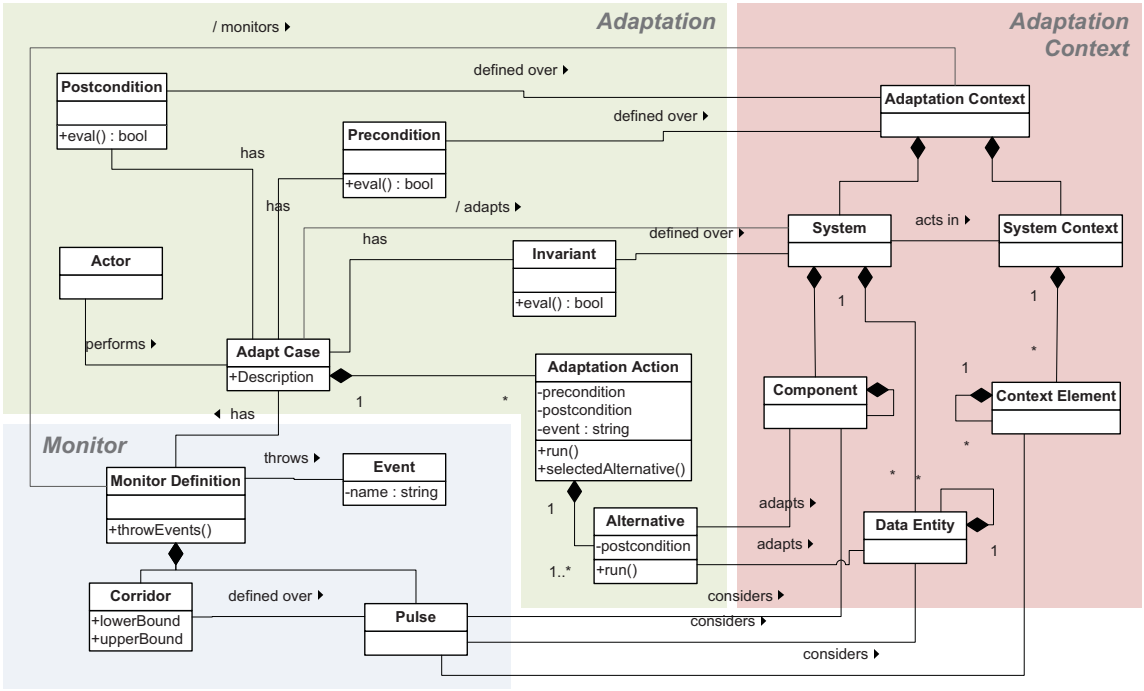

Figure 5: Adapt Case Meta-Model

```
context AdaptCase.action.run()

  inv: adaptCase.invariant.eval()

  pre: adaptCase.monitor.event
         ->contains(e | e.name = event)
    and adaptCase.precondition.eval()
    and self.precondition

  post: not adaptCase.monitor.event
          ->contains(e | e.name = event)
    and adaptCase.postcondition.eval()
    and self.postcondition
    and selectedAlternative().postcondition

def MonitorDefinition.throwEvents(Date t = Date.
    now())
  forall c in corridor
    if c.pulse.get(t) > c.upper.get(t) or
       c.pulse.get(t) < c.lower.get(t) then
        throw createEvent(c,c.pulse)
```

Listing 1: Semantics for ACs

2. The postcondition of the selected alternative holds
3. The corresponding event has been consumed

The semantics specification of the `throwEvents()` method and the pre- and postconditions for the execution of an Adapt Case in Listing 1 is valid for every Adapt Case. To complete the semantic specification, the behavior of every concrete Adapt Case must be specified. In the case of our web shop example, we have to define how the concrete Adapt Case *Adapt Product Offer* reacts on a thrown event. To that extent, Listing 2 specifies the behavior of the

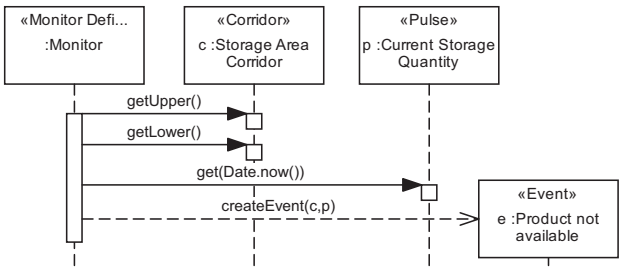

Figure 6: Sequence Diagram for `throwEvents()`

`run()` method of this Adapt Case and the behavior of its adaptation actions.

