Adapt Cases 4 BPM

Adaptivity Engineering für flexible und anpassbare Prozesse

Dissertation

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

an der Fakultät für Elektrotechnik, Informatik und Mathematik

der Universität Paderborn

vorgelegt von

Alexander Teetz

Paderborn, Februar 2019

Zusammenfassung

Diese Dissertation ist im Kontext des *NRW Fortschrittskollegs „Gestaltung von flexiblen Arbeitswelten“* entstanden, das sich mit der digitalen Transformation von Fertigungssystemen auseinandersetzt. Derartige Systeme und deren Prozesse müssen zukünftig durch eine erweiterte Flexibilität gestaltet werden, da immer mehr Eigenschaften von unterschiedlichen, miteinander vernetzten Ressourcen berücksichtigt werden sollen.

Aus der Sicht der Domäne des *Business Process Management* (*BPM*) orchestrieren Prozesse wesentliche Teile dieser Systeme, die neben Verhalten der Anwendungslogik insbesondere auch Verhalten der Anpassungslogik enthalten können. Erst durch die Anpassungslogik wird ein hoher Grad an angestrebter Flexibilität in Prozessen ermöglicht. In bisherigen Methoden des *BPM* ist die getrennte Gestaltung beider Typen von Verhalten nicht vorgesehen, wodurch eine erhöhte Komplexität in der Handhabung und Wartung der Prozesse entsteht.

Diese Arbeit stellt einen neuartigen Ansatz für die Domäne *BPM* vor, der auf Methoden und Techniken des *Adaptivity Engineering* basiert, welches zur getrennten Gestaltung der beiden Typen von Verhalten eingesetzt werden kann. Hierzu wurden die Sprache ACML4BPM, zugehörige Entwurfsmuster sowie eine Methode zur Gestaltung erarbeitet.

Die Sprache ACML4BPM stellt in der Domäne *BPM* eine neuartige Sprache zur Gestaltung von flexiblen und anpassbaren Prozessen dar. Sie unterstützt die getrennte Gestaltung von Aspekten der Anwendungs- und Anpassungslogik. Ferner wurden in ihr domänenspezifische Konzepte integriert, wie z.B. Operationen zur Anpassung oder einen Ansatz zur Reaktion auf Ereignisse. Die Sprache nutzt Konzepte des Defacto-Standards *BPMN2.0* und unterstützt hierdurch die Gestaltung durch Nutzer und Domänenexperten.

Damit die Gestaltung von Flexibilität weiter unterstützt werden kann, wurde sich mit Typen von Flexibilität auseinandergesetzt, für die insgesamt vier grundlegende Entwurfsmuster beschrieben worden sind. Hierdurch wurden diverse Erweiterungen der Sprache ACML4BPM beschrieben und ihre Verwendung gezeigt.

Abschließend wurde eine methodische Integration von zuvor aufgeführten Lösungsteilen in einen BPM-Lebenszyklus beschrieben.

Abstract

This PhD thesis was written in the context of the *NRW Fortschrittskolleg „Gestaltung von flexiblen Arbeitswelten“*, that deals with the digital transformation of manufacturing systems. Such systems and their processes will have to be shaped by enhanced flexibility in the future, as more and more characteristics of different interconnected resources have to be taken into account.

From the point of view of the domain of *Business Process Management* (*BPM*), processes orchestrate significant parts of these systems, which, in addition to the behaviour of the application logic, can include the behaviour of the adaptation logic. Only by the adaptation logic a high degree of desired flexibility in processes can be achieved. Previous methods of the *BPM* do not provide the separate design of both types of behaviour, resulting in an increased complexity in the handling and maintenance of the processes.

This work presents a novel approach to the *BPM* domain, based on methods and techniques of *Adaptivity Engineering*, which can be used to separate the design of these two types of behaviour. For this purpose, the language ACML4BPM, associated design patterns and a method for designing processes were developed.

The language ACML4BPM is a novel language in the *BPM* domain for the design of flexible and adaptable processes. It supports the separate design of aspects of application and adaptation logic. Furthermore, domain-specific concepts have been integrated into it, such as operations for adaptations or an approach to respond to events. The language uses concepts of the de-facto standard *BPMN2.0* and thus supports the design by users and domain experts.

In order to support the design of flexibility, different types of flexibility have been discussed for which a total of four basic design patterns have been described. They contain various extensions of the language ACML4BPM and show their use.

Finally, a methodical integration of the previously listed solution parts into a *BPM* lifecycle is described.

Danksagung

An dieser Stelle möchte ich mich bei den Menschen bedanken, die mich in den letzten Jahren bei der Bearbeitung dieser Arbeit auf unterschiedlichen Ebenen unterstützt haben.

An erster Stelle steht dabei mein Doktorvater Herr Prof. Dr. Gregor Engels. Er hat mir über mehrere Jahre ermöglicht, eine breit aufgestellte Forschung zu betreiben und stellte dabei stets hohe Erwartungen, die maßgeblich die Qualität dieser Arbeit bestimmt haben. An zweiter Stelle möchte ich mich bei der Prüfungskommission in Form der Personen Prof. Dr. Britta Wrede, Prof. Dr. Eric Bodden, Jun.-Prof. Dr. Anthony Anjorin und Dr. Thim Strothmann für die aufgewendete Zeit sowie den inhaltlichen Austausch bedanken.

Während meiner Forschung war ich in unterschiedlichen Projekten eingesetzt. Dabei bin ich neben dem fachlichen Austausch auch für die besondere Gelegenheit, über den eigenen Tellerrand schauen und lernen zu können, dankbar. Aus dem Kontext des *NRW Fortschrittskollegs* ist in diesem Bezug insbesondere Sonja Ötting zu nennen, mit der ich die Bearbeitung verschiedener interdisziplinären Themen durchführen durfte. Darüber hinaus werde ich die gute und spannende gemeinsame Zeit mit Thomas John und Christoph Weskamp in dem Projekt *SMART EM* nie vergessen.

Neben Personen aus den unterschiedlichen Projektkontexten sind insbesondere meine beiden Büronachbarn Dennis Wolters und Stephan Heindorf hervorzuheben, denen ich für viele hilfreichen Gespräche und neue Ideen zwischendurch dankbar bin. Zudem bedanke ich mich aber auch bei Thim Strothmann und Florian Rittmeier für die effiziente Zusammenarbeit und ihr Feedback in Bezug zu Veröffentlichungen bzw. wesentlichen Teilen dieser Arbeit. Als letzte fachbezogene Person möchte ich Mirko Rose hervorheben, mit dem ich stets den Umgang auf Augenhöhe hinsichtlich verschiedener Themen rund um die Organisation der Dissertation pflegen konnte.

Meine tiefste Dankbarkeit bezieht sich jedoch auf meine Lebenspartnerin Elisabeth Herick. Ihre Unterstützung und ihr Glaube an mich hat mich erst dazu befähigt, diese Arbeit zu beginnen und am Ende auch abzuschließen.

– Danke!

Inhaltsverzeichnis

Seite

Anhang 293

Kapitel **1**

Einleitung

In diesem Kapitel wird zunächst in Abschnitt 1.1 der Kontext der Arbeit beschrieben. Anschließend wird in Abschnitt 1.2 auf die in dieser Arbeit angenommene Problemstellung sowie auf die zugehörigen abgeleiteten Forschungsfragen eingegangen. In Abschnitt 1.3 werden Anforderungen für mögliche Lösungsansätze beschrieben. Das Kapitel schließt in Abschnitt 1.4 mit einer Übersicht über den strukturellen Aufbau der vorliegenden Arbeit.

1.1 Motivation

Diese Dissertation ist im Kontext des *NRW Fortschrittskollegs „Gestaltung von flexiblen Arbeitswelten“* entstanden. Dieses Fortschrittskolleg basiert auf der Idee, die wissenschaftliche Weiterqualifizierung in Form einer Promotion durch ein inter- und transdisziplinäres Umfeld zu unterstützen. Als inhaltlicher Schwerpunkt wurden unterschiedliche Aspekte aus einer digitalisierten Arbeitswelt gesetzt. Für diese Arbeit war dieser Aspekt durch neuartige Gestaltungstechniken für flexible Prozesse gegeben. Ein Typ einer digitalisierten Arbeitswelt mit flexiblen Prozessen ist durch den Begriff der *Industrie 4.0* geprägt worden und wird nun nachfolgend als weitere Motivation verwendet.

Als eine Schlüsselrolle im Kontext von *Industrie 4.0* wird ein Paradigmenwechsel in der Gestaltung von neuartigen Produktionsumgebungen gesehen. Derartige Produktionsumgebungen – auch *Smart Factories* genannt – erlauben erstmals, das Konzept der *Mass Customization* unter ökonomisch tragfähigen Bedingungen anzuwenden. *Mass Customization* beschreibt dabei die Möglichkeit zur Produktion von individualisierbaren Produkten (Losgröße 1) [Spa+13]. Für die Fertigung dieser Produkte ist eine erhöhte Flexibilität der beteiligten Produktionssysteme notwendig, sodass stets

schnell und bedarfsgerecht auf geänderte Anforderungen reagiert werden kann.

Dabei stehen Prozessunterstützungssysteme im Vordergrund, durch die die relevanten Fertigungsprozesse IT-gestützt ausgeführt werden. Als Voraussetzungen für die Funktion dieser Systeme sind die vertikale als auch horizontale Vernetzung aller in der Wertschöpfungskette beteiligten Prozesse, Entitäten und Datenobjekte notwendig. Durch den Austausch von relevanten Informationen in Echtzeit können Anpassungen von Produktionsumgebungen durchgeführt werden, sodass Fertigungsprozesse zu optimalen Konditionen ablaufen können.

Bei derartigen Produktionsumgebungen handelt es sich aber nicht um ausschließlich technische Systeme, sondern um sozio-technische Systeme. Der Mensch stellt somit einen nicht zu vernachlässigenden Faktor dar [Wie13]. Somit sind bei der Gestaltung solcher Systeme neben technischen Aspekten der Produktionsumgebung auch intra- und interorganisationale arbeitswissenschaftliche Umstände zu berücksichtigen [Lud+16]. Beispiele hierfür sind digitale Beschreibungen von spezifischen Aspekten der Arbeitnehmerinnen und Arbeitnehmer oder für sie relevante Aspekte der Produktionsumgebung. Eine Berücksichtigung dieser Aspekte in ihrer Gestaltung kann bei der Produktion von individualisierbaren Produkten dazu beitragen, Fertigungsprozesse hinsichtlich ihrer Flexibilität und Effizienz zu verbessern. Darüber hinaus bildet die Berücksichtigung dieser Aspekte zum Zeitpunkt der Gestaltung die Basis für Anpassungen an der Produktionsumgebung und an ihren Fertigungsprozessen zum Zeitpunkt der Ausführung.

Für den Betrieb neuartiger Produktionsumgebungen ist somit nicht mehr nur ein einzelner standardisierter Prozess, sondern vielmehr ein Netzwerk aus flexiblen Prozessen notwendig. Insbesondere im Rahmen möglicher Anpassungen der Produktionsumgebungen und ihrer Fertigungsprozesse müssen eine Vielzahl an unterschiedlichen zu behandelnden Aspekten aus verschiedenen sowohl technischen als auch soziologischen Bereichen berücksichtigt werden. Setzt man als Beispiel ablaufende Prozesse mit menschlicher Beteiligung in den Fokus der Betrachtung, so ergeben sich für die Gestaltung der benötigten Prozessunterstützungssysteme verschiedene allgemeine Fragestellungen, auf die nachfolgend eingegangen wird.

Frage 1: Was sind flexible Arbeitsprozesse?

Industrie 4.0-Anwendungen sind durch eine Vielzahl variierender Faktoren geprägt. So sind die geschäftlichen Abläufe in einer derartigen Anwendung sowohl von intra- als auch interorganisationalen Gegebenheiten ab-

hängig. Ein in diesem Bezug viel genanntes Beispiel ist die erhöhte Produktvariabilität, welche auch besser bekannt ist unter dem Stichwort *Losgröße 1* [Spa+13; Rus13; Las+14; Kau15]. Eine gesteigerte Produktvariabilität hat maßgeblich Einfluss auf die zur Produktion verwendeten Prozesse. So sind je nach Produktvariante bestimmte Teilprozesse an der Produktion beteiligt oder eben nicht. Der Gesamtprozess zur Fertigung eines solchen Produktes muss daher flexibel hinsichtlich der durch den Kunden bestellten Konfiguration eines Produktes sein.

Sind an der Fertigung der Produkte auch Mitarbeiterinnen und Mitarbeiter beteiligt, z.B. in Form von Montageaufgaben, so spricht man in diesem Teil des Fertigungsablaufs auch von Arbeitsprozessen. Somit zeichnen sich Arbeitsprozesse hinsichtlich ihrer Ausführungsform besonders dadurch aus, dass der Mensch und nicht etwa eine Maschine oder ein IT-Service ausschließlich als leistungserbringender Teilnehmer beteiligt ist. Insbesondere unter der Annahme des Vorhandenseins von nicht automatisierbaren Aufgaben bleiben Arbeitsprozesse für die Leistungserbringung in Unternehmen ein wesentlicher Faktor. Gleichwohl es in Arbeitsprozessen Mischformen der Leistungserbringung geben kann, wie z.B. in Form einer Mensch-Maschine-Kollaboration, ist der Mensch hier stets beteiligt.

Berücksichtigt man die steigende Produktvariabilität in Industrie 4.0-Anwendungen, so hat diese auch Einfluss auf die beteiligten Arbeitsprozesse. Wie bereits zuvor für Fertigungsprozesse beschrieben, ist auch für Arbeitsprozesse eine erhöhte Flexibilität notwendig, da Aufgaben für unterschiedliche Produktvarianten wechseln können. Mitarbeiterinnen und Mitarbeiter sehen sich daher zukünftig mit flexiblen Arbeitsprozessen konfrontiert, in denen ihre Aufgaben häufiger als zuvor wechseln können, da je nach Konfiguration des zu fertigenden Produktes einzelne Montageschritte geändert oder hinzugefügt werden oder gar ganz wegfallen können.

Frage 2: Was sind menschenzentrierte Arbeitsprozesse?

Hinsichtlich einer ethischen Perspektive auf Arbeitsprozesse, in der z.B. die Humanisierung der Arbeitswelt oder der demographische Wandel im Vordergrund stehen [Deu+15; KJP15; WRN14], kann die Betrachtung von individuellen Eigenschaften der Arbeitnehmenden für die Gestaltung von Arbeitsprozessen relevant sein. So ist es denkbar, dass gewisse Aufgaben z.B. aufgrund ihrer körperlichen Anforderungen nicht für jede Mitarbeiterin oder jeden Mitarbeiter zu bewerkstelligen sind. Werden individuelle Eigenschaften von Arbeitnehmern in der Gestaltung der Arbeitsprozesse berücksichtigt, kann von menschenzentrierten Arbeitsprozessen gesprochen werden.

Frage 3: Was sind menschenzentrierte, anpassbare Arbeitsprozesse?

In Industrie 4.0-Anwendungen sind die beteiligten Prozesse, Entitäten und Datenobjekte sowohl vertikal als auch horizontal in Wertschöpfungsnetzwerken miteinander vernetzt. Hierdurch wird das Ziel der schnellen und bedarfsgerechten Anpassung (*analog: Adaption*) an sich ändernde Anforderungen verfolgt. Im Rahmen dieser Arbeit wird nachfolgend angenommen, dass sich die ändernden Anforderungen stets durch diskrete Ereignisse einstellen, die sowohl planbar als auch unvorhersehbar sein können. Ein diskretes Ereignis stellt dabei einen festen Zeitpunkt dar, zu dem eine Änderung der genannten Anforderungen eintritt. Für den Fall des Auftretens eines Ereignisses können Regelsätze beschrieben werden, die eine Analyse von aktuellen Umständen und eine Auswahl von zulässigen Anpassungen innerhalb einer Industrie 4.0-Anwendung vornehmen. Das Ziel einer solchen Anpassung von beteiligten Prozessen, Entitäten und Datenobjekten ist die Einhaltung von Anforderungen, die an den Betrieb der Industrie 4.0-Anwendung gesetzt worden sind. Wendet man derartige Regelsätze zur Ausführungszeit auf die beteiligten Prozesse an, handelt es sich um anpassbare Prozesse oder in dem hier angeführten Beispiel insbesondere um menschenzentrierte, anpassbare Arbeitsprozesse.

Frage 4: Was sind mögliche Treiber für flexible und menschenzentrierte, anpassbare Arbeitsprozesse?

Als letztes stellt sich die Frage, was die treibenden Kräfte für die Gestaltung von menschenzentrierten, anpassbaren Arbeitsprozessen in Industrie 4.0-Anwendungen sind. Sie können je nach Organisation vielfältig sein und sind maßgeblich durch deren rechtliche als auch betriebliche Richtlinien motiviert. So ist es vorstellbar, dass zukünftig die Anpassung von Arbeitsprozessen auf Basis veränderter Umstände der Mitarbeiterinnen und Mitarbeiter zur Ausführungszeit arbeitsschutzrechtlich gefordert wird. Die Einführung von flexiblen und anpassbaren Prozessen könnte damit z.B. zur Steigerung der Arbeitssicherheit beitragen.

Aber auch aus der Perspektive des demographischen Wandels können flexible und anpassbare Prozesse einen Mehrwert bieten, indem sie zu einer Beschäftigung bis in ein hohes Lebensalter beitragen, da zum einen individuelle Eigenschaften der Mitarbeitenden berücksichtigt werden können und zum anderen wertvolles Wissen länger im Unternehmen bestehen bleibt [Jes+14].

Neben rechtlichen oder ethischen Treibern sind aber auch wettbewerbliche Treiber identifizierbar. So kann die Fähigkeit zur Anpassung von Arbeitsprozessen sowie ihrer Umgebung dazu beitragen, die Mitarbeitermotivation und die empfundene Lebensqualität am Arbeitsplatz zu steigern. Mitarbeiterinnen und Mitarbeiter könnten somit in Zukunft motiviert sein, sich für Arbeitgeber zu entscheiden, die ihnen einen besonderen Grad an

Menschenzentrierung durch Flexibilität und Anpassbarkeit hinsichtlich ihrer Arbeitsprozesse und Umgebungen anbieten können.

Die im Rahmen der zuvor vorgestellten Fragestellungen genannten Typen von Prozessen stellen eine mögliche Motivation für die Gestaltung von flexiblen und anpassbaren Prozessen dar. Dabei existieren zahlreiche weitere industrielle Anwendungen, in denen die beteiligten Prozesse nicht an den Menschen, sondern an spezifische Eigenschaften weiterer Entitäten wie z.B. Maschinen, IT-Services oder den Prozessen selbst zentriert sind. Die sich hieraus ergebene Problemstellung wird im nachfolgenden Abschnitt beschrieben.

1.2 Problemstellung

Für flexible und anpassbare Prozesse sind Methoden und Techniken notwendig, mit denen sie sich in einem methodischen Rahmen adäquat gestalten lassen. Dabei können derartige Prozesse aus der Perspektive des Software Engineerings auch als eine spezielle Art eines selbst-adaptiven Systems verstanden werden, da hier ausgehend von Ereignissen innerhalb des Systems (hier: *Prozess*) oder seinem Kontext (hier: *beteiligte Entitäten und Ressourcen*) Analysen und Anpassungen ausgeführt werden. Dabei steht stets der zuverlässige und eigenständige Betrieb der Prozesse im Vordergrund. Eine zugehörige Übersicht über Prozesse als selbst-adaptives System ist in Abbildung 1-1 gegeben.

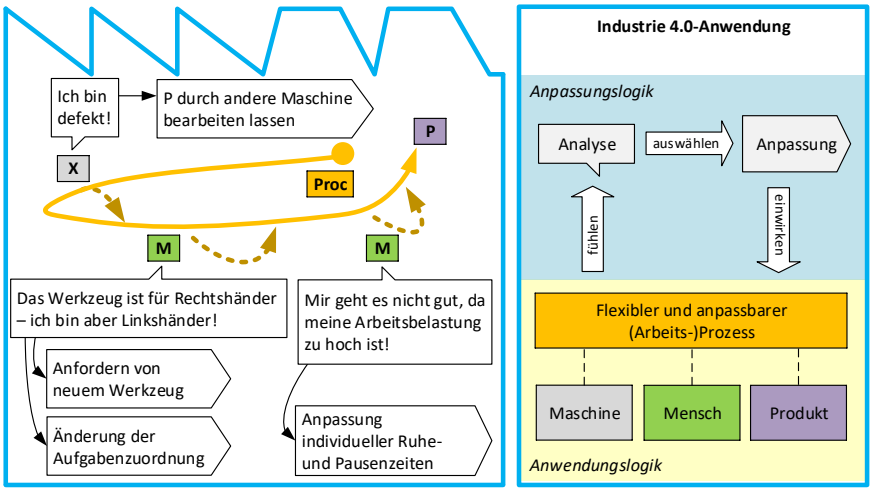


Abbildung 1-1: *Schematische Sicht auf eine Industrie 4.0-Anwendung*

Im rechten Bereich ist das generelle Funktionsprinzip der Industrie 4.0-Anwendung dargestellt, welches das Paradigma einer Kontrollschleife

umsetzt [Bru+09]. Dabei kann eine Reihe von unterschiedlichen Entitäten an dem Prozess beteiligt sein. Beispiele für derartige Entitäten sind hier durch Maschinen, Menschen und Produkte gegeben. Ausgehend von diskreten Ereignissen können Analysen angestoßen werden, die je nach Erfordernis eine Anpassung an den Prozessen sowie an beteiligten Entitäten auswählt und nachfolgend ausführt.

Im linken Bereich der Abbildung ist ein Beispiel für das zuvor beschriebene Funktionsprinzip gegeben. Der flexible und anpassbare Prozess wird hier als *Proc* dargestellt. Er erzeugt die Entität *P*, welche das zu fertigende Produkt repräsentiert. Entlang des Prozesses *Proc* wird die Maschine *X* sowie ein menschlicher Akteur *M* eingebunden. Sowohl *X* als auch *M* sind in der Lage, bestimmte Ereignisse zu erzeugen, auf die durch eine vordefinierte Maßnahme reagiert werden soll.

So ist hier dargestellt, dass die Maschine *X* melden könnte, dass sie einen Defekt aufweist. In diesem Fall wird eine Anpassung an den Prozess durchgeführt, die im weiteren Verlauf das Produkt *P* durch eine andere Maschine bearbeiten lässt. Wie in der Motivation bereits eingeführt wurde, kann aber auch auf Ereignisse ausgehend von menschlichen Akteuren reagiert werden, um einen bestimmten Grad an Menschenzentrierung zu erreichen. So wird in dem hier gezeigten Beispiel die Möglichkeit zur Berücksichtigung der Arbeitsbelastung bedacht. Tritt eine durch einen Menschen *M* angezeigte Überbelastung ein, sollen Maßnahmen zur Anpassung seiner individuellen Ruhe- und Pausenzeiten durchgeführt werden. Ergänzend kann der Mensch *M* aber auch melden, dass er für die Verwendung eines bestimmten Werkzeugs ungeeignet ist, da eine persönliche Eigenschaft die Verwendung des Werkzeugs erschwert oder unmöglich macht. Für diesen Fall sind zwei mögliche Maßnahmen angedacht, durch die entweder ein alternatives Werkzeug angefordert wird oder ein Wechsel der zugeordneten Aufgabe erfolgt.

Für die Gestaltung derartiger selbst-adaptiver Systeme stellt das *Adaptivity Engineering (AE)* nach [Luc+11; LE13; Luc13] bereits einen etablierten Rahmen dar. *Luckey* stellt in seiner Arbeit neben dem Gesamtansatz *Adapt Cases* einen modellbasierten Entwurfsprozess sowie die zugehörige Sprache *Adapt Case Modeling Language (ACML)* zur Gestaltung von selbst-adaptiven Systemen vor.

Das wesentliche Grundprinzip des *Adaptivity Engineering* ist dabei die Trennung der Anwendungs- von der Anpassungslogik. Man spricht in diesem Zusammenhang im Software Engineering auch von *Separation-of-Concerns (SoC)* – also die Dekomposition von unterschiedlich zu spezi-

fizierenden Aspekten eines Softwaresystems. Hierdurch ist es möglich, verschiedene Aspekte des Softwaresystems unter einer spezifischen Sicht und unabhängig voneinander betrachten zu können [Dij76; Par02]. In Abbildung 1-1 ist eine exemplarische Trennung der beiden Logiken bereits dargestellt worden. So umfasst die Anpassungslogik notwendiges Verhalten für die Analyse als auch für auszuwählende Maßnahmen in Form einer Anpassung an Entitäten der Anwendungslogik. Der Ansatz *Adapt Cases* setzt dieses Grundprinzip durch die getrennte Gestaltung von Anpassungs- und Anwendungslogik um. *Luckey* argumentiert in seiner Arbeit, dass durch das *Adaptivity Engineering* die getrennte Modellierung von unterschiedlichen Aspekten, wie der Anpassungs- und Anwendungslogik, die Qualität der jeweils erarbeiteten Modelle erhöht werden kann.

Die Sprache *ACML* stellt laut *Luckey* [Luc13] in diesem Bezug eine sogenannte *General Purpose Language (GPL)* für die Gestaltung von selbstadaptiven Systemen dar. *GPLs* zeichnen sich dabei durch eine Vielzahl an unterschiedlichen Sprachelementen aus, die eine breite Verwendung in der Gestaltung von Systemen aus unterschiedlichen Anwendungsdomänen ermöglichen. Ein alternatives Beispiel für eine *GPL* stellt die *Unified Modeling Language (UML)* [OMG10] dar, die für den Bereich der objektorientierten Gestaltung von Softwaresystemen eingesetzt werden kann.

Aufgrund der Vielzahl an unterschiedlichen Sprachelementen können *GPLs* als sehr anspruchsvoll in ihrer Verwendung betrachtet werden [Voe+13]. Im Kontrast zu *GPLs* existieren daher auch sogenannte *Domain-Specific Languages (DSLs)*. *DSLs* fokussieren dabei die Verwendung in einer ausgewählten Anwendungsdomäne. Die Sprachelemente in *DSLs* sind somit spezifischer auf einen in der Anwendungsdomäne gesetzten Fokus ausgerichtet.

Die Gestaltung von Flexibilität und Anpassbarkeit in Prozessen mit eingeschlossener Trennung der Anpassungs- und Anwendungslogik kann einen solchen Fokus darstellen. Dabei existiert jedoch bisher keine Sprache, die domänenspezifische Konzepte von flexiblen und anpassbaren Prozessen mit Konzepten des *Adaptivity Engineering* so in Verbindung setzt, dass dieser Fokus eingenommen werden kann. Daher stellt sich in diesem Bezug die nachfolgende Forschungsfrage:

Forschungsfrage 1

Wie lassen sich auf Basis von Konzepten des Adaptivity Engineering flexible und anpassbare Prozesse domänenspezifisch gestalten?

Neben der reinen domänenspezifischen Gestaltung von Flexibilität und Anpassbarkeit kann der methodische Rahmen zur adäquaten Gestaltung

weiter unterstützt werden. So ist es gängige Praxis, dass im Software Engineering sogenannte Entwurfsmuster (engl. *Design Patterns*) eingesetzt werden. Entwurfsmuster stellen dabei eine Erfolgsmethode (engl. *Best-Practice*) dar, durch die sich stetig wiederholende Probleme bzw. Aspekte eines Systems mit vorgefertigten Lösungsbausteinen adäquat beschreiben lassen.

Im Rahmen der Gestaltung von Softwaresystemen beschreiben Entwurfsmuster wesentliche Prinzipien, die z.B. einen bestimmten Typ von Verhalten oder der Struktur eines abzubildenden Aspekts betreffen können. Entwurfsmuster können dabei auch die Rolle von Leitfäden bzw. Handlungsempfehlungen einnehmen. Da grundsätzliche Probleme in der Gestaltung bereits durch ein passendes Entwurfsmuster gelöst sein können, ist z.B. durch Wiederverwendung einer Lösungsstruktur eine Unterstützung von Entwicklern bzw. Experten einer Domäne bei ihrer Arbeit möglich.

Neben dieser eher konstruktiven Unterstützung ist auf Basis einer Sammlung von unterschiedlichen Entwurfsmustern auch eine Typisierung von zu gestaltenden Aspekten möglich. Hierdurch kann bereits existierendes Wissen wiederverwendet und somit Aufwand in der Gestaltung eingespart werden.

Für die Gestaltung von flexiblen und anpassbaren Prozessen existieren verschiedene in der Literatur [Sch+08; RW12] bereits vorgestellte Entwurfsmuster. Sie können jeweils als ein Typ von Flexibilität verstanden werden, der in Prozessen vorkommen kann. Unterschiedliche Typen von Flexibilität können dabei oft durch regelbasierte Verfahren realisiert werden. Jedoch sind vornehmlich integrierte Verfahren bekannt, in denen z.B. eine Trennung der Anwendungs- und Anpassungslogik nicht explizit vorgesehen ist. Hierdurch kann es zu komplexen, für den Menschen unübersichtlichen und schwer wartbaren Prozessmodellen kommen, die im Kontext von Industrie 4.0-Anwendungen und ihren flexiblen und anpassbaren Prozessen problematisch sein können. Ferner ist derzeit unbekannt, wie die Gestaltung dieser Flexibilitätsaspekte durch das *Adaptivity Engineering* unterstützt werden kann. Für diese Arbeit ergibt sich aus diesen Umständen daher die nachfolgende Forschungsfrage:

Forschungsfrage 2 *Was sind Flexibilitätsaspekte von Prozessen und wie können sie durch Konzepte eines domänenspezifischen Adaptivity Engineering modellbasiert in Form von Entwurfsmustern beschrieben werden?*

Das *Adaptivity Engineering* ist für die Domäne des *Business Process Management* (*BPM*) eine neuartige Betrachtungsweise auf die zu gestaltenden flexiblen und anpassbaren Prozesse. Daraus resultiert, dass es derzeit

noch unklar ist, wie sich eine zu entwickelnde Sprache sowie die zugehörigen Entwurfsmuster in einen methodischen Rahmen der Domäne *BPM* integrieren lassen. Die Notwendigkeit einer solchen Integration lässt sich hinsichtlich der Verwendbarkeit des *Adaptivity Engineering* begründen. So sollten relevante methodische Aktivitäten und Artefakte des *Adaptivity Engineering* in Relation zu üblichen Verfahrensweisen und Artefakten der Domäne *BPM* gesetzt werden. Erst hierdurch kann verdeutlicht werden, wie eine zu entwickelnde Sprache und zugehörige Entwurfsmuster anzuwenden sind. Die hieraus resultierende Forschungsfrage lautet wie folgt:

Forschungsfrage 3

> *Wie lassen sich Konzepte eines domänenspezifischen Adaptivity Engineering in einen methodischen Rahmen der Domäne BPM integrieren?*

Zuvor wurden Forschungsfragen vorgestellt, die die Ausgangssituation für die vorliegende Arbeit beschreiben. Als Ergänzung werden in dem nachfolgenden Abschnitt 1.3 verschiedene Anforderungen an eine zu erarbeitende Lösung vorgestellt.

1.3 Anforderungen

In diesem Abschnitt werden grundlegende Anforderungen an ein Lösungskonzept vorgestellt. Dabei kann in zwei Arten von Anforderungen unterschieden werden. Zum einen werden Anforderungen beschrieben, die bereits für den Ansatz *Adapt Cases* gesetzt worden waren. Dies lässt sich vornehmlich dadurch begründen, dass sie sowohl für das *Adaptivity Engineering* nach *Luckey* als auch für eine domänenspezifische Variante als notwendig erachtet werden können. Zum anderen kommen aufgrund des domänenspezifischen Bezugs auch neue Anforderungen hinzu, die für den generelleren Ansatz *Adapt Cases* nicht bestanden.

Separation-of-Concerns

In der Sprache *ACML* [Luc+11] werden die Anwendungs- und Anpassungslogik getrennt voneinander gestaltet. Mit einem in dieser Arbeit erstellten Lösungskonzept soll es auch weiterhin möglich sein, das *Separation-of-Concerns* hinsichtlich dieser beiden Logiken durchzuführen.

Kontrollschleife

Ein grundlegendes Paradigma für die Gestaltung von selbst-adaptiven Systemen stellt die Kontrollschleife dar [Bru+09]. Das Paradigma wird bereits durch den Ansatz *Adapt Cases* umgesetzt und soll auch in dem zu erarbeiteten Ansatz in der Gestaltung von flexiblen und anpassbaren Prozessen eingesetzt werden.

Ausdrucksfähigkeit

In der Sprache *ACML* werden *UML Use Case-Diagramme* [OMG10] um weitere Elemente erweitert und zur frühen Spezifikation von einzelnen Funktionen eines Systems eingesetzt. Dabei steht bereits dort die Trennung von Funktionen der Anpassungs- und der Anwendungslogik im Vordergrund. Durch die Nutzung von *UML*-spezifischen Gestaltungstechniken wird der Ansatz *Adapt Cases* hinsichtlich der Spezifikation von Aspekten der beiden Logiken besonders ausdrucksfähig. Um auch die Gestaltung von flexiblen und anpassbaren Prozessen bereits in einer frühen Phase unterstützen zu können, soll auch in einem domänenspezifischen *Adaptivity Engineering* diese Ausdrucksfähigkeit weitestgehend beibehalten werden.

UML-Konsistenz

Die Sprache *ACML* ist weitestgehend in Anlehnung an einzelne Konzepte der Sprache *UML* [OMG10] definiert worden. Hierdurch entsteht eine hohe Konsistenz zwischen der reinen Gestaltung durch Diagramme der *UML* und den durch *ACML* zur Verfügung gestellten Diagrammtypen. Eine Sprache im Rahmen eines domänenspezifischen *Adaptivity Engineering* soll diese Eigenschaft weitestgehend beibehalten.

BPMN2.0-Konsistenz

Eine große Herausforderung stellt die Nutzung von domänenspezifischem Wissen in der Gestaltung von flexiblen und anpassbaren Prozessen dar. Eine fachliche Domäne, in der sich vornehmlich mit unterschiedlichen Aspekten von Prozessen auseinandergesetzt wird, ist das *Business Process Management (BPM)* [Wes12]. Für die Gestaltung von Prozessen im Rahmen des *BPM* ist die Sprache *BPMN2.0* [OMG11] der aktuelle De-facto-Standard. Die Sprache *BPMN2.0* stellt in diesem Bezug eine Implementierung von spezifischen Konzepten der Domäne des *BPM* dar. Für ein domänenspezifisches *Adaptivity Engineering* sollen ausgewählte Konzepte daher in enger Anlehnung an die Sprache *BPMN2.0* definiert werden. Hierbei soll eine hohe Konsistenz zwischen der reinen Nutzung von Diagrammen der Sprache *BPMN2.0* und solcher, die durch eine potentielle Lösung zur Verfügung gestellt werden, erreicht werden.

Musterbasierte Unterstützung in der Gestaltung von Flexibilität

Für häufig wiederkehrende Typen von Flexibilität in Prozessen bietet sich die Bereitstellung von zugehörigen Entwurfsmustern an. In einem domänenspezifischen *Adaptivity Engineering* sollen daher Entwurfsmuster beschrieben werden, die derartige Typen enthalten. Dabei soll insbesondere das *Separation-of-Concerns* hinsichtlich der Trennung von Anpassungs- und Anwendungslogik im Fokus dieser Entwurfsmuster liegen.

Integration in eine spezifische Methode der Domäne BPM

Für die Verwendung der zu entwickelnden Sprache zur Trennung der Anwendungs- und Anpassungslogik sowie zugehöriger Entwurfsmuster ist eine Methode notwendig. Diese Methode sollte relevante Aktivitäten

und Artefakte eines domänenspezifischen *Adaptivity Engineering* in Relation zu Elementen eines bestehenden methodischen Rahmens der Domäne *BPM* setzen.

1.4 Aufbau der Arbeit

Eine Übersicht über die vorliegende Arbeit ist in Abbildung 1-2 dargestellt. Die Arbeit ist insgesamt in die drei Teile *Grundlagen und verwandte Arbeiten*, *Lösungskonzept* sowie *Evaluation, Zusammenfassung und Ausblick* strukturiert. Auf die Inhalte dieser Teile wird nachfolgend kurz eingegangen.

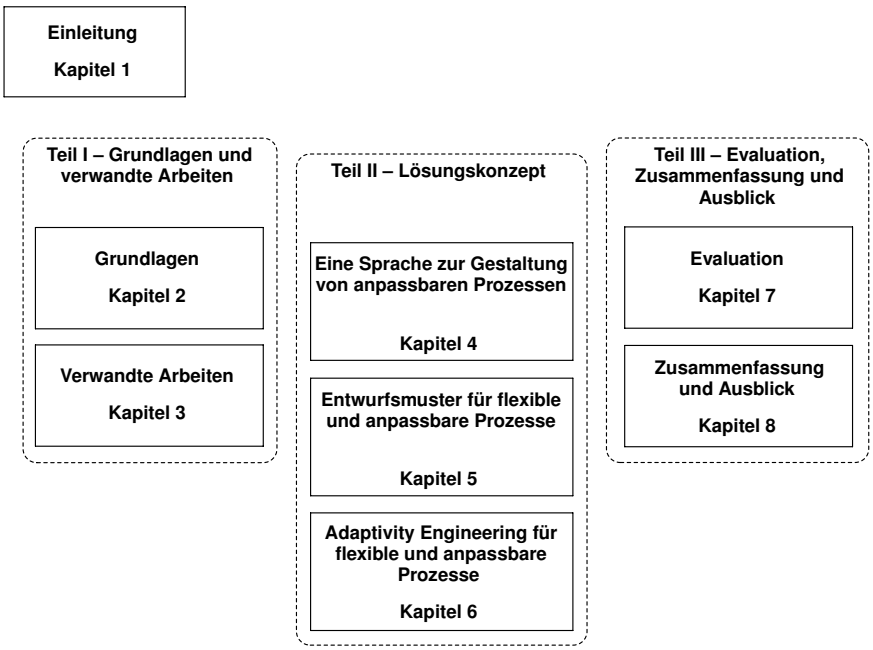


Abbildung 1-2: *Inhalte der Arbeit*

In dem ersten Teil der Arbeit werden sowohl Grundlagen als auch alternative Ansätze für die Gestaltung von Flexibilität in Prozessen vorgestellt. In Kapitel 2 werden die benötigten Grundlagen beschrieben, welche für die Erarbeitung des Lösungsansatzes notwendig waren bzw. unterstützend für das weitere Verständnis der Arbeit sein können. Darunter fällt zunächst die *modellgetriebene Softwareentwicklung (MDSD)*. Ferner sind domänenspezifische Methoden und Techniken notwendig, wie z.B. das *Business Process Management (BPM)* sowie der Bereich der Gestaltung von Prozessen, dem *Business Process Modeling (BPMod)*. Das anschließende Kapitel 3 führt alternative Ansätze für die Gestaltung von flexiblen und anpassbaren Prozessen auf. Der Schwerpunkt dieser Aufführung liegt dabei auf Ansätzen, die vornehmlich in den Domänen *BPM* sowie *BPMod* zu verordnen sind.

Das in dieser Arbeit erarbeitete Lösungskonzept sieht drei wesentliche Lösungsteile vor: Eine domänenspezifische Sprache, die Erarbeitung von relevanten Entwurfsmustern zur Gestaltung von flexiblen und anpassbaren Prozessen und die Beschreibung eines methodischen Vorgehens unter Verwendung der beiden zuvor genannten Lösungsteile. Das Lösungskonzept wird im zweiten Teil der Arbeit beschrieben. Dazu wird zunächst in Kapitel 4 der erste Lösungsteil in Form der domänenspezifischen Sprache *Adapt Case Modeling Language 4 BPM (ACML4BPM)* vorgestellt. Das Kapitel umfasst neben der reinen Beschreibung der Sprache auch die Analyse von Teilen der beiden beteiligten Domänen *BPM* und *BPMod*. Anschließend wird der zweite Lösungsteil in Form von relevanten Entwurfsmustern in Kapitel 5 vorgestellt. Dabei wird für jeden betrachteten Aspekt von Flexibilität in Prozessen zunächst eine umfassende Analyse und anschließend die Beschreibung des zugehörigen Entwurfsmusters vorgenommen. Abschließend wird der dritte Lösungsteil in Form der Methode *Adapt Cases 4 BPM* in Kapitel 6 beschrieben. Diese Methode stellt ein Vorgehen des *Adaptivity Engineering* im Kontext einer spezifischen Entwicklungsmethode der Domäne *BPM* dar. Im Fokus der Methode *Adapt Cases 4 BPM* liegt die Benennung von notwendigen Aktivitäten und Artefakten in Relation zu bestehenden Vorgehensweisen der Domäne *BPM*.

Der dritte und letzte Teil dieser Arbeit umfasst die durchgeführte Evaluation des vorgestellten Lösungsansatzes und stellt anschließend eine Zusammenfassung sowie einen Ausblick auf mögliche zukünftige Forschungsarbeiten vor. Die durchgeführte Evaluation wird in Kapitel 7 beschrieben und gliedert sich in zwei weitere Teile auf. So wird im Rahmen der Evaluation zunächst ein zusammenhängendes Beispiel für die Gestaltung von flexiblen und anpassbaren Prozessen unter Einsatz der erarbeiteten Sprache sowie ausgewählter Entwurfsmuster gegeben. Nachfolgend werden Eigenschaften der beiden erarbeiteten Lösungsteile hinsichtlich verschiedener Kriterien analysiert und bewertet. Im Rahmen der Zusammenfassung und des Ausblicks werden die konkreten Inhalte der Lösungsteile in Kapitel 8 zusammengefasst. Die Arbeit schließt dabei mit einem Ausblick auf mögliche zukünftige Forschungsarbeiten. Dabei wird insbesondere Bezug zur weiteren Verwendung vorgestellter Lösungsteile und auf das *Adaptivity Engineering* genommen.

Teil I

Grundlagen und verwandte Arbeiten

Kapitel **2**

Grundlagen

In diesem Kapitel werden verschiedene Grundlagen vorgestellt, die für die Bearbeitung der Arbeit notwendig gewesen sind und für das weitere Verständnis behilflich sein können. Das Kapitel strukturiert sich dabei in insgesamt drei Abschnitte. Zunächst wird in Abschnitt 2.1 auf grundlegende Aspekte der *modellgetriebenen Softwareentwicklung* eingegangen. Der Fokus liegt hierbei auf der Metamodellierung und auf der Entwicklung von domänenspezifischen Sprachen. Anschließend wird in Abschnitt 2.2 das *Business Process Management (BPM)* vorgestellt, welches als ein methodischer Rahmen zur Verwaltung von Prozessen eingesetzt werden kann. Damit im weiteren Verlauf der Arbeit spezifischer auf die Gestaltung von Prozessen eingegangen werden kann, wird ferner das *Business Process Modeling (BPMod*) als Teildisziplin des *BPM* in Abschnitt 2.3 vorgestellt. Das Kapitel schließt in Abschnitt 2.4 mit einer Vorstellung des durch [Luc13] vorgestellten Ansatzes *Adapt Cases* und der zugehörigen Sprache *Adapt Case Modeling Language (ACML)*, welche beide die Basis für das spätere Konzept zur Lösung bilden.

2.1 Modellgetriebene Softwareentwicklung

Bei der modellgetriebenen Softwareentwicklung (engl. *Model-Driven Engineering (MDE)*) handelt es sich um ein spezielles Entwicklungsverfahren, das insbesondere die Verwendung von Modellen als primäres Artefakt im Entwicklungsprozess betrachtet [BCW17]. Auf wichtige Grundprinzipien des *MDEs* wird nachfolgend detaillierter eingegangen. Darunter befinden sich die *Metamodellierung* (siehe Abschnitt 2.1.1), die *Meta-Object-Facility (MOF)* (siehe Abschnitt 2.1.2) und die *Model-Driven Architecture (MDA)* (siehe Abschnitt 2.1.3). Ferner wird abschließend in Abschnitt 2.1.4 das *Domain-Specific Language Engineering (DSL Engineering)* als methodischer Rahmen zur Entwicklung von *DSLs* beschrieben.

2.1.1 Metamodellierung

Bei der *Metamodellierung* handelt es sich um eine Schlüsselaktivität des *MDEs*. Dabei werden verschiedene Ziele verfolgt, wie z.B. die Spezifikation von Sprachen zur Modellierung und Programmierung, den Austausch und Speichern von Informationen und die Anreicherung bestehender Informationen mit neuen Eigenschaften oder Features [BCW17].

Im Fokus der *Metamodellierung* steht die Definition von sogenannten *Metamodellen* (siehe Abbildung 2-1). Ein *Metamodell* ist eine Menge von *Notationsregeln* zur Beschreibung einer Klasse von Modellen. Dabei bilden *Metamodelle* hierdurch unter anderem die Grundlage für die Definition von Modellierungssprachen. Dies umfasst insbesondere die Spezifikation der abstrakten Syntax (siehe Abschnitt 2.1.4). Weiterführende Ansätze haben sich dabei auch in Teilen mit der Spezifikation der Semantik einer Sprache befasst [Eng+00; Hau05; Sol13].

Bei einem *Modell*, welches auf Basis eines *Metamodells* erstellt worden ist, spricht man auch von einer Instanz des Metamodells. Es handelt sich hierbei um einen Satz der Sprache, die durch das *Metamodell* beschrieben wird.

Die Sprache, die durch ein *Metamodell* beschrieben ist, kann entweder durch direkte Ableitung von *Notationsregeln* auf Basis eines *(realweltlichen) Objekts* ermittelt werden oder durch Ableitung auf Basis einer Menge von (Beispiel-)Modellen [Voe+13; BCW17]. Bei *(realweltlichen) Objekten* handelt es sich um gedachte oder reale Objekte, wie z.B. die Arbeitnehmenden in einer Organisation oder ein Stück Programmquellcode, das modellbasiert beschrieben werden soll.

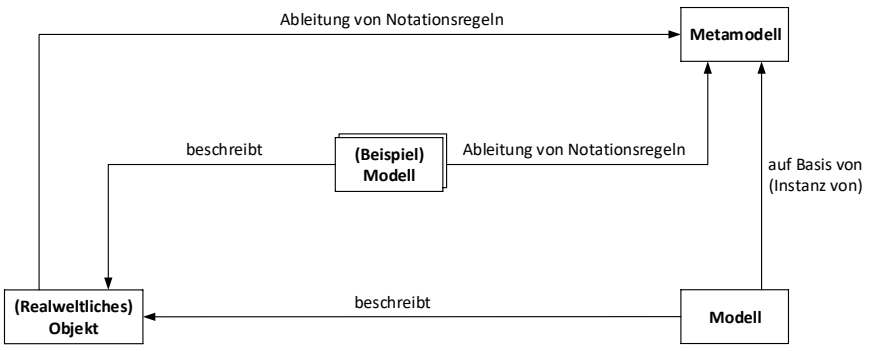


Abbildung 2-1: Beziehungen zwischen (realweltlichen) Objekten, Modellen und Metamodellen

2.1.2 Meta-Object-Facility (MOF)

Bei der *Meta-Object-Facility (MOF)* [OMG15a] handelt es sich um einen durch die *Object Management Group (OMG)* spezifizierten Rahmen für die

Metamodellierung. Er wird zur einheitlichen Spezifikation von Metadaten bzw. Metamodellen eingesetzt und verwendet hierzu vereinfachte Notationsregeln, die durch Konzepte wie z.B. *Klassen*, *Pakete* oder *Assoziationen* der *UML Klassendiagramme* ausgedrückt werden können.

Prinzipiell werden in der Modellierung unendlich viele Abstraktionsebenen durch *MOF* unterstützt, wobei jedoch mindestens zwei Ebenen vorgesehen sind. Klassischerweise wird *MOF* aber in Form einer Vier-Ebenen-Architektur verwendet, die im Folgenden näher beschrieben wird. Hierzu sind diese Architekturebenen in Form von *M0*, *M1*, *M2* und *M3* im linken Bereich von Abbildung 2-2 dargestellt. Im rechten Bereich der Abbildung befinden sich für die genannten Architekturebenen zugehörige Beispiele mit jeweiligen Abhängigkeiten. Nachfolgend wird eine detailliertere Erläuterung dieser Abstraktionsebenen gegeben.

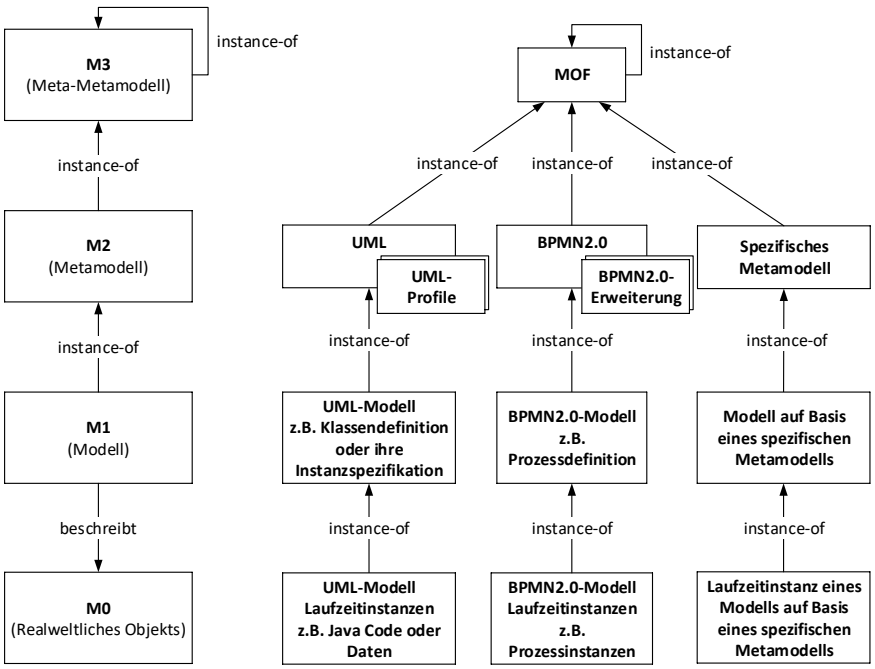


Abbildung 2-2: Übersicht über die Vier-Ebenen-Architektur

Ebene M3

Auf der *Ebene M3* befinden sich *Meta-Metamodelle* (hier: *MOF*). *MOF* bildet dabei die Basis für Definitionen von Metamodellen auf der *Ebene M2*. *MOF* ist rekursiv durch sich selbst definiert. Hierzu wird eine Untermenge von Spracheigenschaften der *UML Klassendiagramme* verwendet. Wie in [BCW17] beschrieben, ist eine weitere höhere Abstraktionsebene in dieser Architektur nicht notwendig, da weitere Ebenen auch weiterhin das Sprachkonstrukt der Klasse enthalten würden. Eine Implementierung einer Untermenge von Spracheigenschaften des Rahmenwerks zur Metamo-

dellierung *MOF* ist gegeben durch das *Eclipse Modeling Framework (EMF)* [Ecl], dessen zugehöriges Metamodell auch *ECore* genannt wird [Eco].

Ebene M2 Auf der *Ebene M2* befinden sich Metamodelle oder auch Sprachspezifikationen, wie die *Unified Modeling Language (UML)* [OMG10], die *Business Model and Notation (BPMN2.0)* [OMG11], die *Adapt Case Modeling Language (ACML)* [Luc13] oder auch verschiedene *Domain-Specific Languages (DSLs)*. Spracherweiterung von *UML*-basierten Sprachen (hier: *UML-Profile*) und *BPMN2.0*-basierten Sprachen (hier: *BPMN2.0*-Erweiterungen) befinden sich ebenfalls auf der *Ebene M2*. Die auf der *Ebene M2* liegenden Metamodelle werden dabei genutzt, um Sprachelemente für die Modellierung von Modellen auf der *Ebene M1* zur Verfügung zu stellen.

Ebene M1 Die *Ebene M1* enthält Modelle, die auf Basis der auf der *Ebene M2* befindlichen Metamodelle erstellt worden sind. Ein Beispiel ist die Gestaltung einer Klasse mit enthaltenen Attributen, die durch ein *UML Klassendiagramm* erstellt worden ist. Ein weiteres Beispiel ist durch eine Prozessdefinition gegeben, die durch ein *Business Process Diagram (BPD)* der Sprache *BPMN2.0* beschrieben ist.

Ebene M0 Die *Ebene M0* enthält schließlich realweltliche Objekte. Diese können, wie in Abbildung 2-3 dargestellt, gegeben sein durch Quellcodes in Form einer Java-Klasse aber auch in Form von Laufzeitinstanzen.

Ein weiterer relevanter De-facto-Standard der *OMG* für die Metamodellierung ist die *Object Constraint Language (OCL)* [RG02; OMG14a] mit der die statische Semantik von *UML-Modellen* durch Ausdrücke, Invarianten, Vor- und Nachbedingungen formal beschrieben werden kann.

2.1.3 Model-Driven Architecture (MDA)

Die *Model-Driven Architecture (MDA)* [OMG14b] ist ein weiterer durch die *OMG* spezifizierter Standard zur Unterstützung des *MDEs*. Kern der *MDA* ist dabei ein modellgetriebener Softwareentwicklungsprozess mit dem Ziel, ausführbaren Code – also die Implementierung in Form einer Anwendung – zu erhalten. Eine Übersicht ist in Abbildung 2-3 gegeben und wird nachfolgend kurz erläutert.

Die *MDA* lässt sich, wie hier gezeigt wird, in die zwei Phasen des *Requirements Engineering* und des *System Designs* unterteilen. Dabei werden Modelle zur Beschreibung der Anwendungen auf unterschiedlichen Abstraktionsebenen beschrieben. Diese Abstraktionsebenen sind das *CIM*, *PIM* und *PSM* und werden im Folgenden detaillierter beschrieben.

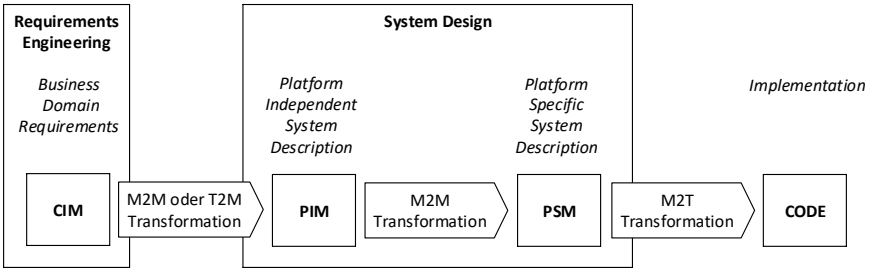


Abbildung 2-3: Übersicht über die Model-Driven Architecture (nach [BCW17])

Computation Independent Model (CIM)

Bei dem *Computation Independent Model (CIM)* handelt es sich um die abstrakteste Ebene der Betrachtung einer zu entwickelnden Anwendung [BCW17]. Im Rahmen des *CIM* werden das Geschäfts- und das Domänenmodell beschrieben, welche jeweils die Kernkonzepte oder auch das Vokabular für die Entwicklung der Anwendung definieren. Dies kann z.B. die Beschreibung von Anforderungen, den Anwendungszweck oder die Kontextinformationen der Anwendung betreffen. Dabei stehen hierbei insbesondere solche Beschreibungen im Vordergrund, die unabhängig von einer möglichen technischen Realisierung sind.

Platform Independent Model (PIM)

Auf Basis des *CIM* kann anschließend das *Platform Independent Model (PIM)* erstellt werden. Es enthält sowohl das Verhalten als auch die Struktur der Anwendung, welche jeweils unabhängig von einer konkreten technischen Zielplattform beschrieben werden. Für die Modellierung des *PIM* werden häufig Sprachen zur Gestaltung eingesetzt. Beispiele für derartige Sprachen sind *UML* [OMG10], *BPMN2.0* [OMG11] oder domänenspezifische Sprachen (engl. *DSL*).

Platform Specific Model (PSM)

Im Rahmen des *Platform Specific Models (PSM)* wird das *PIM* verfeinert. Man spricht von *Verfeinerung*, wenn zu einem späteren Zeitpunkt weitergehende Informationen zu einem gestalteten Modell hinzugefügt werden. Solche Informationen können z.B. in Bezug zu einer technischen Zielplattform stehen. Auf Basis des *PSM* kann schließlich der Programmcode für eine konkrete Zielplattform, wie z.B. *Java*[1] oder *.net*[2], generiert werden.

Die Übergänge von einer Ebene zu einer nachfolgenden Ebene, wie z.B. vom *CIM* zum *PIM*, können durch verschiedene Transformationen mindestens teil- aber vorzugsweise vollautomatisiert realisiert werden. Dabei können auf Basis des Modells einer Abstraktionsebene verschiedene Modelle auf einer nachfolgenden Ebene existent sein. So kann es sinnvoll sein, dass auf Basis eines *PIM* mehrere *PSM* für unterschiedliche technische Zielplattformen erstellt werden. Für die in diesem Kontext eingesetzten

[1] `https://www.oracle.com/de/java/index.html` Letzter Zugriff: 11.12.2018
[2] `https://www.microsoft.com/net` Letzter Zugriff: 11.12.2018

Transformationen, wie z.B. *Model-2-Model (M2M)* oder *Model-2-Text (M2T)*, existieren dabei eine Reihe von verschiedenen Sprachen und Rahmenwerken. Einige Beispiele sind durch *ATL*[3], *Xtext*[4] oder *XPand*[5] gegeben.

2.1.4 Domain-Specific Language Engineering

Das *Domain-Specific Language Engineering (DSL Engineering)* ist eine fachliche Disziplin, die sich mit dem methodischen Vorgehen bei der Entwicklung von domänenspezifischen Sprachen (engl. *DSL*) beschäftigt. Der Inhalt dieses Abschnitts beschäftigt sich zunächst mit einigen Grundkonzepten des *DSL Engineering* und beschreibt anschließend einzelne zugehörige Entwicklungsschritte.

Eine Sprache ist definiert durch ihre *Abstrakte Syntax (AS)* und ihre *Semantik (S)*. Ist zudem eine textuelle oder sprachliche Repräsentation erforderlich, so wird zudem auch noch die *Konkrete Syntax (KS)* definiert. Im Folgenden wird eine detailliertere Beschreibung dieser Begriffe vorgenommen.

Abstrakte Syntax (AS)

Die *Abstrakte Syntax (AS)* beschreibt die Struktur der Sprache. Dies umfasst eine Beschreibung, welche Elemente Teil der Sprache sind und wie sie miteinander kombiniert werden dürfen. Dabei wird jedoch die Art der textuellen oder graphischen Repräsentation ausgeblendet. Die *AS* kann durch Metamodelle (siehe Abschnitten 2.1.1 und 2.1.2) beschrieben werden.

Konkrete Syntax (KS)

Bei der *Konkreten Syntax (KS)* handelt es sich um eine Spezifikation der textuellen oder graphischen Repräsentation einzelner Elemente der Sprache. Dabei wird den einzelnen Elementen oder der Kombination aus mehreren Elementen aus der *AS* vorzugsweise ein eindeutiges textuelles oder graphisches Symbol zugeordnet. So ist z.B. in *UML Klassendiagrammen* dem Element des Typs *Class* ein rechteckiges graphisches Symbol zugeordnet, in dessen Mitte der Wert des Attributs *Name* in textueller Form dargestellt wird.

Semantik (S)

Der Begriff der *Semantik (S)* einer Sprache wird unterschieden. So existiert sowohl der Begriff der *statischen Semantik (StatS)* als auch der Begriff der *Ausführungssemantik (AusS)*. Die *StatS* einer Sprache beschreibt zusätzliche strukturelle Einschränkungen von Elementen und ihrer Kombination sowie der textuellen oder graphischen Repräsentation. Wohingegen die

[3] `http://www.eclipse.org/atl` Letzter Zugriff: 11.12.2018
[4] `http://www.eclipse.org/Xtext` Letzter Zugriff: 11.12.2018
[5] `http://wiki.eclipse.org/XPand` Letzter Zugriff: 11.12.2018

AusS kontextsensitive Abhängigkeiten der Elemente und ihrer Kombination beschreibt. Die *AusS* ist dabei oftmals implizit in den Regeln zur Transformation, wie z.B. *M2T-Transformation*, oder durch die Implementierung der Ausführungsumgebung definiert. Alternativ existieren aber auch Ansätze [Eng+00; Hau05; Fow10; Sol13], mit denen die *AusS* formal definiert werden kann.

Pragmatik (P)

Der Begriff *Pragmatik (P)* – oder alternativ auch *Verwendungsweise* – beschreibt einen Prozess bestehend aus Aktivitäten, die die Nutzung einer Sprache beschreiben. So kann es in Sprachen spezielle Abhängigkeiten von zu erstellenden Artefakten geben. Durch die *P* können solche Abhängigkeiten beschrieben und eine empfohlene Reihenfolge einzelner Aktivitäten definiert werden. Wird für eine Sprache keine *P* angegeben, so wird üblicherweise angenommen, dass die Vorgehensweise durch einen übergeordneten Ansatz übernommen wird. Dies ist insbesondere bei DSLs der Fall, die auf Basis eines bestehenden Ansatzes zur Gestaltung erstellt worden sind.

2.1.4.1 DSL-Typen und Charakterisierung

Im Kontext der modellgetriebenen Softwareentwicklung existieren eine Reihe von unterschiedlichen Perspektiven auf verschiedene Typen von Sprachen. Typischerweise wird bei Sprachen zur Gestaltung in die beiden Typen *Domain-Specific Language (DSL)* und *General-Purpose Language (GPL)* unterschieden. Auf eine Beschreibung und die Unterschiede der beiden Typen wird nachfolgend detaillierter eingegangen.

Domain-Specific Language (DSL)

Bei einer *Domain-Specific Language (DSL)* handelt es sich um eine Sprache, mit der die Konzepte aus einer spezifischen Domäne beschrieben werden können. Man spricht in diesem Kontext auch von einer Anwendungsdomäne. Eine *DSL* kennzeichnet sich daher häufig, jedoch nicht immer, durch wenige aber dafür für die Domäne spezifische Sprachelemente aus. Hiermit können z.B. die Anwendung selbst, ihr Kontext in dem sie eingebettet ist oder weitere Metadaten gemeint sein. Durch diesen spezifischen Bezug ist in einer *DSL* typischerweise relevantes Wissen von der betroffenen Anwendungsdomäne kodiert, welches im Rahmen der Gestaltung wiederverwendet werden kann.

General-Purpose Languages (GPL)

Im Gegensatz zu einer *DSL* existieren aber auch Sprachen, die als *General-Purpose Languages (GPL)* bezeichnet werden. Eine *GPL* kennzeichnet sich durch die Fähigkeit aus, dass sie nicht lediglich für eine spezifische Domäne anwendbar ist. Stattdessen spricht man hier auch davon, dass sie

für einen bestimmten Bereich oder eine Gruppe von Domänen eingesetzt werden können. Ein typisches Beispiel für eine *GPL* ist durch die Sprache *UML* [OMG10] gegeben. Sie wird dabei für den Bereich der (*objektorientierten*) Gestaltung von Softwaresystemen eingesetzt.

Die Trennung von Sprachen in die beiden zuvor aufgeführten Typen *GPL* und *DSL* ist dabei oftmals nicht klar durchführbar. So könnte man z.B. behaupten, dass es sich bei der Entwicklung von Softwaresystemen ebenfalls um eine spezifische Domäne handelt. Daraus könnte der logische Schluss folgen, dass es sich demnach auch bei der *UML* um eine *DSL* handelt. Je nach individueller Perspektive auf die Welt der Sprachen zur Gestaltung ist dieser Aspekt gerechtfertigt. Dennoch würde man aus der Perspektive der Softwareentwicklung die Sprache *UML* eher als eine *GPL* einstufen. Zur besseren Charakterisierung von Sprachen zur Gestaltung kann der durch [Voe+13] beschriebene Vergleich basierend auf einzelnen Charakteristiken von *GPLs* und *DSLs* verwendet werden. Es handelt sich hierbei um einen Versuch, eine methodische Trennung für die beiden Typen *GPL* und *DSL* anhand vorgegebener Dimensionen und deren typischen Belegungen durchführen zu können. Ein Auszug dieser Charakteristiken ist in Tabelle 2-1 dargestellt.

Tabelle 2-1: *Gegenüberstellung von GPLs und DSLs (nach [Voe+13])*

Dimension	**GPLs**	**DSLs**
Domäne	groß und komplex	kleiner und spezifischer definiert
Benutzerdefinierte Abstraktionen	anspruchsvoll	limitiert
Lebensspanne	Jahre oder Jahrzehnte	Monate oder Jahre
Gestaltet durch	Experten oder Komitee	Domänenexperten
Benutzergemeinschaft	groß, anonym und weit verbreitet	klein, zugreifbar und lokal
Evolution	langsam, oft standardisiert	schnelllebig

2.1.4.2 Die Rolle von Domänenmodellen

Wie bereits im vorherigen Abschnitt beschrieben, stützt sich das *DSL Engineering* auf die Existenz des Konzepts des *Domänenmodells*. Typischerweise beschreibt ein *Domänenmodell* relevante Konzepte und deren Beziehungen untereinander, die für das Abbild einer Domäne notwendig sind. Das in einer Domäne zu verwendende Vokabular kann somit durch ein *Domänenmodell* bestimmt werden. Dieses Vokabular einer spezifischen Domäne kann z.B. durch die Konzepte der *Klassen*, *Ereignisse*, *Transitionen* oder *Verhaltensmuster* als typisches Verhalten beschrieben werden.

Ein methodischer Ansatz zur Entwicklung von *Domänenmodellen* ist dabei durch das *Domain-Driven Design (DDD)* [Eva03] gegeben. Das wesentliche Grundprinzip des *DDD* ist eine starke Fokussierung auf die Inhalte einer einzelnen Domäne, wohingegen der Schwerpunkt ansonsten eher auf die Realisierung bezogen ist. Die Beziehung zwischen dem *MDE* und dem *DDD* ist zur besseren Übersicht in Abbildung 2-4 in Anlehnung an *Brambilla et al.* [BCW17] dargestellt.

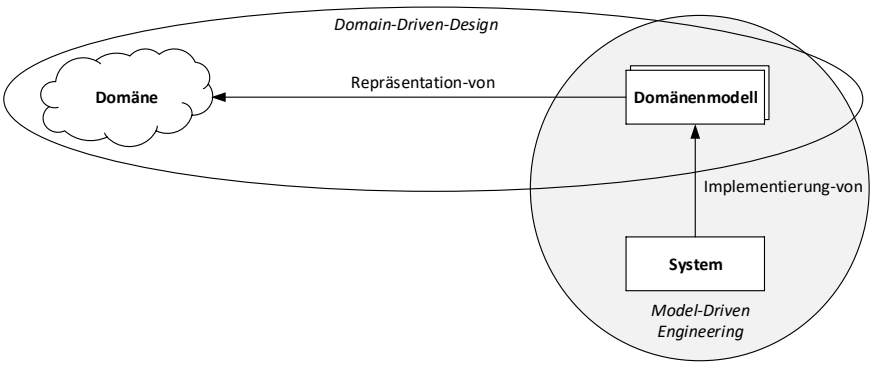


Abbildung 2-4: Zusammenhang zwischen Domain-Driven Design und Model-Driven Engineering (nach [BCW17])

Das *DDD* und *MDE* sind demnach keine konträren Ansätze. So verwenden beide das Konzept des Domänenmodells zur Repräsentation des Wissens einer Domäne. Stattdessen kann das *DDD* vielmehr als eine sinnvolle Ergänzung zum *MDE* verstanden werden ([BCW17]). Es verwendet hierzu spezifische Methoden und Techniken, die unterstützend für die Gestaltung von Domänenmodellen eingesetzt werden können.

2.1.4.3 Spracherweiterungsansätze

Für die Entwicklung einer *DSL* bietet sich neben der Erstellung eines eigenen Metamodells insbesondere auch die Wiederverwendung existierender Sprachen an. Wichtige Beispiele für derartige Sprachen sind durch die *MOF* auf der Meta-Metaebene (*M3*) und durch die *UML* auf der Metaebene (M2) gegeben (siehe Abbildung 2-3). Werden existierende Sprachen erweitert, so spricht man in diesem Bezug auch von einer *Lightweight Extension* oder einer *Heavyweight Extension*. Auf Details der beiden genannten Typen von Erweiterungen von existierenden Sprachen wird nachfolgend eingegangen.

Lightweight Extension

Bei einer *Lightweight Extension (LExt)* handelt es sich um eine Erweiterung einer existierenden Sprache. Dabei verfeinern die Sprachelemente einer *LExt* für sie relevante Sprachelemente der existierenden Sprache, sodass ein domänenspezifisches Konzept oder ein Concern (hier: *Aspekt*) unterstützt wird. Grundsätzliche Eigenschaft einer *LExt* ist dabei, dass keine Elemente der existierenden Sprache hinsichtlich ihrer *AS*, *KS*, oder *S* verändert werden.

Heavyweight Extension

Eine *Heavyweight Extension (HExt)* ist ebenfalls eine Erweiterung einer existierenden Sprache. Im Gegensatz zur *LExt* können aber Elemente der Sprachspezifikation der zu erweiternden Sprache verändert werden. Dies kann sowohl die *AS*, die *KS* als auch die *S* der zu erweiternden Sprache betreffen. Hierdurch können beim Einsatz der erweiternden Sprache unter Umständen auch ungewollte Seiteneffekte entstehen. Ein Beispiel hierfür ist dadurch gegeben, dass existierende Transformationen möglicherweise nicht mehr ordnungsgemäß funktionieren und angepasst werden müssen.

Die Erstellung von *LExt* oder *HExt* kann in verschiedenen Kontexten unterschiedlich realisiert werden. So ist z.B. im Kontext der Sprache *UML* [OMG10] die Erstellung von *UML-Profilen* gängig, wohingegen im Kontext der Sprache *BPMN2.0* [OMG11] die Erstellung von Erweiterungen über einen eigenen Mechanismus durchgeführt wird.

UML-Profile können z.B. durch *UML Profil-Diagramme* [OMG10] definiert werden. Sie spezialisieren Elemente der *UML* durch domänenspezifische Konzepte und Symbole mittels der Mechanismen *Stereotyp*, *Tagged Value* oder *Constraint*. Die einzelnen Mechanismen werden im Folgenden näher erläutert.

Stereotyp

Ein *Stereotyp* definiert ein domänenspezifisches Konzept auf Basis eines existierenden Elements der Sprache *UML*. Beispiele für derartige Elemente sind durch die Elemente *Class*, *State* oder *Component* gegeben. Die existierenden Elemente der Sprache *UML* können in diesem Rahmen auch als *Metaklasse* bezeichnet werden. Die Beziehung zwischen einem *Stereotyp* und seiner *Metaklasse* definiert dabei, dass sich der *Stereotyp* in Anlehnung an die *Metaklasse* verwenden lässt bzw. sich so verhält. Der *Stereotyp* spezialisiert folglich die *Metaklasse* durch Eigenschaften eines domänenspezifischen Konzepts. Ferner können durch Bedingungen – welche z.B. durch die Sprache *OCL* [OMG14b] spezifiziert werden – weitere Einschränkungen hinsichtlich der erlaubten Verwendung neuer Elemente oder der Semantik spezifiziert werden. Für einen *Stereotypen* können im Rahmen der Definition der *KS* auch neue Symbole zugewiesen werden.

UML-Profile beschreiben domänenspezifische Erweiterungen auf Basis von bestehenden Elementen der Sprache *UML*. *Tagged Values* stellen in diesem Zusammenhang eine Möglichkeit dar, die Attribute eines Modellelements durch konkrete Werte vorzugeben. *Tagged Value*

Mittels *Constraints* kann die erlaubte Verwendung oder die Semantik von *Stereotypen* und *Metaklassen* in einem *UML Profildiagramm* weiter eingeschränkt werden. *Constraint*

Im Gegensatz zum Ansatz der Spezialisierung durch *UML-Profile* wird im Kontext der Sprache *BPMN2.0* ein eigener Erweiterungsansatz angeboten. Hier kann über sogenannte *Extension Points* [OMG11] eine Erweiterung der bestehenden Spezifikation der Sprache *BPMN2.0* vorgenommen werden. In einer *BPMN*-Erweiterung, die durch diesen Erweiterungsansatz spezifiziert worden ist, können neue Attribute und Elemente in Form des Elements *ExtensionDefinition* beschrieben werden. Ein solches Element kann an ein beliebiges Element vom Typ *BaseElement* gebunden werden und erweitert so Konzepte der Sprache *BPMN2.0* [OMG11]. Dabei ist durch die Sprachspezifikation der Sprache *BPMN2.0* [OMG11] keine einheitliche Vorgehensweise zur methodischen Erstellung einer *BPMN*-Erweiterung gegeben. Ebenso fehlt es an einer Erweiterungsmöglichkeit der graphischen Notation zur Repräsentation der Struktur einer *BPMN*-Erweiterung. Eine Lösung bieten [Str+11; SCV11] durch den *BPMN+X*[6] Ansatz an, in dem die Definition eines *UML-Profils* zur Spezifikation von *BPMN*-Erweiterungen eingesetzt wird. Auf eine allgemeine Vorgehensweise zur methodischen Erstellung einer *DSL* wird im nachfolgenden Abschnitt eingegangen.

2.1.4.4 DSL-Entwicklungsprozess

Die adäquate Entwicklung von domänenspezifischen Sprachen stellt einen wichtigen Beitrag für die spätere Qualität der Sprache und der auf ihr gestalteten Artefakte dar. Daher sind hier geeignete Entwicklungsprozesse anzuwenden, die dies ermöglichen. In der Literatur sind verschiedene Entwicklungsprozesse existent, mit denen z.B. *UML-Profile* [Sel07; Lag+07; Wim09] oder *BPMN*-Erweiterungen [Str+11; SCV11] erstellt werden können. Die zuvor genannten Arbeiten setzen dabei jeweils unterschiedliche Schwerpunkte, die sich jedoch allgemein zunächst auf die Ermittlung der domänenspezifischen Konzepte und die anschließende Definition der

[6] Werkzeugunterstützung für BPMN+X: `http://code.google.com/p/bpmnx//` Letzter Zugriff: 12.10.2018

Sprache beziehen. An dieser Stelle wird der *DSL-Entwicklungsprozess* nach *Brambilla et al.* [BCW17] exemplarisch in Form von wesentlichen Schritten beschrieben.

Analyse der Domäne

Zu Anfang steht die Analyse einer Domäne im Vordergrund, bei der relevante Konzepte identifiziert werden. Die Durchführung der Analyse stellt dabei besondere Herausforderungen an den Entwicklungsprozess, da irrelevante von relevanten Konzepten getrennt und für die jeweilige Anwendung abstrahiert dokumentiert werden müssen. Damit die Analyse einer Domäne unterstützt werden kann, wurde ein Katalog von insgesamt 26 Richtlinien für die Entwicklung einer *DSL* von [Kar+14] vorgeschlagen. Ein Auszug der wichtigsten Kategorien und zugehörigen Fragestellungen ist nachfolgend aufgeführt.

Zweck	Wofür soll die Sprache eingesetzt werden?
Realisierung	Wie soll die Sprache realisiert werden?
Inhalt	Was soll Teil der Sprache sein?
Abstrakte Syntax	Was ist die (interne) Repräsentation der Sprache?
Konkrete Syntax	Was ist die (externe) Repräsentation der Sprache?

Als Grundlage für die Analyse können ausgesuchte Beispiele für konkrete Sätze der zu entwickelnden Sprache verwendet werden. Solche Beispiele können in Form von z.B. Quellcodes oder zuvor bereits erstellten Modellen verfügbar sein [Sta+06]. Ergebnis der Analyse ist die Dokumentation von Wissen über die Domäne mittels eines initialen Domänenmodells (siehe Abschnitt 2.1.4.2). Dabei löst dieses Domänenmodell die zuvor eingeführten Fragestellungen auf.

Entwurf der Sprache

Der zweite Schritt sieht den Entwurf der zu entwickelnden Sprache vor. Dabei wird das im ersten Schritt der Analyse der Domäne dokumentierte Wissen über die Domäne in die spätere *AS*, *KS* und *S* der zu entwerfenden Sprache überführt. Hiermit ist gemeint, dass z.B. die Zuordnung von Konzepten und Beziehungen der Domäne zu Klassen, Attributen und deren Typen sowie Assoziationen durchgeführt wird. Ferner werden Multiplizität und weitere Einschränkungen zur Verwendung der Sprache definiert. Dies kann z.B. durch die Verwendung der Sprache *Object-Constraint Language (OCL)* [OMG14b] durchgeführt werden. Neben der zuvor beschriebenen Definition der *internen Repräsentation* (*AS*) der Sprache kann auch die externe Repräsentation (*KS*) definiert werden.

Validierung der Sprache

Als letzten Schritt sieht der Entwicklungsprozess die Validierung der entwickelten Sprache vor. Dies kann hinsichtlich verschiedener Eigenschaften, wie z.B. der *Korrektheit*, der *Vollständigkeit*, der *Einfachheit* oder der *Konsistenz* sinnvoll sein. Eine solche Validierung kann durch Anwendung der Sprache, also durch das Erstellen von Modellen auf ihrer Basis, durchgeführt werden. Hier kann geprüft werden, ob sich z.B. ein Attribut einer Klasse mit einem konkreten Wert erstellen lässt oder ob sich spezifische Konzepte in geforderter Relation zueinander stellen lassen. Besonders bietet sich aber an, die im ersten Schritt verwendeten Beispiele mit der entwickelten Sprache zu modellieren, da es sich hierbei um eine Art Referenzmodell handelt.

2.2 Business Process Management

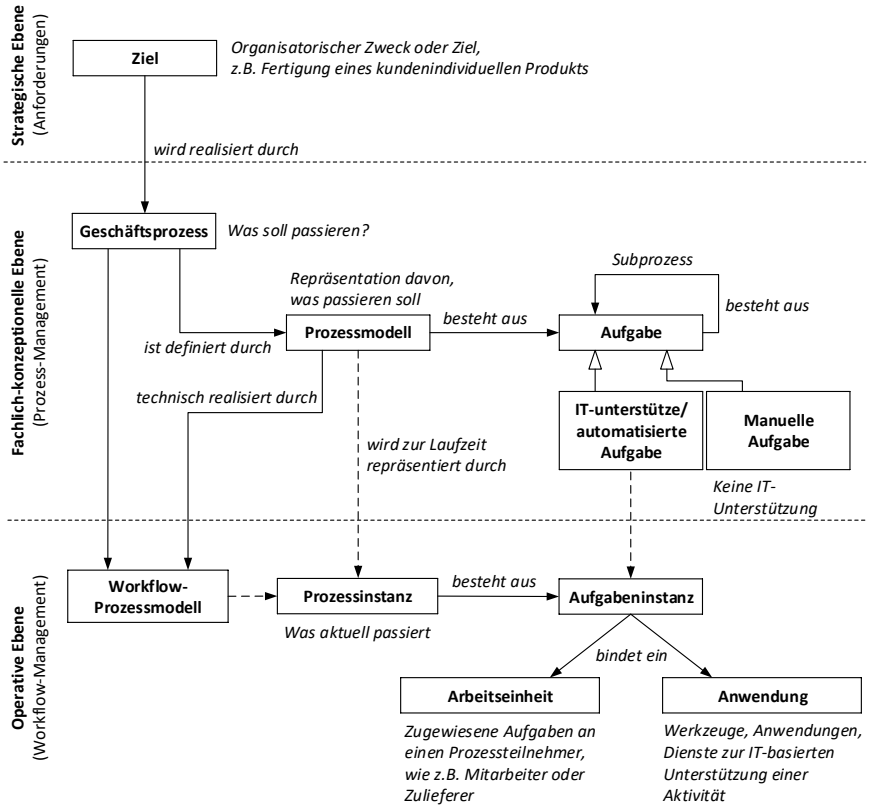
Eine der in dieser Arbeit betrachteten Domänen stellt das *Business Process Management (BPM)* dar. Nachfolgend wird zunächst in Abschnitt 2.2.1 eine Einführung in das *BPM* durch eine Vorstellung grundlegender Begriffe und Konzepte gegeben. Aufbauend werden weitere für diese Arbeit relevante Aspekte von Prozessen im Kontext des *BPM* näher beschrieben. So wird in Abschnitt 2.2.2 zunächst das Modell des *BPM-Lebenszyklus* eingeführt. Darauf folgt in Abschnitt 2.2.3 eine Vorstellung des Begriffs der Flexibilität und Anpassbarkeit in Bezug zu Prozessen.

2.2.1 Einführung in das Business Process Management

Unter dem Begriff *Business Process Management (BPM)* wird allgemein ein systematischer Ansatz verstanden, der sich mit verschiedenen Aspekten von Geschäftsprozessen (hier: *Prozesse*) beschäftigt. Zum besseren und detaillierten Verständnis des *BPM* werden in diesem Abschnitt zunächst relevante Begriffe eingeführt, die am Ende des Abschnitts für eine spezifischere Definition des *BPM* verwendet werden.

Ein Modell der Domäne des *BPM* ist in Abbildung 2-5 zur Darstellung wichtiger Begriffe und ihrer Relationen gezeigt. Es orientiert sich dabei an das durch [Gad08] vorgestellte integrierte Konzept für das Prozess- und Workflow-Management. Dabei können die dargestellten Begriffe hinsichtlich einer *strategischen*, *fachlich-konzeptionellen* und *operativen Ebene* aufgeteilt werden. Auf eine detaillierte Erläuterung dieser Ebenen und der gezeigten Begriffe wird nachfolgend eingegangen.

Abbildung 2-5:
Modell der Domäne BPM



Strategische Ebene

Die *strategische Ebene* bezieht sich auf die für ein Unternehmen relevanten Geschäftsfelder und auf deren kritische Erfolgsfaktoren (engl. *Critical Success Factors (CSF)*). Hierbei werden auf Basis der Unternehmensstrategie – hier dargestellt als *Ziel* – die zentralen Prozesse identifiziert und geplant. Ein Beispiel für ein solches *Ziel* ist die Fertigung eines kundenindividuellen Produkts, für das eine Vielzahl von Prozessen mit Schnittstellen zu unterschiedlichen internen und externen Prozessbeteiligten entlang der Wertschöpfungskette vorhanden sein können. Hiervon können z.B. die Beschaffungs- und die Planungsprozesse, aber auch Arbeits- und Produktionsprozesse betroffen sein. Das *Ziel* bildet dabei also die Basis für die Erstellung von Prozessen, die wiederum dafür vorgesehen sind, das jeweilige *Ziel* zur erreichen.

Fachlich-konzeptionelle Ebene

Auf der *fachlich-konzeptionellen Ebene* werden Prozesse (hier: *Geschäftsprozesse*) durch *Prozessmodelle* definiert. Für den Begriff *Prozessmodell* finden sich in der Literatur synonyme Verwendungen durch den Begriff des *Prozess-Schemas* und der *Prozessdefinition*.

Ein *Prozessmodell* beschreibt in Anlehnung an die an ihn gestellten Anforderungen unter anderem die Struktur eines *realweltlichen Prozesses* (siehe Abschnitt 2.1.1 und Abschnitt 2.1.2). Hiervon betroffen sind z.B. der *Kontroll-* und der *Datenfluss*, die jeweils für den *Prozess* zu beschreiben sind (siehe Abschnitt 2.3.2). So sind z.B. mögliche Pfade, Regeln für die Auswahl eines Pfades und auch die benötigten *Aufgaben* oder Daten inbegriffen. Eine *Aufgabe* in einem *Prozessmodell* beschreibt eine logische Handlung, die weiter detailliert werden kann. Die Detaillierung einer *Aufgabe* wird auch *hierarchischer Prozess* oder *Subprozess* genannt. Lassen sich Handlungen nicht weiter detaillieren, spricht man von *atomaren Aufgaben*, die in Abbildung 2-5 durch *manuelle Aufgaben* oder *IT-unterstützte* bzw. *automatisierte Aufgaben* dargestellt sind.

Da es sich bei *Prozessmodellen* lediglich um eine Repräsentation von dessen, was passieren soll, handelt, kann es notwendig sein, im Rahmen von Verbesserungs- und Optimierungsmaßnahmen das *Prozessmodell* an den *realweltlichen Prozess* neu zu orientieren [AHW03; Ger13] (siehe auch Abschnitt 2.2.2).

Operative Ebene

Die *operative Ebene* beschäftigt sich speziell mit der IT-gestützten Ausführung von Prozessen. So kann ein *Prozessmodell* zuvor für unterschiedliche Zwecke erstellt worden sein. Hier wird zwischen der Dokumentation eines Prozesses und seiner IT-gestützten Ausführung unterschieden [FR14].

Sollen *Prozesse* IT-gestützt ausgeführt werden, so können deren *Prozessmodelle* bzw. *Workflow-Prozessmodelle* im Ganzen oder in bestimmten Teilen in *Prozessinstanzen* überführt werden. Man spricht hier auch davon, dass eine *Prozessinstanz* ein solches Modell zur Laufzeit repräsentiert und weitere für die Ausführung notwendige Informationen enthält. Ein einfaches Beispiel einer derartigen Eigenschaft ist durch den Laufzeitzeiger gegeben, der die zu einem bestimmten Zeitpunkt aktive Aufgabe identifiziert.

IT-unterstützte Teile einer *Prozessinstanz* sind in Form von *Aufgabeninstanzen* gegeben. Eine solche *Aufgabeninstanz* bildet entweder eine *Arbeitseinheit*, die z.B. durch einen *Prozessteilnehmer* ausgeführt werden kann. Alternativ kann aber auch eine weitere *(Software-)Anwendung* eingebunden werden, die die benötigte Funktion für die Aufgabe zur Verfügung stellt. Eine solche *Anwendung* kann durch einen *IT-Service* oder durch *Werkzeuge* gegeben sein, die für die technische Realisierung notwendig sind. Ferner kann eine *Aufgabeninstanz* aber auch *Prozessteilnehmer* – wie z.B. eine Mitarbeiterin bzw. einen Mitarbeiter, eine Fachabteilung oder einen Zulieferer – einbinden, die jeweils in verschiedenen Rollen agieren können und ebenfalls *Aufgabeninstanzen* ausführen.

Sowohl das *Workflow-Prozessmodell* als auch seine zugehörige *Prozessinstanz* werden in der Literatur für die *operative Ebene* oftmals mit dem Begriff des *Workflow-Prozesses* vereinheitlicht. Nachfolgend ist in Definition 2.2.1 eine häufig verwendete Beschreibung eines *Workflow-Prozesses* in Anlehnung an [Coa96] gegeben.

Definition 2.2.1. *(Workflow-Prozess)*

Ein Workflow-Prozess ist die vollständige oder teilweise Automatisierung eines Prozesses, durch den Dokumente, Informationen oder Aufgaben (engl. Tasks) von einem Prozessbeteiligten zu einem anderen geleitet werden mit dem Zweck der Handlung und in Abhängigkeit zu prozeduralen Regeln.

Neben dem *Workflow-Prozess* wurde ebenso der Begriff des *Geschäftsprozesses* für die *fachlich-konzeptionelle Ebene* eingeführt. Eine Übersicht über unterschiedliche Aspekte der beiden zuvor eingeführten Begriffe des *Geschäfts-* und *Workflow-Prozesses* wird in Tabelle 2-2 gegeben. Hier werden diese Begriffe hinsichtlich ihrer wichtigsten Merkmale gegenübergestellt.

Tabelle 2-2: Gegenüberstellung von Geschäftsprozess und Workflow-Prozess (nach [Gad08])

Kriterium	**Geschäftsprozess**	**Workflow-Prozess**
Zielsetzung	Analyse und Gestaltung von Arbeitsabläufen im Sinne gegebener (strategischer) Ziele	Spezifikation der technischen Ausführung von Arbeitsabläufen
Gestaltungsebene	Fachlich-konzeptionelle Ebene mit Bezug zur Geschäftsstrategie	Operative Ebene mit Bezug zu unterstützender Technologie
Detaillierungsgrad	In einem Zug von einer Mitarbeiterin bzw. einem Mitarbeiter ausführbare Arbeitsschritte	Konkretisierung von Arbeitsschritten hinsichtlich Arbeitsverfahren sowie personeller und technologischer Ressourcen

Sowohl für die *fachlich-konzeptionelle Ebene* als auch für die *operative Ebene* können verschiedene unterstützende Werkzeuge eingesetzt werden. So kann auf der *operativen Ebene* von *Workflow Management Systemen (WfMS)* gesprochen werden. In Definition 2.2.2 ist eine Definition eines solchen Systems in Anlehnung an [Coa96] gegeben.

Definition 2.2.2. *(Workflow Management System)*

Ein Workflow Management System (WfMS) ist ein Werkzeug, das die Ausführung von Workflow-Prozessen durch die Verwendung von Software definiert, erzeugt und verwaltet.

Die *Workflow-Prozesse* werden dabei auf einer oder mehreren sogenannten *Workflow-Engines* (*WfE*) ausgeführt. Eine *WfE* ist dabei in der Lage, *Workflow-Prozesse* zu interpretieren, mit Workflow-Teilnehmern zu interagieren und – wo benötigt – die Einbindung von weiteren IT-Werkzeugen und Anwendungen durchzuführen.