Figure 7 shows the call sequence of the method `AdaptationAction.run()` that is continuously executed to poll for events.[3] If an event that matches the specified name is found, the adaptation action retrieves the attached data, e. g. the identifier of the product that ran out of availability. Next, the action evaluates the conditions for the different adaptation alternative if multiple exist. Depending on the result the corresponding alternative is executed. In our example, the button is hidden or the product page is disabled.

Given the described meta-model and OCL expressions, software engineers are able to formally specify adaptation requirements. Due to the fixed semantics, the specified Adapt Cases are semantically unambiguous and may be further used e. g. , for automated requirements validation and model-driven development. Since we use OCL expressions to

[3]We use but—due to space restrictions—did not model the setter and getter methods for the corresponding class attributes.

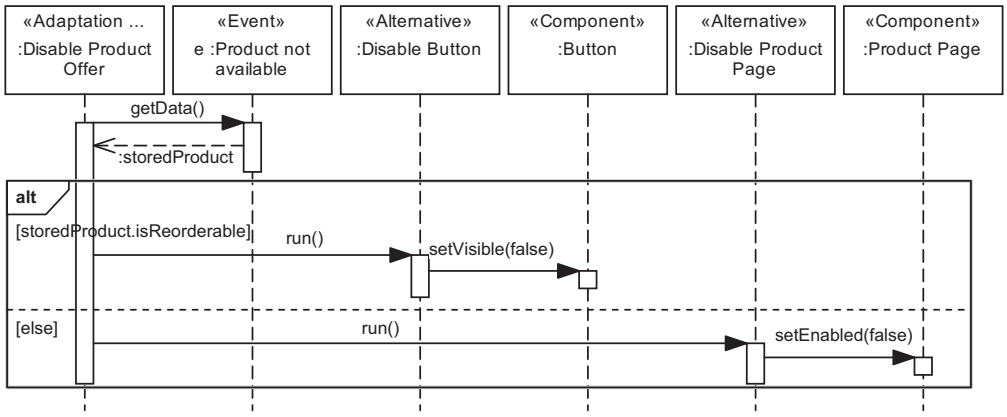

Figure 7: Sequence Diagram for `AdaptationAction.run()`

```
def AdaptCase.action.DisableProductPage.run()
 component->select(cmp|cmp.type = "Button"
  and cmp.name = "Buy")
  and cmp.relatedProduct = event.productID)
   .isVisible = false

def AdaptCase.action.DisableButton.run()
  component->select(cmp|
    cmp.type = "Product Page" and
  cmp.presentedProduct = event.productID)
  .isEnabled = false

def AdaptCase.action.run()
  if storedProduct.isReorderable then
    DisableProductPage.run()
  else
    DisableButton.run()
  endif
```

Listing 2: Implementation for the Web Shop AC

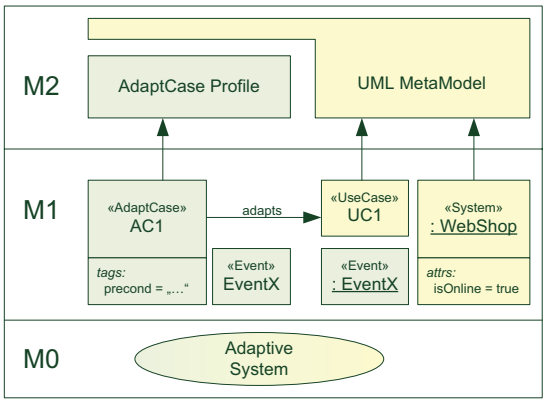

Figure 8: UML Profile for Adapt Cases

describe the concrete adaptation action, we currently cannot define compositional adaptation actions directly. However, compositional adaptation is expressible indirectly using parametric adaptation by providing different system configuration plans which are electable via parameters [4]. In Section 3, we show how Adapt Cases are used by applying the approach to a real world example.

2.3 Implementation

We implemented the meta-model for Adapt Cases using UML Profiling (see Figure 8). Using the UML Profile, we are able to fully integrate the concept of Adapt Cases into the UML, thus being able to use Adapt Case with every software development process that uses UML for specifying requirements. In addition, using a UML Profile ensures an easy integration into model-based development approaches.

3. CASE STUDY

In the previous section, we introduced and described the usage of Adapt Cases for the explicit modeling of adaptivity for adaptive systems. In order to validate our approach, we conducted the Znn.com case study introduced in [7] with our approach. In previous work, the adaptivity of Znn.com has been specified on the technical design level providing concrete solutions for the Rainbow Framework (using the language Stitch). We would like to use Adapt Cases to elicit and model the adaptivity in the logical design for Znn.com. More concrete, we track the following three research questions in order to show that adaptivity can be modeled using Adapt Cases.

RQ1 Could the models of the analysis and technical design phase be modeled? In particular, can control loops conveniently be reflected by Adapt Cases?

RQ2 Can Adapt Cases be integrated with the standard development process? How do Adapt Cases perform together with use cases?

RQ3 Do Adapt Cases fill the gap between analysis and technical design? Are the transitions between the different phases sufficiently small?

3.1 Case Study Design

Znn.com is a web-based service that provides news content to its customers. Znn.com aims for three business objectives, which are prioritized in the enumerated order:

[4]We are currently working on a full UML implementation of Adapt Cases which uses UML Actions for the specification of compositional adaptation.

1. News content provision in a reasonable response time
2. High quality content presentation
3. Minimization of operating server costs

As a web application, Znn.com is acting in a continuously changing context (e.g. number of customers requesting the news service). Thus, the trade-off between customer satisfaction (1,2) and economical aspects (3) strongly depends on changes in the system context.

Example: A high number of customers is requesting news content at the same time, resulting in an high server load. The server load leads to unacceptable response times on client side, lowering the customer satisfaction. Additional servers could reduce the response time but would increase the operating costs for the server pool.

In order to handle these trade-offs with respect to environmental changes, two high-level adaptations should be available:

- Change the level of content fidelity
- Adjust the number of server in the server pool

3.2 Case Study Results

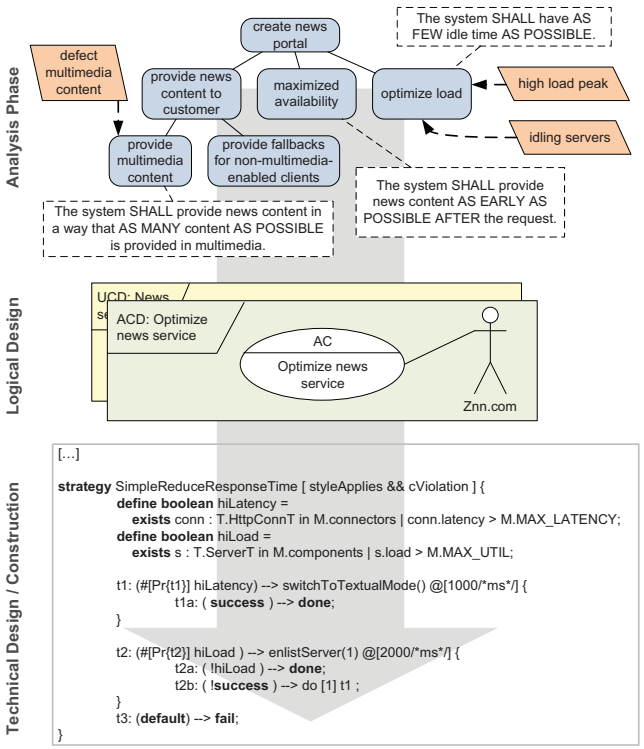

Figure 9: Adapt Cases in the Development Process

For the case study, we created goal graphs and RELAX-ed requirements for the Znn.com example. Next we transformed those artifacts into high-level use cases and Adapt Cases, which were then refined to reflect enough information to be transformed into Rainbow Stitch code [7]. At the moment all these steps are performed manually (see Section 5 for future work). Figure 9 depicts the different artifacts within the phases clarifying how Adapt Cases are embedded into the design process of adaptive systems.

The top of Figure 9 depicts the goal graph that we identified for the Znn.com example. Goals and requirements are refined into use cases and Adapt Cases in the logical design phase. Within this phase, both use cases and Adapt Cases may be refined to capture more specific information. Finally, use cases and Adapt Cases may be mapped to software components in the technical design or—as suggested in Figure 9—be transformed to Rainbow's Stitch code that describes adaptation strategies textually.