In der Literatur [AHW03] wird das *BPM* heutzutage als eine Erweiterung zum *Workflow-Management* gesehen. Dies lässt sich dadurch begründen, dass es sich nicht nur auf die Phase der *Ausführung* von *Workflow-Prozessen*, sondern auch auf weitere Phasen bezieht. In derartigen Phasen können z.B. die *Evaluation und Verbesserung* von bestehenden Prozessen fokussiert werden (siehe Abschnitt 2.2.2). So kann das *BPM* ebenfalls durch verschiedene IT-gestützte Technologien und Systeme unterstützt werden. Derartige Systeme werden auch als *Business Process Management Systeme (BPMS)* bezeichnet und lassen sich nach [AHW03], wie in Definition 2.2.3 angegeben, definieren.

Definition 2.2.3. *(Business Process Management System)*

> *Ein Business Process Management System (BPMS) ist ein generisches Softwaresystem, welches die Gestaltung, Ausführung und insbesondere die Verwaltung von operationalen Prozessen unterstützt.*

Das in dieser Arbeit im Fokus stehende *BPM* kann auf Basis der zuvor gegebenen Beschreibung wichtiger Begriffe und in enger Anlehnung an [AHW03] wie durch Definition 2.2.4 gegeben definiert werden.

Definition 2.2.4. *(Business Process Management)*

> *Das Business Process Management (BPM) ist eine Methode zur Unterstützung von Geschäftsprozessen (hier: Prozessen). Dabei kommen Methoden, Techniken und Software zum Zweck des Designs & Analyse, der Konfiguration, der Ausführung, der Evaluation von operationalen Prozessen unter Berücksichtigung von Menschen, Organisationen, Anwendungen, Dokumenten und anderen Informationsquellen zum Einsatz.*

Die in Definition 2.2.4 aufgeführten Phasen *Design & Analyse*, *Konfiguration*, *Ausführung* und *Evaluation* zur Verwaltung von Prozessen bilden dabei einen möglichen Lebenszyklus von Prozessen. Im folgenden Abschnitt wird detailliert auf den sogenannten *BPM-Lebenszyklus* von Prozessen nach [Wes12] eingegangen, der die zuvor aufgeführten Phasen wieder aufgreift.

2.2.2 Der BPM-Lebenszyklus

Modelle für Lebenszyklen stellen einen möglichen methodischen Rahmen zur Unterstützung in der Verwaltung von Prozessen dar. Solche Modelle beschreiben wichtige Phasen und deren Übergänge, enthaltene Aufgaben und zu erstellende Artefakte. In der Domäne des *BPM* haben sich je nach Anwendungsgebiet insgesamt zwei solcher Modelle durchgesetzt [Wes12; Dum+18]. Aufgrund der inhaltlichen Nähe zu der im vorherigen Abschnitt eingeführten Definition wird in diesem Abschnitt der *BPM-Lebenszyklus* nach [Wes12] bevorzugt.

In Abbildung 2-6 ist hierzu eine eigene Darstellung der relevanten Phasen *Design & Analyse*, *Konfiguration*, *Ausführung* und *Evaluation* gezeigt. Jede dieser vier Phasen ist weiter unterteilt in typische Aktivitäten, die basierend auf [Wes12] im Folgenden näher beschrieben werden.

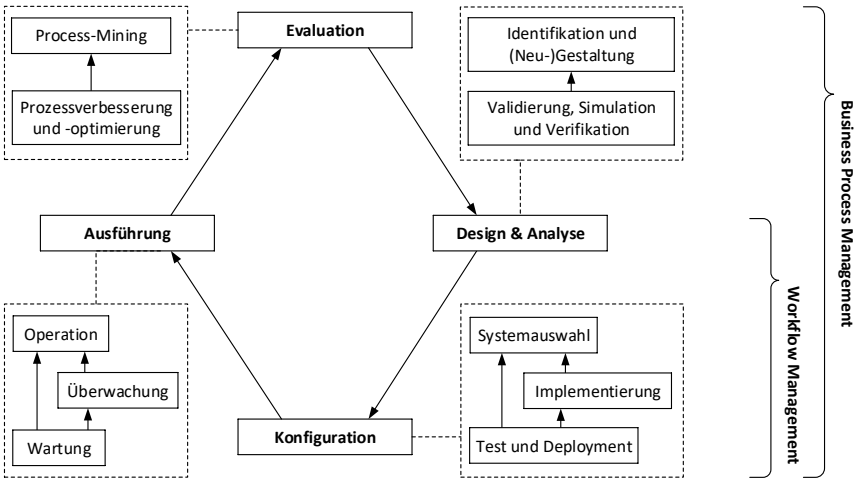


Abbildung 2-6: BPM-Lebenszyklus mit Differenzierung zum Workflow Management (nach Weske [Wes12] bzw. van der Aalst [AHW03])

Phase Design & Analyse

In der ersten Phase des *BPM-Lebenszyklus* werden zunächst in der Gestaltung die abzubildenden Abläufe der Prozesse identifiziert und durch entsprechende Prozessmodelle beschrieben. Für die Beschreibung von Prozessmodellen können unterschiedliche Sprachen eingesetzt werden, auf die in Abschnitt 2.3 näher eingegangen wird. Neben den Abläufen innerhalb eines geschäftlichen Umfeldes spielen aber auch die organisatorischen und technischen Gegebenheiten bei der Identifikation eine wichtige Rolle. Daher werden neben den Modellen für die Prozesse weitere Modelle erstellt, so z.B. Organigramme zur Beschreibung personeller Abhängigkeiten oder die organisatorische Einbettung der Prozesse.

Im Anschluss an die Gestaltung werden die erstellten Modelle hinsichtlich spezifischer Eigenschaften analysiert. Bei dieser Analyse kommen fachliche Validierungs- und Verifikationstechniken sowie Simulationen zum Einsatz. Neben fachlichen Validierungen – wie z.B. durch Experten in Workshops – stellen Simulationen ein wichtiges methodisches Werkzeug dar. So können durch entsprechende Werkzeuge auch komplexe Prozessmodelle analysiert werden, bei denen manuelle Verfahren zu aufwendig wären.

Phase Konfiguration

In der *Phase Konfiguration* wird die Implementierung der zuvor entworfenen und analysierten Prozessmodelle durchgeführt. Dabei bezieht sich der Begriff der Implementierung nicht zwangsweise auf die Umsetzung in Form von Software. So können die Prozessmodelle auch in Form von Richtlinien oder als dokumentiertes Beispiel für eine Erfolgsmethode (engl. *Best-Practice*) eingesetzt werden, die die Mitarbeiterinnen und Mitarbeiter umsetzen bzw. befolgen sollen. Soll der gestaltete Prozess durch den Einsatz von IT unterstützt werden, bieten sich spezielle Unterstützungssysteme an, wie z.B. die in Abschnitt 2.2.1 eingeführten *Workflow-Engines*.

Dabei müssen die Prozessmodelle um technische Informationen ergänzt werden, um ausgeführt werden zu können. Die Implementierung richtet sich dabei maßgeblich an Anforderungen des Unterstützungssystems. Nachdem die Zielumgebung bzw. das Unterstützungssystem ausgewählt und die Implementierung durchgeführt worden sind, kann der implementierte Prozess durch etablierte Verfahren aus dem Bereich des *Software Engineering* getestet werden. Ein Ziel kann hierbei die Sicherstellung des korrekt implementierten und erwarteten Verhaltens des Gesamtsystems sein. Ferner können auch Tests hinsichtlich potentieller Laufzeitprobleme – wie z.B. *Performance* oder *Speicherauslastung* – durchgeführt werden.

Phase Ausführung

Im Anschluss an die *Konfiguration und Implementierung* von Prozessen können Prozessinstanzen in der Phase der Ausführung (engl. *Enactment*) ausgeführt werden. Im Fall einer Instanziierung eines Prozesses folgt der Ablauf dem in der Phase *Design & Analyse* erfassten geschäftlichen Ablauf. Dabei wird der Zweck verfolgt, ein organisatorisches Ziel zu erfüllen – z.B. die Fertigung eines kundenindividuellen Produktes. Neben der eigentlichen Ausführung einer Prozessinstanz umfasst diese Phase jedoch auch noch die beiden Aktivitäten der *Überwachung* und der *Wartung*. Im Rahmen der Überwachung werden aktuelle Statusinformationen von Prozessinstanzen zur Laufzeit überwacht. Das Ziel ist hierbei die Sicherstellung von Anforderungen an die Prozessinstanzen, die erst zur Laufzeit überprüft werden können.

Beispiele im Rahmen von Arbeitsprozessen sind z.B. die Arbeitsgeschwindigkeit im Akkordbetrieb oder die Qualitätssicherung hinsichtlich einer Montage von (Teil-)Produkten. Neben diesen eher traditionellen Anforderungen können aber auch Anforderungen hinsichtlich einer Menschenzentrierung von Prozessen geprüft werden. So kann bspw. der körperliche Zustand oder die aktuelle Umgebungsbeschaffenheit in einer derartigen Anforderung betroffen sein. Für die Überwachung von Anforderungen zur Laufzeit können Softwarewerkzeuge eingesetzt werden, die die Überwachung automatisiert durchführen und im Fall einer Verletzung einen vordefinierten Handlungsplan zur Abstellung der Anforderungsverletzung umsetzen. Durch die Ausführung und Überwachung von Prozessinstanzen zur Laufzeit entsteht darüber hinaus eine Reihe von Daten, die in der Phase der Evaluation zur Analyse und Verbesserung von existierenden Prozessen eingesetzt werden kann. Derartige Daten können z.B. beschreiben, zu welchen Zeitpunkten die ausgeführten Prozessinstanzen, Tasks oder Aufgaben gestartet und beendet wurden. Ferner können aber auch zusätzliche Informationen enthalten sein, wie etwa die Anwendung eines Handlungsplans oder sonstige Fehler.

Phase Evaluation

Der *BPM-Lebenszyklus* nach [Wes12] schließt mit der Phase der *Evaluation* ab. In dieser Phase wird jedoch zuvor eine Retrospektive auf Basis der in der Phase *Ausführung* erhobenen Daten durchgeführt. Für derartige Analysen können unterschiedliche Methoden eingesetzt werden, wie etwa das sogenannte *Process-Mining* [AWM04; Aal16]. Das Ziel ist es hierbei die kontinuierliche Prozessoptimierung und -verbesserung zu unterstützen. Ein Beispiel im Rahmen von Arbeitsprozessen ist hierfür z.B. die Analyse von Teamzuteilungen einzelner Mitarbeiterinnen und Mitarbeiter. So können Daten in Bezug zu Teamzuteilungen und ermittelter Arbeitsgeschwindigkeit auf eine vorteilhafte oder ungünstige Einsatzplanung hinweisen. Die Evaluation bestehender Prozesse auf Basis von Daten aus der Phase der *Ausführung* ist dabei insbesondere für Arbeitsprozesse ein wichtiger und damit essentieller Schritt, um ein gewolltes Maß an Menschenzentrierung umzusetzen und zu gewährleisten.

Neben den eigentlichen Phasen des *BPM-Lebenszyklus* geht *Weske* darüber hinaus auf verschiedene beteiligte Rollen ein. Da dieser Aspekt auf die späteren Lösungsteile keinen wesentlichen Einfluss hat, wird an dieser Stelle abstrahiert und auf die Literatur verwiesen [Wes12].

2.2.3 Flexibilität in Prozessen

Die Betrachtung von Flexibilität in Prozessen ist ein weit verbreitetes Themenfeld der Domäne *BPM*. Dabei existieren in der Literatur verschiedene Taxonomien [Sof05; RSS06; Sch+08; RW12], die je nach Anwendungszweck relevant für die Gestaltung von Prozessen sein können. Nachfolgend werden Flexibilitätsaspekte von Prozessen in Anlehnung an [Sch+08] und [RW12] vorgestellt und an sinnvollen Stellen miteinander verglichen. In Abbildung 2-7 werden im oberen Bereich Flexibilitätsaspekte nach [Sch+08] und im unteren Bereich nach [RW12] gezeigt. Die dargestellten Pfeile weisen dabei auf vergleichbare Eigenschaften der Flexibilitätsaspekte hin, auf die nachfolgend in den Beschreibungen der beiden Taxonomien eingegangen wird. Zunächst folgt die Beschreibung der Taxonomie nach Schonenberg et. al [Sch+08].

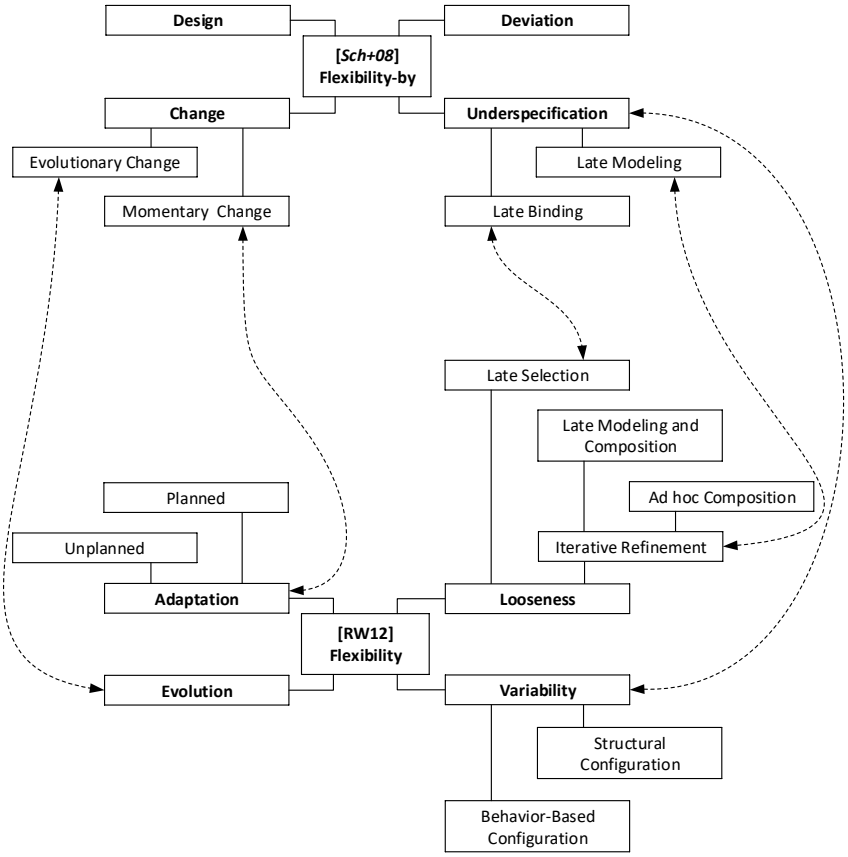


Abbildung 2-7: *Flexibilitätsaspekte im Vergleich*

Flexibility-by Design

Schonenberg et. al [Sch+08] führen den Flexibilitätsaspekt *Flexibility-by Design* an. Darunter verstehen die Autoren die Fähigkeit, alternative Ausführungspfade im Prozessmodell beschreiben zu können. Auf Basis dieser Ausführungspfade ist während der Ausführung ein geeigneter Pfad innerhalb der Prozessinstanz auswählbar. Zudem werden dabei verschiedene Vorschläge für Realisierungen gegeben, wie z.B. *Parallelität*, *Auswahl*, *Schleifen*, *Verschachtelung*, *nebenläufige Instanzen* und der *Abbruch* von einzelnen Aufgaben.

Es existieren Sprachen zur Gestaltung von Prozessen, die die zuvor genannten Realisierungen mindestens in Teilen unterstützen. Zwei Beispiele sind durch die Sprachen *BPMN2.0* [OMG11] und die *UML Aktivitätsdiagramme* [OMG15b] gegeben.

Flexibility-by Deviation

Flexibility-by Deviation beschreibt die Fähigkeit einer Prozessinstanz, von den im Prozessmodell beschriebenen Kontrollflusspfaden abzuweichen. Dabei werden keine Änderungen am ausgehenden Prozessmodell vorgenommen. Derartige Abweichungen können z.B. im Rahmen der Behandlungen von Fehlern oder Ausnahmen (engl. *Error- and Exception Handling*) sinnvoll sein [Cas+99; AWG05].

Flexibility-by Change

Ein weiterer Flexibilitätsaspekt beschreibt die Fähigkeit zur Anpassung von Prozessen. Von einer derartigen Anpassung können eine oder alle auf einem Prozessmodell basierenden und aktuell ausgeführten Prozessinstanzen betroffen sein. Anpassungen können – müssen aber nicht – auch die Anpassung des Prozessmodells einschließen. Siehe auch zur Differenzierung die beiden folgenden Untertypen *Momentary Change* und *Evolutionary Change*.

Momentary Change

Der erste Untertyp *Momentary Change* bezieht sich auf die Anpassung einer oder mehrerer ausgewählter Prozessinstanzen. Es handelt sich folglich um eine Anpassung, die nach Beendigung der betroffenen Prozessinstanzen für zukünftige Prozessinstanzen nicht mehr relevant ist. *Momentary Change* kann dem durch [RW12] vorgestellten Typ *Adaptation* bzw. *Ad-hoc Change* zugeordnet werden.

Evolutionary Change

Sollen von einer Anpassung auch zukünftige Prozessinstanzen betroffen sein, so muss die Anpassung auch auf dem Prozessmodell durchgeführt werden. Neue Prozessinstanzen auf Basis des geänderten Prozessmodells enthalten somit die bereits durchgeführten Anpassungen. Bei derartigen Anpassungen spricht man von *Evolutionary Change*. Sie lassen sich dem durch [RW12] vorgestellten Typ *Evolution* zuordnen.

Flexibility-by Underspecification

Der letzte von *Schonenberg et. al* [Sch+08] beschriebene Flexibilitätsaspekt bezieht sich auf die Fähigkeit eine Prozessinstanz auszuführen, die auf Basis eines unvollständigen Prozessmodells instanziiert worden ist. Dies ist insbesondere dann sinnvoll, wenn erst während der Ausführung alle benötigten Informationen über notwendige (Teil-)Aufgaben vorhanden sind. Prozessmodelle, die hinsichtlich *Flexibility-by Underspecification* erstellt worden sind, enthalten daher oft sogenannte *Platzhalter*. Dabei werden konkrete Realisierungen für derartige Elemente erst während der Ausführung instanziiert. Derartige Realisierungen werden in der korrespondierenden Literatur auch *Prozessfragmente* genannt. *Flexibility-by Underspecification* kann ferner in die beiden Typen *Late Binding* und *Late Modeling* unterschieden werden.

Flexibility-by Underspecification ist ein Flexibilitätsaspekt, der nicht allein durch ein geeignetes Prozessmodell bzw. die eingesetzte Sprache zur Gestaltung realisiert werden kann. Komplementär muss auch die Ausführungsumgebung eine geeignete Unterstützung anbieten. Beispiele für konkrete Konzepte zur Unterstützung von *Late Binding* und *Late Modeling* sind z.B. durch [Mur+13; CMT10] gegeben.

Late Binding

Bei *Late Binding* werden an der Stelle von *Platzhaltern* vordefinierte Funktionsblöcke, wie z.B. in Form von *Subprozessen* bzw. *Prozessfragmenten*, verlinkt und ausgeführt. *Late Binding* wird in der Literatur alternativ auch *Late Selection* genannt [RW12].

Late Modeling

Bei *Late Modeling* werden ebenso wie bei *Late Binding* an der Stelle von Platzhaltern Funktionsblöcke eingebunden. Im Gegensatz zum *Late Binding* sind diese Funktionsblöcke beim *Late Modeling* aber nicht vordefiniert. Daher müssen sie zunächst gestaltet werden. Diese Funktionsblöcke können an der unterspezifizierten Position innerhalb der Prozessinstanz verlinkt und anschließend ausgeführt werden. *Late Modeling* schließt damit auch die Einbindung eines Nutzers bzw. Domänenexperten mit ein.

Die vier zuvor beschriebenen Flexibilitätsaspekte der Taxonomie nach [Sch+08] beziehen sich teilweise auf das Prozessmodell oder die darauf gebildeten Prozessinstanzen. Ferner können sie zu unterschiedlichen Zeitpunkten im *BPM-Lebenszyklus* relevant sein. Komplementär zu den zuvor dargestellten Flexibilitätsaspekten von Prozessen wird nachfolgend auf die Taxonomie nach [RW12] eingegangen. Dabei werden die im unteren Bereich von Abbildung 2-7 dargestellten vier Flexibilitätsaspekte *Variability*, *Adaptation*, *Looseness* und *Evolution* aufgeführt.

Adaptation Beim Flexibilitätsaspekt *Adaptation* werden in Anlehnung an auftretende Ereignisse Anpassungen von Prozessen vorgenommen. Derartige Ereignisse können im Kontext von Prozessen z.B. bei technischen oder semantischen Fehlern, Zeitüberschreitungen oder der Nichtverfügbarkeit von Ressourcen auftreten. Der Flexibilitätsaspekt *Adaptation* wird insbesondere im Kontext von Fehler- und Ausnahmebehandlungen betrachtet und bietet Konzepte für die Behandlung von vorhersehbaren und unvorhersehbaren Ereignissen an. Vergleichbar mit *Flexibility-by Change* wird ebenso zwischen kurzzeitigen und dauerhaften Anpassungen unterschieden. Kurzzeitige Anpassungen beziehen sich dabei auf Prozessinstanzen. Sie werden auch als *Ad-hoc-Anpassungen* bezeichnet. Für den Flexibilitätsaspekt werden eine Reihe von Operationen zur Anpassung [WRR07; WRR08] vorgestellt, welche auch Anpassungsmuster genannt werden. Sind von einer Anpassung nicht nur Prozessmodelle, sondern auch Prozessinstanzen betroffen, so muss das Unterstützungssystem, wie z.B. eine Workflow-Engine, auch eine entsprechende Funktionalität bieten.

Looseness Der Flexibilitätsaspekt *Looseness* wird insbesondere mit wissensintensiven Prozessen in Verbindung gesetzt. Hier sind Reihenfolgen von Aktivitäten hochgradig spezifisch für eine konkrete Situation, sodass sie zum Zeitpunkt der Gestaltung nicht oder nur eingeschränkt gestaltbar sein können. *Looseness* sieht somit einen gewissen Grad an unterspezifizierten Prozessen vor und ist daher vergleichbar mit *Flexibility-by Underspecification*. Es werden die vier Typen *Late Selection*, *Iterative Refinement*, *Late Modeling and Composition* und *Ad-hoc Composition* von *Looseness* unterschieden.

Late Selection *Late Selection* ist dabei unmittelbar vergleichbar mit *Late Binding*. Es sieht die Verwendung von *Platzhaltern* vor, die zur Ausführungszeit durch *Subprozesse* bzw. *Prozessfragmente* ersetzt werden können.

Iterative Refinement Bei *Iterative Refinement* können zur Laufzeit weitere Aktivitäten hinsichtlich der Gestaltung von Prozessen ausgeführt werden. Bei *Late Modeling and Composition* und *Ad-hoc Composition* handelt es sich um spezielle Typen von *Iterative Refinement*. Dabei ist *Late Modeling and Composition* vergleichbar mit dem durch [Sch+08] vorgestellten Flexibilitätsaspekt *Late Modeling*. Es werden aber zusätzlich Aspekte der Komposition von neuen Funktionsblöcken auf Basis existierender und neu zu gestaltender Prozessfragmente betrachtet.

Ad-hoc Composition Bei *Ad-hoc Composition* werden zur Laufzeit einzelne Prozessfragmente sowie Bedingungen hinsichtlich ihrer erlaubten Kombinationen erstellt. Darauf aufbauend können zur Laufzeit Prozessfragmente durch Nutzerinnen und Nutzer zusammengestellt werden.

Evolution

Der *BPM-Lebenszyklus* sieht eine iterative Weiterentwicklung von Prozessen in Anlehnung an sich ändernde Anforderungen oder im Kontext eines sogenannten *kontinuierlichen Verbesserungsprozesses (KVP)* vor. Unter dem Flexibilitätsaspekt *Evolution* werden Anpassungen von Prozessmodellen sowie Prozessinstanzen verstanden. Dies ist vergleichbar mit dem durch [Sch+08] vorgestellten Flexibilitätsaspekt *Flexibility-by Change* sowie seinem Untertyp und *Evolutionary Change*.

Variability

Bei dem letzten Flexibilitätsaspekt *Variability* wird der Umgang mit verschiedenen *Prozessvarianten* verstanden. *Prozessvarianten* teilen sich einen gemeinsamen *Kernprozess*, auf dem aufbauend weitere Teile im Rahmen der Phase *Konfiguration* hinzugefügt werden können. *Variability* kann daher als eine spezielle Variante der Typen *Late Selection* bzw. *Late Binding* verstanden werden, die neben den Phasen *Design & Analyse* und *Ausführung* auch eine Rolle in der Phase *Konfiguration* spielen kann. So können Prozessvarianten in beiden vorgestellten Ansätzen in Anlehnung an einen konkreten Kontext bereits in der Phase *Konfiguration* selektiert werden. Der daraus resultierende konfigurierte Prozess benötigt in diesem Fall keine Prüfung von Bedingungen zur Laufzeit. Dies kann relevant sein, wenn das Unterstützungssystem keine Anpassung von Prozessen zur Laufzeit unterstützt. Dabei wird zwischen dem *verhaltensbasierten* und dem *strukturbasierten Konfigurationsansatz* unterschieden, auf die bspw. in [Tor+12] vertieft eingegangen wird.

Verhaltensbasierter Konfigurationsansatz

Beim *verhaltensbasierten Konfigurationsansatz* werden Prozessvarianten und zugehörige Bedingungen in einem gemeinsamen Prozessmodell beschrieben. Bei der Ausführung einer zugehörigen Prozessinstanz werden die Bedingungen in Anlehnung an den bestehenden Kontext geprüft und eine entsprechende Prozessvariante selektiert und ausgeführt. Existierende Ansätze, die diesen Ansatz unterstützen, sind z.B. durch *Configurable Event-driven Process Chains (C-EPC)* [RA07] und *C-YAWL* [Got+08] gegeben.

Strukturbasierter Konfigurationsansatz

Der *strukturbasierte Konfigurationsansatz* sieht die Trennung der Prozessvarianten von den Bedingungen vor. So existiert ein sogenannter Basisprozess, welcher an spezifischen Punkten sogenannte *Variation Points* enthält. Ein *Variation Point* kann durch spezifische Operationen zur Anpassung manipuliert werden. *Variation Points* sind dabei vergleichbar mit den aus *Late Selection* bekannten *Platzhaltern*. Eine Manipulation eines *Variation Point* führt zu der Herleitung einer spezifischen Prozessvariante. Ansätze, die den strukturellen Konfigurationsansatz umsetzen sind durch *Provop* [HBR10] und *vBPMN* [DZK11] gegeben.

[Tor+12] argumentieren, dass der *strukturbasierte Konfigurationsansatz* insbesondere bei größeren Prozessmodellen Vorteile gegenüber dem *verhaltensbasierten Konfigurationsansatz* bietet. So kann durch die Trennung des Basisprozesses von den Bedingungen (siehe auch *SoC*) eine reduzierte Komplexität hinsichtlich der Gestaltung dieser Aspekte erreicht werden. Ferner ist es beim *strukturbasierten Konfigurationsansatz* nicht notwendig, dass bereits zur Laufzeit alle Prozessvarianten bekannt sind. Hierdurch ist insbesondere die Erweiterungsfähigkeit der beteiligten Modelle positiv betroffen.

Zuvor wurden weitere Flexibilitätsaspekte von Prozessen vorgestellt. In Teilen verhalten sich die beiden aufgezeigten Taxonomien komplementär zueinander. So nimmt die durch [RW12] vorgestellte Taxonomie bspw. keinen direkten Bezug auf den Flexibilitätsaspekt *Flexibility-by Design*. Dafür fügen *Reichert und Weber* den Flexibilitätsaspekt *Variability* hinzu, der wiederum in [Sch+08] keine Berücksichtigung findet. Die restlichen Flexibilitätsaspekte beinhalten ähnliche Eigenschaften, die sich nur geringfügig unterscheiden.

2.3 Business Process Modeling

Die zweite in dieser Arbeit betrachtete Domäne stellt das *Business Process Modeling (BPMod)* dar. Dabei kann das *BPMod* als eine untergeordnete Disziplin des *BPM* betrachtet werden. Sie umfasst dabei zahlreiche Aspekte, die in der Gestaltung von Prozessen relevant sind. Nachfolgend wird zunächst in Abschnitt 2.3.1 eine Einführung in das *BPMod* mit dem Fokus auf der Vorstellung von unterschiedlichen Arten von Sprachen zur Gestaltung von Prozessen gegeben. Ergänzend wird in Abschnitt 2.3.2 auf unterschiedliche Perspektiven von Prozessen eingegangen. In Abschnitt 2.3.3 und Abschnitt 2.3.4 werden jeweils Beispiele und eine Übersicht der gängigsten Elemente von *UML Aktivitätsdiagrammen* sowie des De-facto-Standards *BPMN2.0* gegeben.

2.3.1 Einführung in das Business Process Modeling

Für die Phase *Design & Analyse* des *BPM-Lebenszyklus* werden geeignete Sprachen zur Gestaltung von Prozessen benötigt. So haben sich für diese beiden Bereiche einige bekannte Standards etabliert. Hierzu werden die Sprache *BPEL* [OAS07] und seit 2011 auch die Sprache *BPMN2.0* [OMG11] gezählt. Dabei hat sich über die Jahre die Sprache *BPMN2.0* als De-facto-Standard in der Industrie durchgesetzt. Wissenschaftliche Vertreter von

Ansätzen zur Gestaltung von Prozessen sind beispielsweise durch die Ansätze *ADEPT1* und *ADEPT2* [RD09], *YAWL* [AT05] oder *Petri-Netze* [Mur89] gegeben.

Neben den bereits zuvor genannten Ansätzen und Sprachen existiert eine Vielzahl an weiteren Sprachen, die zur Gestaltung von flexiblen Prozessen eingesetzt werden kann. Diese Sprachen lassen sich in die drei Kategorien *datenflussorientiert*, *kontrollflussorientiert* und *objektorientiert* einteilen. Eine Zusammenfassung von diagrammbasierten Sprachen mit Zuordnung zu einer der drei zuvor genannten Kategorien ist in Tabelle 2-3 nach [Gad08] dargestellt.

Orientierung	**Methode**
Objektorientiert	Aktivitätsdiagramm (UML) | Activitychart-Diagramm | Statechart-Diagramm | Use Case-Diagramm (UML) | Objektorientierte EPK | Integrationsdiagramm (SOM) | Vorgangsereignisschema (SOM)
Kontrollflussorientiert	Petri-Netze | Folgestruktur- und Folgeplan | Aufgabenkettendiagramm (PROMET) | GPM Diagramm | Struktogramme | Swimlane-Diagramm | Erweiterte EPK | Picture | Business Process Model and Notation (BPMN)
Datenflussorientiert	IDEF-Diagramm | Datenflussdiagramm (SSA) | Flussdiagramm (SADT)

Tabelle 2-3: *Sprachen zur Gestaltung von Prozessen (nach [Gad08])*

Eine Sprache für die Gestaltung von Prozessen sollte dabei je nach Anforderung in der Lage sein, geschäftliche Abläufe durch grundlegende Konzepte, wie etwa die Benennung von notwendigen Informationen, den einzelnen Tätigkeiten, Ablaufbeziehungen sowie die Zuordnung von Rollen, zu beschreiben.

2.3.2 Perspektiven in Geschäftsprozessmodellen

Moderne Prozesse berücksichtigen eine Vielzahl unterschiedlicher Aspekte. Die adäquate Gestaltung dieser Aspekte von Prozessen kann dabei eine hohe Komplexität aufweisen. Zur Reduzierung dieser Komplexität werden Prozesse häufig aus verschiedenen Perspektiven beschrieben. Eine Perspektive kann dazu genutzt werden, um einen einzelnen oder eine Gruppe von ausgesuchten Aspekten fokussieren zu können.

Der Vorteil bei der Verwendung von unterschiedlichen Perspektiven auf einen Prozess ist die (teilweise) Trennung der relevanten Aspekte. Das unterliegende Konzept ist dabei vergleichbar mit dem in Abschnitt 1.2 eingeführten Konzept des *Separation-of-Concerns (SoC)*. Durch Perspektiven können verschiedene Fragen beantwortet werden, die unterschiedliche Ausprägungen haben können. Auf eine Auswahl dieser Fragen nach [CKO92] wird nachfolgend eingegangen.

- Welche Handlung soll stattfinden?
- Wer wird die Handlung ausführen?
- Wann und wo wird die Handlung stattfinden?
- Wie und warum soll die Handlung stattfinden?
- Wer ist von der Handlung betroffen?

Durch *Curtis* [CKO92] werden insgesamt die vier folgenden Klassen von Perspektiven unterschieden. In der Gesamtheit aller Klassen wird ein Prozess als *integriert*, *vollständig* bzw. *konsistent* verstanden. Auf Details dieser Perspektiven wird nachfolgend eingegangen.

Funktion Die Perspektive *Funktion* beschreibt, was in den einzelnen Schritten des Prozesses getan werden muss. Ferner werden auch Abhängigkeiten zu verschiedenen Informationen, wie z.B. Daten, Artefakten oder Produkten, betrachtet.

Verhalten Die Perspektive *Verhalten* beschreibt den Kontrollfluss des Prozesses. Sie bezieht sich damit auf zeitliche und logische Abhängigkeiten zwischen verschiedenen Elementen des Prozesses.

Organisation Durch die Perspektive *Organisation* wird festgelegt, welcher Aufgabenträger in einem Prozess an welchem Schritt beteiligt ist. Klassischerweise sind hiervon die drei Typen *Mensch*, *Maschine* und *Anwendung* betroffen. Ferner können auch Gruppen, Kategorien, Rollen oder organisatorische Einheiten sowie ihre Beziehungen als relevante Aspekte des Prozesses darstellbar sein.

Information Die Perspektive *Information* beschreibt den Datenfluss eines Prozesses. Damit fokussiert sie die Daten und Informationen, die durch Schritte des Prozesses erzeugt oder manipuliert werden. Neben dieser Beschreibung von Ein- und Ausgaben stehen aber auch die Struktur der Daten und Informationen sowie ihre Beziehungen im Fokus dieser Perspektive.

Weitere Perspektiven auf Prozesse

In der Literatur [JB96; AJ00; AHW03; ARD07] werden weitere Perspektiven oder modifizierte Klassen von Perspektiven betrachtet. So unterscheiden [JB96; AJ00] insgesamt fünf verschiedene Klassen. Dabei werden die durch *Curtis* vorgestellten Perspektiven *Funktion* und *Verhalten* in einer neuen Perspektive *Prozess* zusammengefasst. Die Ansätze führen weitere Perspektiven durch *Operation* und *Integration* ein. Die Perspektive *Operation* beschreibt elementare Operationen, die von den an einem Prozess beteiligten Ressourcen und Anwendungen ausgeführt werden. Beispiele für derartige Operationen geben die Autoren durch das Erstellen, Lesen oder Modifizieren von Daten aus der Perspektive *Information* an. Die Perspektive *Integration* führt die einzelnen Elemente der Perspektive *Prozess* mit den Elementen der anderen Perspektiven zusammen. So werden z.B. identifizierte Aufgaben zu Rollen, Gruppen oder organisatorischen Einheiten aus der Perspektive *Organisation* referenziert.

Der in [AHW03] vorgestellte Ansatz unterscheidet die fünf Perspektiven *Funktion*, *Prozess*, *Organisation*, *Information* und *Operation*. Eine Unterscheidung zu den in [CKO92] vorgestellten Perspektiven ist durch die Perspektiven *Prozess* und *Operation* möglich. Die Perspektive *Prozess* kann der Perspektive *Verhalten* zugeordnet werden, da sie einen vergleichbaren Zweck erfüllt. Analog zu dem in [JB96; AJ00] dargestellten Ansatz werden in der Perspektive *Operation* elementare Operationen beschrieben.

Der letzte hier vorgestellte Ansatz [ARD07] beschreibt zunächst die vier Perspektiven *Prozess*, *Daten*, *Ressource* und *Task*. Dabei lassen sich diese Perspektiven hinsichtlich ihrer Bedeutung auf die durch [CKO92] vorgestellten Perspektiven anwenden. So lässt sich *Prozess* zu *Verhalten*, *Task* zu *Funktion*, *Daten* zu *Information* und *Ressource* zu *Organisation* zuordnen. Auf dieser Grundlage beschreiben [AHW03], dass eine Reihe weiterer Perspektiven je nach Anwendungszweck als sinnvoll erachtet werden kann. Sie führen daher die sogenannten höherwertigen Perspektiven *Kontext*, *Ziel*, *Performance* und *Produkt* bzw. *Dienstleistung* ein. Im Rahmen der Perspektive *Kontext* kann das Umfeld, für den der Prozess beschrieben wird, gestaltet werden. So kann die Ausführung einzelner Aufgaben oder die Einbindung von Ressourcen z.B. von der Jahres- oder Tageszeit abhängig sein. Die Perspektiven *Ziel*, *Performance* und *Produkt* bzw. *Dienstleistung* beschreiben weitere Aspekte des Prozesses. Die Autoren verweisen darauf, dass diese drei Perspektiven auch oftmals in klassischen Perspektiven berücksichtigt werden können.

Die in diesem Abschnitt vorgestellten Konzepte für Perspektiven bilden die Basis für die in Abschnitt 4.3.3 durchgeführte Analyse der Sprache

BPMN2.0. Dabei muss an dieser Stelle angemerkt werden, dass nicht jede der aufgeführten Perspektiven auch in einer solchen Sprache enthalten sein muss. Die durch [CKO92] genannten Perspektiven *Funktion* und *Verhalten* sind typischerweise aber Bestandteil einer entsprechenden Sprache. Andere Perspektiven wie *Organisation* und *Information* können als optional betrachtet werden.

In den nachfolgenden Abschnitten 2.3.3 und 2.3.4 werden zwei Beispiele für Sprachen zur Gestaltung von Prozessen gegeben. So werden zunächst *UML Aktivitätsdiagramme* [OMG15b] vorgestellt, welche die Basis für die Gestaltung von Verhalten im Rahmen des Ansatzes *Adapt Cases* bilden (siehe Abschnitt 2.4). Anschließend wird die Sprache *BPMN2.0* [OMG11] vorgestellt, welche in der Domäne *BPM* den De-facto-Standard zur Gestaltung von Prozessen bildet und auch in dem in dieser Arbeit vorgestellten Lösungsansatz verwendet wird.

2.3.3 UML Aktivitätsdiagramm

Die *Unified Modeling Language (UML)* [OMG10] stellt einen industriellen Standard dar, der unterschiedlichste Aspekte der objektorientierten Gestaltung von modernen Softwaresystemen beinhaltet. Dabei existiert die Möglichkeit, dass sowohl die Struktur als auch das Verhalten von derartigen Systemen gestaltet werden können. Die in dieser Arbeit fokussierten Prozesse stellen eine Form für ein solches Verhalten dar. Hierzu bietet die *UML* verschiedene Diagramme zur Gestaltung von Verhalten an, wie z.B. das *Zustandsdiagramm* oder das *Aktivitätsdiagramm*.

In Abbildung 2-10 ist eine Übersicht ausgesuchter Elemente von *Aktivitätsdiagrammen* dargestellt. Nachfolgend wird auf eine detaillierte Beschreibung dieser Elemente sowie auf ein zugehöriges Beispiel eingegangen.

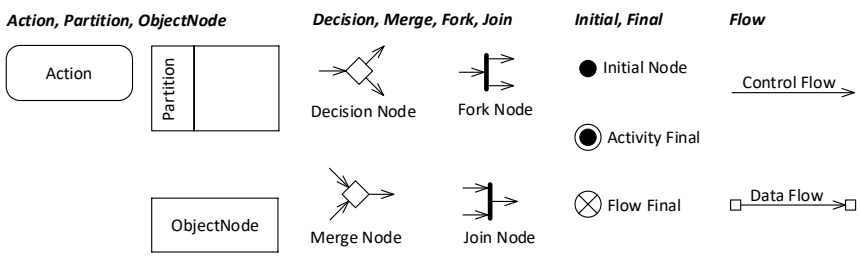


Abbildung 2-8: Elemente von UML Aktivitätsdiagrammen

Mit Hilfe eines *Aktivitätsdiagramms* kann ein Ablauf von einzelnen Aufgaben in Form einer Sequenz von Aktivitäten (hier: *Action*) beschrieben werden. Ein Element des Typs *Action* kann weiteres Verhalten enthalten, wodurch die übergeordnete Aktivität verfeinert wird. Elemente des Typs *Action* können durch ein Element des Typs *Partition* gruppiert werden. Ein Element des Typs *Partition* kann z.B. zur Beschreibung von einer organisationalen Zugehörigkeit oder einer Rolle verwendet werden. Bei der Ausführung von Aktivitäten können Daten benötigt oder erzeugt werden. Durch ein Element des Typs *ObjectNode* können derartige Daten beschrieben werden.

Im Rahmen des Kontrollflusses können *Verzweigungen* vorkommen, mit deren Hilfe können konditional-abhängige Sequenzen (*Decision*) beschrieben werden. Durch weitere Konstrukte der Sprache sind aber auch parallele (Teil-)Sequenzen (*Fork*) möglich. Ein verzweigter Kontrollfluss kann entweder durch ein Element des Typs *MergeNode* oder durch ein Element des Typs *JoinNode* zusammengeführt werden.

Der Start des Verhaltens in einem *Aktivitätsdiagramm* wird durch ein Element des Typs *InitialNode* beschrieben. Es werden insgesamt zwei mögliche Beendigungen des Verhaltens unterschieden. Zum einen kann durch Elemente des Typs *ActivityFinal* das Verhalten der aktuellen Aktivität beendet werden, wohingegen durch Elemente des Typs *FlowFinal* lediglich der jeweilige (Teil-)Pfad des Verhaltens beendet wird.

Um den Kontrollfluss zu beschreiben kann das Flusselement vom Typ *ControlFlow* verwendet werden. Ein solches Element verbindet jeweils zwei Elemente miteinander, wie z.B. Aktivitäten oder die zuvor eingeführten Kontrollelemente. Um Datenfluss zu beschreiben, kann das Flusselement vom Typ *DataFlow* verwendet werden.

Ein Beispiel eines *Aktivitätsdiagramms* ist in Abbildung 2-9 gezeigt. Auf eine Beschreibung des dargestellten Verhaltens wird nachfolgend eingegangen.

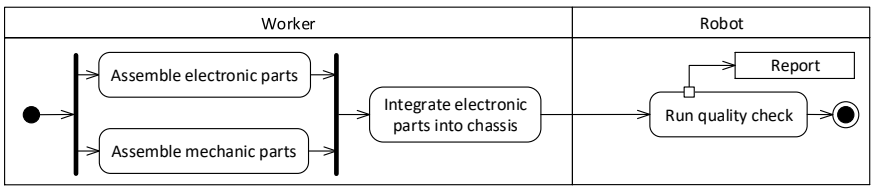


Abbildung 2-9: Beispiel eines UML Aktivitätsdiagramms

Das dargestellte Verhalten beschreibt einen Montageprozess, dessen Verhalten sich auf die beiden Rollen *Worker* und *Robot* aufteilt. Durch die Rolle *Worker* wird dabei die eigentliche Montage durchgeführt. Die Rolle *Robot* übernimmt anschließend Maßnahmen zur Qualitätssicherung und er-

stellt einen zugehörigen Bericht. Dabei wird das Verhalten zunächst durch die Rolle *Worker* gestartet. Hier wird der Kontrollfluss in zwei parallele Pfade aufgeteilt. Hierdurch ist eine parallele Ausführung der dargestellten Aktivitäten des nachfolgenden Kontrollflusses möglich. Ob diese Aktivitäten tatsächlich parallel oder beliebig versetzt ausgeführt werden, ist durch ein *Aktivitätsdiagramm* nicht beschreibbar. Werden beide Aktivitäten beendet, werden die einzelnen Pfade wieder zu einem Pfad zusammengeführt. Die letzte Aktivität wird durch die zweite Rolle *Robot* ausgeführt. Dabei wird ein neues Datenobjekt mit der Bezeichnung *Report* erzeugt und das Gesamtverhalten anschließend beendet.

2.3.4 BPMN2.0

In der Gestaltung von Prozessen stellt die Sprache *Business Model and Notation (BPMN)* [OMG11] derzeit den De-facto-Standard dar. Die Entwicklung der Sprache *BPMN2.0* wird durch die *Object Management Group (OMG)* verwaltet. Durch die Verwendung der Sprache *BPMN2.0* wird das Ziel verfolgt, den Übergang von der Gestaltung von Prozessen hin zu deren Ausführung zu verkürzen bzw. zu vereinfachen. Hierzu enthält die Sprache zahlreiche Sprachelemente für die Domäne *BPM*. Die Gestaltung von Prozessen kann bereits so umfangreich sein, dass der gestaltete Prozess oftmals durch ein entsprechendes Prozessunterstützungssystem, wie z.B. eine Workflow-Engine, direkt ausgeführt werden kann.