For reasons of brevity, we skip the analysis phase in the description of the case study and start in the logical design phase with the use cases which were inferred from the goals. As shown in Figure 10 customers and the Znn.com system are the relevant actors in this scenario. The customer sends a news request and Znn.com provides the requested news. As described, the efficient provision of content is affected by contextual changes. Handling such changes, the high-level Adapt Case *Optimize news service* adapts the use case *Provide news content to customer*.

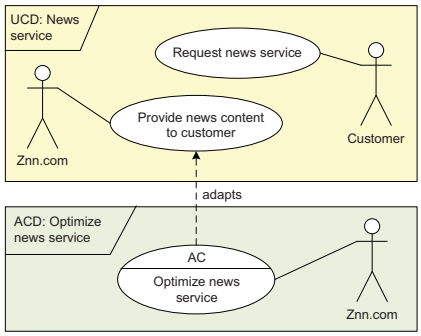

Figure 10: Znn.com Business Use Case

The high-level Adapt Case optimizes the news service for maintaining the Znn.com business objectives even in a changing system context. To recognize relevant context changes, elements in the context and the system itself are monitored. For instance, the client response time is checked against a threshold for the maximal acceptable response time addressing the customer satisfaction. In addition, the average server load in the Znn.com system is monitored.

Comparable to use case modeling, Adapt Cases may be refined to reflect more (technical) details. This allows the smooth transition from logical design to technical design via refinement and transformation. For example, refined Adapt Cases might be assigned to software components in the technical design phase. In Figure 11, the business use case *Provide news content to customer* has been refined to the system use case *Handle news request* that includes the dispatching of incoming requests to the server pool and the presentation of news content to the customer according to his request. The Adapt Case *Optimize news service* is refined into the Adapt Case *Optimize request response time* adapting the system use case *Handle news request*.

To provide a precise definition of the Adapt Case, we use the developed meta-model described in Section 2.1 and specify the *Optimize request response time* Adapt Case. The resulting model is illustrated in Figure 12. In order to optimize the request response time with respect to customer satisfaction and server costs, elements in the context and the systems needs to be considered. For this purpose, the *Server load monitoring* defines two corridors and pulses. The *Client response time corridor* and the *Average response ti-*

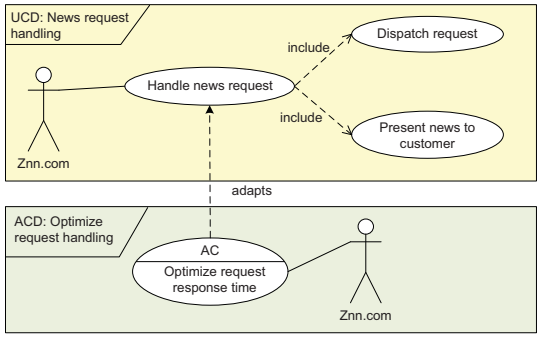

Figure 11: Znn.com System Use Case

me are measured at the client side. The *Server load corridor* and the pulse *Average server load* consider the current load of the server pool in the Znn.com system.

According to changes of monitored elements, the monitor definition throws events. The event *Response time exceeds threshold* indicates that the average response time exceeds the defined upper bound in the *Client response time corridor*. If the *Average server load* is below the defined performance buffer that was defined in the lower bound of the *Server load corridor*, the event *Server load capacity higher than required* is thrown.

To handle the defined events, adaptation actions with different alternatives are specified. If the event *Response time exceeds threshold* is thrown, the response time needs to be decreased. The adaptation action *Decrease response time* defines two alternative adaptation. With respect to customer satisfaction, increasing the server pool is the preferred alternative. If the pool already reached the maximal number of servers, the content fidelity is reduced to textual representation. The semantics of the adaption action *Decrease response time* are defined in Listing 3.

```
def IncreaseNumberOfServerInPool()
 component->select(cmp|cmp.type =
  "LoadBalancer").addServerToPool()

def SwitchToTextualMode()
  component->select(cmp|cmp.type =
      "ContentPresentationController")
   .setContentFidelity(text);

def DecreaseResponseTime.run()
  if ServerPool.isMaximized() then
    SwitchToTextualMode.run()
  else
    IncreaseNumberOfServerInPool.run()
  endif
```