Die Sprache *BPMN2.0* verfügt seit der *Version 2.0* über insgesamt vier unterschiedliche Diagrammtypen. Jeder Diagrammtyp fokussiert dabei einen unterschiedlichen Aspekt in der Gestaltung von Prozessen. Als zentraler Diagrammtyp können *Business Process-Diagramme (BPD)* betrachtet werden. Da an Abläufen häufig unterschiedliche inter- und intraorganisatorische Rollen beteiligt sind, existieren zudem drei weitere Diagrammtypen zur Beschreibung von Kollaborationen, Konversationen und Choreographien. Der Fokus der weiteren Diagrammtypen liegt auf der Gestaltung von Zusammenhangkomponenten zwischen unterschiedlichen Prozessen oder beteiligten Rollen. So kann hier z.B. stärker auf den Nachrichtenfluss zwischen beteiligten Prozessrollen durch ein Kollaborationsdiagramm Bezug genommen werden.

In dieser Arbeit liegt der Fokus auf der Gestaltung von Prozessen unter Verwendung von *BPD*. Nachfolgend wird daher zunächst eine Auswahl von zugehörigen Elementen sowie ein Beispiel eines Prozesses durch ein *BPD* gegeben. Auf eine detaillierte Übersicht über weitere Diagrammtypen wird an dieser Stelle verzichtet.

Das *Business Process Diagram (BPD)* stellt einen zentralen Typ von Diagrammen in der Sprache *BPMN2.0* dar. Ein *BPD* beschreibt dabei den Ablauf eines Prozesses durch Aktivitäten bzw. Tasks. Dabei werden in der Beschreibung von Abläufen eines Prozesses zum Teil vergleichbare Konzepte verwendet, wie sie auch in *UML Aktivitätsdiagrammen* vorkommen (siehe Abschnitt 2.3.3). In Abbildung 2-10 werden ausgesuchte Elemente zur Gestaltung von Prozessen durch ein *BPD* gezeigt. Auf ihre Bedeutung wird nachfolgend detaillierter eingegangen.

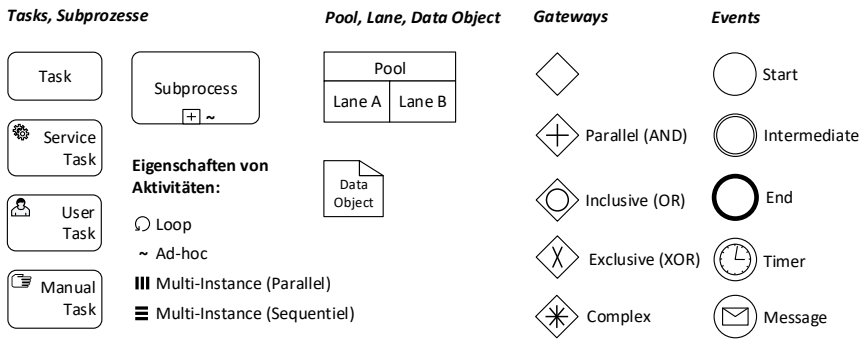


Abbildung 2-10: *Elemente eines Business Process Diagram*

Das Verhalten eines durch ein *BPD* gestalteten Prozesses enthält eine oder mehrere Aktivitäten. Aktivitäten können entweder durch Elemente des Typs *Task* oder des Typs *Subprocess* (*deutsch: Unterprozess*) beschrieben werden. Dabei handelt es sich bei *Tasks* um eine atomare und bei *Subprocess* um eine dekomponierbare Aufgabeneinheit. Ferner sieht die Sprache *BPMN2.0* zahlreiche Untertypen für *Tasks* vor, für die jeweils ein spezieller Zweck vorgesehen ist. Beispiele hierfür sind durch die Typen *ServiceTask*, *UserTask* und *ManualTask* gegeben. Der Typ *ServiceTask* ist dafür vorgesehen, dass ein IT-Dienst die jeweilige Aktivität übernehmen soll. Dementsprechend können *UserTasks* dafür eingesetzt werden, dass ein menschlicher Akteur die Aktivität unter Verwendung einer IT-Unterstützung ausführt. Ein besonderer Typ von *Tasks* ist durch *ManualTask* gegeben. Hierbei wird angenommen, dass die zugehörige Aktivität durch einen menschlichen Akteur, aber ohne IT-Unterstützung, durchgeführt wird.

Ein Element des Typs *Subprocess* kann dabei wiederum weitere Elemente zur Beschreibung von Verhalten enthalten. Das durch ihn beschriebene Verhalten ist daher dekomponierbar. Ein solches Element kann sowohl aufgeklappt als auch zusammengeklappt darstellt werden. In Abbildung 2-10 ist ein zusammengeklapptes Element gezeigt, was durch das umrandete Plussymbol gezeigt wird. Eine zusammengeklappte Darstellung kann z.B. dann verwendet werden, wenn das enthaltene Verhalten erst später gestaltet wird. Alternativ kann hierdurch aber auch bereits beschriebenes Ver-

halten ausgeblendet werden, sodass die Fokussierung auf weitere Aspekte des Prozesses unterstützt werden kann.

Sowohl Tasks als auch Unterprozesse können darüber hinaus auch mit einer Reihe von unterschiedlichen Eigenschaften versehen werden. So kann z.B. für einen Task spezifiziert werden, dass mehrere seiner Instanzen parallel oder sequentiell ausgeführt werden sollen. Alternativ kann auch eine wiederholte Ausführung durch die Eigenschaft *Loop* beschrieben werden. Die Eigenschaft *Ad-hoc* hingegen kann z.B. bei Unterprozessen eingesetzt werden, wenn die enthaltenen Aktivitäten in einer nur geringfügig vorbestimmten Reihenfolge ausgeführt werden dürfen.

Durch ein Element des Typs *Pool* werden Aktivitäten gruppiert, die zu einem Teilnehmer eines Prozesses gehören. Kann ein Teilnehmer eines Prozesses weiter aufgeteilt werden, so bietet sich die Verwendung von Elementen des Typs *Lane* an. Ein Element des Typs *Lane* ist dabei Teil eines *Pools* und enthält Aktivitäten, die zu einem untergeordneten Teilnehmer bzw. zu einer untergeordneten Rolle gehören. In Prozessen können ebenfalls die Erzeugung und Verwendung unterschiedlichster Daten notwendig sein. Daten können durch Elemente des Typs *DataObject* im Verlauf des Prozesses beschrieben werden.

In der Sprache *BPMN2.0* können entlang des Kontrollflusses Elemente des Typs *Gateway* eingesetzt werden. Hierdurch wird der Verlauf des Kontrollflusses zur Ausführungszeit gesteuert, sodass bspw. die Auswahl eines alternativen Pfades oder mehrerer paralleler Pfade ermöglicht wird. Dabei existieren Gateways für *XOR-*, *OR-* und *AND-*Operationen. Als Ergänzung dieser gängigen Typen von Gateways können aber auch Elemente des Typs *ComplexGateway* eingesetzt werden. Durch ein solches Gateway können komplexe Konditionen ausgewertet werden, sodass bspw. eine Auswahl von mehreren ausgehenden Pfaden ermöglicht wird.

Events sind spezielle Konstrukte einer Sprache, die Ereignisse beschreiben. In der Sprache *BPMN2.0* wird dabei in die drei grundlegenden Typen Startereignis (*Start*), Zwischenereignis (*Intermediate*) und Endereignis (*End*) unterschieden. Ferner sind weitere Typen von Ereignissen für Spezialfälle verfügbar. So können z.B. auch zeitgesteuerte Ereignisse (*TimerEvent*) eingesetzt werden, die sowohl als ein Start- als auch als ein Zwischenereignis eingesetzt werden können. Neben Elementen des Typs *TimerEvent* ist ein grundlegender weiterer Typ durch *MessageEvent* gegeben. Ein solches Element stellt einen wesentlichen Teil eines Nachrichtenaustauschs zwischen Kommunikationspartnern dar.

Die zuvor beschriebenen Elemente können durch verschiedene Elemente miteinander verbunden werden. Eine Auswahl derartiger Elemente ist in Abbildung 2-11 dargestellt. Durch ein Element des Typs *SequenceFlow* wird die Reihenfolge einzelner Aktivitäten eines Prozesses festgelegt. Hierbei können z.B. Elemente der Typen *Task*, *Subprocess* oder *Gateway* miteinander verbunden werden. Werden Gateways mit Entscheidungen eingesetzt, so bietet sich die Möglichkeit der Kennzeichnung, dass es sich bei einem beteiligten Element des Typs *SequenceFlow* um die Standardwahl (*default*) handelt.

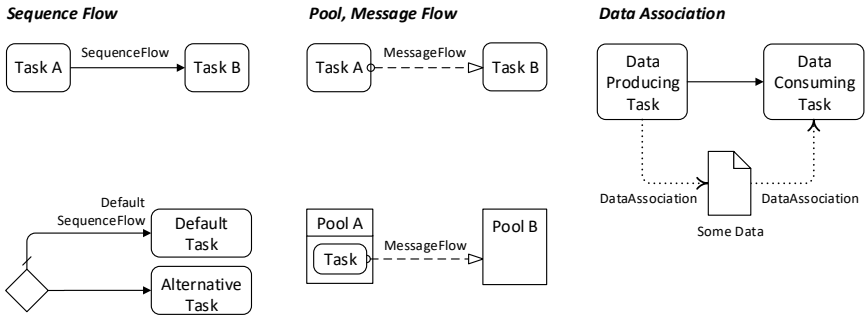


Abbildung 2-11:
Weitere Elemente eines Business Process Diagram

Durch ein Element des Typs *MessageFlow* wird der Austausch von Nachrichten (engl. *Message*) beschrieben. Nachrichten können zwischen Aktivitäten und Elementen des Typs *Pool* ausgetauscht werden. Ein Beispiel für den Nachrichtenaustausch ist in Abbildung 2-11 gegeben. Hier ist ein Nachrichtenaustausch von der Aktivität *Task A* zur Aktivität *Task B* dargestellt. Durch den dritten Typ *DataAssociation* können Datenobjekte mit Aktivitäten verbunden werden. Ein Datenobjekt kann – wie in Abbildung 2-11 dargestellt – z.B. durch eine Aktivität erzeugt oder von ihr für die weitere Verarbeitung verlangt werden.

Ein Beispiel für einen Prozess, der durch ein *BPD* der Sprache *BPMN2.0* beschrieben worden ist, wird in Abbildung 2-12 gezeigt. Das Verhalten orientiert sich dabei an dem gegebenen Beispiel für *UML Aktivitätsdiagramme* (siehe Abbildung 2-9). Ein wesentlicher Unterschied zur Gestaltung mittels *UML Aktivitätsdiagrammen* ist hier vornehmlich durch die Differenzierung unterschiedlicher Typen von Aktivitäten erkennbar. So lässt sich bspw. ausdrücken, dass bestimmte Tasks IT-gestützt oder ohne IT-Unterstützung durchgeführt werden sollen. Ferner lässt sich das Verhalten der beiden Rollen *Worker* und *Robot* durch eine nachrichtenbasierte Kommunikation entkoppeln. So ist eine Beendigung des (Teil-)Verhaltens der Rolle *Worker* möglich, sobald die letzte zugehörige Aktivität beendet worden ist.

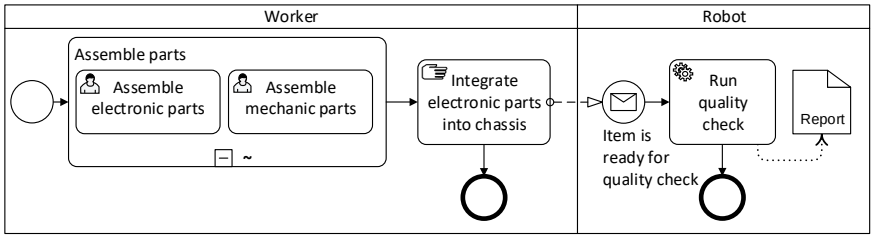


Abbildung 2-12: *Beispiel eines Business Process Diagram*

Ferner ist durch den Unterprozess *Assemble parts* ein Beispiel für einen sogenannten *Ad-Hoc-Prozess* gegeben. Das in ihm gezeigte Verhalten wird auch als schwach strukturiert bezeichnet. Hierdurch ist die reale Reihenfolge der Aktivitäten *Assemble electronic parts* und *Assemble mechanic parts* durch die Rolle *Worker* frei wählbar.

2.4 Adapt Cases

Der Ansatz *Adapt Cases* und das mit ihm verbundene *Adaptivity Engineering* wurde erstmals durch *Luckey* [Luc+11] eingeführt. Dabei wird die Gestaltung von Funktionen zur Anpassung von selbst-adaptiven Softwaresystemen in einer frühen Phase des *Software Development Process (SDP)* bzw. Entwicklungsprozesses verstanden. In *Adapt Cases* wird dabei insbesondere die Trennung der Anpassungs- von der Anwendungslogik fokussiert. Durch die Wiederverwendung von Grundprinzipien des Gestaltungsansatzes der *UML Use Cases* kann deren hohe Ausdrucksfähigkeit übernommen werden. Ferner lässt sich der Ansatz *Adapt Cases* durch weit verbreitete Gestaltungstechniken in eine Vielzahl von bestehenden Entwicklungsprozessen integrieren.

Das *Adaptivity Engineering* nach *Luckey* enthält neben einem konstruktiven Verfahren zur getrennten Gestaltung der Anwendungs- und Anpassungslogik zudem auch eine Methode zur Qualitätssicherung von relevanten Artefakten. Der Ansatz *Quality Assurance For Adaptive Systems (QUAASY)* stellt ein Verfahren zur Analyse von verschiedenen Systemeigenschaften dar und kann in diesem Bezug zur Überprüfung verschiedener Qualitätsanforderungen eingesetzt werden. Er basiert auf etablierten Techniken aus dem Bereich der Graphtransformationen [Eng+00] und des Model-Checkings [Ren03; ESW07]. Da der Fokus dieser Arbeit auf der konstruktiven Gestaltung von Prozessen liegt, wird für weiterführende Informationen auf die Arbeit von *Luckey* [Luc13] verwiesen.

Nachfolgend wird zunächst in Abschnitt 2.4.1 das generelle Prinzip des Ansatzes *Adapt Cases* und der zugehörigen Sprache *ACML* vorgestellt. An-

schließend wird in Abschnitt 2.4.3 auf die abstrakte Syntax eingegangen. In Abschnitt 2.4.2 wird die konkrete Syntax der Sprache *ACML* anhand eines Beispiels veranschaulicht. Abschließend wird in Abschnitt 2.4.4 eine exemplarische Integration in einen *SDP* beschrieben.

2.4.1 Überblick

Die Sprache *Adapt Case Modeling Language (ACML)* kann zur getrennten Gestaltung von Aspekten der Anpassungs- und Anwendungslogik eingesetzt werden. Dabei werden Konzepte aus der Gestaltung von selbstadaptiven Systemen wiederverwendet. In Abbildung 2-13 ist das Grundprinzip des Ansatzes *Adapt Cases* nach *Luckey* [Luc+11; LE13] dargestellt.

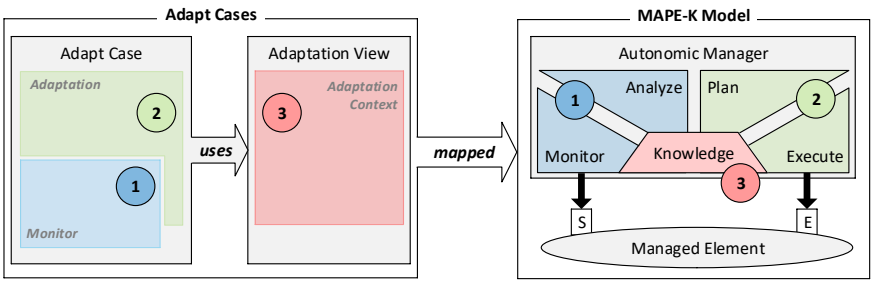


Abbildung 2-13: Prinzip des Ansatzes Adapt Cases (nach Luckey [Luc+11])

In dem Ansatz *Adapt Cases* wird die Referenzarchitektur *MAPE-K* als Sicht auf das zu gestaltende Softwaresystem eingesetzt. *MAPE-K* wurde durch [KC03] eingeführt und stellt in der Domäne der selbst-adaptiven Systeme eine der am häufigsten eingesetzten Modelle zur Beschreibung von wesentlichen Funktionen und Abläufen in Bezug zur eigenständigen Anpassung eines Systems dar. Das Prinzip des Modells stellt eine Rückkopplungsschleife (engl. *Feedback Loop*) dar, entlang der eine Reihe von verschiedenen Funktionen ausgeführt wird.

Das Modell sieht dabei die Überwachung und bedarfsorientierte Anpassung eines sogenannten *Managed Element* durch das Konzept des *Autonomic Managers* vor. Für die Überwachung und Anpassung stellt das *Managed Element* wohldefinierte Schnittstellen in Form eines Sensors (*S*) und eines Effektors (*E*) zur Verfügung. Durch diese Schnittstellen werden der lesende Zugriff (*S*) auf Daten und der schreibende Zugriff (*E*) auf relevante Teile des Systems und deren Umgebung unterstützt.

Der *Autonomic Manager* enthält eine Reihe von Funktionen, die zur Erkennung, Auswahl und Ausführung von Anpassungen an Teile des *Managed Element* notwendig sein können. *Luckey* partitioniert diese Funktionen in

drei Bereiche, die in Abbildung 2-13 farbig dargestellt sind. Ferner sieht er für die Gestaltung die beiden Modelle *Adapt Case Model (AVM)* und *Adaptation View Model (ACM)* vor. Nachfolgend wird zunächst auf die Partitionen und anschließend auf die beiden Modelle eingegangen.

Die erste Partition gruppiert die beiden Funktionen *Monitor* und *Analyze* des Modells *MAPE-K*. *Luckey* nennt diese Partition *Monitor*. Der Fokus dieser Partition bezieht sich auf die Erhebung und Aggregation von Daten unter Verwendung von Operationen der Schnittstelle *S*. Ferner ist eine weiterführende Analyse dieser Daten vorgesehen, deren Ergebnis anzeigt, ob Anpassungen notwendig sind oder nicht. Falls eine Anpassung notwendig ist, so werden Funktionen aus der zweiten Partition *Adaptation* eingesetzt.

Durch die zweite Partition werden die beiden Funktionen *Plan* und *Execute* des Modells *MAPE-K* gruppiert. Der Fokus dieser Partition liegt auf der Beschreibung von Verhalten, durch das eine Anpassung am *Managed Element* ausgeführt werden kann. *Luckey* sieht hier die Beschreibung von unterschiedlichen *Alternativen* für eine Anpassung vor. Diese *Alternativen* lassen sich in dem Modell *MAPE-K* der Funktion *Plan* zuordnen. Ferner kann eine *Alternative* spezifische Operationen zur Anpassung des *Managed Element* enthalten, die durch die Schnittstelle *E* angeboten werden.

Die dritte und letzte Partition *Adaptation Context* beinhaltet das Konzept *Knowledge*. Hierbei handelt es sich weniger um eine Funktion. Vielmehr wird durch die vier zuvor aufgeführten Funktionen gemeinsam genutztes Wissen beschrieben. Dieses Wissen kann im Modell *MAPE-K* im einfachsten Fall durch gemeinsam genutzte Daten vorhanden sein. *Luckey* verwendet die Partition *Adaptation Context* neben dem zuvor genannten Fall insbesondere zur Beschreibung der Systemarchitektur in Hinsicht auf anzupassende (Teil-)Komponenten. Im Rahmen des Verhaltens der beiden anderen Partitionen *Monitor* und *Adaptation* wird auf Eigenschaften dieser Komponenten zurückgegriffen.

Die Inhalte der Partitionen *Monitor* und *Adaptation* werden im Rahmen des *ACM* beschrieben. Dabei wird auf zuvor beschriebene Inhalte der Partition *Adaptation Context* in Form des *AVM* zurückgegriffen. So lassen sich Anpassungen hinsichtlich relevanter Ausschnitte des betrachteten Systems und seiner Umgebung fokussiert beschreiben.

In den nachfolgenden Abschnitten 2.4.2 und 2.4.3 werden für die Sprache *Adapt Case Modeling Language* zunächst die konkrete Syntax anhand eines Beispiels und anschließend die abstrakte Syntax vorgestellt. Die beschriebenen Inhalte bilden dabei die Basis für die in dieser Arbeit vorgestellten Sprache *ACML4BPM* (siehe Kapitel 4).

2.4.2 Konkrete Syntax der Sprache ACML am Beispiel

Die konkrete Syntax der Sprache *ACML* wird in diesem Abschnitt in Anlehnung an ein Beispiel beschrieben. Teile des Beispiels basieren dabei auf den durch *Luckey* [LE13] beschriebenen Anwendungsfall. Dabei wird die Steuerung von verschiedenen Modi, in denen ein Server Dienste ausführt, beschrieben. Dabei soll je nach Auslastung des Servers entweder in einen effizienten (*eco*) oder einen performanten (*performance*) Modus gewechselt werden können. Ferner soll aber auch der manuelle Wechsel in den performanten Modus ermöglicht werden. Analog zum vorherigen Abschnitt wird für das Beispiel zunächst auf das *AVM* und anschließend auf das *ACM* eingegangen.

In Abbildung 2-14 ist ein Beispiel für ein *AVM* dargestellt. Das Beispiel besteht demnach aus der Systemkomponente *Server* und der Umgebungskomponente *HumanMachineInterface*. Die Auslastung des Servers kann durch die Sensorschnittstelle *ServerLoad* bereitgestellt werden. Um den Status des Servers zu ändern, wird die dargestellte Effektorschnittstelle *ServerMode* verwendet. Die Notwendigkeit für einen manuellen Wechsel wird durch das dargestellte Signal *ManualPerformanceCall* angedeutet.

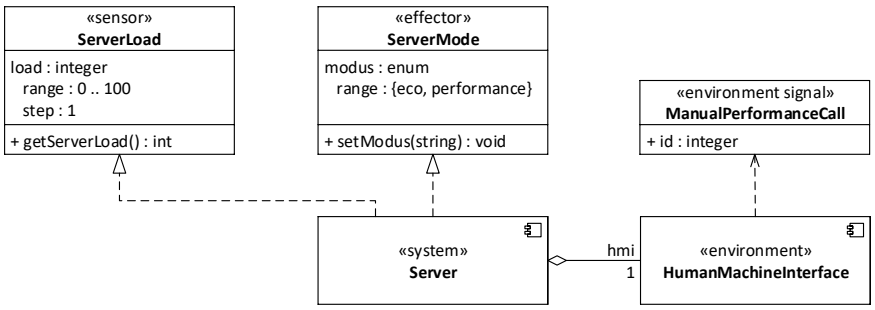


Abbildung 2-14: Konkrete Syntax der Sprache ACML am Beispiel eines AVM (nach Luckey [LE13])

Bei der Verwendung von einer gängigen Gestaltung durch U*ML Komponentendiagramme* spricht *Luckey* auch von einer hohen Konsistenz zu bestehenden Methoden. Dies lässt sich dadurch begründen, dass der Einstieg in die Gestaltung von Eigenschaften des Systems und seiner Umgebung für Anwender, die *UML* bereits erfolgreiche einsetzen, eine geringe Hürde darstellt. Hierdurch lässt sich die Erstellung des *AVM* in bereits bestehende Entwicklungskontexte besonders leicht erreichen.

Die Erstellung des *ACM* sieht die beiden aufeinanderfolgenden Aktivitäten der *High-Level-Gestaltung* und der *Low-Level-Gestaltung* vor. Eine integrierte Sicht der Ergebnisse dieser Aktivitäten ist in Abbildung 2-14 dargestellt.

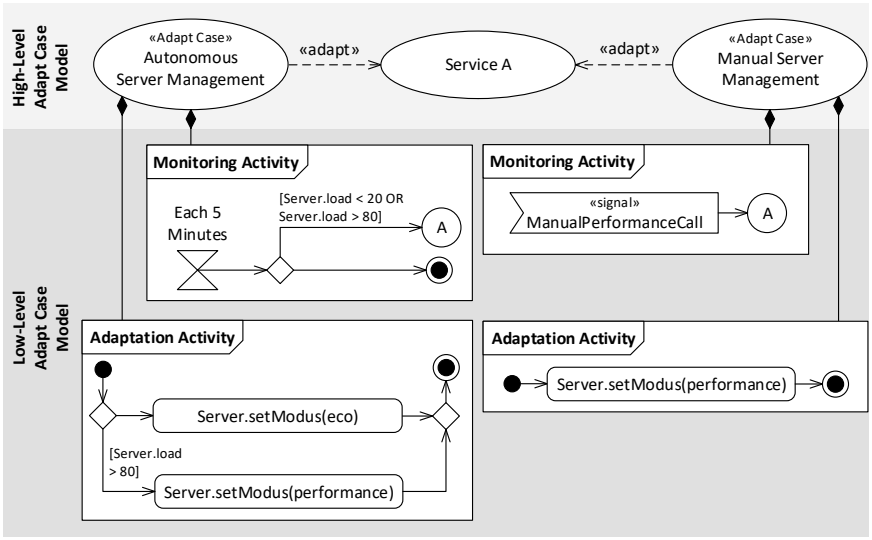


Abbildung 2-15: *Konkrete Syntax der Sprache ACML am Beispiel eines ACM (nach Luckey [LE13])*

Das *High-Level ACM* ist hier im oberen Bereich gezeigt. *Luckey* sieht bei dieser Erstellung des *ACM* die Verwendung von erweiterten *UML Use Case-Diagrammen* vor. So wird das Konzept des Anwendungsfalls (*Adapt Case*) in Anlehnung an *UML Use Cases* dargestellt. Zwischen Funktionen der Anwendungslogik (hier: *Service A*) und Funktionen zur Anpassung (hier: *Adapt Cases*) können Assoziationen eingesetzt werden. Die Aufschrift *«adapts»* einer solchen Assoziation beschreibt, dass ein ausgehender *Adapt Case* eine andere Funktion anpasst. Dabei kann das Ziel einer solchen Assoziation sowohl ein weiterer *Adapt Case* als auch eine Funktion der Anwendungslogik in Form eines *UML Use Case* sein.

Das *Low-Level ACM* ist im unteren Bereich dargestellt. Es enthält dabei Verfeinerungen der beiden Anpassungsfälle in Form von Beobachtungs- und Anpassungsaktivitäten. Das Verhalten des *Autonomous Server Managements* ist dabei so verfeinert worden, dass durch die dargestellte Beobachtungsaktivität zeitgesteuert alle 5 Minuten die Auslastung des Servers geprüft wird. Wird eine zu geringe (*<20%*) oder zu hohe (*>80%*) Auslastung detektiert, so wird die zugehörige Anpassungsaktivität aufgerufen. Hierfür bedient sich *Luckey* einer sogenannten *Call Adaptation Activity*, welche durch einen Kreis mit der Aufschrift *A* dargestellt wird. Die Anpassungsaktivität enthält dabei Verhalten, welches je nach Auslastung einen Wechsel des Modus, in dem der Server operiert, hervorruft. Die Beobachtungs- und Anpassungsaktivität des Anpassungsfalls *Manual Server Management* ist im rechten Bereich dargestellt. Hier wird im Rahmen der Beobachtungsaktivität das durch die Umgebungskomponente *HumanMachineInterface* bereit-

gestellte Signal *ManualPerformanceCall* als Startsignal eingesetzt. Die zugehörige Anpassungsaktivität passt als Folge den Modus des Servers an.

Es sei an dieser Stelle darauf hingewiesen, dass in dem zuvor beschriebenen Verhalten auf Elemente der Sensor- und Effektorschnittstellen aus dem *AVM* zurückgegriffen wird. Dies stellt ein Beispiel für die Verwendung von Elementen des *AVM* im Rahmen der Gestaltung des *ACM* dar. Umfangreichere Anwendungen aus der Praxis können so auch eine höhere Anzahl an Elementen im Rahmen des *AVM* vorsehen, die dann Verwendung in der Gestaltung des Verhaltens durch ein *ACM* finden. Dabei können im Rahmen des *ACM* auch mehrere Anpassungsaktivitäten im Rahmen der Gestaltung eines einzelnen *Adapt Case* vorkommen, die je nach Situation entsprechend alternatives Verhalten zum Zweck der Anpassung bereitstellen.

2.4.3 Abstrakte Syntax der Sprache ACML

In diesem Abschnitt werden Ausschnitte der abstrakten Syntax der Sprache *ACML* für die beiden Modelle *ACM* und *AVM* vorgestellt. In Abbildung 2-16 sind wesentliche Konzepte des *AVM* in Form eines Metamodells dargestellt. Dabei sind Konzepte der Sprache *ACML* farblich unterlegt hervorgehoben. Nicht hervorgehobene Konzepte stammen aus dem Metamodell der *UML*.

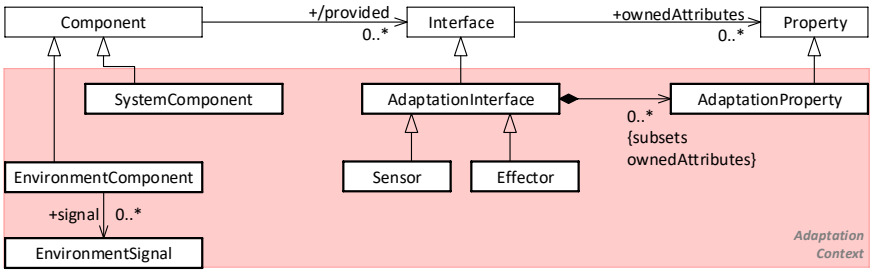


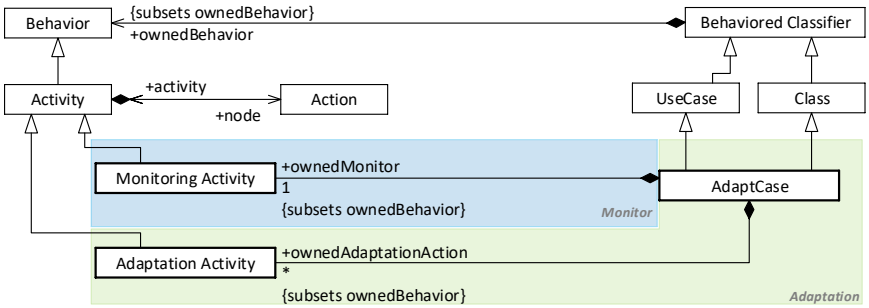
Abbildung 2-16: Auszug aus dem AVM-Metamodell (nach Luckey [LE13])

Durch die Partition *Adaptation Context* werden wesentliche strukturelle Informationen über den Kontext der Anpassung (hier: *Adaptation Context*) gestaltet. Das von *Luckey* beschriebene Konzept basiert dabei vornehmlich auf Konzepten der *UML Komponentendiagramme*, um die Struktur des Systems und seiner Umgebung zu beschreiben. Man spricht dabei im Rahmen der Gestaltung von selbst-adaptiven Systemen von den beiden Konzepten *System* und *Environment*, für die jeweils ein neuer Typ von Komponenten eingeführt wird. Dabei können Komponenten des Typs *EnvironmentComponent* eine Reihe von Signalen (*EvironmentSignal*) senden, die zur Auslösung einer möglichen Anpassung eingesetzt werden können.

Ein System und seine Umgebung können aus einer beliebigen Anzahl an Komponenten bestehen, die jeweils zwei Typen von Schnittstellen für den kontrollierten lesenden und schreibenden Zugriff auf Inhalte der zuvor eingeführten Typen von Komponenten ermöglichen. Im Kontext der selbst-adaptiven Systeme spricht man dabei vorrangig von Sensor- und Effektorschnittstellen. Dabei können diese Schnittstellen Operationen oder Eigenschaften (*AdaptationProperty*) enthalten, die im Rahmen der Gestaltung des *ACM* eingesetzt werden können.

Das Metamodell des *ACM* ist in Abbildung 2-17 dargestellt. Dabei sind abermals die Konzepte der Sprache *ACML* farblich unterlegt hervorgehoben. Nicht hervorgehobene Konzepte stammen aus dem Metamodell der *UML* und stellen im Wesentlichen Elemente der *UML Aktivitätsdiagramme* und *UML Use Case-Diagramme* dar.

Abbildung 2-17: *Auszug aus dem ACM-Metamodell (nach Luckey [LE13])*



Wesentliches Konzept des Ansatzes ist das Konzept des Anpassungsfalls (hier: *AdaptCase*). Bei einem Anpassungsfall handelt es sich um einen spezialisierten Typ eines *UML Use Case*. Erstmals wurde dieses Konzept durch *Luckey* [Luc+11] vorgestellt. Die Verwendung des Konzepts des Anpassungsfalls ist für die Beschreibung von Funktionen zur Anpassung von Eigenschaften der System- oder Umgebungskomponenten gedacht. Eine solche Anpassung lässt sich hinsichtlich der beiden Partitionen *Monitor* und *Adaptation* weiter unterteilen. So wird durch das Konzept der Beobachtungsaktivität (hier: *MonitoringActivity*) Verhalten beschrieben, das zur Aggregation und Analyse von Daten der System- und Umgebungseigenschaften eingesetzt wird. Tritt die Notwendigkeit für eine Anpassung auf, so kann das Konzept der Anpassungsaktivität (hier: *AdaptationActivity*) genutzt werden, um entsprechendes Verhalten zu beschreiben. Für die Beschreibung des Verhaltens der Beobachtungs- als auch der Anpassungsaktivität wird vorrangig auf existierende Elemente der *UML Aktivitätsdiagramme* zurückgegriffen.

2.4.4 Integration in einen Entwicklungsprozess

Die Verwendung der Sprache *ACML* schließt die Erstellung von verschiedenen Artefakten auf unterschiedlichen Ebenen entlang eines Softwareentwicklungsprozesses (*SDP*) mit ein. Dabei steht stets die Trennung von Aspekten der Anpassungs- von der Anwendungslogik im Vordergrund. In enger Anlehnung an [LE13] sind in Abbildung 2-18 Teile des Ansatzes *Adapt Cases* entlang relevanter Phasen eines *SDP* gezeigt. Auf den Zusammenhang gezeigter Phasen und Artefakte wird nachfolgend eingegangen.

Dabei steht die Spezifikation von Aspekten der Anwendungslogik im Vordergrund, die in Abbildung 2-18 einem beispielhaften Verlauf folgt. So wird zunächst die Beschreibung von Anforderungen an das System vorgenommen. Üblich ist es zudem, dass ein Domänenmodell erstellt wird, das Kernkonzepte der jeweiligen Anwendungsdomäne beschreibt und in Relation zueinander setzt. Hierfür können bspw. *UML Klassendiagramme* oder *Mind-Maps* eingesetzt werden.

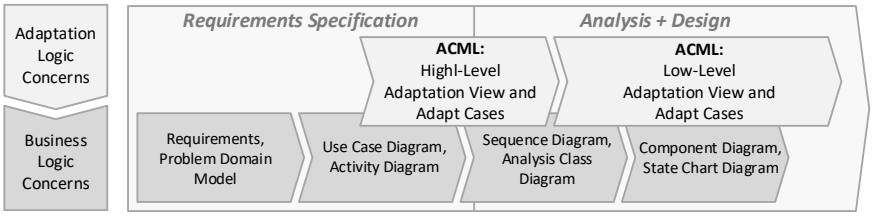


Abbildung 2-18: Software Development Process unter Verwendung der Sprache ACML (nach Luckey [LE13])

Aufbauend können erste Funktionen durch *UML Use Case-Diagramme* und *UML Aktivitätsdiagramme* beschrieben werden, wobei diese stets die zuvor beschriebenen Anforderungen adressieren. In einem weiteren Schritt können die Funktionen und das Verhalten anschließend durch *UML Sequenzdiagramme* und *UML Analyseklassendiagramme* verfeinert werden. Im letzten dargestellten Schritt vor der Implementierung wird die Architektur durch *UML Komponentendiagramme* beschrieben. Dabei wird häufig auch der Lebenszyklus von wichtigen Objekten durch *UML Zustandsübergangsdiagramme* beschrieben.

Die Gestaltung von Aspekten der Anpassungslogik kann komplementär zu den letzten drei beschriebenen Schritten erfolgen. Die Basis bildet hierbei das *AVM*. Es enthält einen Ausschnitt des zuvor beschriebenen Systems, der ausschließlich für die Anpassungslogik relevante Aspekte beschreibt. Wie bereits beschrieben, handelt es sich hierbei z.B. um relevante Komponenten, Sensor- und Effektorschnittstellen oder um mögliche Signale zur Auslösung eines Anpassungsfalls. Nachfolgend kann mit der *High-Level-Gestaltung* von *Adapt Cases* begonnen werden, die mit beste-

henden *Use Cases* der Anwendungslogik in Relation stehen. Im zweiten Schritt der Gestaltung von Aspekten der Anpassungslogik kann es notwendig sein, das *AVM* anzupassen. *Luckey* begründet dies damit, dass zum Zeitpunkt des Erstentwurfs des *AVM* einzelne Funktionen oftmals noch nicht adäquat auf Komponenten verteilt sein können. So entsteht erst im weiteren Verlauf das Wissen, das eine adäquatere Gestaltung von System- und Umgebungskomponenten sowie von einer Verteilung von Funktionen ermöglicht. Auf Basis des *Low-Level-AVM* kann anschließend das zugehörige *Low-Level-ACM* erstellt werden, in dem die *Adapt Cases* durch Beobachtungs- und Anpassungsaktivitäten verfeinert werden können.

Durch die konsequente Nutzung bestehender Gestaltungselemente und -techniken der *UML* sind die durch die *ACML* erstellten Artefakte in verschiedene Phasen und Schritten des *SDP* stets mit den Artefakten der Anwendungslogik integrierbar. Hierdurch wird die Verwendung der Sprache *ACML* für *UML*-erfahrende Nutzer vereinfacht und vor allem auch anwendbar.

Kapitel 3

Verwandte Arbeiten

In diesem Kapitel werden unterschiedliche wissenschaftliche Ansätze vorgestellt, die als verwandte Arbeiten betrachtet werden können. Dabei kann Flexibilität in Prozessen durch verschiedene Arten erreicht werden, sodass sich auf insgesamt drei Felder bezogen wird. Zunächst werden in Abschnitt 3.1 ausgesuchte Arbeiten vorgestellt, die Flexibilität und Anpassbarkeit von Prozessen für diverse Zwecke unterstützen. Anschließend wird in Abschnitt 3.2 auf Ansätze von flexiblen und anpassbaren Prozessen im Kontext von *Industrial Internet-of-Things (IIoT)* und Industrie 4.0-Anwendungen eingegangen. Es folgen in Abschnitt 3.3 Arbeiten, in denen Konzepte des *Autonomic Computing* ebenfalls auf Prozesse angewendet werden.

3.1 Flexible und anpassbare Prozesse

Flexibilität in Prozessen ist in der Domäne *BPM* ein weitreichend erforschtes Feld. Daher wurden bereits in Abschnitt 2.2 grundlegende Ansätze für flexible und anpassbare Prozesse vorgestellt, da sie als Grundlage für die Ausarbeitung dieser Arbeit angesehen werden. Nachfolgend vorgestellte Arbeiten haben sich anschließend mit konzeptionellen Weiterentwicklungen oder technischen Realisierungen dieser Grundlagen beschäftigt.

Eine Arbeit, die sich sowohl mit Entwurfsmustern als auch deren technischen Realisierung beschäftigt hat, wurde durch *Döhring et. al* [DZK11] vorgestellt. Dabei wird die Sprache *BPMN2.0* um adaptive Workflow-Segmente erweitert, an denen zur Laufzeit vordefinierte Anpassungsmuster (engl. *Adaptation Pattern*) gebunden werden können. In dem Ansatz werden Regeln in der Form *Wenn-Dann-Anders* für die Auswahl eines solchen Anpassungsmusters eingesetzt. Derartige Segmente sind vergleichbar mit den für den Typ *Late Selection* eingeführten *Platzhaltern* (siehe

Abschnitt 2.2.3). Die Autoren stellen ebenfalls eine prototypische Implementierung vor, die auf der Verwendung der *Drools Rule-Engine* und *Workflow-Engine*[1] beruht. Die Anwendung des Ansatzes kann insbesondere dann eingesetzt werden, wenn mehrere Varianten eines Prozesses verwaltet werden sollten. Alternative Ansätze aus diesem Bereich sind durch [Ayo+16] oder im spezielleren Bereich für die Versionierung von Prozessen [Sai+15] gegeben.

In der Arbeit von *Milanovic et. al* [MG09] wurde die Sprache *BPMN2.0* um diverse Eigenschaften erweitert, sodass ein gesteigerter Grad an Flexibilität in der Gestaltung ermöglicht wird. Der Ansatz fokussiert dabei die Gestaltung von Entscheidungen durch Regeln. Als Resultat wird das neue Metamodell *rBPMN* vorgestellt, über das eine Komposition von Prozessen (*BPMN*) und Regeln (*R2ML*) durchgeführt wird. Hierzu führen die Autoren ein erweitertes regelbasiertes Gateway (*RuleGateway*) ein, das mit höherwertigen Regeln versehen werden kann. Der Ansatz fokussiert damit vorwiegend den Flexibilitätsaspekt *Flexibility-by Design*.

Weitere Arbeiten, die insbesondere im Kontext der eingeführten Taxonomien für Flexibilität in Prozessen entstanden sind, werden im Rahmen der Evaluation aufgeführt und mit Fähigkeiten des eigenen Ansatzes verglichen. Beispiele für derartige Arbeiten sind durch *ADEPT1* [RRD03], *YAWL* [AT05; Ada+06; Ada+07], *FLOWer* [AWG05] oder *Declare* [PA06; Pes+07] gegeben.

3.2 Flexible und anpassbare Prozesse im IIoT

Existierende Ansätze unterstützen die Gestaltung von flexiblen und anpassbaren Prozessen bereits auf verschiedenen Ebenen in einem einzelnen oder zwischen mehreren Unternehmen. Ein verbreitetes Vorgehen stellt dabei die Erstellung von Erweiterungen einer Sprache dar, wie z.B. der *BPMN2.0*. Hierdurch können neue und relevante Konzepte in der Gestaltung derartig berücksichtigt werden, dass die enthaltene Flexibilität als umfassender betrachtet werden kann. Einige Beispiele im Kontext von *Industrial Internet-of-Things (IIoT)* bzw. von Industrie 4.0-Anwendungen werden nachfolgend erörtert.

So stellen *Meyer et. al* [MRM13; MRH15] einen Ansatz vor, in dem grundlegende Konzepte des *Internet-of-Things* auf der Sprachebene zur Gestaltung von Prozessen eingeführt werden. Der Ansatz basiert auf dem be-

[1] `http://www.drools.org/` Letzter Zugriff: 11.12.2018

reits durch *Sperner et. al* [SMM11] eingeführten Konzept der entitätsbasierten Gestaltung von Prozessen. Das Resultat ist eine Erweiterung der Sprache *BPMN2.0*, durch die Eigenschaften von IoT-Geräten gestaltet werden können. Zur Integration derartiger Geräte wird in dem Ansatz ein neuer Typ von Ressourcen eingeführt, der klassische Komponenten, wie z.B. Sensoren und Aktuatoren auf der Ebene der Prozesse, repräsentiert. Durch das zugehörige Metamodell kann die Gestaltung von IoT-Geräten durchgeführt werden. Als Ergänzung zur Integration verwenden *Meyer et. al* ontologiebasierte Techniken zur Spezifikation der Semantik von eingesetzten Ressourcen in den gestalteten Prozessmodellen.

Zukünftige Prozesse sollen auf Veränderungen von einer Vielzahl von vernetzten Ressourcen eingehen können. Hierdurch wird das Ziel verfolgt, die Prozesse und ihre Ausführung möglichst flexibel gestalten zu können. Dabei kann diese Art von Flexibilität nur dann erreicht werden, wenn bereits auf der Ebene der Sprache spezifische Eigenschaften auch gestaltet werden können.

Ein weiterer Ansatz, der auf der Metamodellierung beruht und zum Zweck der Verwaltung von Ressourcen eingesetzt werden kann, ist durch *Bocciarelli et. al* [Boc+17] vorgestellt worden. Der durch *Bocciarelli et. al* gegebene Ansatz führt ebenfalls einen neuen Typ von Ressourcen in der Sprache *BPMN2.0* ein. Durch diesen Typ sind Eigenschaften der Umgebung eines Prozesses mit einer Fokussierung auf *Cyber-Physische Systeme (CPS)* und *Smart Factories* gestaltbar. Beispiele für derartige Ressourcen sind durch menschliche Akteure, der eingesetzten Software oder Hardware gegeben. Die Erweiterung ist Teil eines modellgetriebenen Rahmenwerks, das neben der Gestaltung ebenso die Analyse von Leistung (engl. *Performance*) und von Zuverlässigkeit (engl. *Reliability*) von Prozessen und beteiligten Ressourcen unterstützt. Der Ansatz greift auf verschiedene Vorarbeiten der Autoren zurück [Boc+14a; Boc+14b; BDP14; Boc+16].

In der vorliegenden Arbeit wurde sich auch mit der Gestaltung unterschiedlicher Auslöser (hier: *Ereignis*) für eine Anpassung beschäftigt. Eine weitere Arbeit, die sich mit der Integration von unterschiedlichen Ereignissen beschäftigt, ist durch *Mandal et. al* [MHW17] gegeben. Es wird ein Rahmenwerk vorgestellt, das sich mit einer erweiterten Ereignisverarbeitung auseinandersetzt. Die Autoren sprechen von sogenannten real-weltlichen Ereignissen, die in oder im Kontext von Prozessen vorkommen können. Dabei wurden spezielle Techniken entwickelt, um auf einer technischen Ebene der Realisierung auf aufkommende Ereignisse adäquat reagieren und erforderliche Maßnahmen verarbeiten zu können. So ist eine bidirek-

tionale Reaktion bzw. Steuerung möglich. Hierdurch können Prozesse auf aufkommende Ereignisse aus ihrer Umgebung adäquat reagieren oder das Verhalten von IoT-Geräten steuern.

Zor et. al [ZLS11] stellen eine Erweiterung der Sprache *BPMN2.0* vor, welche weitere Konzepte der Domäne der industriellen Fertigung abdeckt. Dabei werden ebenfalls weitere Typen von Ressourcen eingeführt, die zur Gestaltung von z.B. Maschinen, Werkzeugen oder Teilen eines Produktes eingesetzt werden können. Hierdurch können Prozesse unter Berücksichtigung von Eigenschaften dieser Ressourcen gestaltet werden. Ferner werden weitere Konzepte eingeführt, wie z.B. spezielle Gateways, mit denen der Materialfluss beschrieben werden kann. Dies ist sinnvoll, wenn die Transformation von Material in Abhängigkeit zur Ein- und Ausgabe einer Aktivität beschrieben werden soll. Elemente des Materialflusses können dabei als eine zusätzliche Perspektive von Prozessen verstanden werden, die bspw. über die Möglichkeiten der Perspektive *Information* hinausgeht (siehe Abschnitt 2.3.2).

Durch *Graja et. al* [Gra+16] wird eine weitere Erweiterung der Sprache *BPMN2.0* vorgestellt, mit der die Gestaltung von Cyber-physischen Prozessen adressiert wird. Die Erweiterung *BPMN4CPS* trennt das in den Prozessen beschriebene Verhalten hinsichtlich unterschiedlicher Logiken auf. So wird in *Cyber Tasks* und *Physical Tasks* unterschieden. Organisatorische Unterschiede können durch die Verwendung von verschiedenen Pools dargestellt werden. Hierdurch können sowohl *Cyber-*, *Physical-* und *Controlling-spezifische* Prozesse unterschieden werden. Durch diese Trennung unterstützt dieser Ansatz ebenso das *Separation-of-Concerns (SoC)*. Jedoch steht hierbei die Differenzierung in Bezug zu den genannten Kernkonzepten der Domäne *IIoT* hinsichtlich der zu gestaltenden Prozesse und nicht die Trennung der Anpassungs- von der Anwendungslogik im Fokus.

3.3 Selbst-adaptive Prozesse

Neben den zuvor beschriebenen Ansätzen, in denen vorwiegend eine ressourcenzentrierte Perspektive gewählt worden ist, existieren aber auch Ansätze, die sich eher aus der Perspektive von selbst-adaptiven Systemen der Gestaltung von Prozessen nähern.

Ein Ansatz, der auf Konzepte aus dem Bereich des *Autonomic Computing* [KC03] zurückgreift, ist durch [Sei+15; Sei+16; SHS18] vorgestellt worden. *Seiger et. al* beschreiben ein Rahmenwerk für die Gestaltung und Ausführung von Prozessen in intelligenten Cyber-physischen Umgebungen. Da-

bei wird im Rahmen von Anpassungen das Paradigma der *MAPE-K Kontrollschleife* eingesetzt. Derartige Anpassungen werden in dem Ansatz dabei hinsichtlich der Einbindung von IT-Diensten durchgeführt. Eine Anpassung des Kontroll- oder Datenflusses ist nicht vorgesehen. Der Ansatz verfügt darüber hinaus über Konzepte für die weitere Analyse zur Identifikation von Inkonsistenzen zwischen Unterschieden von aufkommenden und erwarteten Effekten einer ausgeführten Anpassung von Prozessen.

Marrella und Mecella [MM17] stellen einen Ansatz zur Verwaltung von adaptiven Cyber-physischen Prozessen vor. Durch diesen Ansatz können Prozesse automatisiert in der Phase *Ausführung* angepasst werden. Dabei werden Techniken aus dem Bereich der künstlichen Intelligenz für die konkrete Entscheidungsfindung eingesetzt. Der Ansatz ist in der Lage, Unterschiede zwischen dem realweltlichen Ergebnis einer Anpassung und einer zuvor erwarteten Anpassung zu ermitteln. Hierdurch ist eine Interaktion durch Nutzer oder Domänenexperten nicht mehr nötig. Der Ansatz geht dabei verstärkt auf den Aspekt der automatisierten und unüberwachten Anpassung ein. Dabei werden konkrete Fähigkeiten zur Anpassung von Prozessen, wie z.B. Kontroll- oder Datenfluss, nicht detailliert beschrieben. Der Ansatz wird daher als komplementär betrachtet, da er eine fortgeschrittene Technik für Analyse- und Entscheidungsfindungsfunktionalität beschreibt.

Teil II

Lösungskonzept

Kapitel **4**

Eine Sprache zur Gestaltung von anpassbaren Prozessen

In diesem Kapitel wird eine Sprache für die Gestaltung von anpassbaren Prozessen vorgestellt. Zunächst wird das Lösungskonzept für diese Sprache in Abschnitt 4.1 beschrieben. Dies umfasst neben einer Skizze der Lösung auch grundlegende Fragestellungen, die die Basis für die weiteren erforderlichen Bestandteile der entwickelten Sprache bilden. Das Lösungskonzept basiert auf dem durch *Luckey* [Luc+11] vorgestellten Ansatz *Adapt Cases* (siehe Abschnitt 2.4). Es sieht insgesamt zwei domänenspezifische Teilsprachen vor. Diese werden in Bezug zum *ACM* in Abschnitt 4.2 und in Bezug zum *AVM* in Abschnitt 4.3 detailliert vorgestellt. Abschließend wird in Abschnitt 4.4 eine Zusammenfassung sowie eine Diskussion hinsichtlich der in Abschnitt 4.1 vorgestellten Fragestellungen gegeben.

4.1 Übersicht

In diesem Abschnitt wird eine Übersicht über die in dieser Arbeit vorgestellte Sprache für die Gestaltung von anpassbaren Prozessen gegeben. Die Sprache wird *Adapt Case Modeling Language 4 BPM (ACML4BPM)* genannt. Es handelt sich bei dieser Sprache um eine domänenspezifische Redefinition der durch *Luckey* [Luc+11] vorgestellten Sprache *ACML* (siehe Abschnitt 2.4). Dabei sieht das Lösungskonzept die Integration von verschiedenem domänenspezifischem Wissen vor. Für die Erarbeitung dieses Wissens und die anschließende Integration wurde sich an verschiedenen Fragestellungen orientiert, die sich auf Kernkonzepte der ursprünglichen Sprache *ACML* beziehen. Nachfolgend wird eine kurze Übersicht über diese Fragestellungen gegeben.

Fragestellung 1 *Was sind relevante System- und Umgebungskomponenten in der Domäne BPM?*

Fragestellung 2 *Wie kann auf Informationen innerhalb dieser System- und Umgebungskomponenten zugegriffen werden?*

Fragestellung 3 *Wie lassen sich Eigenschaften der System- und Umgebungskomponenten in der Domäne BPM adäquat anpassen?*

Fragestellung 4 *Was sind mögliche Ereignisse, die die Notwendigkeit einer Anpassung von Prozessen andeuten?*

Eine schematische Darstellung der zu den Fragestellungen zugehörigen Lösungsteile für die Sprache *ACML4BPM* ist in Abbildung 4-1 gegeben. Die Sprache *ACML4BPM* sieht dabei in Anlehnung an die von *Luckey* [LE13] vorgestellte Sprache *Adapt Case Modeling Language (ACML)* die Konzeptionierung von zwei Teilsprachen zur Beschreibung eines *Adapt Case Model 4 BPM (ACM4BPM)* und eines *Adaptation View Model 4 BPM (AVM4BPM)* vor. Die beiden Teilsprachen zur Erstellung des *ACM4BPM* und des *AVM4BPM* werden in den nachfolgenden Abschnitten 4.2 und 4.3 detailliert beschrieben.

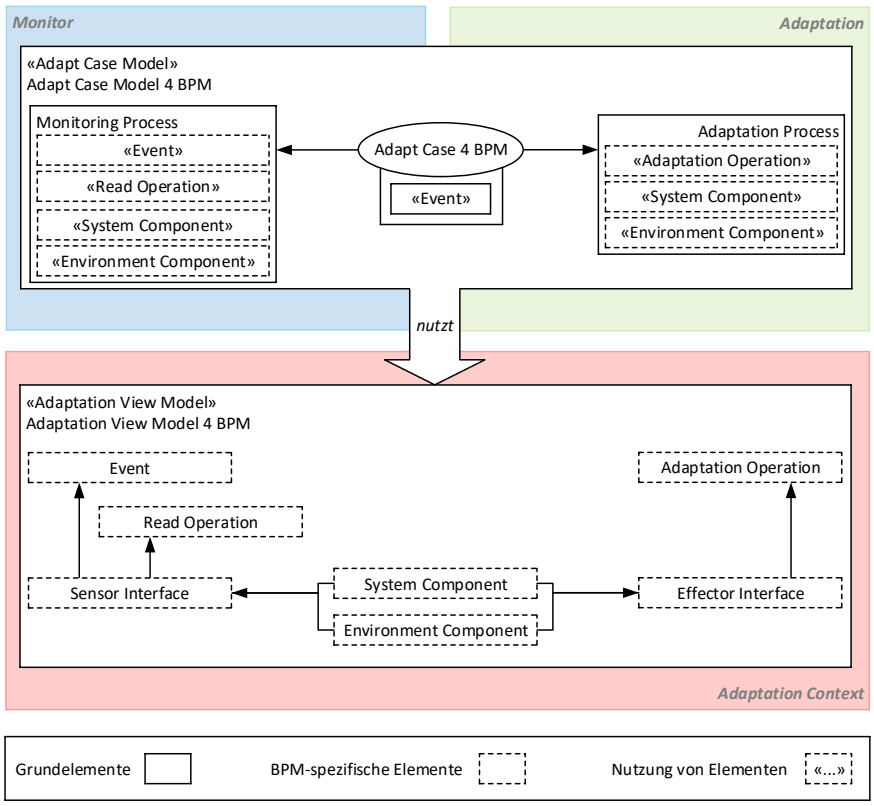


Abbildung 4-1: Konzept der Sprache Adapt Case Modeling Language 4 BPM

Beschreibung der Struktur des Systems

Das *AVM4BPM* enthält identifizierte System- und Umgebungskomponenten der Domäne *BPM*. Jede dieser System- und Umgebungskomponenten kann die beiden Typen von Schnittstellen *Sensor Interface* und *Effector Interface* anbieten. Diese Schnittstellen können spezifische Operationen zum Lesen und zur Anpassung von Eigenschaften der Komponenten unterstützen. So beschreibt die Schnittstelle *Sensor Interface* zum einen Ereignisse (*Event*), die für mögliche Anpassungen von Prozessen als Auslöser vorkommen können. Ferner können Operationen *(Read Operation)* für den lesenden Zugriff von Eigenschaften der beiden Typen von Komponenten beschrieben werden. Die Schnittstelle *Effector Interface* enthält Operationen (*Adaptation Operation*), die zur Anpassung von Eigenschaften der System- und Umgebungskomponenten eingesetzt werden können. Die in der Sprache *ACML4BPM* vorgesehenen domänenspezifischen Operationen betreffen insbesondere die Anpassung von Eigenschaften der betroffenen Prozesse.

Beschreibung von Verhalten des Systems

Durch die Teilsprache *ACM4BPM* können Anpassungsfälle (engl. *Adapt Cases*) beschrieben werden. Das zugehörige domänenspezifische Konzept wird *Adapt Case 4 BPM (AC4BPM)* genannt. Bei der Gestaltung von Anpassungsfällen werden die zuvor beschriebenen Sprachelemente des *AVM4PBM* eingesetzt. So können einem Anpassungsfall die im Rahmen des *AVM4BPM* spezifizierten Ereignisse zugeordnet werden. Hierdurch wird ausgedrückt, dass der Anpassungsfall beim Vorkommen des betreffenden Ereignisses angewendet wird. Das Verhalten eines Anpassungsfalls kann durch spezifischeres Verhalten in Form eines *Monitoring Process* und *Adaptation Process* verfeinert werden.

Die Anwendung eines Anpassungsfalls sieht dabei zunächst den Aufruf eines Beobachtungsprozesses *(Monitoring Process)* vor. Durch einen *Monitoring Process* kann das Verhalten zur Beobachtung der System- und Umgebungskomponenten beschrieben werden. Dabei werden Bedingungen hinsichtlich erforderlicher Eigenschaften der genannten Komponenten ausgewertet und, falls notwendig, ein entsprechender *Adaptation Process* gestartet, welcher eine spezifizierte Anpassung durchführt.

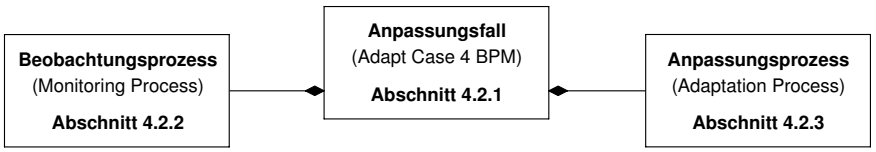
Ein *Adaptation Process* beschreibt das Verhalten zur Anpassung von Eigenschaften der System- und Umgebungskomponenten. Er kann durch einen *Monitoring Process* gestartet werden. Dabei werden im Rahmen des in ihm spezifizierten Verhaltens Operationen zur Anpassung angewendet, die durch die *Effector Interfaces* der System- und Umgebungskomponenten bereitgestellt werden.

4.2 Adapt Case Model 4 BPM

Durch das *Adapt Case Model 4 BPM (ACM4BPM)* wird das Verhalten hinsichtlich der Fähigkeit zur Anpassung beschrieben. Hierbei liegt der Fokus auf der Beschreibung von Anpassungsfällen, die im Rahmen der Gestaltung von anpassbaren Prozessen notwendig sind. In den folgenden Abschnitten werden die wesentlichen Konzepte für die Domäne *BPM* eingeführt.

In Abbildung 4-2 ist eine Übersicht über diese Konzepte dargestellt. Dabei wird zunächst in Abschnitt 4.2.1 das Konzept des Anpassungsfalls (*Adapt Case 4 BPM*) beschrieben. Nachfolgend werden die Konzepte des Beobachtungsprozesses (*Monitoring Process*) (siehe Abschnitt 4.2.2) sowie des Anpassungsprozesses (*Adaptation Prozess*) (siehe Abschnitt 4.2.3) vorgestellt.

Abbildung 4-2: Inhalte des Adapt Case Model 4 BPM



4.2.1 Adapt Case 4 BPM

Ein wesentliches Konzept für die Gestaltung von anpassbaren Prozessen stellt das Konzept des *Adapt Case 4 BPM (AC4BPM)* dar, das in diesem Abschnitt beschrieben wird. Ein *AC4BPM* stellt eine Erweiterung des durch *Luckey* [Luc+11] vorgestellten Konzepts des *Adapt Case* dar (siehe auch Abschnitt 2.4). Die Verwendung dieses Konzepts ermöglicht eine getrennte Beschreibung der Anwendungs- und Anpassungslogik in einer frühen Phase der Gestaltung. Mit einem *AC4BPM* lässt sich die Funktion zur Anpassung von Prozessen fallbasiert und getrennt von der Anwendungslogik beschreiben. Ein *AC4BPM* stellt in diesem Bezug die konsequente und domänenspezifische Weiterentwicklung eines *Adapt Case* für die Domäne *BPM* und damit für die Beschreibung von Anpassungen von Prozessen und ihrer Umgebung dar.

Eine konzeptionelle Darstellung verschiedener Verwendungsweisen von *AC4BPM* mit Bezug zu einer Funktion der Anwendungslogik ist in Abbildung 4-3 dargestellt. So besteht die hier durch ein *Use Case-Diagramm* beschriebene Funktion eines Systems aus einem *Use Case* und verschiedenen *AC4BPM*. Es existieren die beiden Rollen *Actor A* und *Actor B*, die spezifische dargestellte Funktionen ausführen.

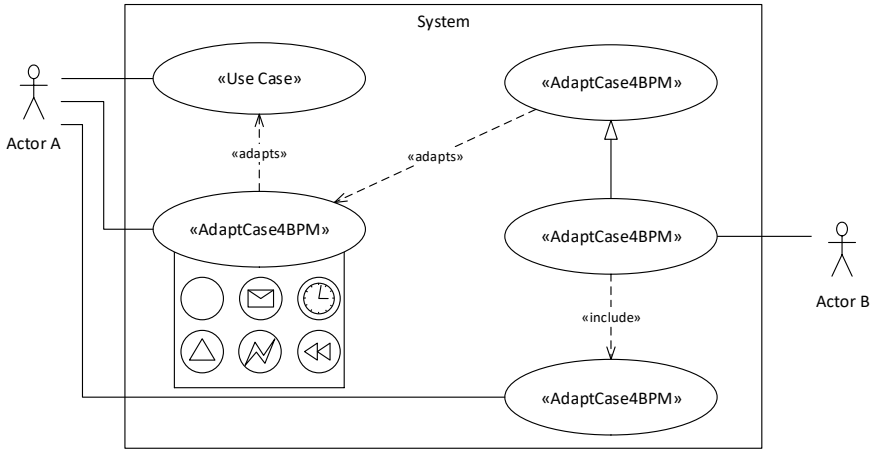


Abbildung 4-3: *Konzeptionelle Darstellung des Konzepts Adapt Case 4 BPM*

Ein *AC4BPM* wird in Anlehnung an *UML Use Cases* in Form einer Ellipse dargestellt. Sind in einer frühen Phase der Gestaltung bereits auslösende Ereignisse für die Anpassung an einen Prozess bekannt, so können verschiedene in Abschnitt 4.3.4 eingeführte Ereignisse der Domäne *BPM* mit einem *AC4BPM* assoziiert werden. Hierbei wurde sich für eine Darstellung entschieden, bei der mögliche Ereignisse im Rahmen eines Containers in Form eines Rechtecks als Teil des *AC4BPM* dargestellt werden. Sind Ereignisse hingegen noch nicht bekannt, wird der Container nicht visualisiert. Ereignisse können im weiteren Verlauf der Gestaltung ergänzt oder verfeinert werden. Ein *AC4BPM* kann ferner verschiedene Beziehungen zwischen weiteren *AC4BPM* und *Use Cases* haben.

«adapts»

Soll unter einer Bedingung, die durch einen *Use Case* gegebene Funktion erweitert werden, wird anstelle einer Beziehung mit der Bezeichnung *«extends»* eine Beziehung mit der Bezeichnung *«adapts»* verwendet. Die Darstellung dieser Beziehung unterscheidet sich außer in der Bezeichnung nicht zu der sonst gebräuchlichen Darstellung der Beziehung mit der Bezeichnung *«extends»*. Ein *AC4BPM* stellt selbst wieder eine Funktion dar, die unter bestimmten Bedingungen durch einen weiteren *AC4BPM* angepasst werden kann. Soll ein *AC4BPM* angepasst werden, so können weitere Beziehungen mit der Bezeichnung *«adapts»* verwendet werden, um weitere *AC4BPM* zu diesem Zweck miteinander in Verbindung zu setzen. Durch eine Beziehung mit der Bezeichnung *«adapts»* lassen sich somit Funktionen zur Anpassung an anderen Funktionen eines Systems referenzieren. Der Pfeil der Beziehung zeigt dabei stets auf die anzupassende Funktion.

inheritance

Durch eine *Inheritance*-Beziehung können Eigenschaften einer Anpassung an weitere Anpassungen in Form von *AC4BPM* vererbt werden. Ein erben-

der *AC4BPM* kann geerbte Eigenschaften überschreiben und neue beinhalten. Die Verwendung einer Beziehung mit der Bezeichnung *Inheritance* kann insbesondere dann sinnvoll sein, wenn sich mehrere *AC4BPM* Eigenschaften teilen oder diese verfeinern.

«include»

Die Beziehung mit der Bezeichnung *«include»* kann verwendet werden, wenn beschrieben werden soll, dass eine Funktion zur Anpassung weitere Funktionen zur Anpassung einschließt. Dabei wird die Funktion eines *AC4BPM* in der Funktion eines anderen *AC4BPM* eingeschlossen, von dem ausgehend die Beziehung mit der Bezeichnung *«include»* dargestellt wird.

Verfeinerung von Anpassungsfällen und abstrakte Syntax

Die durch einen *AC4BPM* gegebene Funktion zur Anpassung kann darüber hinaus auch verfeinert werden. Hierfür werden weitere Konzepte benötigt. In Abbildung 4-4 wird eine Übersicht über das Metamodell hinsichtlich des Konzepts *Adapt Case 4 BPM* gegeben. Durch das hier als Klasse *AdaptCase4BPM* dargestellte Konzept lassen sich – in Anlehnung an den durch *Luckey* [Luc13] vorgestellten Ansatz – Anpassungsfälle für anpassbare Prozesse beschreiben.

Auslösende Ereignisse

Wie bereits zuvor beschrieben, kann für einen *AC4BPM* eine Reihe von Ereignissen definiert werden, die für die Kennzeichnung einer Auslösung des Anpassungsfalls verwendet werden. Derartige Ereignisse werden in der Abbildung unter dem Typ *AdaptationRequestEvent* vom Typ *AdaptCase4BPM* referenziert.

Anpassungsfälle können derartig verfeinert werden, dass zum einen ein *Beobachtungsprozess* (*MonitoringProcess*) und zum anderen eine Reihe von *Anpassungsprozessen* (*AdaptationProcess*) enthalten sein können.

Beobachtungsprozess (Monitoring Process)

Durch einen *Beobachtungsprozess* können Analyse- und Planungsfunktionen beschrieben werden, die für eine Anpassung von Prozessen notwendig sein können. Die für einen Anpassungsfall definierten Ereignisse vom Typ *AdaptationRequestEvent* dienen als mögliche Startereignisse (*StartEvent*) für einen *Beobachtungsprozess*. Wird im Rahmen seiner Funktion festgestellt, dass eine Anpassung notwendig ist, wird ein Ereignis vom Typ *CallAdaptationProcessEvent* ausgelöst, dass den *Beobachtungsprozess* beendet und einen *Anpassungsprozess* aufruft.

Anpassungsprozess (Adaptation Process)

Soll eine Anpassung von Prozessen beschrieben werden, kann das Konzept des *Anpassungsprozesses* (*AdaptationProcess*) genutzt werden. Dieser wird durch die Beendigung eines *Beobachtungsprozesses* und das Auslösen eines spezifischen Ereignisses aufgerufen. Bei einem *Anpassungsprozess* handelt es sich um eine Abfolge von spezifischen Operationen (*AdaptationOperation*) zur Anpassung von Prozessen. Beispiele für derartige

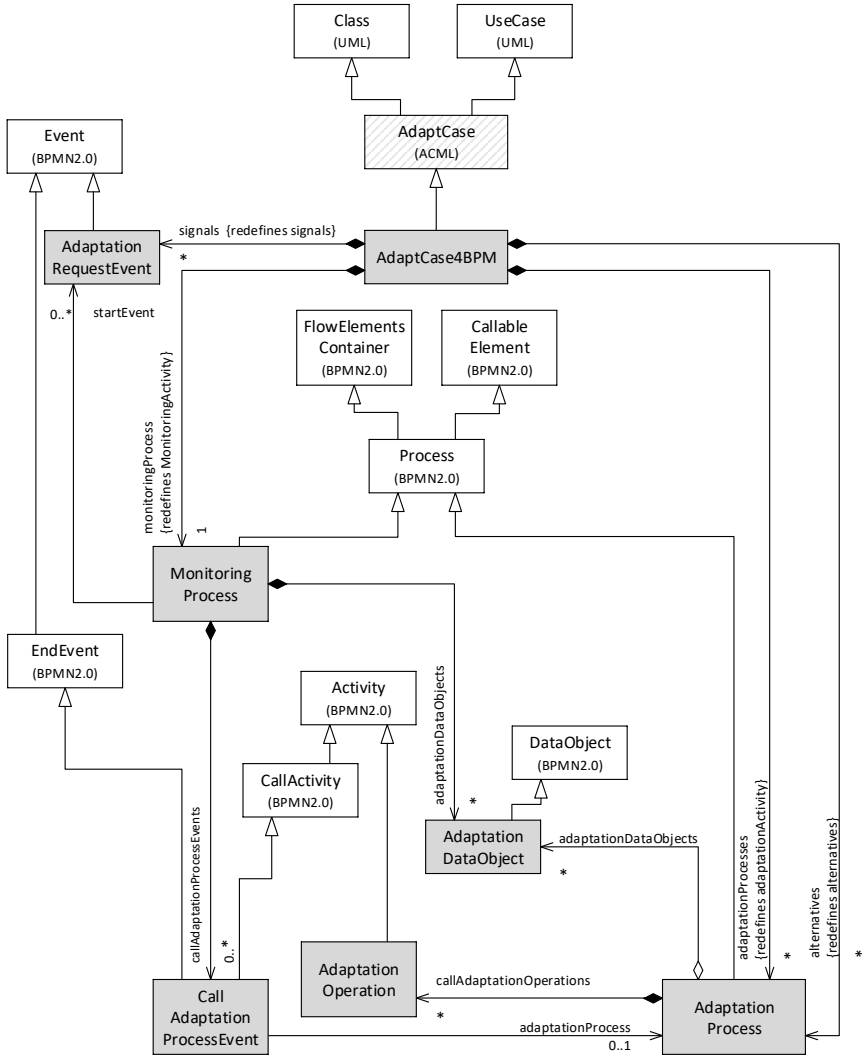


Abbildung 4-4: Übersicht über das Metamodell des Konzepts Adapt Case 4 BPM

Operationen werden in Abschnitt 4.3.3 im Rahmen der Beschreibung des *AVM4BPM* gegeben. Daneben sind weitere Operationen möglich, die im Rahmen der Beschreibung von System- und Umgebungskomponenten und deren Schnittstellen beschrieben werden können.

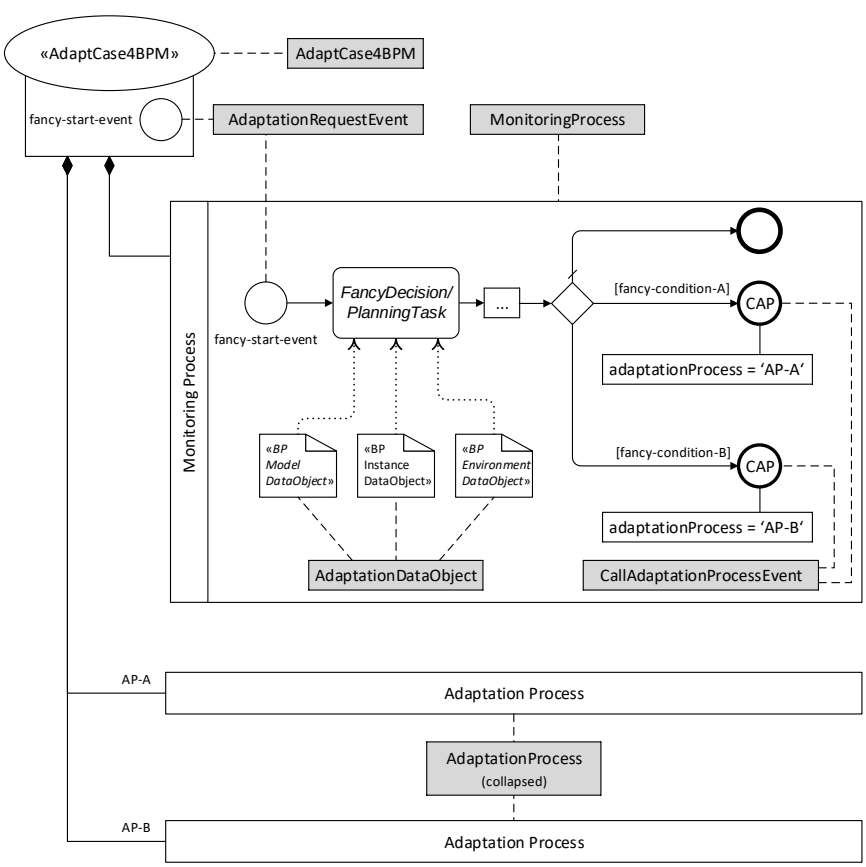
Sowohl der Typ *MonitoringProcess* als auch der Typ *AdaptationProcess* erben vom Typ *Process*. Hierdurch handelt es sich bei beiden Elementen um Containerelemente, welche weitere Elemente eines Prozesses in der Sprache *BPMN2.0* enthalten können. Auf konzeptionelle Details der beiden eingeführten Prozesstypen sowie weitere in Abbildung 4-4 dargestellte Konzepte wird in den Abschnitten 4.2.2 und 4.2.3 eingegangen.

4.2.2 Beobachtungsprozess

Durch einen Beobachtungsprozess (*MonitoringProcess*) kann Verhalten zur Beobachtung und Analyse von Prozessen und deren Umgebung beschrieben werden. Er verfeinert die durch einen *AC4BPM* gegebene Funktion. Dem Ansatz *Adapt Cases* folgend werden so die Funktionalitäten *Monitor* und *Analyze* der Referenzarchitektur *MAPE-K* [KC03] realisiert. Im Rahmen eines Beobachtungsprozesses wird also auf Ereignisse reagiert, die unmittelbar durch Prozesse oder durch ihre Umgebung ausgelöst wurden, sodass im Bedarfsfall eine Anpassung durchgeführt werden kann.

Eine konzeptionelle Darstellung eines Anpassungsfalls mit Fokus auf den dargestellten Beobachtungsprozess ist in Abbildung 4-5 gezeigt. Für eine Beschreibung der dargestellten zusammengeklappten Anpassungsprozesse (*Adaptation Process*) wird auf Abschnitt 4.2.3 verwiesen.

Abbildung 4-5: Konzeptionelle Darstellung des Beobachtungsprozesses (Monitoring Process)



Start eines Beobachtungsprozesses

Für den dargestellten Anpassungsfall ist ein einfaches auslösendes Ereignis (*AdaptationRequestEvent*) definiert worden, welches auch das Startereignis des Beobachtungsprozesses (*MonitoringProcess*) darstellt. Ein Beobachtungsprozess wird grafisch in Anlehnung an *BPMN*-spezifische Pools dargestellt. Das enthaltene Verhalten kann durch das Aufkommen eines der im Rahmen des Anpassungsfalls definierten auslösenden Ereignisse gestartet werden. Der Kontrollfluss eines Beobachtungsprozesses kann verschiedene Aktivitäten zur Beschreibung von Beobachtungs- und Analysefunktionen enthalten. Dabei kann auf Informationen, wie hier dargestellt in Form von Datenobjekten des Typs *AdaptationDataObject*, zugegriffen werden. Für eine detaillierte Beschreibung für den Zugriff auf Informationen wird auf die Beschreibung von Sensor- und Effektorschnittstellen in Abschnitt 4.3.2 verwiesen.

Alternative Startereignisse

Durch die dargestellten Sprachelemente für die Verfeinerung eines Anpassungsfalls in Form eines Beobachtungsprozesses ist eine an die Sprache *BPMN2.0* angelehnte Gestaltung von Beobachtungs- und Analysefunktionen möglich. Für weitere Beispiele für die Verfeinerung von Anpassungsfällen unter Verwendung von spezifischen Ereignissen sind in Abbildung 4-6 mögliche auslösende Ereignisse durch die Typen *TimerEvent*, *MessageEvent* und *Signal* dargestellt. Dabei handelt es sich bei den hier dargestellten Typen von Ereignissen um eine ausgesuchte Auswahl. Es kann neben den in Abschnitt 4.3.4 eingeführten Ereignissen der Sprache *BPMN2.0* auch weitere Ereignisse geben, die im Rahmen des *Adapt Case View Model 4 BPM* angegeben und in der Gestaltung von Anpassungsfällen verwendet werden können.

Beendigung eines Beobachtungsprozesses

Ein Beobachtungsprozess kann, wie in Abbildung 4-5 dargestellt, auf verschiedene Weise beendet werden. So können auf Basis der Beobachtungs- und Analysefunktion Entscheidungen hinsichtlich anzuwendender Maßnahmen beschrieben werden. Der dargestellte Kontrollfluss kann, sofern die Bedingungen *fancy-condition-A* und *fancy-condition-B* nicht positiv ausgewertet werden, ohne eine Anpassung von Prozessen beendet werden.

Alternativ kann eine der beiden Anpassungen durch Ereignisse des Typs *CallAdaptationProcessEvent* aufgerufen und der Beobachtungsprozess beendet werden. Derartige Ereignisse werden in Anlehnung an *BPMN*-spezifische Endereignisse mit der Beschriftung *CAP* dargestellt. Ein Beobachtungsprozess kann dabei mehrere Ereignisse des Typs *CallAdaptationProcessEvent* enthalten, sodass z.B. für verschiedene kontext- oder prozessspezifische Eigenschaften andere Anpassungsprozesse aufgerufen werden können. Zur Bestimmung, welche Anpassung in Form eines Anpassungs-

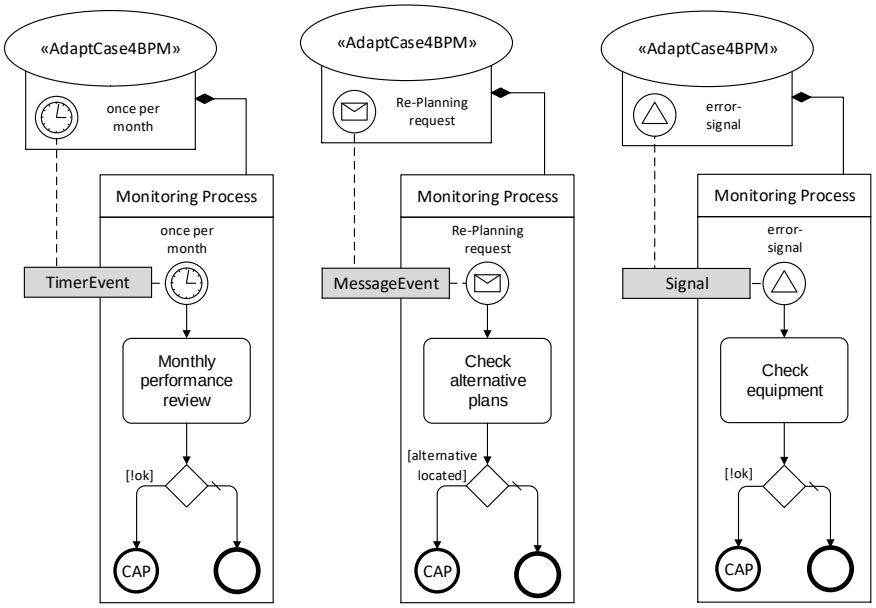


Abbildung 4-6: Beispiele für auslösende Ereignisse in einem Beobachtungsprozess (Monitoring Process)

prozesses aufgerufen werden soll, ist hierbei jedoch das Attribut *adaptationProcess* mit dem Namen des zu referenzierenden Anpassungsprozesses zu setzen.

Abstrakte Syntax

In Anlehnung an den in Abbildung 4-4 gezeigten Ausschnitt des Metamodells für das Konzept *Adapt Case 4 BPM* ist ein Beobachtungsprozess vom Typ *MonitoringProcess* und erbt von dem Typ *Process*. Ferner ist er Teil des Typs *AdaptCase4BPM* und agiert in der Rolle *monitoringProcess*, welche die ursprüngliche Rolle der *monitoringActivity* ersetzt. Durch diese *BPM*-spezifische Redefinition des Konzepts handelt es sich bei dem Typ *MonitoringProcess* um eine *BPMN*-spezifische Prozessdefinition. Diese kann neben einer einfachen Spezifikation von Kontroll- und Datenflüssen innerhalb des Beobachtungsprozesses auch deren organisatorische Einbettung in Form von *Pools* und *Lanes* enthalten. Hierdurch ist es möglich, auch komplexe Beobachtungs- und Analysefunktionen zu beschreiben, deren einzelne Schritte durch verschiedene Rollen ausgeführt werden können. Wurde durch beschriebene Beobachtungs- und Analysefunktionen die Notwendigkeit einer Anpassung von Prozessen oder an ihrer Umgebung detektiert, so lassen sich im Rahmen der Gestaltung des Beobachtungsprozesses verschiedene Aufrufe von Anpassungsprozessen (*Adaptation Process*) (siehe Abschnitt 4.2.3) definieren.

4.2.3 Anpassungsprozess

Durch einen Anpassungsprozess (*AdaptationProcess*) kann Verhalten zur Anpassung von Prozessen und deren Umgebung beschrieben werden. Er verfeinert die durch einen *AC4BPM* gegebene Funktion und ergänzt das Verhalten von Beobachtungsprozessen. Ein Anpassungsprozess kann dabei die Funktion *Execute* der Referenzarchitektur *MAPE-K* [KC03] realisieren, in dem die in Abschnitt 4.3.2 beschriebenen Effektorschnittstellen und deren Operationen zur Anpassung von Eigenschaften der System- und Umgebungskomponenten verwendet werden.

Eine konzeptionelle Darstellung eines Anpassungsfalls mit Fokus auf die dargestellten Anpassungsprozesse mit der Bezeichnung *AP-A* und *AP-B* ist in Abbildung 4-7 gezeigt. Sie stellen zueinander jeweils einen alternativen Anpassungsprozess dar. Bei dem zusammengeklappten Beobachtungsprozess handelt es sich um den in Abschnitt 4.2.2 beschriebenen Prozess in einer alternativen Darstellungsweise.

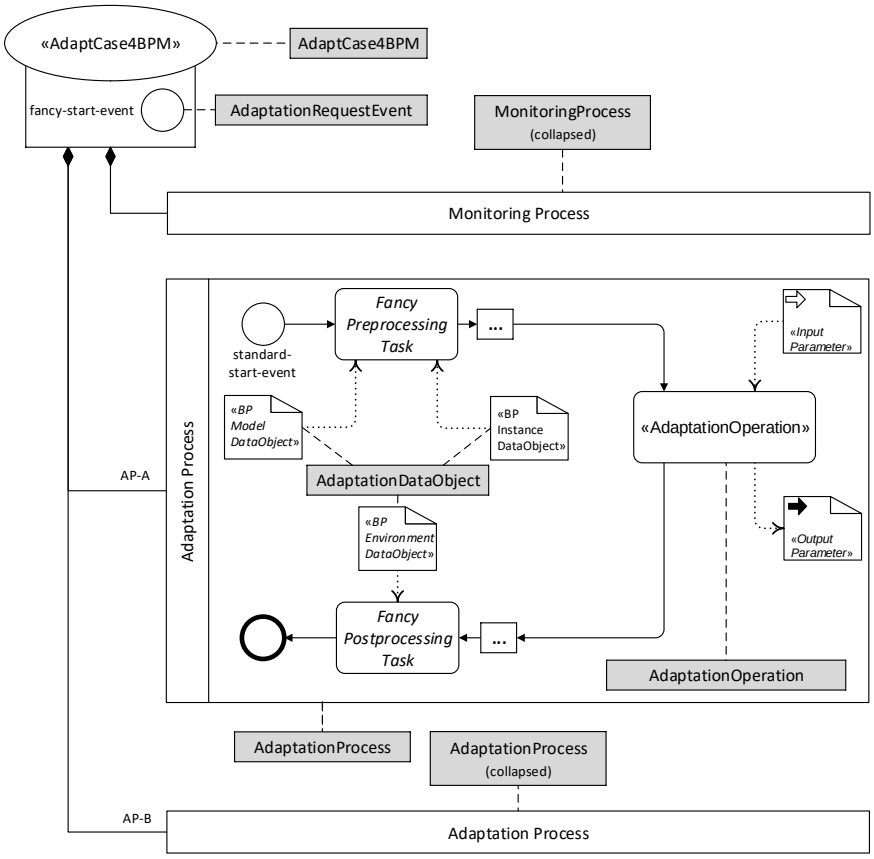


Abbildung 4-7: Konzeptionelle Darstellung des Anpassungsprozesses (Adaptation Process)

Ein Anpassungsprozess wird grafisch in Anlehnung an *Pools* der Sprache *BPMN2.0* dargestellt. Er beginnt typischerweise mit einem Startereignis der Sprache *BPMN2.0*, da er durch einen Beobachtungsprozess explizit aufgerufen wird. Der Kontrollfluss eines Beobachtungsprozesses kann verschiedene Aktivitäten (*AdaptationOperation*) zur Beschreibung von Anpassungen von Prozessen enthalten. Für diese Operationen können verschiedene Ein- und Ausgabeparameter notwendig sein, die in Anlehnung an *BPMN*-spezifische Ein- und Ausgabeobjekte dargestellt sind. Ferner ist es möglich, auch weitere Operationen zur Anpassung von durch System- und Umgebungskomponenten gekapselten Inhalten vorzunehmen, sofern sie im Rahmen des *AVM4BPM* gestaltet worden sind. Für Beispiele für mögliche Operationen wird auf Abschnitt 4.3.3 verwiesen.

Daneben können aber auch Aktivitäten sinnvoll sein, die die Anpassung von Prozessen vor- oder nachbereiten. Diese sind in der konzeptionellen Darstellung als *FancyPreprocessingTask* bzw. *FancyPostprocessingTask* abgebildet. Die Aktivitäten des Kontrollflusses des Anpassungsprozesses können ebenfalls Datenobjekte vom Typ *AdaptationDataObject* verwenden, die in Anlehnung an die Operationen der in Abschnitt 4.3 beschriebenen System- und Umgebungskomponenten abgeleitet werden können. Der Zugriff auf Daten wird in der Sprache *ACML4BPM* durch Datenobjekte (*AdaptationDataObject*) repräsentiert, die durch Sensorschnittstellen angeboten werden (siehe Abschnitt 4.3.2).

In Anlehnung an den durch Abbildung 4-4 gezeigten Ausschnitt des Metamodells für das Konzept *Adapt Case 4 BPM* ist ein Anpassungsprozess vom Typ *AdaptationProcess* und erbt von dem Typ *Process*. Ferner ist er Teil des Typs *AdaptCase4BPM* und agiert in der Rolle *adaptationProcess*, welche die ursprüngliche Rolle der *adaptationActivity* ersetzt. Sind alternative Anpassungsprozesse vorhanden, agieren sie in der Rolle *alternatives*, welche die ursprüngliche Rolle *alternatives* ersetzt.

Durch diese *BPM*-spezifische Redefinition des Konzepts handelt es sich bei dem Typ *AdaptationProcess* um eine *BPMN*-spezifische Prozessdefinition. Diese kann neben einer einfachen Spezifikation von Kontroll- und Datenflüssen innerhalb des Anpassungsprozesses auch deren organisatorische Einbettung in Form von *Pools* und *Lanes* enthalten. Hierdurch ist es möglich auch komplexe Anpassungen von Prozessen zu beschreiben, deren einzelne Schritte durch verschiedene Rollen vorgenommen werden können.

4.3 Adaptation View Model 4 BPM

Durch das *Adaptation View Model 4 BPM (AVM4BPM)* werden strukturelle Informationen eines Systems beschrieben. Hierbei liegt der Fokus auf den System- und Umgebungskomponenten und ihren angebotenen Schnittstellen mit möglichen Ereignissen und Operationen zum Lesen oder Anpassen von Eigenschaften. In Anlehnung an die Referenzarchitektur *MAPE-K* [KC03] lassen sich Komponenten grundlegend in die Typen *System* und *Umgebung* unterscheiden. Neben diesen Komponenten sind aber auch die von ihnen angebotenen Schnittstellen, die vorhandenen Ereignisse sowie die angebotenen Operationen zum Lesen und zur Anpassung von Eigenschaften von besonderem Interesse. Im zuvor genannten Ansatz werden dabei die beiden grundlegenden Typen *Sensor* und *Effector* für Schnittstellen genannt.