Listing 3: Semantics for Act. *Decrease response time*

The event *Server load capacity higher than required* triggers the *Decrease server load capacity* action. As specified in Listing 4 the event can be handled in two different ways. Addressing the most important business objective of customer satisfaction, the available performance should preferably be used to increase the content fidelity. If the content is already presented in multimedia fidelity, the number of servers in the pool is decreased to minimize the operating costs.

```
def DecreaseNumberOfServerInPool()
 component->select(cmp|cmp.type =
  "LoadBalancer").removeServerFromPool()

def SwitchToMultimediaMode()
  component->select(cmp|cmp.type =
      "ContentPresentationController")
   .setContentFidelity(multimedia);

def DecreaseServerLoadCapacity.run()
  if ContentPresentationController.isHighFidelity
      () then
    DecreaseNumberOfServerInPool.run()
  else
    SwitchToMultimediaMode.run()
  endif
```

Listing 4: Semantics for *Decrease server load capacity*

Note that in addition to the semantics of the different adaptation actions, the Adapt Case semantics itself are defined by a generic frame as described in Section 2.1, Listing 1.

Using the described model, the defined Adapt Case considers the business objectives for Znn.com and their maintenance in a changing context. Further, the model includes the previous mentioned high-level adaptations to react to context changes.

3.3 Case Study Discussion

The presented case study clarifies several benefits of our approach. First of all, specifying adaptivity requirements for the Znn.com scenario is possible using Adapt Cases. Especially, utilizing the monitor concept, complex event-condition-action rules can be described easily while *adaptation cause and reaction* can still be *distinguished* in a reasonable manner. The underlying control-loop concept (given by the Stitch code used to implement Znn.com for Rainbow [7]) could successfully be reflected using Adapt Cases. Therefore, we can answer **RQ1** positively.

This distinction between adaptation cause and reaction allows an easy translation (in terms of MDA) to rule-based adaptation languages, such as Rainbow's Stitch. In addition, the smooth integration in the use case modeling method showed that it is fairly easy to integrate Adapt Cases with standard development processes which use UML. Thus, we can answer **RQ2** positively, too.

Finally, the case study has shown that Adapt Cases allow the *explicit* specification of adaptivity making it more visible in early project phases. We realized strong advantages in using Adapt Cases when it comes to inferring concrete technical system designs from the requirements. The ability to model adaptivity on *different abstraction levels* helps in conceiving the described adaptivity in the same way as can be recognized in traditional use case modeling. Further, as depicted in Figure 9, Adapt Cases provide the ability to model the adaptivity in the logical design. The consideration and specification narrows the gap between requirements and technical design of adaptive systems. Hence, we also answer **RQ3** positively.

Summarizing the result, the case study showed that Adapt Cases satisfy the defined requirements for an adaptivity modeling language.

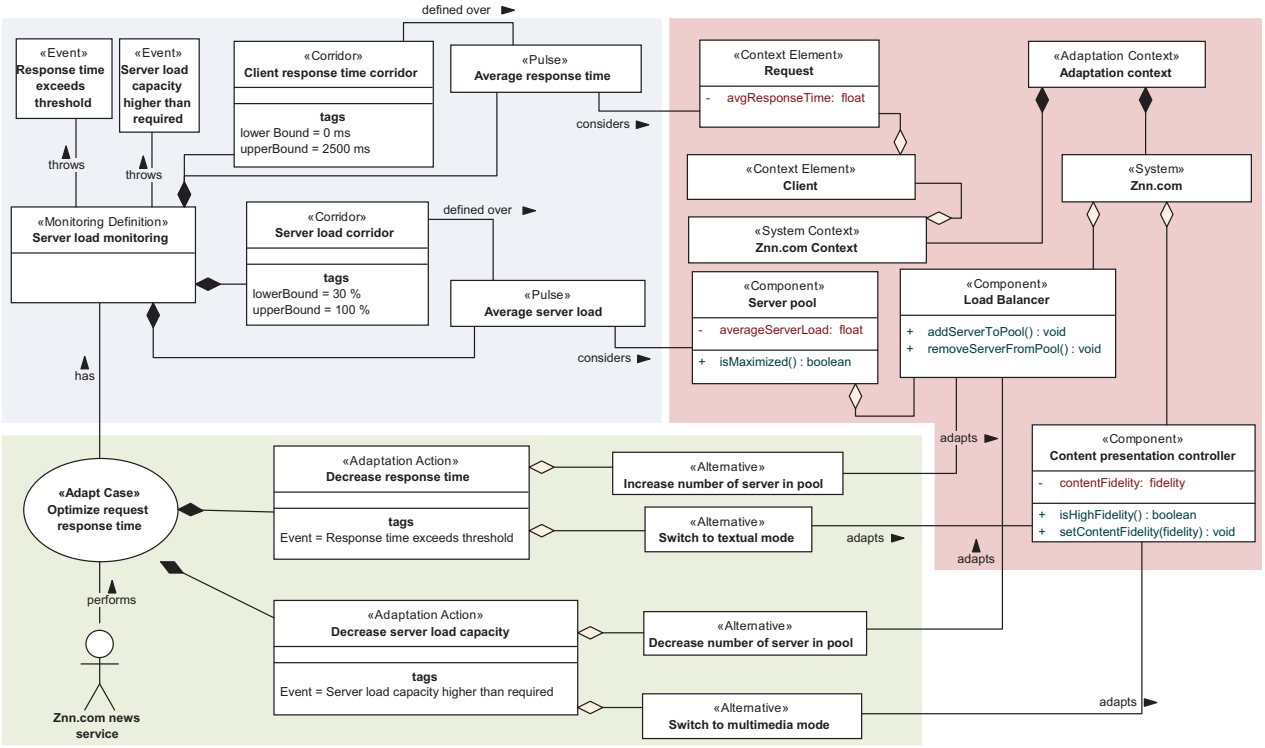

Figure 12: Znn.com Adapt Case Model

4. RELATED WORK AND DISCUSSION

Recent research provides various approaches that support the engineering of adaptive systems in different phases of the design process. The approaches and their assignment to the software engineering process are shown in Figure 13.

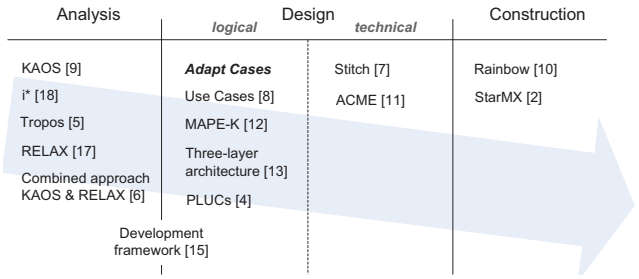

Figure 13: Related approaches

In the analysis phase goal-oriented modeling approaches like i* [18], TROPOS [5] and KAOS [9] support the specification of high-level goals and their incremental refinement to subgoals and requirements. Using goal models, alternative behaviors can be specified.

Considering and specifying the aspect of uncertainty in requirements, RELAX [17] provides an extended controlled natural language. RELAX allows the relaxation of requirements using phrases like "AS EARLY AS POSSIBLE", thus accounting for the uncertainty which might prevent the original unrelaxed requirement to be met. RELAX introduces means to express uncertainty in requirements. Furthermore, relevant properties in the system context and dependencies between requirements can be expressed.

Cheng et al. present a combined, goal-based modeling approach in [6] using the KAOS methodology that explicitly considers uncertainty and the context the system is acting in. Identified uncertainty is modeled using the RELAX notation. The system context is defined by a domain model and referenced in the goal-model.

All these approaches take care of adaptivity in requirements and thus are input for Adapt Cases in the development process. In our opinion, these approaches cannot replace Adapt Cases, which support the logical design of adaptive systems.

In [15] the Tropos methodology is used in a development framework for autonomous systems. This approach explicitly addresses the system context and the correlations between different kinds of goals and entities in the system and the system context. In order to make these relations more concrete a concept of conditions is introduced. Although this approach supports the specification of aspects of the logical design, it is not clear in which way monitoring definitions and concrete adaptation actions can be specified without creating huge and unclear goal lattices.

Use cases [8] are a well-known modeling technique for the logical system design. Using „traditional" use cases, the described adaptivity of the system is hidden within the attached activity or sequence diagrams. As a result, adaptation is not sufficiently taken into account during the logical design. Adapt Cases extend use case modeling by making the explicit specification of adaptation actions available.