In Abbildung 4-8 sind die Inhalte der nachfolgenden Abschnitte in Anlehnung an die zuvor genannten Elemente dargestellt. So wird zunächst in Abschnitt 4.3.1 auf System- und Umgebungskomponenten der Domäne *BPM* eingegangen. Nachfolgend werden in Abschnitt 4.3.2 die von diesen Komponenten angebotenen Sensor- und Effektorschnittstellen vorgestellt. Abschließend werden die von den Schnittstellen angebotenen Operationen (siehe Abschnitt 4.3.3) und Ereignisse (siehe Abschnitt 4.3.4) beschrieben.

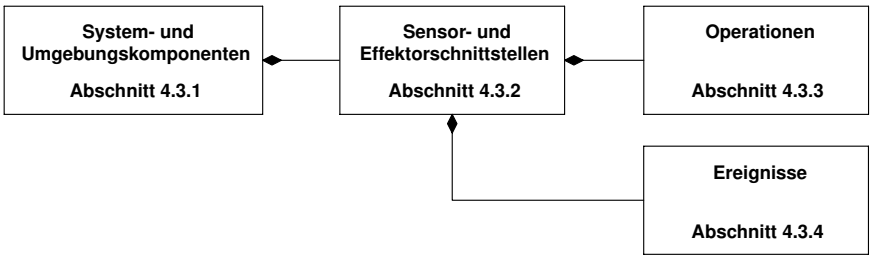


Abbildung 4-8: Inhalte des Adaptation View Model 4 BPM (AVM4BPM)

4.3.1 System- und Umgebungskomponenten

Durch die System- und Umgebungskomponenten können strukturelle Informationen über das System beschrieben werden. Für anpassbare Prozesse muss hierzu zunächst analysiert werden, welche Komponenten sinnvoll sein können. Ein mögliches methodisches Vorgehen für die Analyse ist dadurch gegeben, den *BPM-Lebenszyklus* (siehe Abschnitt 2.2) mit den im Vordergrund stehenden Artefakten *Prozessmodell* und *Prozessinstanz* zunächst näher zu betrachten.

Hierzu ist in Abbildung 4-9 der *BPM-Lebenszyklus* nach *Weske* [Wes12] mit den im Fokus stehenden Artefakten und Treibern zur Anpassung von Prozessen dargestellt. Insgesamt werden in dieser Analyse vier Treiber zur Anpassung von Prozessen betrachtet. Sie beziehen sich je nach Phase des Lebenszyklus entweder auf Anpassungen von Prozessmodellen oder auf Prozessinstanzen. Im Folgenden wird näher auf die vier Treiber *Definition*, *Verfeinerung*, *Adaption* bzw. *Flexibilisierung* und *Verbesserung* eingegangen.

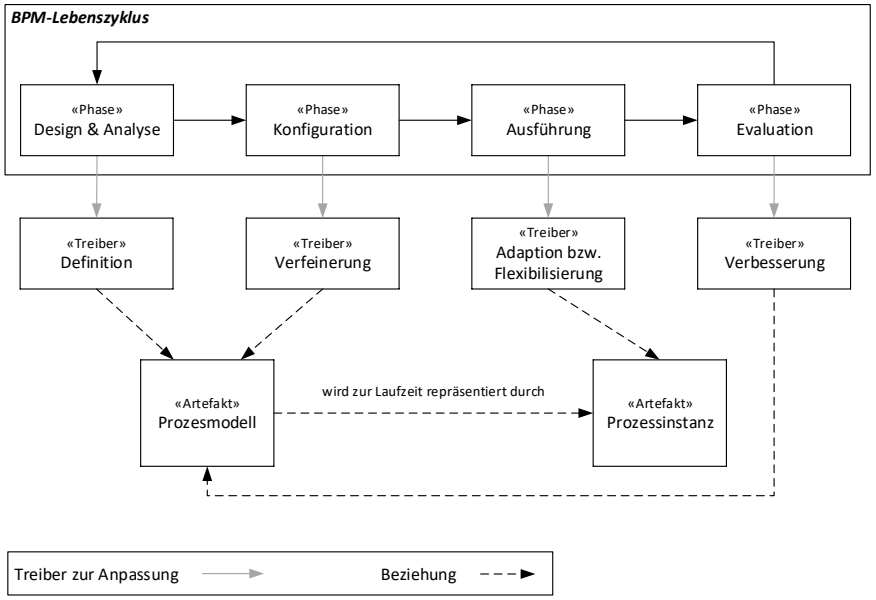


Abbildung 4-9: BPM-Lebenszyklus mit den im Fokus stehenden Artefakten und möglichen Treibern zur Anpassung von Prozessen

Definition In der ersten Phase des *BPM-Lebenszyklus Design & Analyse* werden Prozessmodelle definiert. Je nach Perspektive kann bei dieser Definition von Prozessen auch von Anpassung gesprochen werden, da ein initiales bzw. zunächst leeres Prozessmodell nach und nach im Rahmen der Gestaltung der Prozesse mit Sprachelementen angereichert wird. Jede Anreicherung eines Prozessmodells mit neuen Sprachelementen stellt eine Anpassung im Bezug zum ausgehenden Prozessmodell dar.

Verfeinerung In der zweiten Phase *Konfiguration* werden Prozessmodelle für die Ausführung konfiguriert bzw. verfeinert. Dies kann z.B. durch eine Implementierung oder das Hinzufügen von plattformspezifischen Informationen geschehen. Ferner können Prozesse in frühen Phasen des *BPM-Lebenszyklus* auch hinsichtlich der Flexibilität verfeinert werden, wie es z.B. bei der Variabilität von Prozessen durchgeführt werden kann (siehe Abschnitt 2.2.3). Betrachtet man diese Zwecke, kann auch hier von Anpassungen auf Basis existierender Prozessmodelle ausgegangen werden. Aufgrund des Hinzu-

fügens neuer Informationen bzw. der Selektion bestimmter Teilprozesse für einen konkreten Kontext wird in diesem Bezug auch von einer Verfeinerung gesprochen.

Adaption bzw. Flexibilisierung

Während der Ausführung werden Prozessmodelle durch Prozessinstanzen repräsentiert. Ferner können zur Ausführung Ereignisse aufkommen, die geplant oder ungeplant eine Anpassung von sowohl Prozessmodellen als auch Prozessinstanzen notwendig machen können. Im Rahmen dieser Arbeit sind hier die ausschlaggebenden Treiber durch *Adaption* bzw. *Flexibilisierung* von Prozessen dargestellt, die dementsprechend in der dritten Phase *Ausführung* vorkommen. Ereignisse können hier die Anpassung von Eigenschaften der Prozessmodelle und -instanzen auslösen.

Verbesserung

Der *BPM-Lebenszyklus* sieht in der letzten Phase *Evaluation* die Analyse und Bewertung bestehender Prozesse zur Verbesserung vor. Dabei kann z.B. auf die Historie von abgeschlossenen Prozessinstanzen zurückgegriffen werden, welche zu dem genannten Zweck der Verbesserung analysiert werden. Auf Basis der Analyseergebnisse können neue Anforderungen zur Verbesserung der Prozesse in der nächsten Iteration des *BPM-Lebenszyklus* zur Verfügung gestellt werden. Hierdurch ist es möglich, vorhandene Prozesse kontinuierlich zu verbessern und an neue Umgebungsfaktoren anzupassen. Eine Verbesserung eines Prozesses zur Laufzeit kann durch geeignete Mechanismen im Rahmen der *Adaption* bzw. *Flexibilisierung* vorgenommen werden.

Entlang der Phasen des *BPM-Lebenszyklus* und der vorgestellten Treiber für Anpassungen von Prozessen lassen sich zwei Kontexte identifizieren, welche durch die Anpassung von Prozessmodellen und Prozessinstanzen gegeben sind. Die Anpassung von Prozessmodellen kann wie beschrieben in allen genannten Phasen sinnvoll sein. Wobei die Anpassung von Prozessinstanzen vorwiegend in der Phase *Ausführung* als sinnvoll erachtet werden kann.

Konkrete Syntax für Komponenten

Aufgrund der verschiedenen erläuterten Kontexte, in denen Anpassungen von Prozessen möglich sind, werden unterschiedliche Komponenten benötigt. Hierdurch können für einen jeweiligen Kontext spezifische Operationen und Ereignisse angeboten werden. Das in dieser Arbeit vorgestellte Konzept sieht dabei insgesamt drei unterschiedliche Komponenten vor. Darunter befinden sich zwei Systemkomponenten und eine Umgebungskomponente, deren konkrete Syntax in Abbildung 4-10 gezeigt ist. Die drei Typen von Komponenten werden in Anlehnung an *UML-Komponenten* [OMG10] dargestellt und können jeweils über ein individuelles Symbol unterschieden werden. Dabei tragen die Symbole eine Ab-

kürzung ihrer jeweiligen Bezeichnung der Typen. So trägt z.B. das Symbol für den Typ *BPModelComponent* die Abkürzung *MC*. Die in Abbildung 4-10 dargestellten Sensor- und Effektorschnittstellen werden ebenso analog zu *UML-Schnittstellen (Interfaces)* [OMG10] dargestellt.

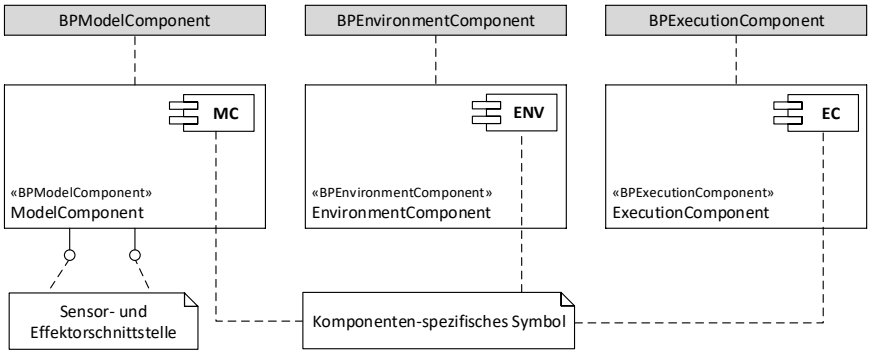


Abbildung 4-10: Konkrete Syntax von System- und Umgebungskomponenten (AVM4BPM)

Die im linken Bereich dargestellte Systemkomponente vom Typ *BPModelComponent* ist für die Kapselung von Prozessmodellen gedacht. Die zweite im rechten Bereich dargestellte Systemkomponente *BPExecutionComponent* kapselt die zur Laufzeit bestehenden Prozessinstanzen.

Für industrielle Umgebungen kann es eine Reihe von unterschiedlichen Umgebungskomponenten geben, die aktiv oder passiv mit anpassbaren Prozessen interagieren und diese beeinflussen können. Beispiele für derartige Umgebungskomponenten sind z.B. Sensoren oder Aktoren in einer Produktionsumgebung, die Eigenschaften von Ressourcen wahrnehmen, prüfen oder anpassen können. Derartige Umgebungskomponenten können dabei im Rahmen der Prozessausführung bzw. -steuerung in unmittelbarem Zusammenhang mit den Prozessen stehen. Zur Kapselung derartiger Umgebungskomponenten werden in der Sprache *ACML4BPM* Umgebungskomponenten vom Typ *BPEnvironmentComponent* verwendet.

In Abbildung 4-11 ist die abstrakte Syntax der erarbeiteten Typen der System- und Umgebungskomponenten der Teilsprache *AVM4BPM* dargestellt. Dabei werden bestehende Konzepte des Ansatzes *Adapt Cases* in der Farbe *Weiß* und erarbeitete domänenspezifische Konzepte des Ansatzes *Adapt Cases 4 BPM* in der Farbe *Grau* dargestellt.

BPModelComponent

Für Anpassungen von Prozessmodellen ist die Komponente *BPModelComponent* vorgesehen. Eine derartige Komponente referenziert eine beliebige Menge von Prozessmodellen, welche hier als vom Typ *Process* dargestellt werden. Sie dient der Kapselung von Prozessmodellen, welche für

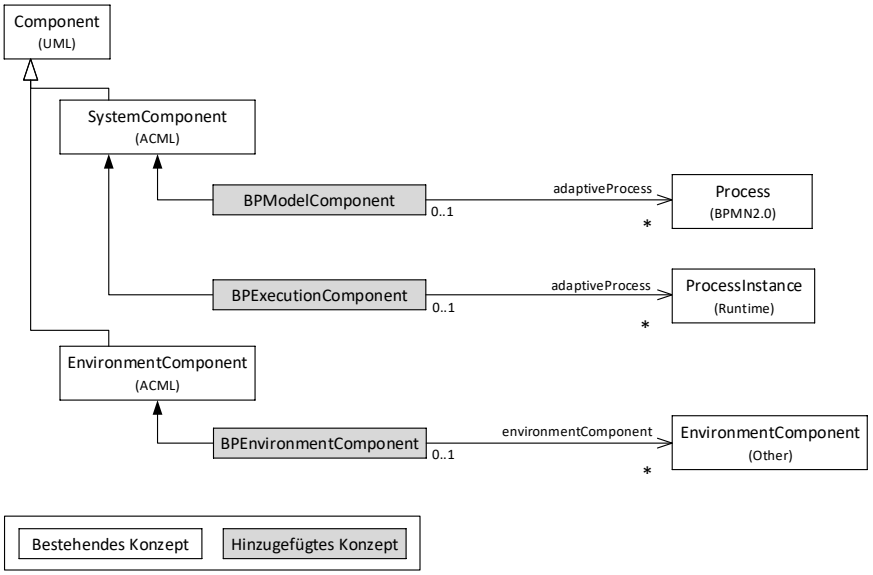


Abbildung 4-11: System- und Umgebungskomponenten (AVM4BPM)

Anpassungen vorgesehen sind, weshalb sie in der Rolle *adaptiveProcess* agieren. Der Typ *Process* stellt ein sogenanntes Containerelement für die Definition von Prozessen in der Sprache *BPMN2.0* [OMG11] dar. In ihm können weitere Elemente der Sprache *BPMN2.0*, wie z.B. *Pools*, *Lanes* oder das Verhalten des Prozesses, enthalten sein.

BPExecutionComponent

Durch die zweite Systemkomponente *BPExecutionComponent* werden die zur Laufzeit bestehenden Prozessinstanzen gekapselt. Hier ist darauf hinzuweisen, dass innerhalb der Spezifikation der Sprache *BPMN2.0* keine Sprachelemente für die Beschreibung von Prozessinstanzen vorhanden sind. Daher ist die Klasse *ProcessInstance* nicht analog zur Klasse *Process* mit einem Containerelement der Sprache *BPMN2.0* zu referenzieren. Die Klasse *ProcessInstance* repräsentiert daher lediglich eine Abstraktion einer konkreten internen Repräsentation einer Prozessinstanz innerhalb einer beliebigen Laufzeitumgebung (*Runtime*) für Prozesse.

Im Rahmen der Phase *Konfiguration* und der Auswahl einer konkreten Plattform zur Ausführung müssen daher gegebenenfalls Eigenschaften und Funktionalitäten ergänzt bzw. verfeinert werden. Dies betrifft insbesondere auch die angebotenen Operationen zur Anpassung (siehe Abschnitt 4.3.3) und Ereignisse (siehe Abschnitt 4.3.4).

BPEnvironmentComponent

In diesem Ansatz wird der Typ *BPEnvironmentComponent* für die Repräsentation einer beliebigen Umgebungskomponente (*Other*) verwendet. Der Typ *BPEnvironmentComponent* stellt dabei eine Realisierung der Klasse *Environment* dar. Aufgrund der Diversität an möglichen zu kapselnden In-

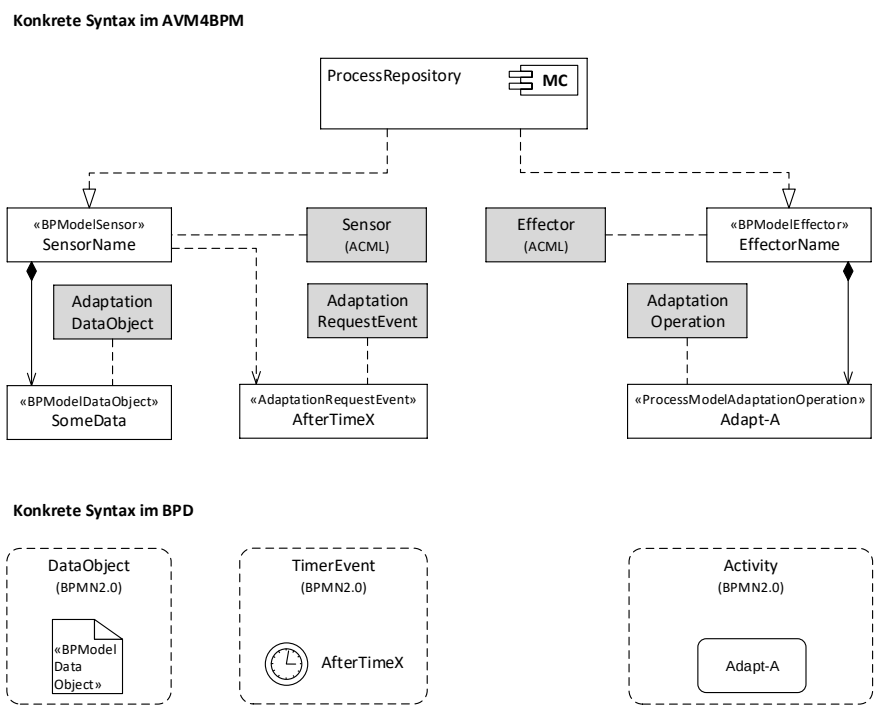
halten wird dieser Typ in dieser Arbeit nicht detaillierter spezifiziert. Hierzu lässt sich vornehmlich die Fokussierung von Prozessen als Begründung anbringen.

Sowohl die System- als auch die Umgebungskomponenten bieten Schnittstellen der Typen *Sensor* und *Effector* an. Hierdurch wird ein kontrollierter Zugriff auf gekapselte Inhalte ermöglicht. Die Beschreibung dieser Schnittstellen wird in dem nachfolgenden Abschnitt 4.3.2 vorgenommen.

4.3.2 Sensor- und Effektorschnittstellen

In diesem Abschnitt werden Sensor- und Effektorschnittstellen beschrieben, die von den im vorherigen Abschnitt vorgestellten System- und Umgebungskomponenten angeboten werden. Insgesamt sind im Rahmen des *AVM4BPM* drei Konzepte für den kontrollierten Zugriff auf gekapselte Inhalte vorgesehen. Dabei handelt es sich um Ereignisse zur Auslösung einer Anpassung (*AdaptationRequestEvent*), den Zugriff auf Daten (*AdaptationDataObject*) und Operationen zur Anpassung (*AdaptationOperation*). Beispiele für diese Konzepte hinsichtlich des Typs von Systemkomponenten *BPModelComponent* sind in Abbildung 4-12 durch ihre konkrete Syntax gegeben.

Abbildung 4-12: Konkrete Syntax von Sensor- und Effektorschnittstellen (AVM4BPM)



Gezeigt ist eine konkrete Systemkomponente (*ProcessRepository*), die zwei Schnittstellen realisiert: eine Sensorschnittstelle (*BPModelSensor*) und eine Effektorschnittstelle (*BPModelEffector*). Die Sensorschnittstelle bietet dabei für eine Anpassung notwendige Daten an (*AdaptationDataObject*). Ferner definiert sie zudem ein Ereignis (*AdaptationRequestEvent*), das für die Auslösung einer Anpassung genutzt werden kann. Die Effektorschnittstelle bietet eine Operation (*AdaptationOperation*) zur Anpassung an. Die konkrete Syntax der Elemente des *AVM4BPM* folgt dabei einer an die *UML* angelehnte Darstellung. Werden die im Rahmen des *AVM4BPM* gestalteten Elemente im Rahmen des *ACM4BPM* (siehe Abschnitt 4.2) eingesetzt, so ändert sich die Darstellung hinsichtlich einer entsprechenden Weise für die Sprache *BPMN2.0*. Hierfür sind Beispiele im unteren Bereich von Abbildung 4-12 gegeben. Für weitere Beispiele wird auf die Beschreibung des *ACM4BPM* in Abschnitt 4.2 und auf das im Rahmen der Evaluation gegebene Szenario in Abschnitt 7.1 verwiesen.

Eine Übersicht über des zu den Schnittstellen zugehörigen Metamodells ist in Abbildung 4-13 gezeigt. Die erarbeiteten Schnittstellen werden dort als Realisierungen der Typen *Sensor* und *Effector* der Sprache *ACML* dargestellt. Nachfolgend wird auf damit zusammenhängende Konzepte eingegangen.

Typen von Schnittstellen

Es bestehen die drei Sensorschnittstellen *BPModelSensor*, *BPExecutionSensor* und *BPEnvironmentSensor*. Ferner werden als Effektorschnittstellen die Typen *BPModelEffector*, *BPExecutionEffector* und *BPEnvironmentEffector* vorgestellt. Durch die Verwendung der genannten Schnittstellen kann ein kontrollierter lesender (*Sensor*) und schreibender Zugriff (*Effector*) auf Funktionen und Eigenschaften der System- bzw. Umgebungskomponenten realisiert werden. Die zuvor genannten Typen von Sensor- und Effektorschnittstellen werden nachfolgend detaillierter beschrieben.

Sensorschnittstellen

Die dargestellten Sensorschnittstellen stellen spezielle Schnittstellen dar, mit deren Hilfe auf verschiedene Eigenschaften der jeweiligen Komponenten und der gekapselten Inhalte zugegriffen werden kann. Dabei liegt der Fokus einer solchen Schnittstelle auf dem lesenden Zugriff von Eigenschaften, sodass bei ihrer Verwendung keine Anpassung durchgeführt wird. Die Zugehörigkeit einer Sensorschnittstelle lässt sich in Anlehnung an die Benennung ihres Typs ableiten, sodass bspw. eine Schnittstelle vom Typ *BPModelSensor* einer Komponente vom Typ *BPModelComponent* zugehörig ist. Weitere Zuordnungen können entsprechend gleichartig vorgenommen werden.

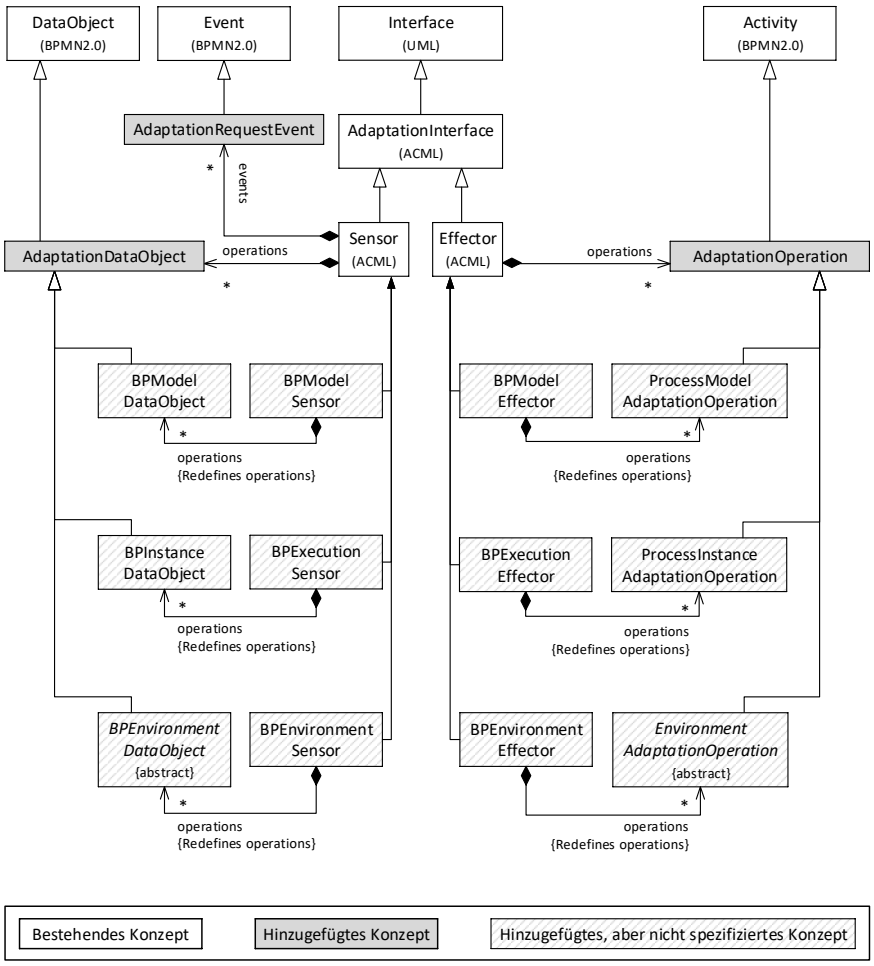


Abbildung 4-13: Sensor- und Effektorschnittstellen für anpassbare Prozesse (AVM4BPM)

Zugriff auf Daten

Im Rahmen der Sprache *ACML4BPM* wird der lesende Zugriff auf Eigenschaften der Komponenten durch Datenobjekte vom Typ *AdaptationDataObject* realisiert. Bei diesem Typ handelt es sich um eine spezifische Variante des Typs *DataObject* aus der Sprache *BPMN2.0*. Für die zuvor beschriebenen Sensorschnittstellen werden jeweils spezifische Typen von Datenobjekten eingeführt. So bietet bspw. die Sensorschnittstelle vom Typ *BPModelSensor* den lesenden Zugriff auf Eigenschaften durch Datenobjekte vom Typ *BPModelDataObject*.

Auslösung einer Anpassung

Neben dem lesenden Zugriff auf Eigenschaften bieten Sensorschnittstellen aber auch die Möglichkeit zur Spezifikation von Ereignissen, die ausgehend von der Komponente eine mögliche Anpassung auslösen können. Diese Ereignisse werden in Abbildung 4-13 durch den Typ *AdaptationRe-*

questEvent repräsentiert, der von dem Typ *Event* der Sprache *BPMN2.0* erbt. Hierdurch ist es möglich verschiedenste Typen von Ereignissen der Sprache *BPMN2.0* im Rahmen des *AVM4BPM* zu beschreiben und anschließend im Rahmen des *ACM4BPM* zu nutzen. Auf Details möglicher Ereignisse wird in Abschnitt 4.3.4 näher eingegangen.

Effektorschnittstellen

Neben Sensorschnittstellen sind aber insbesondere auch Effektorschnittstellen notwendig, mit denen Anpassungen von den Eigenschaften der Komponenten und somit an den Prozessmodellen und -instanzen, aber auch an denen in ihrer Umgebung möglich sind. Jede Effektorschnittstelle kann eine Reihe von verschiedenen Operationen vom Typ *AdaptationOperation* zur Anpassung von gekapselten Inhalten anbieten.

In dieser Arbeit liegt der Fokus dabei insbesondere auf den Operationen zur Anpassung von Prozessen. Entsprechend der zuvor eingeführten Typen von Komponenten existieren in diesem Bezug die beiden Effektorschnittstellen *BPModelEffector* und *BPExecutionEffector*, die jeweils Operationen für die Anpassung von Prozessmodellen bzw. Prozessinstanzen anbieten können. Die Benennung der Schnittstellen gibt dabei darüber Aufschluss, welchem Typ von Komponente sie zugehörig ist.

So bietet bspw. eine Effektorschnittstelle vom Typ *BPModelEffector* Operationen für die Anpassung von Eigenschaften einer Komponente vom Typ *BPModelComponent* an. Zur Anpassung von Prozessinstanzen kann die Schnittstelle vom Typ *BPExecutionEffector* und deren Operationen vom Typ *ProcessInstanceAdaptationOperation* verwendet werden. Auf die von diesen beiden Effektorschnittstellen angebotenen Operationen zur Anpassung von Prozessmodellen *(ProcessModelAdaptationOperation)* und Prozessinstanzen *(ProcessInstanceAdaptationOperation)* wird in Abschnitt 4.3.3 näher eingegangen. Ebenso bietet die Schnittstelle vom Typ *BPEnvironmentEffector* Operationen zur Anpassung gekapselter Inhalte an; diese liegen jedoch nicht im Fokus dieser Arbeit, weshalb die Schnittstelle an dieser Stelle nur der Vollständigkeit halber aufgeführt wird.

Verwendung von Inhalten des AVM4BPM im ACM4BPM

Die durch die Sensor- und Effektorschnittstellen bereitgestellten Ereignisse, Datenobjekte und Operationen sollen in der Teilsprache *ACM4BPM* zur Gestaltung von Anpassungsfällen *(AC4BPM)* eingesetzt werden. Sie bilden somit das Bindeglied zwischen den beiden Teilsprachen *AVM4BPM* und *ACM4BPM*. Einzelne Operationen, die durch eine der vorgestellten Schnittstellen angeboten werden, können dabei in der Gestaltung eines *Monitoring Process* und eines *Adaptation Process* verwendet werden (siehe Abschnitt 4.2).

4.3.3 Operationen

Für die Anpassung von Eigenschaften der durch System- und Umgebungskomponenten gekapselten Inhalte sind entsprechende Operationen notwendig. Derartige Operationen werden durch Effektorschnittstellen (siehe Abschnitt 4.3.2) zur Verfügung gestellt. In diesem Abschnitt werden Operationen fokussiert, die zur Anpassung von Prozessen eingesetzt werden können. Generell lassen sich derartige Operationen dabei in drei Typen unterscheiden, die nachfolgend vorgestellt werden.

Add Anpassungsoperationen vom Typ *Add* fügen einem Prozess neue Eigenschaften hinzu. Hinsichtlich der hier betrachteten Prozesse könnten z.B. Elemente oder Eigenschaften des Kontroll- oder Datenflusses betroffen sein.

Remove Durch Anpassungsoperationen vom Typ *Remove* lassen sich Eigenschaften aus einem Prozess entfernen. Analog zu den Anpassungsoperationen vom Typ *Add* könnten auch hiervon Elemente oder Eigenschaften des Kontroll- und Datenflusses betroffen sein.

Modify Anpassungsoperationen vom Typ *Modify* fügen weder Eigenschaften hinzu noch entfernen sie diese. Stattdessen ist eine Manipulation bestehender Elemente oder ihrer Eigenschaften, z.B. in Form der Änderung des Wertes eines Attributs, möglich.

Für die in dieser Arbeit betrachtete Domäne *BPM* existieren bereits eine Reihe verschiedener Operationen, die zur Anpassung von Prozessen eingesetzt werden können. Einige Beispiele sind durch [WRR07; WRR08; Ger13] gegeben. Dabei fokussieren existierende Ansätze oftmals wenige ausgesuchte Eigenschaften von Prozessen.

Existierende Ansätze für Operationen

So legen [WRR08; WRR07] fest, dass die von ihnen beschriebenen Anpassungsoperationen lediglich für die Anwendung hinsichtlich des Kontrollflusses innerhalb eines Prozesses anwendbar sind. *Gerth* [Ger13] hingegen führt eine sogenannte *Intermediate Representation* ein, welche eine Abstraktion gängiger Prozessbeschreibungen darstellt. Auf Basis dieser Abstraktion führt er eine Reihe elementarer und erweiterter Operationen zur Anpassung ein, die sich aber ebenso auf Eigenschaften und Elemente des Kontrollflusses konzentriert. Für anpassbare Prozesse ist aber nicht nur der Kontrollfluss zu berücksichtigen, sondern auch weitere Eigenschaften und Elemente, wie z.B. aus den Perspektiven *Organisation* und *Information*

(siehe Abschnitt 2.3.2). Daher ist eine Analyse der Domäne *BPM* hinsichtlich möglicher Anpassungen notwendig, bevor Operationen der Sprache *ACML4BPM* vorgestellt werden können.

Grundlage für die Analyse der Domäne BPM

Als Grundlage für eine Analyse können verschiedene Artefakte entlang des *BPM-Lebenszyklus* dienen. Alternativ kann aber auch die Methode, also der *BPM-Lebenszyklus* selbst, Aufschluss über mögliche Umstände geben, die eine Anpassung erfordern. So stellen, wie bereits zuvor in Abbildung 4-9 beschrieben, unterschiedliche Treiber zur Anpassung von Prozessen entlang des *BPM-Lebenszyklus* und die von ihnen referenzierten Artefakte selbst bereits Umstände einer möglichen Anpassung dar. Für eine konkretere Analyse der Domäne *BPM* wird daher an dieser Stelle eine Auswahl einiger weniger Gegenstände getroffen. So wurde sich für die Referenzimplementierung des Metamodells der Sprache *BPMN2.0* im Rahmen des Projekts *BPMN2 Modeler*[1] entschieden.

In Abbildung 4-15 ist das bereits in Abschnitt 2.3.4 eingeführte Beispiel eines *Business Process Diagram* (*BPD*) der Sprache *BPMN2.0* gezeigt. Es wurde um Einfärbungen hinsichtlich der Zugehörigkeit von Elementen zu Perspektiven ergänzt, die die Basis für die weiterführende Analyse bilden.

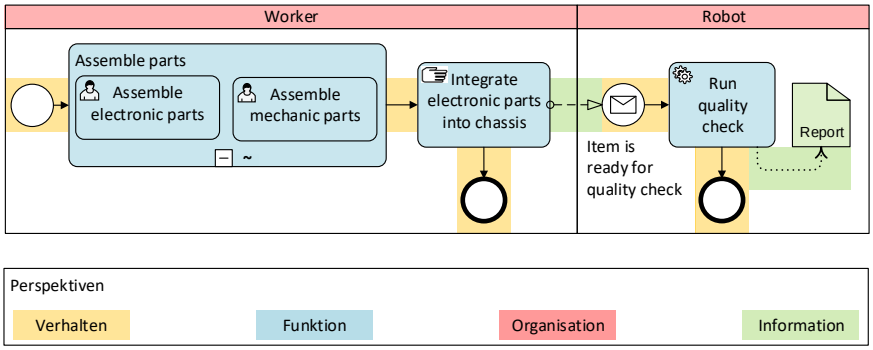


Abbildung 4-14: Analyse von Perspektiven in Prozessen auf Basis eines BPD der Sprache BPMN2.0

Eine zugehörige Einfärbung von Elementen kann ebenfalls auf Basis eines Metamodells durchgeführt werden. In Abbildung 4-15 ist dies auf Basis des Metamodells des Projekts *BPMN2 Modeler* gezeigt. Die Zugehörigkeit der einzelnen Elemente zu den von *Curtis* [CKO92] eingeführten Perspektiven ist dabei farbig dargestellt. Bei den in der Farbe *Weiß* dargestellten Elementen handelt es sich um abstrakte Klassen, die in der Sprache *BPMN2.0* keine grafische Darstellung besitzen.

[1] Projekt *BPMN2 Modeler*
`https://www.eclipse.org/bpmn2-modeler/`
Letzter Zugriff 15.12.2018

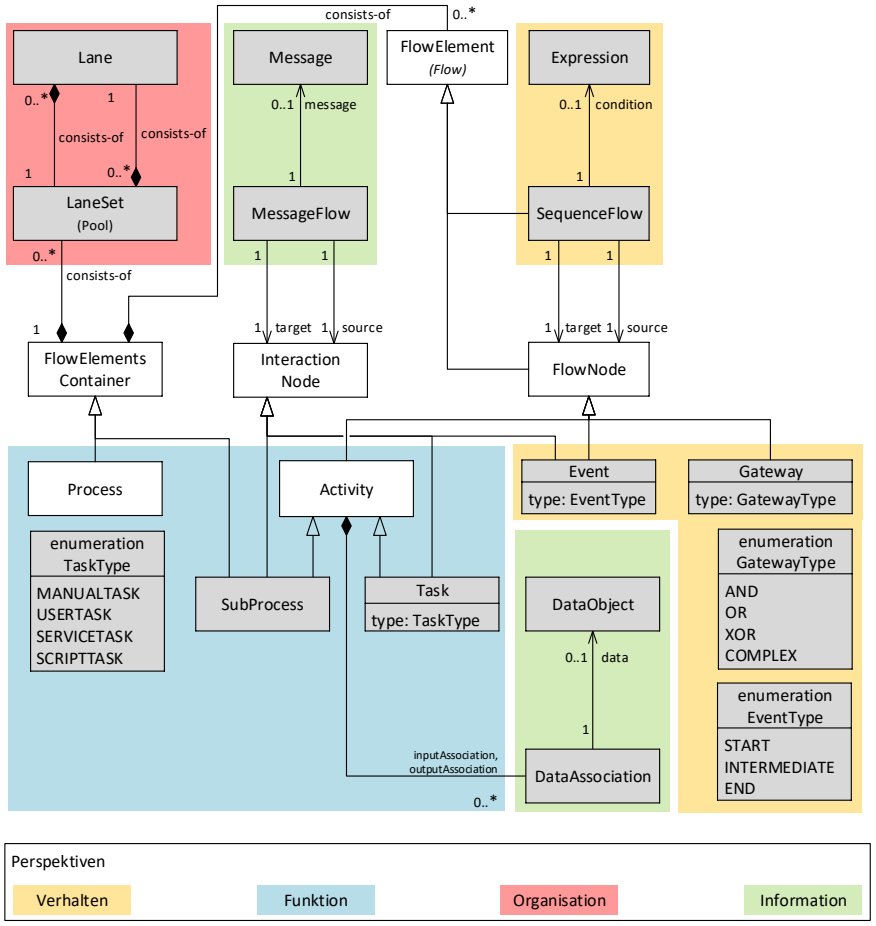


Abbildung 4-15: Analyse von Perspektiven in Prozessen auf Basis des Metamodells des Projekts BPMN 2.0 Modeler

Elemente der Perspektive Organisation

Die Wurzelklasse *Process* ist dabei der Perspektive *Funktion* zugehörig und agiert als Container für die hier dargestellte Repräsentation der Domäne *BPM*. In Abbildung 4-14 ist diese Einfärbung zur Übersichtlichkeit nicht vorgenommen. Es können aus der Perspektive *Organisation* Elemente vom Typ *Pool* und *Lane* zur Gestaltung der organisatorischen Einbettung weiterer Elemente verwendet werden. Dabei kann ein Element vom Typ *Pool* weitere Elemente vom Typ *Lane* enthalten, die wiederum weitere Elemente vom Typ *Pool* enthalten können. Das BPD beschreibt hierzu die beiden Rollen *Worker* und *Robot*.

Elemente der Perspektive Funktion

Neben Elementen der Perspektive *Organisation* kann ein Element des Typs *Process* aber auch aus Elementen anderer Perspektiven bestehen. So sind weitere Elemente der Perspektive *Funktion* durch verschiedene Aktivitäten, wie z.B. in Form der Typen *SubProcess* und *Task*, gegeben. Der hier gezeigte Ausschnitt der Domäne *BPM* unterstützt verschiedenste Unter-

typen der genannten Aktivitäten, wovon ein Auszug in Abbildung 4-15 durch die Enumeration *TaskType* angegeben ist.

Elemente der Perspektive Verhalten

Elemente der Perspektive *Funktion* können mit Elementen der Perspektiven *Verhalten* und *Information* zu verschiedenen Zwecken verbunden sein. So können verschiedene Typen von Ereignissen *(Event)* vorkommen, die den Beginn *(Start)*, den Abschluss *(End)* oder beliebige Zwischenereignisse *(Intermediate)* darstellen. Für Verzweigungen lassen sich unterschiedliche Typen von *Gateways* einsetzen. Die hier dargestellte Enumeration *GatewayType* beschreibt dabei einen Auszug der Domäne *BPM* in Bezug zu Typen von Gateways. Aktivitäten der Perspektive *Funktion* lassen sich mit den Elementen vom Typ *Event* und *Gateway* über Kanten vom Typ *SequenceFlow* verbinden. Eine Kante vom Typ *SequenceFlow* kann darüber hinaus auch mit einer Bedingung *(Expression)* versehen werden, mit der ausgedrückt werden kann, dass die Kante nur unter Einhaltung dieser Bedingung zu dem referenzierten Element führt. Das in Abbildung 4-14 gezeigte BPD enthält verschiedene Start- und Endereignisse sowie ein Ereignis vom Typ *MessageEvent* der Perspektive *Verhalten*.

Elemente der Perspektive Information

Die Perspektive *Information* ist durch zwei unterschiedliche Bereiche in dem hier dargestellten Ausschnitt des Metamodells und des BPD dargestellt. So kann für die Ausführung von Aktivitäten die Eingabe von Daten notwendig sein. Darüber hinaus ist es aber auch möglich, dass Aktivitäten Daten erzeugen. Hierfür sind in dem dargestellten Ausschnitt der Domäne *BPM* Elemente vom Typ *DataObject* vorgesehen, die über Kanten vom Typ *DataAssociation* mit den Aktivitäten verbunden werden können. Dabei deutet eine zu einer Aktivität zugehörige Kante vom Typ *DataAssociation* an, ob es sich bei dem referenzierten Datenelement um eine Eingabe *(inputAssociation)* oder um eine Ausgabe *(outputAssociation)* handelt. Das in Abbildung 4-14 gezeigte BPD zeigt das Datenobjekt *Report* und die zugehörige Kante in der Rolle *outputAssociation*. Neben den zuvor beschriebenen Assoziationen zwischen Daten und Aktivitäten lassen sich aber auch Nachrichten zwischen Aktivitäten austauschen. Ein hierdurch gegebener Nachrichtenaustausch kann durch ein Element vom Typ *MessageFlow* ausgedrückt werden. Ein Element vom Typ *MessageFlow* kann dabei ein Element vom Typ *Message* referenzieren, welches die zu versendende Nachricht beschreibt. Das BPD zeigt das zur Perspektive *Information* zugehörige Element vom Typ *MessageFlow*.

Lösungsansatz für Operationen

Bei den zuvor beschriebenen Elementen aus den verschiedenen Perspektiven von Prozessen handelt es sich um eine geringe Auswahl vorhandener Sprachelemente, die durch die Repräsentation der Domäne *BPM* in Form

der Sprache *BPMN2.0* zur Verfügung gestellt werden. Sie dienen daher lediglich als Beispiel möglicher Ansatzpunkte zur Spezifikation von Operationen für die Anpassung von Prozessen. Auf Basis der zuvor durchgeführten Analyse sind in Abbildung 4-16 Operationen zur Anpassung von Prozessen dargestellt. Jede hier dargestellte Operation *(AdaptationOperation)* wird dabei durch eine der durch *Curtis* [CKO92] vorgestellten Perspektiven angeboten.

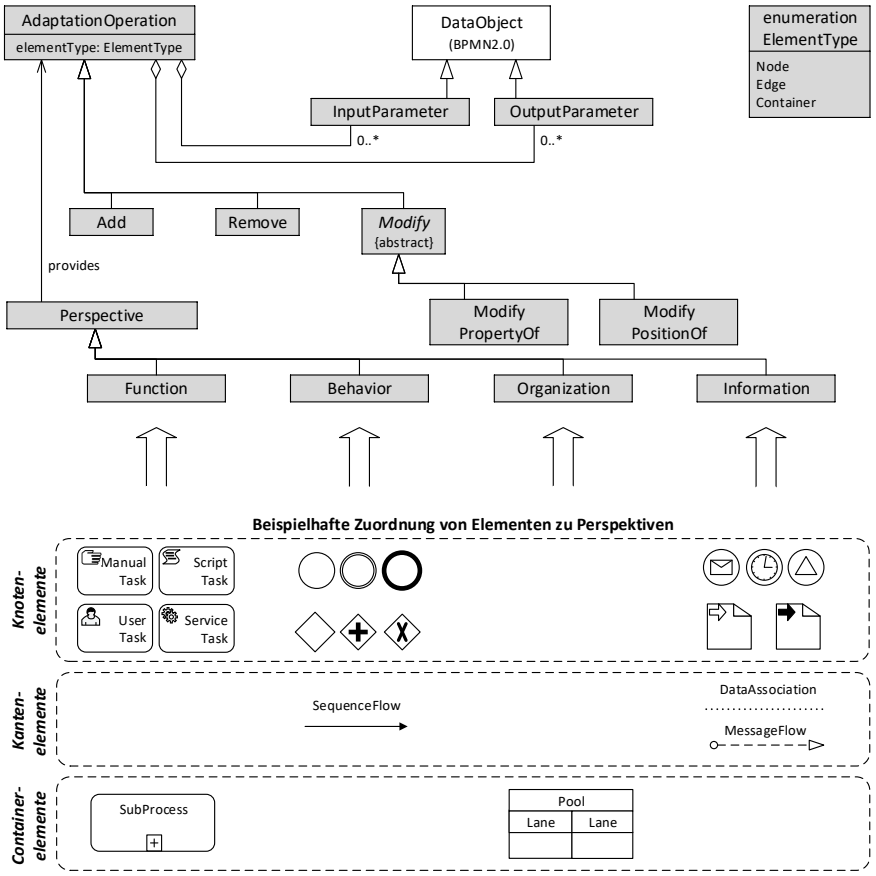


Abbildung 4-16: Operationen zur Anpassung von Prozessen in Anlehnung an eine Zuordnung von BPMN2.0-Elementen zu Perspektiven (AVM4BPM)

Klassifikation von Elementen

In jeder Perspektive sind Elemente enthalten, die sich in drei grundlegende Typen klassifizieren lassen. So existieren *Knotenelemente*, *Kantenelemente* und als spezielle Variante von Knotenelementen auch *Containerelemente*. Ein *Containerelement* kann dabei wiederum weitere Elemente der genannten Typen enthalten. Im unteren Bereich sind einige dieser Elemente in ihrer jeweiligen Notation in Anlehnung an die Zuordnung zu einer Perspektive dargestellt.