Extended use case approaches for software product lines [4] provide the specification of variations which can be related to simple condition concepts. The description of adaptation actions are much more complex than the modeling

of variations since additional aspects like monitored elements in the system and the system context need to considered and are not supported by these approaches. Adapt Cases provide the detailed modeling of the system and the system context. Thereby, adaptation actions considering changes in the system and the system context can be defined.

Existing architectural approaches for adaptive systems like MAPE-K [12] and the three-layer reference architecture proposed in [13] describe abstract logical concepts for realizing adaptivity. These approaches specify aspects and characteristics of adaptive behavior on a high level of abstraction and cannot be used for specifying concrete adaptation actions. Nonetheless, these approaches lay down the foundations for the specification of adaptivity in Adapt Cases.

In the technical design, formal specification languages are used to design platform-specific models for adaptivity. Stitch provides a language [7] that allows the specification of adaptation concepts and can be combined with the architecture description language ACME [11] that specifies systems and contexts in formalized models. These models implement the adaptivity specified in the logical design for a specific platform in contrast to platform-independent Adapt Cases. The technical design models can be implemented by frameworks like Rainbow [10] and StarMX [2].

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a model-based approach that allows the explicit modeling of the adaptivity a particular system shall exhibit. For our modeling language, named Adapt Cases, which is based on traditional use case modeling, we presented a meta-model to formally define the language's elements and defined its semantics. We showed how Adapt Cases can be used to model adaptivity on different levels of abstraction and how they facilitate the distinction of adaptation cause and reaction. Additionally, we sketched how Adapt Case modeling can be introduced into existing engineering processes that are based on the UML. Finally, we showed the applicability of Adapt Cases in a case study that uses the well-known and accepted example of a *slashdot*-like news system, called Znn.com [7].

Currently, our future work is twofold. First, we are working on a more detailed formalization that provides the quality assurance of adaptive systems modelled with Adapt Cases. Second, we are working on the integration of the Adapt Cases into the engineering process of adaptive systems. This addresses the systematic derivation of Adapt Cases from requirements and the inference of the technical design for different frameworks like Rainbow [10] or StarMX [2].

6. REFERENCES

[1] G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In H.-W. Gellersen, editor, *Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*, pages 304–307. Springer Berlin / Heidelberg, 1999.

[2] R. Asadollahi, M. Salehie, and L. Tahvildari. StarMX: A Framework for Developing self-managing Java-based Systems. *Proceedings of the 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 58–67, 2009.

[3] L. Baresi and L. Pasquale. Live Goals for Adaptive Service Compositions. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 114–123, 2010.

[4] A. Bertolino, A. Fantechi, S. Gnesi, and G. Lami. Product Line Use Cases: Scenario-Based Specification and Testing of Requirements. In T. Käköla and J. Duenas, editors, *Software Product Lines*, pages 425–445. Springer, 2006.

[5] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[6] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In *Proceedings of MODELS*, volume 5795 of *LNCS*, pages 468–483. Springer, 2009.

[7] S. Cheng. *Rainbow: cost-effective software architecture-based self-adaptation*. PhD thesis, Carnegie Mellon University, 2008.

[8] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 2000.

[9] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[10] D. Garlan, S. Cheng, and B. Schmerl. Increasing System Dependability through Architecture-based Self-repair. In *Architecting Dependable Systems*, pages 61–89. Springer, 2003.

[11] D. Garlan, R. T. Monroe, and D. Wile. Acme: An Architecture Description Interchange Language. In *CASCON*, pages 169–183, 1997.

[12] J. Kephart and D. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[13] J. Kramer and J. Magee. Self-Managed Systems: An Architectural Challenge. In *Future of Software Engineering (FOSE)*, pages 259–268, 2007.

[14] A. Lapouchnian, W. Robinson, V. Souza, and J. Mylopoulos. Awareness Requirements for Adaptive Systems. Technical report, Department of Information Engineering and Computer Science, University of Trento, Italy, 2010.

[15] M. Morandini, L. Penserini, and A. Perini. Towards goal-oriented development of self-adaptive systems. In *Proceedings of the 2008 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 9–16, 2008.

[16] OMG. Model Driven Architecture, 2001. `http://www.omg.org/mda/`.

[17] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J. Bruel. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In *Proceedings of the 17th IEEE International Requirements Engineering Conference (RE)*, pages 79–88. IEEE Computer Society, 2009.

[18] E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, pages 226–235, 1997.