Für die von *Curtis* [CKO92] vorgestellten Perspektiven lassen sich spezifische Operationen in Anlehnung an die zuvor vorgestellten Basisoperationen *Add, Remove* und *Modify* beschreiben. Dabei wurden die beiden zusätzlichen Untertypen *ModifyPropertyOf* und *ModifyPositionOf* für den Typ *ModifyOperation* hinzugefügt. Mittels *ModifyPropertyOf* lassen sich Eigenschaften eines Elements, wie etwa der Wert der Eigenschaft *Name*, modifizieren. Dahingegen ändert die Operation *ModifyPositionOf* die Position eines Elementes entlang des Kontroll- oder Datenflusses sowie der organisationalen Einbettung.

Für jeden der genannten Typen von Elementen lassen sich auf dieser Basis Operationen bestimmen, die in Anlehnung an die Basisoperationen dann bspw. für das Hinzufügen von Knoten in der Form *AddNode* oder das Entfernen von Containerelementen in der Form *RemoveContainer* benannt werden. Diese Operationen können für die Anpassungen verschiedener Elementtypen von anpassbaren Prozessen, wie z.B. Knoten, Kanten oder Container, eingesetzt werden.

Eine Übersicht über die resultierende Menge an grundlegenden Operationen für die Anpassung von Prozessen ist in Abbildung 4-17 dargestellt. Dabei lassen sich auf Basis der zuvor beschriebenen Analyse insgesamt 24 Operationen bestimmen, die verschiedene Artefakte im *BPM-Lebenszyklus* betreffen können.

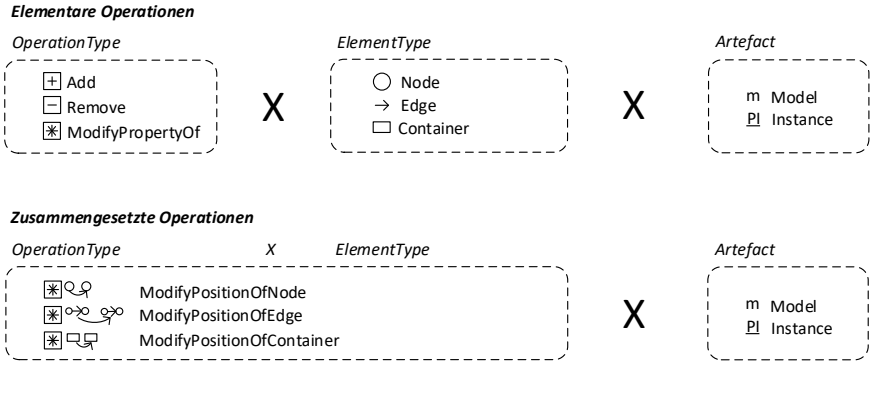


Abbildung 4-17: *Menge von Operationen zur Anpassung von Prozessen*

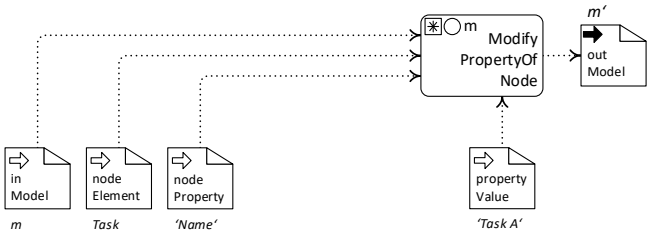
Die in Abbildung 4-17 gezeigte Übersicht beschreibt die Komposition einer Operation aus den Eigenschaften *Typ der Operation (OperationType)*, *Typ des Elements* (*ElementType*) und *betreffendes Artefakt* (*Artifact*). Dabei gilt die Besonderheit, dass für die Operationen für die Modifizierung der Position eines Elements (*ModifyPositionOf*) die Komposition bereits für den Typ

der Operation und den Typ des Elements aufgrund der Komplexität des eingesetzten Symbols komponiert worden ist. Ferner handelt es sich dabei um zusammengesetzte Operationen, die durch elementare Operationen ersetzt werden könnten.

Ein Beispiel für eine sich ergebene Operation ist in Abbildung 4-18 gezeigt. Es wird sowohl die Signatur als auch die konkrete Syntax der Operation *ModifyPropertyOfNode* beschrieben. Eine Operation vom Typ *ModifyPropertyOfNode* modifiziert den Wert einer Eigenschaft eines Knotenelements aus einem Prozessmodell.

Abbildung 4-18: Signatur und konkrete Syntax der Operation ModifyPropertyOfNode

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	nodeElement	*NodeElement*
	nodeProperty	*Property*
	propertyValue	*Value*
OUT :	*outModel*	*ProcessModel*



Signatur der Operation ModifyPropertyOfNode

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Weitere Parameter der Operation sind durch das betreffende Knotenelement (*nodeElement*), seine zu modifizierende Eigenschaft (*nodeProperty*) und den zugehörigen Wert (*propertyValue*) gegeben. Ein Bezeichner der zu modifizierenden Eigenschaft wird durch ein String-Literal angegeben. Der Typ der zu ändernden Werte ist in der dargestellten Signatur generisch als *Value* angegeben, da es verschiedene Typen wie z.B. String, Integer oder auch komplexe Datentypen geben könnte.

Anwendung der Operation ModifyPropertyOfNode

Eine Anwendung der in Abbildung 4-18 spezifizierten Operation ist in Abbildung 4-19 anhand einer schematischen und *BPMN*-spezifischen Darstellung durch ein BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. Es besteht aus insgesamt

drei Knotenelementen *N*, die in einer Sequenz durch zwei Assoziationen *E* miteinander verbunden sind. Im zugehörigen BPD sind ein Startereignis, gefolgt von einem Task und abschließend mit einem Endereignis in einer Sequenz durch Assoziationen vom Typ *SequenceFlow* verbunden. Die Anpassung von Prozessmodell *m* hin zu Prozessmodell *m'* ändert die Eigenschaft *'Name'* des Knotenelements *Task*, sodass der neue Wert der Eigenschaft *'Name' 'Task A'* ist.

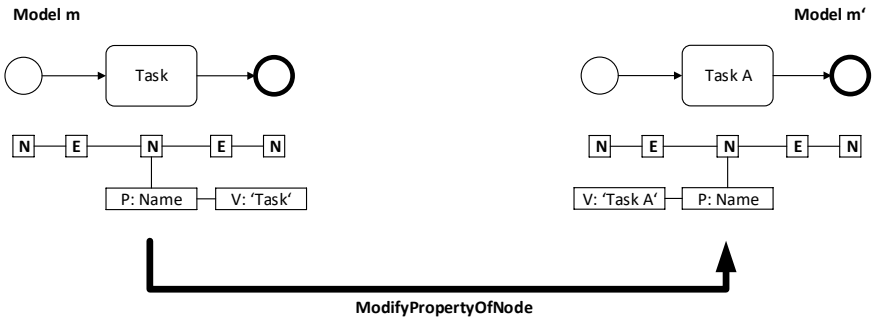


Abbildung 4-19: *Beispielhafte Anwendung der Operation ModifyPropertyOfNode*

Die resultierende Menge an Operationen enthält Beispiele für Operationen, mit denen sich unterschiedliche Artefakte von Prozessen, wie z.B. Prozessmodelle und deren Instanzen, anpassen lassen. Dabei wird an dieser Stelle angenommen, dass für ein konkretes Projekt spezifische Operationen im Rahmen des *AVM4BPM* beschrieben und anschließend implementiert werden müssen. Ein wesentlicher Grund hierfür sind plattformspezifische Gegebenheiten von IT-Unterstützungssystemen, deren Berücksichtigung im Rahmen der hier vorliegenden Beschreibung nicht sinnvoll wäre. Für weiterführende Beschreibungen von Beispielen der einzelnen Operationen durch ihre Signatur, der konkreten Syntax und operationalen Semantik wird auf Anhang A verwiesen.

4.3.4 Ereignisse

Neben Komponenten, Schnittstellen und den von ihnen angebotenen Operationen sind für die Anpassung von Prozessen zudem auch spezifische Ereignisse notwendig. Erst ihr Aufkommen löst eine mögliche Anpassung aus, sodass sie auch als Auslöser (engl. *Trigger*) bezeichnet werden. Im Rahmen dieser Arbeit werden dabei zwei verschiedene Typen von Ereignissen unterschieden.

Explizite Ereignisse Werden im Rahmen der Gestaltung von Prozessen Ereignisse als Teil des Kontroll- oder Datenflusses verwendet, so kann hier von expliziten Ereignissen ausgegangen werden. So stellen z.B. die Elemente vom Typ *StartEvent* bzw. vom Typ *EndEvent* Ereignisse dar, die den Start bzw. das Ende eines Prozesses beschreiben. Ferner sind auch weitere Ereignisse möglich, die im Verlauf des Kontroll- bzw. des Datenflusses explizit für die Auslösung einer Anpassung in der Gestaltung von Prozessen verwendet werden können. Weitere Beispiele für explizite Ereignisse sind zahlreich vertreten und werden in Auszügen in Abschnitt 4.3.4.1 näher beschrieben.

Implizite Ereignisse Es können aber auch weitere Ereignisse auftreten, die nicht im Rahmen der Gestaltung von Prozessen als Teil des Kontroll- oder Datenflusses verwendet werden können. Derartige Ereignisse sind oftmals eng gekoppelt an die operationale Semantik einzelner Elemente von Prozessen. Beispiele für derartige Ereignisse können entlang des Lebenszyklus von Aktivitäten in der Sprache *BPMN2.0* betrachtet werden. So kann z.B. die Aktivierung einer Aktivität selbst ein Ereignis darstellen, wodurch eine Anpassung von Prozessen notwendig wird. Derartige Ereignisse werden am Beispiel des Lebenszyklus von Aktivitäten in Abschnitt 4.3.4.2 beschrieben.

4.3.4.1 Explizite Ereignisse

Explizite Ereignisse stellen ein wichtiges Konzept innerhalb der Sprache *BPMN2.0* dar. Sie können explizit in der Gestaltung von Prozessen als mögliche Auslöser einer Anpassung verwendet werden. Dabei kann weiter zwischen zwei Typen von Ereignissen unterschieden werden. So existieren zum einen Ereignisse vom Typ *ThrowEvent* und zum anderen vom Typ *CatchEvent*, die entweder ein aufkommendes Ereignis darstellen oder auf ein solches Ereignis reagieren. Soll eine Anpassung an einem Prozess ausgeführt werden, können Ereignisse des Typs *ThrowEvent* zur Auslösung dessen Anpassung eingesetzt werden. Eine zu dem aufkommenden Ereignis vom Typ *ThrowEvent* zugehörige Anpassung muss dabei – bis auf wenige Ausnahmen – ein zugehöriges Ereignis vom Typ *CatchEvent* enthalten. Ausnahmen bilden Ereignisse vom Typ *CatchEvent*, die bspw. in Abhängigkeit zu einer konditionalen Auswertung stehen. Konkrete Beispiele hierfür sind durch Ereignisse vom Typ *TimerEvent* oder *ConditionalEvent* gegeben. Im Folgenden wird auf eine Auswahl durch die Sprache *BPMN2.0* gegebener Ereignisse und auf ihre Relevanz für Anpassungen von Prozessen eingegangen.

Timer

Durch Ereignisse vom Typ *TimerEvent* ist es möglich, den Aufruf einer Anpassung von Prozessen in Abhängigkeit zu Zeiträumen, Zeitpunkten oder Zyklen zu setzen. Der Typ *TimerEvent* kann somit zur Beschreibung von Ereignissen verwendet werden, für die kein zugehöriges Ereignis vom Typ *ThrowEvent* aufgekommen sein muss, da eine Auslösung in Abhängigkeit zur Auswertung einer zeitbezogenen Bedingung steht.

Message

Durch Nachrichten (engl. *Messages*) lassen sich Ereignisse beschreiben, die neben dem reinen Aufkommen eines Ereignisses zudem auch noch Informationen enthalten. Derartige Informationen können z.B. für die Prüfung hinsichtlich einer vorzunehmenden Anpassung von Prozessen eingesetzt werden. Für Nachrichten ist es üblich, dass sie neben den zu sendenden Informationen lediglich für einen bestimmten Empfänger gedacht sind.

Signal

Ein weiterer Typ von Ereignissen ist durch *Signal* gegeben. Für Ereignisse dieses Typs existieren sowohl Typen des Typs *CatchEvent* als auch des Typs *ThrowEvent*. Sie können wie Nachrichten Informationen enthalten, die für die weitere Auswertung im Rahmen einer Anpassung von Prozessen notwendig sein können. Dabei unterscheidet sich der Typ *Signal* von dem Typ *MessageEvent* insofern, als dass der Empfänger *(CatchEvent)* des Typs *Signal* nicht explizit gesetzt ist. Hierdurch ist es möglich, dass es mehrere Empfänger (*CatchEvent*) geben kann, die durch ein zugehöriges *ThrowEvent* vom Typ *Signal* ausgelöst werden können.

Error

Die Ausführung von Prozessen kann das Vorkommen von Fehlern explizit durch Ereignisse des Typs *ErrorEvent* berücksichtigen. Das Aufkommen eines solchen Ereignisses kann dabei das Starten einer Anpassung von Prozessen zum Zweck der Fehlerbehandlung auslösen.

Termination

Soll die Ausführung eines Prozesses beendet werden, obwohl Token auf parallelen Ausführungspfaden existent sind, lassen sich Ereignisse vom Typ *TerminationEvent* einsetzen. Eine solche Beendigung eines Prozesses kann dabei der Auslöser einer Anpassung von Prozessen sein.

Compensation

Werden Prozesse ausgeführt, kann ein späterer Abbruch oder eine Rückabwicklung abgeschlossener Aktivitäten notwendig sein. Ein Ereignis vom Typ *CompensationEvent* kann dabei eingesetzt werden, um eine Anpassung von Prozessen durch die Durchführung einer vordefinierten Abbruchprozedur, einer Rückabwicklung oder aber der situativen Anpassung solcher Prozeduren zu starten.

Für die zuvor beschriebenen Ereignisse existieren zum Teil zahlreiche Varianten, deren Beschreibung an dieser Stelle nicht sinnvoll wäre. Für eine vollständige Referenz wird daher auf deren Spezifikation [OMG11] ver-

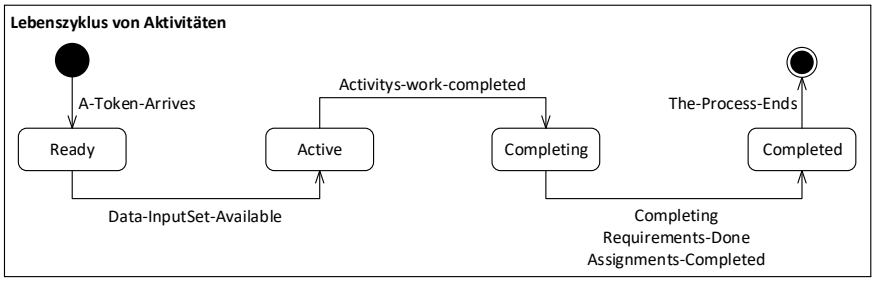
wiesen. Beispiele für den Einsatz der genannten Typen von Ereignissen werden darüber hinaus in Abschnitt 4.2 beschrieben.

4.3.4.2 Implizite Ereignisse

Ein weiterer Typ von Ereignissen ist durch implizite Ereignisse gegeben. So existieren Elemente in der Sprache *BPMN2.0*, für die eine Beschreibung ihres Lebenszyklus gegeben ist. In diesem Bezug stellt der Lebenszyklus von Aktivitäten ein wichtiges Beispiel dar. Es wird im weiteren Verlauf des Abschnitts als Referenz verwendet.

Als Einführung ist im oberen Teil von Abbildung 4-20 ein Auszug des Lebenszyklus für Aktivitäten der Sprache *BPMN2.0* in Form eines *UML Zustandsdiagramms* [OMG10] gezeigt. So werden eine Reihe von verschiedenen Zuständen durchlaufen, bevor der Lebenszyklus endet. Diese Zustände sind durch *Ready*, *Active*, *Completing* und *Completed* gegeben. Sie stellen einen Auszug für den erfolgreichen Ablauf des Lebenszyklus ohne Behandlung von Fehlern, Abbrüchen oder Kompensationen dar. Jeder Übergang von einem Zustand in den nächsten Zustand wird durch ein spezifisches Ereignis ausgedrückt. So kann der Lebenszyklus begonnen werden, wenn durch das Ereignis *A-Token-Arrives* ausgedrückt wird, dass ein Token die Aktivität erreicht hat. Auf die Bedeutung der einzelnen Zustände und Ereignisse wird im Folgenden kurz eingegangen.

Abbildung 4-20: *Lebenszyklus von Aktivitäten in der Sprache BPMN2.0 in Form eines UML Zustandsdiagramms*



Ready Eine Aktivität befindet sich im Zustand *Ready*, wenn die benötigte Anzahl an Token zur Aktivierung verfügbar gewesen ist. Ein Wechsel in diesen Zustand wird durch das Ereignis *A-Token-Arrives* ausgelöst.

Active In dem Zustand *Active* wird die für die Aktivität angedachte Funktion ausgeführt. Ein Wechsel in diesen Zustand wird durch das Ereignis *Data-InputSet-Available* ausgedrückt. Es kommt auf, wenn alle Dateneingaben verfügbar sind.

Completing Wurde die für die Aktivität angedachte Funktion erfüllt und die Ausführung beendet, wechselt der Lebenszyklus in den Zustand *Completing*. Das zugehörige Ereignis ist *Activitys-work-completed*. In dem Zustand *Completing* werden assoziierte Abhängigkeiten, wie z.B. gebundene Behandlung von Ereignissen, aufgelöst.

Completed In den Zustand *Completed* wird gewechselt, sobald gebundene Abhängigkeiten aufgelöst worden sind. Das zugehörige Ereignis ist dabei durch *Completing-Requirements-Done-Assignments-Completed* gegeben. Der dargestellte Verlauf endet, wenn bereitgestellte Token von nachfolgenden Elementen übernommen worden sind. Das zugehörige Ereignis ist durch *The-Process-Ends* gegeben.

Die zuvor beschriebenen Zustandsübergänge und Ereignisse können genutzt werden, um spezifisches Verhalten der Anpassungslogik aufzurufen. Sollen Nutzer und Domänenexperten durch die Verwendung ihrer gewohnten Sprache in Form von *BPMN2.0* unterstützt werden, ist eine mögliche Lösung durch die Transformation von Zuständen und Ereignissen des *UML Zustandsdiagramms* (siehe Abbildung 4-20) in explizite Ereignisse der Sprache *BPMN2.0* möglich.

Transformierte Ereignisse

Ein mögliches Ergebnis einer solchen Transformation ist in Abbildung 4-21 in Form eines BPD gezeigt. Das dargestellte BPD besteht aus Ereignissen, die zu einer der in Abbildung 4-20 entsprechenden Sequenz verbunden worden sind. Dabei wurden dem dargestellten Kontrollfluss für jeden Zustand des *UML Zustandsdiagramms* und für die Start- und Endknoten ein Ereignis vom Typ *Signal* hinzugefügt. Bei dem Ereignis handelt es sich um ein Zwischenereignis (*Intermediate*) des Typs *ThrowEvent* (siehe Abschnitt 4.3.4.1). Jedes Ereignis stellt den unmittelbaren Wechsel in den jeweiligen Zustand bzw. den Start oder das Ende der Aktivität dar.

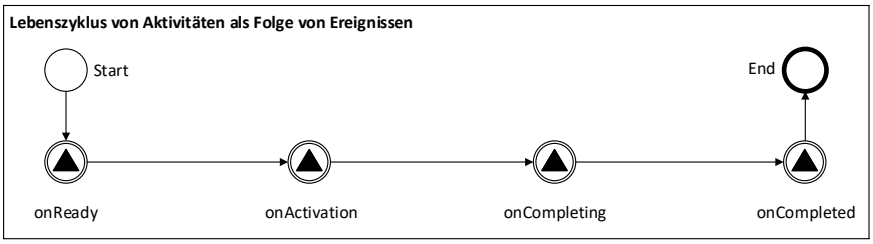


Abbildung 4-21: *Lebenszyklus von Aktivitäten in der Sprache BPMN2.0 als Folge von Ereignissen in Form eines BPD*

Als Ergänzung zu der Transformation von Zuständen können auch die Ereignisse des *UML Zustandsdiagramms* durch explizite Ereignisse der Sprache *BPMN2.0* dargestellt werden. In alternativen Transformationen könnten die zusätzlichen Ereignisse z.B. auch beim Verlassen eines Zustandes eingefügt werden. Hierbei wäre die Namensgebung der Ereignisse von der Form *on** zu *after** anzupassen. Hierdurch würde z.B. das Ereignis *onReady* nachgefolgt durch das Ereignis *afterReady*. Problematisch wäre dieses Vorgehen im Fall des Ereignisses vom Typ *EndEvent*, da hier der Kontrollfluss bereits beendet wäre.

Abstrakte Syntax für transformierte Ereignisse

Für den vollständigen Lebenszyklus von Aktivitäten in der Sprache *BPMN2.0* ist ein Konzept zur Integration von impliziten Ereignissen hinsichtlich der Zustände für die Lösungsvariante *on** in Abbildung 4-22 dargestellt. Der Lebenszyklus *(Lifecycle)* wird hier als Teil einer Aktivität *(Activity)* dargestellt. Der Lebenszyklus besteht aus einer Reihe von Zuständen *(LifecycleState)*, die wiederum einen spezifischen Namen besitzen. Die durch die im Rahmen der Spezifikation der Sprache *BPMN2.0* eingeführten Zustände des Lebenszyklus von Aktivitäten sind hier durch die Enumeration *StateName* dargestellt. Zu jedem dieser Zustände wird ein neues Ereignis vom Typ *LifecycleEvent* eingeführt, welches das Wechseln in den jeweiligen Zustand ankündigt *(pronounces)*. Die Ereignisse werden gemäß ihrer Zugehörigkeit zu Zuständen des Lebenszyklus benannt. So wird z.B. ein Ereignis, das das Betreten des Zustandes *Ready* signalisiert, mit dem Bezeichner *onReady* dargestellt. Die Enumeration *EventName* beschreibt die zugehörigen Ereignisse hinsichtlich des Betretens von Zuständen.

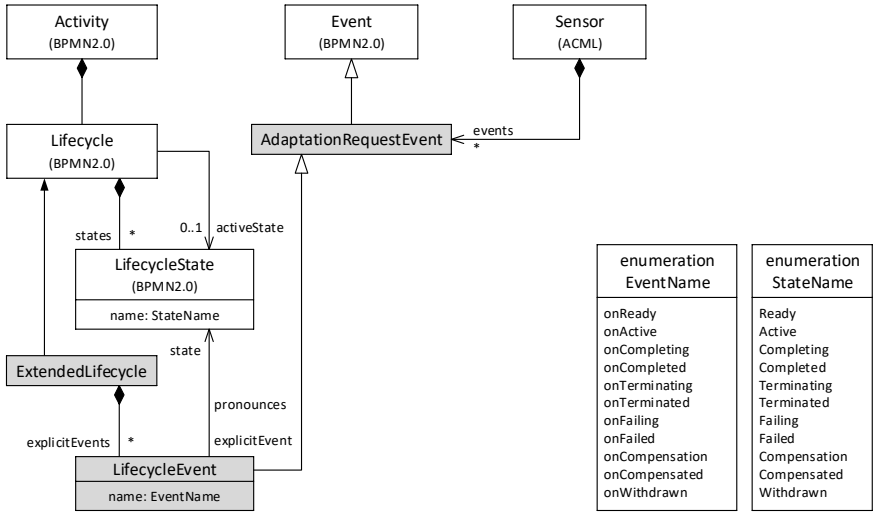


Abbildung 4-22: Integration von impliziten Ereignissen (AVM4BPM)

Bei den transformierten Ereignissen (*LifecycleEvent*) handelt es sich ferner um Ereignisse des Typs *AdaptationRequestEvent* (siehe Abschnitt 4.3.2), die durch Sensorschnittstellen zur Verfügung gestellt werden können. Hierdurch können auch implizite Ereignisse in der Gestaltung von anpassbaren Prozessen berücksichtigt werden, indem sie durch die Sprache *ACML4BPM* als Teil dieser Schnittstellen beschrieben werden.

Neben den in Abbildung 4-22 dargestellten Ereignissen können weitere implizite Ereignisse für die Auslösung von Anpassungen sinnvoll sein. Dabei können aufgrund der Verwendung von Ereignissen des Typs *Signal* mehrere Anpassungen pro aufkommendes Ereignis angestoßen werden. Sollen zeitgleich mehrere Instanzen eines Prozesses aktiv sein, kann daher eine *Unifizierung* des Ereignisses notwendig sein. Ferner kann der Lebenszyklus weitere Zustandswechsel durchführen, obwohl die Ausführung einer Anpassung noch aktiv ist. Daher kann es erforderlich sein, im Anschluss an eine Anpassung eine Rückkopplung durchzuführen.

Transformierte Ereignisse mit Unifizierung und Rückkopplung

Eine Lösungsvariante unter Verwendung von Rückkopplungen ist in Abbildung 4-23 dargestellt. In dem hier dargestellten BPD ist erneut ein Auszug des Lebenszyklus einer Aktivität auf Basis des in Abbildung 4-20 vorgestellten Ablaufs gezeigt. Dabei wurden jedoch nicht Ereignisse des Typs *Signal* verwendet. Um eine *Unifizierung* und eine Rückkopplung zu ermöglichen werden stattdessen Nachrichten (*Message*) eingesetzt.

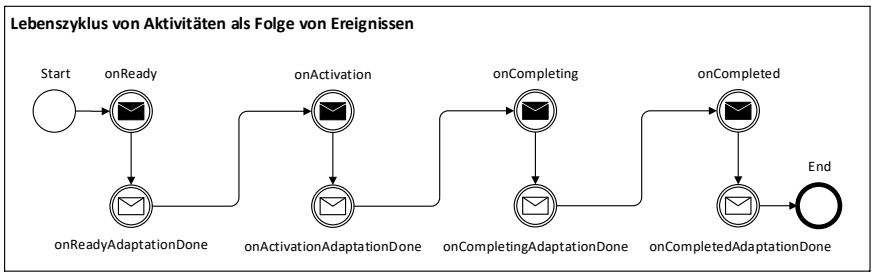


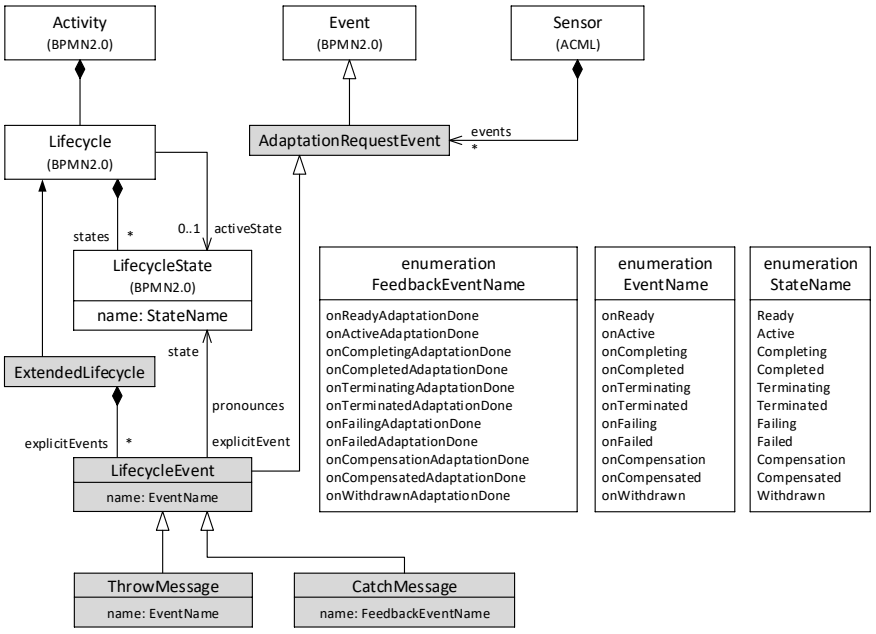
Abbildung 4-23: *Lebenszyklus von Aktivitäten als Folge von Ereignissen in Form eines BPD mit Verwendung von Rückkopplung*

Durch die Eigenschaft, dass Nachrichten neben zu übermittelnden Informationen auch einen speziellen Empfänger enthalten, können ungewollte Seiteneffekte beim Vorhandensein mehrerer Instanzen reduziert werden. Für die Rückkopplung im Fall einer abgeschlossenen Anpassung wurde pro Zustand ein Ereignis des Typs *CatchEvent* eingefügt. Im Rahmen des Kontrollflusses wird somit stets vor dem Betreten eines Zustandes ein Ereignis zur Anpassung ausgelöst. Anschließend wird auf eine Rückkopplung durch ein weiteres Ereignis gewartet, das über die Abgeschlossenheit einer Anpassung informiert.

Abstrakte Syntax für transformierte Ereignisse

Das Konzept zur Integration von impliziten Ereignissen unter Verwendung von Rückkopplung ist in Abbildung 4-24 dargestellt. Wie zuvor bereits beschrieben, wurden die beiden Typen *ThrowEvent* und *CatchEvent* für Nachrichten *(Message)* verwendet. Ereignisse vom Typ *ThrowMessage* werden zur Auslösung einer möglichen Anpassung verwendet. Ferner werden Ereignisse vom Typ *CatchEvent* zur Rückkopplung mit einer möglichen Anpassung verwendet. Die Benennung der Ereignisse ist in Anlehnung an den jeweiligen Zustand gewählt worden, in den nach der Rückkopplung gewechselt werden soll. Eine Übersicht gibt hierzu die Enumeration *FeedbackEventName*.

Abbildung 4-24: Integration von impliziten Ereignissen mit Rückkopplung (AVM4BPM)



Konkrete Syntax von impliziten Ereignissen

Für implizite Ereignisse wurde keine gesonderte konkrete Syntax in dieser Arbeit entwickelt. Dies lässt sich dadurch begründen, dass implizite Ereignisse in explizite Ereignisse der Sprache *BPMN2.0* transformiert werden können. In diesem Bezug werden für transformierte implizite Ereignisse die konkrete Syntax von expliziten Ereignissen eingesetzt.

Unidirektionale Transformation

Für die Integration von impliziten Ereignissen in das *AVM4BPM* der Sprache *ACML4BPM* wurde eine methodische Transformation beschrieben. Die Transformation sieht die Abbildung des Betretens von Zuständen des Lebenszyklus von Aktivitäten auf explizite Ereignisse vor. Da es sich um eine unidirektionale Transformation handelt, kann ein erweiterter Aufwand

bei der Auswahl einer Plattform, der zugehörigen Konfiguration und Implementierung bestehen. Eine Lösung im Rahmen einer Implementierung kann durch die Verwendung von *Event-Handlern* durchgeführt werden. Da in dieser Arbeit die frühe Gestaltung von anpassbaren Prozessen fokussiert wird, liegt dies an dieser Stelle außerhalb des für diese Arbeit gesetzten Rahmens.

Sonstige Ergänzung

Neben den beiden zuvor beschriebenen Varianten lassen sich auch andere implizite Ereignisse in der Sprache *ACML4BPM* berücksichtigen. Ein Beispiel ist hier durch Elemente aus der Perspektive *Information* gegeben. So können die Initialisierung oder die Löschung von Datenobjekten ebenfalls durch Ereignisse dargestellt werden, bei deren Aufkommen eine Anpassung von Prozessen notwendig sein kann. Dabei existieren in der Sprache *BPMN2.0* nicht immer etwaige Lebenszyklen, auf deren Basis eine Erweiterung möglich ist. In diesen Fällen muss ein Lebenszyklus entsprechend gegebener Anforderungen durch Nutzer oder Domänenexperten in enger Anlehnung an das später eingesetzte IT-Unterstützungssystem entwickelt werden.

4.4 Zusammenfassung

In den vorherigen Abschnitten wurde die Sprache *Adapt Case Modeling Language 4 BPM* (*ACML4BPM*) vorgestellt. Sie kann für die Gestaltung von anpassbaren Prozessen und zur Durchführung des *Separation-of-Concerns* hinsichtlich der Anwendungs- und Anpassungslogik eingesetzt werden. Die Sprache basiert auf dem durch *Luckey* beschriebenen Ansatz *Adapt Cases* [Luc+11] (siehe Abschnitt 2.4).

Die entwickelte Sprache gliedert sich dabei in zwei Teilsprachen zur Erstellung des *Adapt Case Model 4 BPM* (siehe Abschnitt 4.2) und des *Adapt Case View Model 4 BPM* (siehe Abschnitt 4.3). Beide Teilsprachen berücksichtigen spezifische Konzepte der Domäne *BPM*. In diesem Abschnitt wird kurz auf die Vollständigkeit der Sprache hinsichtlich der in Abschnitt 4.1 gegebenen Fragestellungen eingegangen. Hierzu ist in Tabelle 4-1 eine Auflistung von zugehörigen Zielen sowie deren Erfüllung dargestellt. Eine Erläuterung sowie Diskussion werden im Folgenden gegeben.

Allgemein lässt sich die Frage nach der Vollständigkeit nur in Abhängigkeit zu dem gewählten Fixpunkt beantworten. In den vorherigen Abschnitten wurden für verschiedene Elemente der Sprache *ACML4BPM* Fixpunkte gewählt, die entweder durch Artefakte entlang des *BPM-Lebenszyklus* oder aber durch eine existierende Repräsentation der Domäne *BPM* in

Tabelle 4-1: Übersicht über gesetzte Ziele und deren Erfüllung für die entwickelte Sprache zur Gestaltung von anpassbaren Prozessen

Fragestellung	Ziel	Erfüllung
1	Identifikation von relevanten System- und Umgebungskomponenten in der Domäne *BPM*	✓
2	Identifikation von Schnittstellen für den Zugriff auf Informationen innerhalb dieser System- und Umgebungskomponenten	✓
3	Beschreibung von Operationen zur adäquaten Anpassung der Eigenschaften dieser System- und Umgebungskomponenten	✓
4	Identifikation möglicher Ereignisse zur Beschreibung der Notwendigkeit einer Anpassung von Eigenschaften dieser System- und Umgebungskomponenten	✓

Form der Sprache *BPMN2.0* gegeben wurden. Es ist klar, dass durch die Wahl anderer Fixpunkte als Analysegrundlage, wie z.B. für die Repräsentation der Domäne *BPM* die Sprache der *UML Aktivitätsdiagramme* [OMG15b], *BPEL* [OAS07] oder neuere Sprachen wie *CMMN* [OMG16a] oder *DMN* [OMG16b], andere Elemente einer Sprache zur Gestaltung von anpassbaren Prozessen notwendig werden. Die zuvor beschriebenen Inhalte können in diesem Bezug als Leitfaden für ein mögliches Vorgehen beim *Adaptivity Engineering* auf Basis des durch *Luckey* [Luc+11; LE13] vorgestellten Ansatzes *Adapt Cases* dienen und sollten bei der nachfolgenden Diskussion berücksichtigt werden.

System und Umgebungskomponenten der Domäne BPM

Aus der Arbeit von *Luckey* lässt sich ableiten, dass es zwei grundlegende Typen von Komponenten geben kann. Dabei handelt es sich zum einen um Systemkomponenten, auf denen der Fokus der Betrachtung liegt, und zum anderen um Umgebungskomponenten, mit denen die Systemkomponenten interagieren. Sollen anpassbare Prozesse gestaltet und ausgeführt werden, können für die Domäne *BPM* zwei wesentliche Systemkomponenten benannt werden. Es handelt sich hierbei um Systemkomponenten zur Kapselung von Prozessmodellen und zur Kapselung von Prozessinstanzen. Derartige Systemkomponenten können mit einer Vielzahl an unterschiedlichen Umgebungskomponenten interagieren. Diese können entweder durch verschiedene reine Softwaresysteme, wie z.B. Anwendungs- und Datenbanksysteme, gegeben sein oder weitere (Misch-)Systeme, die eine Kopplung mit physischen bzw. real-weltlichen Entitäten, wie z.B. Maschinen, bilden. Auf die Frage nach relevanten System- und Umgebungskomponenten konnte somit in Abschnitt 4.3.1 durch die Einführung spezifischer Typen von Komponenten eine Antwort gegeben werden. Es sei hierbei jedoch angemerkt, dass durch eine Veränderung des gewählten Fixpunktes oder für die Entwicklung einer konkreten prozessspezifischen Anwendung die Notwendigkeit für die Einführung zusätzlicher Typen von

Umgebungskomponenten gegeben ist. Da der Fokus der vorliegenden Arbeit jedoch auf der Gestaltung von flexiblen und anpassbaren Prozessen liegt, wird auf diesen Aspekt nicht weiter eingegangen. Beispiele für mögliche Umgebungskomponenten werden jedoch im Rahmen der Evaluation und des dort verwendeten Szenarios gegeben (siehe Abschnitt 7.1).

Schnittstellen

Gemäß des Ansatzes *Adapt Cases* sind für die Gestaltung von Anpassungsregeln die beiden Schnittstellentypen *Sensor* und *Effector* notwendig. Sie ermöglichen zum einen den kontrollierten lesenden Zugriff auf Eigenschaften der durch die System- und Umgebungskomponenten gekapselten Inhalte und zum anderen die Funktion der Anpassung von Eigenschaften dieser Inhalte. Hierzu wurden in Abschnitt 4.3.2 in Anlehnung an die identifizierten System- und Umgebungskomponenten entsprechende Schnittstellen vorgestellt.

Operationen zur Anpassung

Die vorgestellten Typen von Schnittstellen bieten Operationen zur Anpassung von gekapselten Eigenschaften an. Analog zu der durch [Ger13] vorgestellten Arbeit wurde sich hier für einen Fixpunkt in Form einer Sprache (hier: *BPMN2.0*) zur Gestaltung von Prozessen entschieden. Bei der Auswahl des Fixpunktes wurden die durch *Curtis* [CKO92] eingeführten vier Perspektiven von Prozessen als Analysegrundlage verwendet. Es konnte gezeigt werden, dass existierende Arbeiten sich lediglich auf einige wenige Perspektiven konzentrieren und damit die Anpassung wesentlicher Inhalte von Prozessen nicht ermöglichen. Daher wurden in Anlehnung an die Perspektiven Operationen zur Anpassung von Prozessen eingeführt, die eine Anwendung auf die Elemente aller durch *Curtis* [CKO92] vorgestellten Perspektiven bietet. Es wird davon ausgegangen, dass durch diese Vorgehensweise eine Abdeckung der vier Perspektiven ermöglicht werden kann. Hierdurch können für ausgesuchte Elemente adäquate Anpassungen an den durch die System- und Umgebungskomponenten gekapselten Inhalte ermöglicht werden.

Ereignisse zur Auslösung einer Anpassung

Im Rahmen des Ansatzes *Adapt Cases* stellen Ereignisse den wesentlichen Mechanismus zur Andeutung einer potentiell notwendigen Anpassung dar. Daher wurden in Anlehnung an die durch die Sprache *BPMN2.0* gegebene Repräsentation der Domäne *BPM* verschiedene Klassen von Ereignissen vorgestellt, die entweder explizit oder implizit die Notwendigkeit einer Anpassung andeuten können. Durch die beiden gegebenen Klassen von Ereignissen lässt sich eine Vielzahl an möglichen auslösenden Ereignissen gestalten.

Wie zuvor diskutiert bieten die in diesem Abschnitt vorgestellten Inhalte einen möglichen Ansatz zur Gestaltung von anpassbaren Prozessen unter

Verwendung des De-facto-Standards *BPMN2.0*. Dabei ist eine vollständige Abdeckung einer Repräsentation einer Domäne nur bedingt möglich. Daher wurde versucht, auf verschiedene Perspektiven von Prozessen Bezug zu nehmen, sodass das beschriebene Vorgehen als ein Leitfaden für das *Adaptivity Engineering* in Bezug zu anpassbaren Prozessen betrachtet werden kann.

Kapitel **5**

Entwurfsmuster für flexible und anpassbare Prozesse

Neben der Anpassung von Prozessen durch entsprechende Operationen wird in dieser Arbeit zudem auch Flexibilität betrachtet, die in Prozessen auf weitere Arten umgesetzt werden kann. Für diese unterschiedlichen Arten von Flexibilität existieren in der Literatur verschiedene Arbeiten, in denen Flexibilität in Prozessen beschrieben wird. Dieser Abschnitt befasst sich mit der Vorstellung einiger ausgesuchter Arten von Flexibilität, die nachfolgend auch Entwurfsmuster genannt werden. Dabei liegt der Fokus insbesondere auf derartigen Entwurfsmustern, die explizit Entscheidungspunkte vorsehen, sodass eine Trennung der Anpassungs- und Anwendungslogik sinnvoll sein kann. Nachfolgend werden in diesem Bezug in Abschnitt 5.1 zunächst eine Reihe von Forschungsfragen und die weitere Struktur des Kapitels vorgestellt. Die erarbeiteten Entwurfsmuster werden anschließend in den Abschnitten 5.2 bis 5.5 beschrieben. Dabei wird jeweils eine spezifische Analyse des Typs an Flexibilität durchgeführt. Darauf aufbauend wird für die Gestaltung von flexiblen und anpassbaren Prozessen unter Verwendung der Sprache *ACML4BPM* ein Lösungsweg vorgestellt. Das Kapitel schließt in Abschnitt 5.6 mit einer Zusammenfassung der vorgestellten Entwurfsmuster ab.

5.1 Übersicht

Unter Verwendung der in Kapitel 4 eingeführten Sprache *ACML4BPM* wird nachfolgend gezeigt, wie die Trennung von unterschiedlichen Typen von Verhalten in der Gestaltung von Prozessen vorgenommen werden kann. Hierbei wird insbesondere auf die Einhaltung der Anforderung des

Separation-of-Concerns (*SoC*) hinsichtlich der Trennung der Anpassungs- und Anwendungslogik eingegangen. Ferner wird versucht, die nachfolgenden Forschungsfragen im Rahmen der einzelnen Abschnitte zu beantworten.

Fragestellung 1 *Was sind Flexibilitätsaspekte von Prozessen?*

Fragestellung 2 *Wie kann die Gestaltung von flexiblen und anpassbaren Prozessen hinsichtlich bestehender Flexibilitätsaspekte durch die Sprache ACML4BPM in Form von Entwurfsmustern unterstützt werden?*

Fragestellung 3 *Werden für diese Unterstützung Erweiterungen der Sprachen BPMN2.0 und ACML4BPM notwendig?*

Fragestellung 4 *Werden für diese Unterstützung Erweiterungen von Methoden zur Gestaltung notwendig?*

Als Grundlage für einen möglichen Suchraum für die Auswahl von verschiedenen Arten von Flexibilität dienten dabei die Arbeiten von [Sch+08] und [WRR08; RW12]. Bei den Arbeiten handelt es sich jeweils um zentrale Arbeiten hinsichtlich der Flexibilität von Prozessen in der wissenschaftlichen Domäne *BPM*. Insbesondere die Arbeit von [Sch+08] stellte sich als die vielversprechendste Arbeit hinsichtlich verschiedener Arten von Flexibilität heraus, die durch die Sprache *ACML4BPM* bereits frühzeitig in der Gestaltung von flexiblen und anpassbaren Prozessen durch entsprechende Entwurfsmuster unterstützt werden können. Dabei sind beide Gruppen von Arbeiten hinsichtlich der Arten von Flexibilität in Prozessen teilweise deckungsgleich. Die Arbeiten von [WRR08; RW12] sind jedoch an spezifischen Stellen derartig umfangreich, dass sich stattdessen für eine Definition von Entwurfsmustern auf Basis von [Sch+08] entschieden werden musste. In Abbildung 5-1 ist eine Übersicht über die in der vorliegenden Arbeit betrachteten Entwurfsmuster für flexible Prozesse dargestellt.

Im Folgenden wird detaillierter auf die in Abbildung 5-1 abgebildeten Entwurfsmuster zur Realisierung verschiedener Arten von Flexibilität in Prozessen eingegangen. Hier wird zwischen zwei Gruppen von Entwurfsmustern unterschieden. So lässt sich Flexibilität bereits früh in der Phase *Design & Analyse* des *BPM-Lebenszyklus* von Prozessen in Form des Flexibilitätsaspekts *Flexibility-by Design* durch eine geeignete Art zu gestalten umsetzen. Hierauf wird in Abschnitt 5.2 detailliert eingegangen. Anschließend werden Entwurfsmuster der zweiten Gruppe vorgestellt, für die zum einen Spracherweiterungen und zum anderen zusätzliche Operationen zur

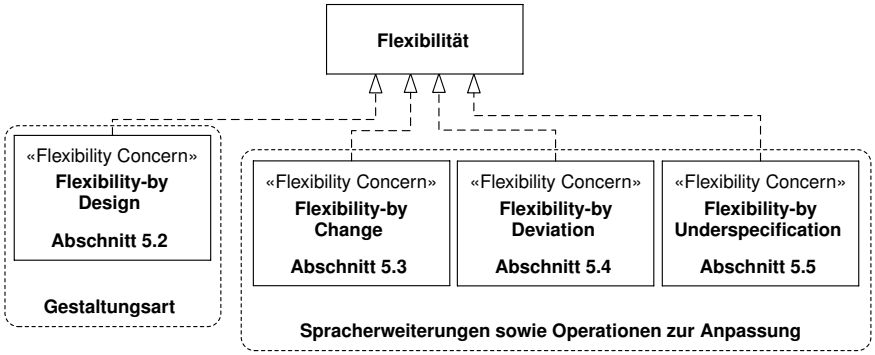


Abbildung 5-1: *Übersicht über Aspekte von flexiblen und anpassbaren Prozessen*

Anpassung notwendig sind. Hierzu gehören zunächst die Flexibilitätsaspekte *Flexibility-by Change* (siehe Abschnitt 5.3) und *Flexibility-by Deviation* (siehe Abschnitt 5.4). Abschließend wird in Abschnitt 5.5 der letzte Flexibilitätsaspekt *Flexibility-by Underspecification* vorgestellt, der einige Inhalte der zuvor genannten Flexibilitätsaspekte wiederverwendet.

5.2 Flexibility-by Design

Im Rahmen von *Flexibility-by Design* werden bereits verschiedene Entwurfsmuster für flexible und anpassbare Prozesse zur Verfügung gestellt. Dabei wird sich vornehmlich auf Aspekte von Flexibilität und Anpassbarkeit konzentriert, die sich durch ein geeignetes Vorgehen in der Phase *Design & Analyse* in Hinsicht auf die Gestaltung von Prozessen in Form von Prozessmodellen bezieht. *Schonenberg et. al* [Sch+08] motivieren *Flexibility-by Design* durch die Anforderung, verschiedene Alternativen für Ausführungspfade zu unterstützen. Hierdurch kann zur Ausführungszeit der Ausführungspfad ausgewählt werden, der für den jeweiligen Kontext am geeignetsten ist. Eine an *Schonenberg et. al* [Sch+08] angelehnte Definition des Flexibilitätsaspekts *Flexibility-by Design* wird in Definition 5.2.1 gegeben.

Definition 5.2.1. *(Flexibility-by Design)*

Flexibility-by Design beschreibt die Fähigkeit zur Integration von alternativen Ausführungspfaden innerhalb eines Prozessmodells in der Phase Design & Analyse des BPM-Lebenszyklus. Dabei wird das Ziel verfolgt, in der Phase Ausführung, also zur Zeit der Ausführung, einen geeigneten Ausführungspfad wählen zu können.

Flexibility-by Design lässt sich in unterschiedliche Aspekte der Gestaltung unterteilen, die bereits frühzeitig in der Phase *Design & Analyse* des *BPM-Lebenszyklus* berücksichtigt werden können. Eine Übersicht über diese Aspekte ist in Abbildung 5-2 gegeben. Eine für diese Arbeit verwendete Interpretation dieser Aspekte sowie deren Analyse hinsichtlich einer möglichen Verwendung der Sprache *ACML4BPM* in ihrer Gestaltung wird in Abschnitt 5.2.1 detaillierter erläutert. In den nachfolgenden Abschnitten 5.2.2 bis 5.2.4 werden Konzepte für die Gestaltung einzelner Aspekte unter Verwendung der Sprache *ACML4BPM* vorgestellt. Abschließend wird in Abschnitt 5.2.5 eine Zusammenfassung sowie eine Diskussion gegeben.

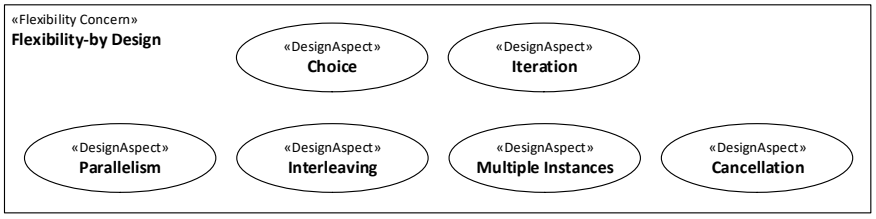


Abbildung 5-2: Gestaltungsaspekte für flexible und anpassbare Prozesse in Hinsicht auf Flexibility-by Design

5.2.1 Gestaltungsaspekte von Flexibility-by Design

In diesem Abschnitt werden die sechs Aspekte (siehe Abbildung 5-2) zur Gestaltung von flexiblen Prozessen in Hinsicht auf den Flexibilitätsaspekt *Flexibility-by Design* vorgestellt. Das Ziel soll hierbei die Analyse hinsichtlich der Erkennung des Potentials zur Trennung von Anpassungs- und Anwendungslogik sein, sodass die in Kapitel 4 vorgestellte Sprache *ACML4BPM* unterstützend in der Gestaltung eingesetzt werden kann. Hierzu werden zunächst die Aspekte zur Gestaltung in den Abschnitten 5.2.1.1 bis 5.2.1.6 beschrieben und bewertet. In Abschnitt 5.2.1.7 wird mit einer Zusammenfassung hinsichtlich einer möglichen Verwendung von der Sprache *ACML4BPM* abgeschlossen.

5.2.1.1 Choice

Unter dem Aspekt *Choice* wird die Fähigkeit verstanden, im Rahmen des Kontrollflusses spezifische Entscheidungspunkte zu beschreiben, durch deren Auswertung alternative Kontrollflusspfade gewählt werden können. Beispiele für konkrete Entscheidungspunkte in der Sprache *BPMN2.0* sind durch Knotenelemente der Typen *Inclusive Gateway* und *Exclusive Gateway* (siehe Abschnitt 2.3.4) gegeben.

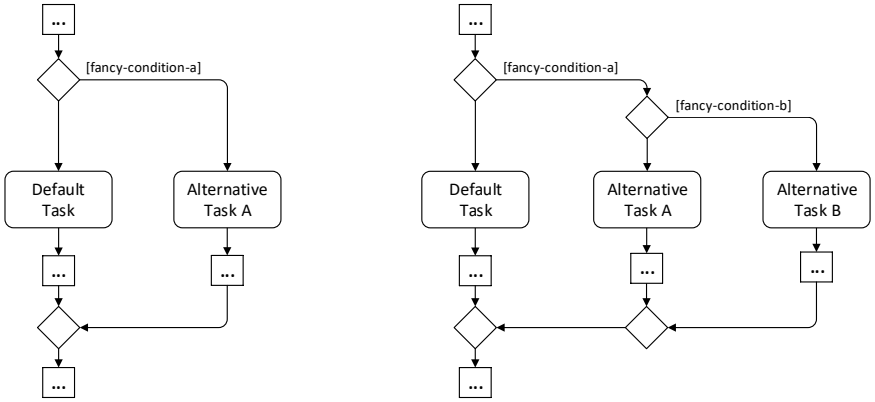


Abbildung 5-3: *Beispiele für den Aspekt Choice in der Sprache BPMN2.0*

Beispiele für den Aspekt Choice

Zur besseren Veranschaulichung des Aspekts *Choice* werden in Abbildung 5-3 zwei Beispiele gegeben. Im linken Bereich wird hierzu eine einfache Auswahl von zwei Kontrollflusspfaden dargestellt. Dabei wird auf Basis einer Auswertung der Bedingung *fancy-condition-a* einer der beiden dargestellten Kontrollflusspfade gewählt. Hierdurch ist es möglich, flexibel eine Auswahl einer Ausführung des Tasks mit der Bezeichnung *Default Task* oder *Alternative Task* zu treffen. Im rechten Bereich ist ein erweitertes Beispiel für den Aspekt *Choice* dargestellt. So können im Rahmen einer geschachtelten Auswahl, zunächst bestehend aus der Bedingung *fancy-condition-a* und anschließend aus der Bedingung *fancy-condition-b*, verschiedene Kontrollflusspfade gewählt werden.

Bewertung für den Aspekt Choice

Flexibilität ist in den Beispielen in Abbildung 5-3 durch die Auswahl verschiedener Kontrollflusspfade gegeben. Dabei werden Anwendungs- und Anpassungslogik gemeinsam integriert dargestellt. Dies kann z.B. insbesondere im Rahmen von einer großen Anzahl an Möglichkeiten zur Auswahl oder tieferen Verschachtelungen sowohl die Wartbarkeit als auch das Verständnis von in Abhängigkeit stehender Umstände erschweren. Hierzu wird in Abschnitt 5.2.2 eine alternative Darstellung unter Verwendung der in Kapitel 4 eingeführten Sprache *ACML4BPM* gegeben.

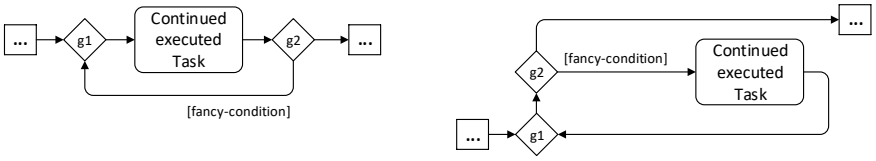
5.2.1.2 Iteration

Der Aspekt *Iteration* beschreibt die Fähigkeit, dass ein einzelner Kontrollflusspfad wiederholt durchlaufen werden kann. Typischerweise wird hierbei an einem Entscheidungspunkt eine Bedingung ausgewertet, dessen Ergebnis einen wiederholten Durchlauf eines Kontrollflusses bedingt. Der Aspekt *Iteration* kann dabei als spezielle Variante des Aspekts *Choice* verstanden werden, da ebenfalls in Anlehnung an die Auswertung einer Bedingung ein Kontrollflusspfad gewählt werden kann.

Beispiele für den Aspekt Iteration

In Abbildung 5-4 sind hierzu zwei Beispiele für den Aspekt *Iteration* in der Sprache *BPMN2.0* dargestellt. Im linken Bereich ist eine sogenannte fußgesteuerte und im rechten Bereich eine kopfgesteuerte Iteration dargestellt. Beide Iterationen unterscheiden sich darin, an welcher Stelle im Kontrollfluss eine Bedingung für den Abbruch ausgewertet wird. So ist im Fall einer fußgesteuerten Iteration, wie hier dargestellt, eine Ausführung des Tasks mit der Bezeichnung *Continued executed Task* stets vor der Auswertung der Bedingung *fancy-condition*. Im Fall der kopfgesteuerten Iteration wird vor dem Betreten einer jeden Iteration geprüft, ob die Bedingung *fancy-condition* noch erfüllt ist. Ist dem nicht so, wird der gesamte Vorgang abgebrochen und weiter im Kontrollfluss verfahren.

Abbildung 5-4: Beispiele für den Aspekt Iteration in der Sprache BPMN2.0



Bewertung für den Aspekt Iteration

Flexibilität ist in dem dargestellten Beispiel des Aspekts *Iteration* gegeben, da jede weitere Iteration die Einbindung eines zusätzlichen Teils im Kontrollfluss bedingt. Diese Art von Flexibilität ist somit sehr ähnlich zu dem Aspekt *Choice*. Ferner werden Anwendungs- und Anpassungslogik in dem dargestellten Beispiel nicht voneinander getrennt, wodurch bereits bekannte Probleme auftreten können (siehe 5.2.2). Wie der Aspekt *Iteration* durch die Sprache *ACML4BPM* gestaltet werden kann, wird in Abschnitt 5.2.3 beschrieben.

5.2.1.3 Parallelism

Durch den Aspekt *Parallelism* ist es möglich, mehrere Kontrollflusspfade zu beschreiben, deren spätere Ausführung parallel stattfinden soll. Hierfür werden Teilungs- (engl. *Fork*) und Vereinigungspunkte (engl. *Join*) benötigt, an denen ein einzelner Kontrollflusspfad geteilt bzw. mehrere Kontrollflusspfade vereinigt werden. Ein Beispiel eines Sprachelements aus der Sprache *BPMN2.0* ist durch das Knotenelement des Typs *ParallelGateway* gegeben (siehe Abschnitt 2.3.4).

Beispiel für den Aspekt Parallelism

In Abbildung 5-5 ist ein Beispiel des Aspekts *Parallelism* in der Sprache *BPMN2.0* gegeben. Dabei sollen ausgehend vom Gateway *g1* mehrere Kontrollflusspfade parallel ausgeführt werden. Sobald die Ausführung auf den parallelen Kontrollflusspfaden beendet worden ist, werden sie durch das Gateway *g2* wieder zu einem Kontrollflusspfad zusammengeführt.

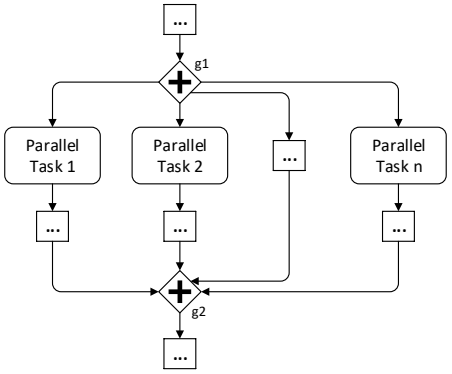


Abbildung 5-5: Beispiel für den Aspekt Parallelism in der Sprache BPMN2.0

Bewertung für den Aspekt Parallelism

Der Aspekt *Parallelism* kann dabei als eine weitere Art von Flexibilität betrachtet werden. So ist im Gegensatz zu den bisher betrachteten Aspekten *Choice* und *Iteration* keine konditionale Auswahl von Kontrollflusspfaden vorhanden. Eine direkte Trennung der Anwendungs- und Anpassungslogik ist somit für den Aspekt *Parallelism* nicht möglich, da keine Bedingung oder Auswahl eingeschlossen ist. Dennoch sind selbstverständlich sonstige Anpassungen an den für die parallele Ausführung angedachten Kontrollflusspfaden möglich. So lässt sich jeder parallele Kontrollflusspfad bspw. hinsichtlich seiner enthaltenen Elemente anpassen. Alternativ lässt sich aber auch die Anzahl von parallelen Kontrollflusspfaden anpassen. Hierbei handelt es sich jedoch nicht um eine für den Aspekt *Parallelism* spezifische Anpassung. Auf eine Beschreibung für die Verwendung der Sprache *ACML4BPM* wird daher in diesem Bezug nachfolgend verzichtet (siehe auch Abschnitt 5.2.1.7)

5.2.1.4 Interleaving

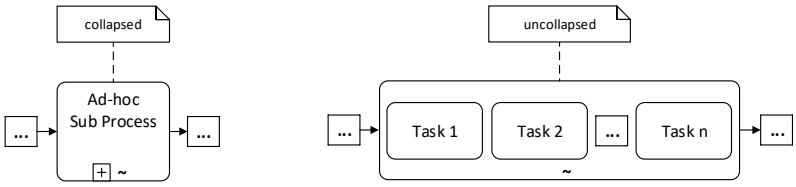
Ein weiterer Aspekt ist durch *Interleaving* gegeben. *Interleaving* beschreibt eine Fähigkeit, durch die es ermöglicht wird, die Sequenz einer Ausführung einzelner Tasks oder Kontrollflusspfade so zu bestimmen, dass sie in einer beliebigen Sequenz ausgeführt werden. Die Sprache *BPMN2.0* bietet für einmalige und unbestimmte Reihenfolgen der Ausführung von Tasks das Sprachelement des *Ad-hoc Subprozesses* (siehe Abschnitt 2.3.4) an.

Beispiele für den Aspekt Interleaving

In Abbildung 5-6 sind zwei Beispiele für *Ad-hoc-Subprozesse* dargestellt. Bei dem *Ad-hoc-Subprozess* handelt es sich um einen speziellen Typ eines *Subprozesses*, der sich sowohl ein- (*collapsed*) als auch ausgeklappt (*uncollapsed*) darstellen lässt. Der eingeklappte *Ad-hoc Subprozess* wird mit einem eingerahmten Plus-Symbol dargestellt. Das Symbol des *Ad-hoc Subprozesses* selbst ist das Tilde-Symbol (~). Im linken Bereich ist ein eingeklappter *Ad-*

hoc Subprozess dargestellt. Im rechten Bereich hingegen wird der gleiche *Ad-hoc Subprozess* ausgeklappt gezeigt. In dem ausgeklappten *Ad-hoc Subprozess* sind verschiedene Tasks (*Task 1* bis *Task n*) enthalten, welche in einer nicht näher spezifizierten Reihenfolge einmalig ausgeführt werden können. Wurde der letzte der genannten Tasks ausgeführt, terminiert der *Ad-hoc Subprozess*.

Abbildung 5-6: Beispiele für den Aspekt Interleaving in der Sprache BPMN2.0



Bewertung für den Aspekt Interleaving

Flexibilität ist auch bei dem Aspekt *Interleaving* losgelöst von einer auswertbaren Bedingung, die eine Trennung von der Anpassungs- und Anwendungslogik ermöglicht. Ebenso wie zuvor für den Aspekt *Parallelism* lassen sich aber sonstige Anpassungen durchführen. So ist es z.B. denkbar, dass die Anzahl der eingebetteten Tasks im *Ad-hoc Subprozess* durch Operationen der Typen *Add*, *Remove* und *Modify* angepasst werden können (siehe Abschnitt 4.3.3). Auf eine Beschreibung für die Verwendung der Sprache *ACML4BPM* wird daher auch für diesen Aspekt nachfolgend verzichtet, da sie hinsichtlich der Funktionsweise des Aspekts *Interleaving* nicht spezifisch wäre.

5.2.1.5 Multiple Instances

Sollen von einem Task mehrere Instanzen gleichzeitig ausgeführt werden, kann dies durch den Aspekt *Multiple Instances* umgesetzt werden. Die Sprache *BPMN2.0* bietet hierzu die Möglichkeit zur entsprechenden Konfiguration eines einzelnen Tasks.

Beispiele für den Aspekt Multiple Instances

In Abbildung 5-7 sind zwei Beispiele für die Konfiguration von Tasks in der Sprache *BPMN2.0* hinsichtlich des Aspekts *Multiple Instances* gegeben. So bietet die Sprache *BPMN2.0* die Möglichkeit zu spezifizieren, dass von einem Task mehrere Instanzen ausgeführt werden sollen. Dabei können verschiedene Konfigurationen angegeben werden, durch die die Anzahl der Instanzen (*loopCardinality*) und die Art der Ausführung (*isSequential*) angegeben werden. Hierbei werden zwei grundsätzliche Arten der Ausführung unterschieden. Verschiedene Instanzen eines Tasks können einerseits sequentiell und andererseits parallel ausgeführt werden. Beide Typen

können unterschieden werden, indem ein Symbol bestehend aus drei horizontalen Strichen die sequentielle und ein Symbol bestehend aus drei vertikalen Strichen die parallele Art der Ausführung kennzeichnet.

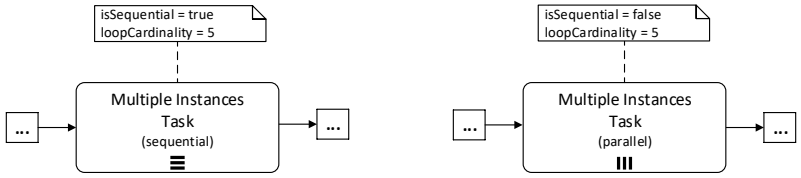


Abbildung 5-7: Beispiele für den Aspekt Multiple Instances in der Sprache BPMN2.0

Bewertung für den Aspekt Multiple Instances

Flexibilität ist abermals bei dem Aspekt *Multiple Instances* losgelöst von einer auswertbaren Bedingung. Anpassungen im Rahmen des Aspekts *Multiple Instances* sind aber dennoch denkbar, sodass z.B. die genannten Eigenschaften *isSequential* und *loopCardinality* anpassbar sind. So kann durch die Änderung des Wertes der Eigenschaft *loopCardinality* die Anzahl der auszuführenden Instanzen des Tasks geändert werden. Auch für den Aspekt *Multiple Instances* wird daher auf die Beschreibung der Verwendung der Sprache *ACML4BPM* verzichtet (siehe Abschnitt 5.2.1.7)

5.2.1.6 Cancellation

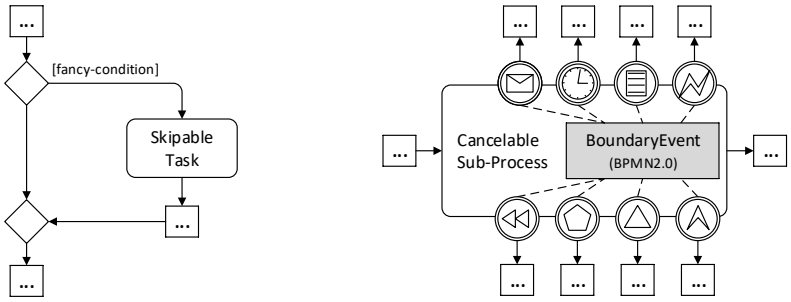
Der letzte Aspekt von *Flexibility-by Design* ist durch *Cancellation* gegeben. Durch *Cancellation* wird die Fähigkeit zum Überspringen bzw. zum Abbrechen eines Abschnitts im Kontrollflusspfad beschrieben. Hierzu existieren zwei Varianten, die betrachtet werden können. Zum einen kann *Cancellation* als Variante des Aspekts *Choice* verstanden werden. So kann hierbei durch die Auswertung einer Bedingung ein Kontrollflusspfad übersprungen werden und zum nächsten gemeinsamen Kontrollflussteilpfad gewechselt werden. Das Überspringen eines Teilpfades wird auch als *Skip* bezeichnet (siehe auch Abschnitt 5.4). Zum anderen ist es aber auch denkbar, die Ausführung eines Subprozesses bzw. Tasks vorzeitig zu beenden. Hierzu bietet die Sprache *BPMN2.0* Ereignisse des Typs *Boundary-Event* an.

Beispiele für den Aspekt Cancellation

In Abbildung 5-8 sind die beiden Varianten des Aspekts *Cancellation* in zwei Beispielen gezeigt. Im linken Bereich ist hierzu die Variante des Aspekts *Cancellation* durch den Aspekt *Choice* dargestellt. Je nach Ergebnis der Auswertung der Bedingung *fancy-condition* kann die Ausführung des Teilpfades, auf dem der Task mit der Bezeichnung *Skipable Task* enthalten ist, übersprungen werden. Eine Übersicht über ausgesuchte Beispiele für Arten des Aspekts *Cancellation* repräsentiert durch Ereignisse des

Typs *BoundaryEvent* ist im rechten Bereich gegeben. So kann bspw. beim Aufkommen eines Ereignisses des Typs *MessageEvent* die weitere Ausführung des Subprozesses abgebrochen werden. Für eine detaillierte Beschreibung aller dargestellten Ereignisse wird auf die Spezifikation der Sprache *BPMN2.0* verwiesen [OMG11].

Abbildung 5-8: Beispiele für den Aspekt Cancellation in der Sprache BPMN2.0



Bewertung für den Aspekt Cancellation

Flexibilität im Rahmen des Aspekts *Cancellation* ist dabei derartig gegeben, dass die Möglichkeit zum Abbruch des vorgesehenen Kontrollflusspfades besteht. Ferner können auch alternative Kontrollflusspfade gewählt werden. Der Aspekt *Cancellation* gibt darüber hinaus die Möglichkeit zur Behandlung von Ausnahmen – sogenannten *Exceptions*. So können bspw. durch ein Ereignis des Typs *TimerEvent* Ausnahmebehandlungen definiert werden, wenn die Ausführung eines Tasks oder Subprozesses eine gewisse Zeitdauer überschreitet. Für die Variante des Aspekts *Cancellation*, bei der ein Teilpfad übersprungen werden kann, ist eine Trennung der Anpassungs- und Anwendungslogik möglich. So können beteiligte Teilpfade und die auszuwertenden Bedingungen getrennt voneinander gestaltet werden. Für die Varianten des Aspekts *Cancellation* zum Abbruch eines Subprozesses oder Tasks ist je nach aufkommendem Ereignis vom Typ *BoundaryEvent* zu differenzieren, ob eine Trennung möglich oder sinnvoll ist.

Werden die verschiedenen Typen von Ereignissen als *BoundaryEvent* eingesetzt, kann hierdurch der Kontrollfluss derartig geändert werden, dass die von dem eingesetzten Ereignis ausgehende Kante des Kontrollflusses schaltet, sodass die zugehörige Aktivität bzw. der Subprozess vorzeitig beendet werden kann. Im Rahmen der Sprache *BPMN2.0* werden für diesen Zweck verschiedene Typen von Ereignissen unterstützt, so z.B. *MessageEvent*, *TimerEvent*, *EscalationEvent*, *ErrorEvent*, *CompensationEvent*, *ConditionalEvent*, *SignalEvent*, *MultipleEvent* und *ParallelEvent*. In Abschnitt 5.2.4 wird für eine Auswahl der zuvor genannten Ereignisse und unter Verwendung der Sprache *ACML4BPM* eine Gestaltung des Aspekts *Cancellation* beschrieben.

5.2.1.7 Zusammenfassung der Bewertung

Zuvor wurden die durch [Sch+08] gegebenen Aspekte von *Flexibility-by Design* zunächst beschrieben. Ferner wurden verschiedene Beispiele zur Gestaltung der eingeführten Aspekte auf Basis der Sprache *BPMN2.0* gegeben. Hierzu ist in Tabelle 5-1 eine Übersicht über die beschriebenen Aspekte sowie eine Einschätzung hinsichtlich der Möglichkeit einer Trennung von Anwendungs- und Anpassungslogik (*SoC*) gegeben.

Aspekt	SoC möglich	Sonstige Anpassungen möglich
Choice	✓	✓
Iteration	✓	✓
Parallelism	–	✓
Interleaving	–	✓
Multiple Instances	–	✓
Cancellation	✓	✓

Tabelle 5-1: *Übersicht über die Möglichkeit der Trennung von Anpassungs- und Anwendungslogik in Bezug zu einzelnen Aspekten von Flexibility-by Design*

Bewertung von Aspekten

Für die Aspekte *Parallelism*, *Interleaving* und *Multiple Instances* ist eine spezifische Verwendung der Sprache *ACML4BPM* zur Trennung der Anwendungs- von der Anpassungslogik nicht möglich, da diese Aspekte typischerweise keine Bedingungen enthalten, die ausgewertet werden müssen und durch die alternative Kontrollflusspfade bestimmt werden können. Stattdessen beschreiben sie spezielle Arten von Ausführungen, wie etwa die parallele und die einmalig versetzte Ausführung von verschiedenen Tasks oder Subprozessen. Eine Anpassung von Prozessen, die diese Aspekte umsetzen, ist aber dennoch möglich. So können Elemente oder Eigenschaften auf den enthaltenen Kontrollflusspfaden angepasst werden, sofern dies notwendig ist.

Eine Trennung von Anpassungs- und Anwendungslogik ist insgesamt bei den drei Aspekten *Choice*, *Iteration* und *Cancellation* möglich. Dies lässt sich vornehmlich dadurch begründen, dass in diesen Aspekten stets mindestens eine Auswertung einer Bedingung zur Auswahl eines alternativen Kontrollflusspfades vorkommt. Für diese Gruppe von Aspekten von *Flexibility-by Design* ist die Verwendung der in Kapitel 5 eingeführten Sprache *ACML4BPM* möglich. Hierdurch kann eine Trennung der Anpassungs- und Anwendungslogik ermöglicht werden.

Die Gestaltung dieser Aspekte kann dabei auf zwei unterschiedliche Arten durchgeführt werden. Auf der einen Seite kann die Anpassungs- und Anwendungslogik durch Beobachtungs- und Anpassungsprozesse umgesetzt werden. Auf der anderen Seite können Beobachtungs- und insbe-

sondere Anpassungsprozesse aber auch dazu verwendet werden, den Gesamtprozess derartig anzupassen, dass eine benötigte Funktion integriert wird. Bei der letzten Art der Gestaltung handelt es sich jedoch um das Entwurfsmuster *Flexibility-by Change*, das in Abschnitt 5.3 detailliert beschrieben wird. Daher wird im Folgenden die Umsetzung der beiden Logiken ohne Anpassungen am Gesamtprozess gezeigt. Stattdessen wird gezeigt, wie Anpassungs- und Anwendungslogik durch Beobachtungs- und Anpassungsprozesse umgesetzt werden können.

Nachfolgend werden für diese Aspekte Beispiele unter Verwendung der in dieser Arbeit entwickelten Sprache gegeben. Die Beispiele können anschließend für die Gestaltung von flexiblen und anpassbaren Prozessen als Entwurfsmuster verwendet werden.

5.2.2 Gestaltung von Choice

Die Möglichkeit zur Gestaltung einer Auswahl von verschiedenen Kontrollflusspfaden kann bereits frühzeitig in der Phase *Design & Analyse* des *BPM-Lebenszyklus* durch den Aspekt *Choice* unterstützt werden. Bei der Gestaltung dieses Aspekts kann die in Kapitel 4 vorgestellte Sprache *ACML4BPM* eingesetzt werden. Ferner lässt sich hierbei eine Trennung von Anpassungs- und Anwendungslogik erreichen. Für die Trennung der beiden Logiken ist es zunächst notwendig, dass zunächst jeweils zugehörige Elemente identifiziert werden, sodass in einem nachfolgenden Schritt eine getrennte Gestaltung unter der Verwendung der Sprache *ACML4BPM* ermöglicht werden kann.

In Abbildung 5-9 ist das Ergebnis einer Identifikation von Elementen der Anpassungs- und Anwendungslogik auf Basis des in Abbildung 5-3 eingeführten Beispiels dargestellt. Das Beispiel beschreibt dabei einen Auszug eines im Folgenden genannten Gesamtprozesses. Dabei liegt der Fokus auf einem Ausschnitt, der durch die Punkte *s* und *e* gegeben ist.

Elemente der Anpassungslogik

Elemente des Kontrollflusses der Anpassungslogik sind in der Farbe *Grün* hinterlegt dargestellt. Im Fall des in Abbildung 5-3 dargestellten Beispiels handelt es sich um Entscheidungspunkte, die durch *Gateways* und durch die angehängten *Bedingungen* abgebildet sind.

Elemente der Anwendungslogik

Ferner sind die einzelnen Teile des Kontrollflusses, die die Anwendungslogik darstellen, in der Farbe *Blau* hinterlegt dargestellt. Dabei handelt es sich in dem gezeigten Beispiel um die dargestellten Tasks bzw. Kontrollflusspfade, auf denen sie vorkommen.

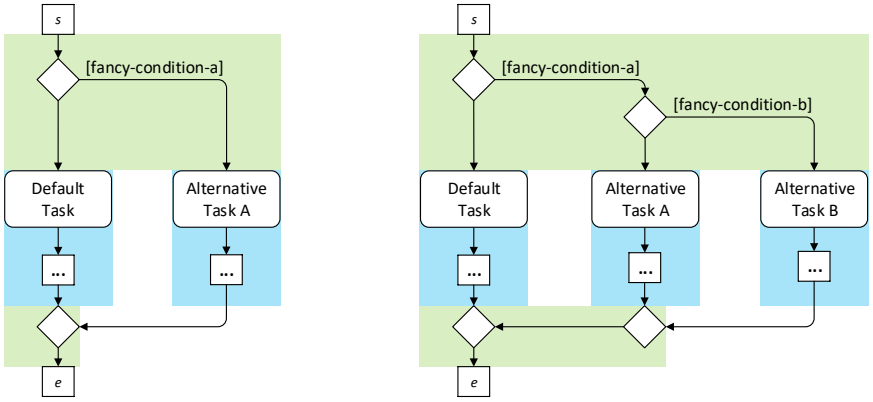


Abbildung 5-9: Elemente der Anpassungs- und Anwendungslogik

Integration in den Gesamtprozess

Eine weitere Fragestellung beschäftigt sich mit der Integration des zwischen den Punkten *s* und *e* liegenden Verhaltens in den Gesamtprozess. Wie in Abschnitt 4.2.2 bereits beschrieben, wird ein *AC4BPM* bzw. ein Beobachtungsprozess typischerweise durch ein Ereignis ausgelöst. Hierzu können sowohl explizite als auch implizite Ereignisse eingesetzt werden (siehe Abschnitt 4.3.4). In Abbildung 5-10 werden mögliche Varianten vorgestellt, die für die Auslösung eines Beobachtungsprozesses und damit für die Integration in den Gesamtprozess eingesetzt werden können. Sie umfassen die Verwendung von expliziten und impliziten Ereignissen sowie die Umsetzung mit und ohne notwendiger Anpassung.

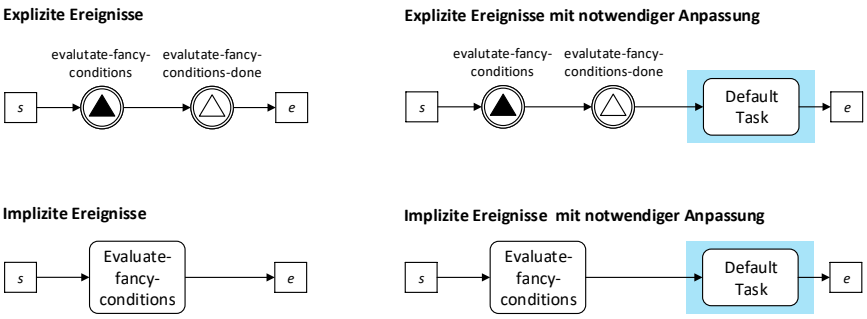


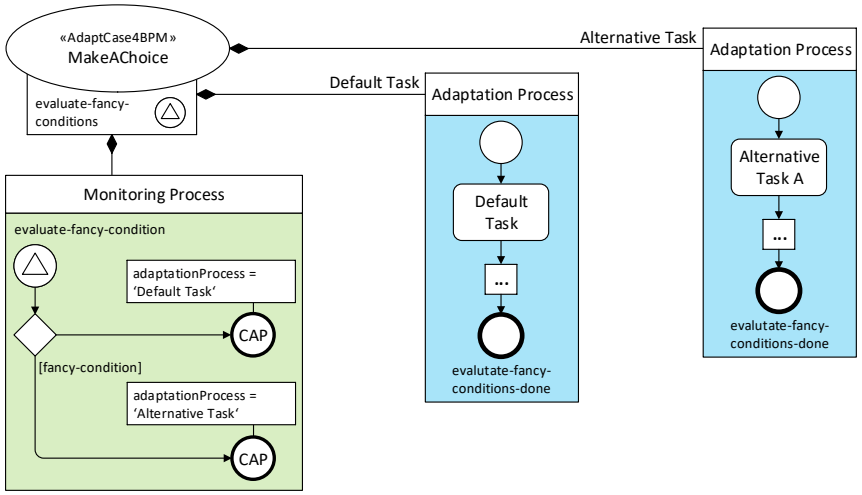
Abbildung 5-10: Explizite und implizite Ereignisse zur Auslösung eines AC4BPM mit und ohne notwendiger Anpassung

Integration ohne Anpassung durch explizite Ereignisse

So lässt sich ein *AC4BPM* durch eine in dem Kontrollfluss des Gesamtprozess vorgesehene Rückkopplung einbinden, die hier durch die beiden Ereignisse der Typen *ThrowEvent* und *CatchEvent* dargestellt sind. Die generelle Funktionsweise lässt sich dabei derartig beschreiben, dass durch das Ereignis mit der Bezeichnung *evaluate-fancy-conditions* (*ThrowEvent*) ein *AC4BPM* ausgelöst werden kann. Ist die Bearbeitung durch den zugehörigen Beobachtungs- oder Anpassungsprozess abgeschlossen, so wird ein Ereignis mit der Bezeichnung *evaluate-fancy-conditions* (*CatchEvent*) ausgelöst und die Rückkopplung ist abgeschlossen.

Ein Beispiel für einen zugehörigen *AC4BPM* ist in Abbildung 5-11 beschrieben. Zum besseren Verständnis sind die Elemente der Anpassungs- und Anwendungslogik abermals in den zuvor eingeführten Farben *Grün* und *Blau* hinterlegt. So ist ersichtlich, dass die Anpassungslogik des Aspekts *Choice* vollständig durch den dargestellten Beobachtungsprozess gestaltet worden ist. Je nach Ergebnis einer Auswertung der dargestellten Bedingung wird der dazugehörige Anpassungsprozess aufgerufen, durch den die jeweils benötigte Funktion gestaltet worden ist.

Abbildung 5-11: *Beispiel einer Alternative für den Aspekt Choice in ACML4BPM*



Werden verschachtelte Bedingungen wie im rechten Bereich von Abbildung 5-9 verwendet, so lässt sich diese Anpassungslogik ebenfalls durch einen Beobachtungsprozess beschreiben. Weitere alternative Funktionen können durch das Hinzufügen weiterer Anpassungsprozesse mit entsprechender Funktionsbeschreibung gestaltet werden.

Integration ohne Anpassung durch implizite Ereignisse

Ein *AC4BPM* lässt sich alternativ aber auch durch implizite Ereignisse auslösen. In Abbildung 5-10 ist hierzu im linken unteren Bereich ein Beispiel für eine derartige Integration dargestellt. Dabei bietet sich die in Abschnitt 4.3.4.2 vorgestellte Variante von transformierten impliziten Ereignissen an, in denen eine Rückkopplung vorgesehen ist. Die Funktionsweise gleicht anschließend der durch explizite Ereignisse beschriebenen Integration.

Ein Beispiel für einen zugehörigen *AC4BPM* ist in Anlehnung an Abbildung 5-12 beschrieben. Zum besseren Verständnis sind die Elemente der Anpassungs- und Anwendungslogik abermals in den zuvor eingeführten Farben *Grün* und *Blau* hinterlegt. In dem Beobachtungsprozess ist die verschachtelte Anpassungslogik des in Abbildung 5-9 eingeführten Beispiels

gestaltet worden. Dabei muss an dieser Stelle erneut darauf aufmerksam gemacht werden, dass aufgrund der Rückkopplung für die Variante ein höherer Aufwand in der weiteren Gestaltung bzw. Implementierung berücksichtigt werden sollte (siehe Abschnitt 4.3.4.2).

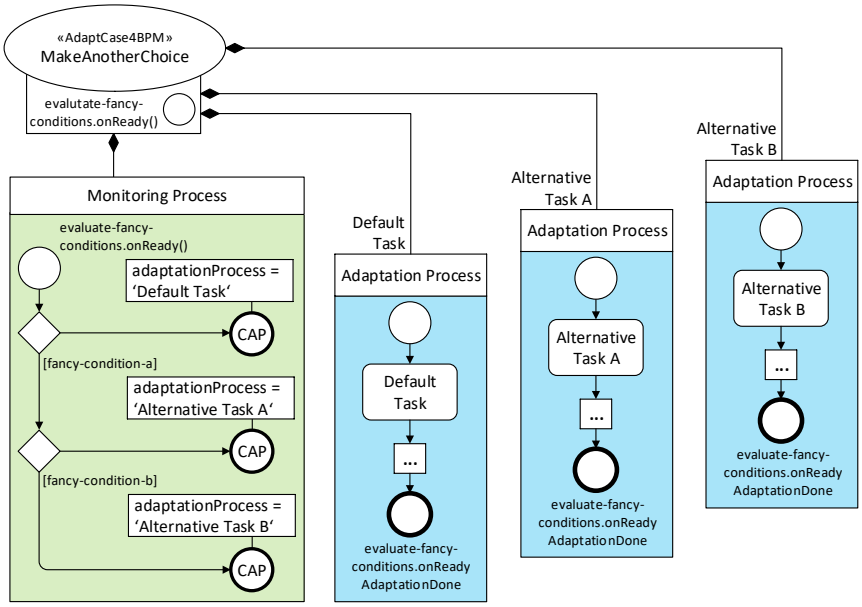


Abbildung 5-12: *Beispiel multipler Alternativen für den Aspekt Choice in ACML4BPM*

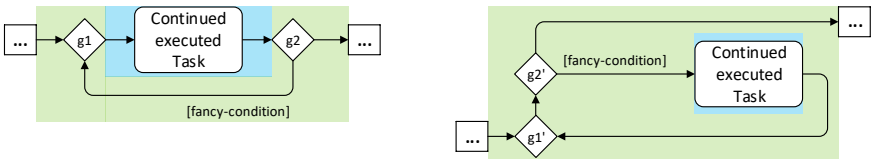
Integration durch explizite und implizite Ereignisse mit Anpassung

Darüber hinaus ist ebenso eine Integration durch explizite oder implizite Ereignisse mit Anpassung möglich. Im rechten oberen bzw. rechten unteren Bereich von Abbildung 5-10 ist dies gezeigt. Dabei ist die Standardfunktion (hier: *Default Task*) Teil des Ausschnittes zwischen den Punkten *s* und *e*. Die zuvor beschriebenen beiden Varianten von Verwendungsweisen für eine mögliche Auslösung eines *AC4BPM* können dabei dazu verwendet werden, eine Anpassung am gezeigten Ausschnitt des Kontrollflusses durchzuführen. Eine derartige Anpassung kann so z.B. den Task mit der Bezeichnung *Default Task* aus dem Kontrollfluss entfernen und einen alternativen Task, wie *Alternative Task A* oder *Alternative Task B*, einführen. Durch die in Abbildung 5-10 dargestellten Mechanismen zur Rückkopplung ist dieser Austausch einer Funktionalität vor ihrer jeweiligen Aktivierung möglich. Auf eine Beschreibung von Beispielen wird verzichtet, da die Funktionsweise der Integration vornehmlich der zuvor beschriebenen beiden Prinzipien folgt. Eine Abweichung ist lediglich hinsichtlich der Verwendungsweise des Anpassungsprozesses vorhanden, in dem anstelle der Ausführung einer Funktion, wie z.B. *Default Task*, das Entfernen und Hinzufügen einer solchen Funktion beschrieben steht.

5.2.3 Gestaltung von Iteration

Durch den Aspekt *Iteration* kann die Möglichkeit zur Gestaltung eines iterativ ausgeführten Kontrollflusspfades bereits frühzeitig in der Phase *Design & Analyse* des *BPM-Lebenszyklus* unterstützt werden. Dabei handelt es sich um eine spezielle Variante des Aspekts *Choice*, weshalb im Folgenden lediglich auf Besonderheiten des Aspekts *Iteration* eingegangen wird. Bei der Gestaltung dieses Aspekts kann ebenfalls die in Kapitel 4 vorgestellte Sprache *ACML4BPM* eingesetzt werden, sodass sich eine Trennung von Anpassungs- und Anwendungslogik erreichen lässt. Für die Trennung der beiden Logiken ist es zunächst notwendig, dass jeweils zugehörige Elemente in einem ersten Schritt identifiziert werden, sodass in einem nachfolgenden Schritt eine getrennte Gestaltung unter Verwendung der Sprache *ACML4BPM* ermöglicht werden kann. Hierzu ist in Abbildung 5-13 das Ergebnis einer Identifikation von Elementen der Anpassungs- und Anwendungslogik auf Basis des in Abbildung 5-4 eingeführten Beispiels dargestellt.

Abbildung 5-13: Identifizierung von Anpassungs- und Anwendungslogik zur Unterstützung des Aspekts Iteration



Elemente der Anpassungslogik

Dabei sind wesentliche Elemente des Kontrollflusses, die für die Anpassungslogik vorhanden sind, in der Farbe *Grün* hinterlegt dargestellt. Im Fall der in Abbildung 5-4 dargestellten Beispiele handelt es sich um Entscheidungspunkte, die durch die Gateways *g1* und *g2* bzw. *g1'* und *g2'* und die angehängten Bedingungen abgebildet sind.

Elemente der Anwendungslogik

Ferner sind die einzelnen Teile des Kontrollflusses, die die Anwendungslogik darstellen, in der Farbe *Blau* hinterlegt dargestellt. Dabei handelt es sich in den gezeigten Beispielen um den Task mit der Bezeichnung *Continued executed Task*.

Integration in den Gesamtprozess

Wie bereits in Abschnitt 5.2.2 beschrieben, lassen sich für eine Umsetzung der Anpassungs- und Anwendungslogik durch Beobachtungs- und Anpassungsprozesse mehrere Möglichkeiten zur Integration der beiden Logiken in den Gesamtprozess anwenden. Da es sich bei dem Aspekt *Iteration* um eine spezielle Variante des Aspekts *Choice* handelt, wird in Anlehnung an eine mögliche Integration in den Gesamtprozess lediglich auf die Realisierung über explizite Ereignisse ohne Anpassung eingegangen.

In Abbildung 5-14 wird hierzu eine Variante für die Auslösung eines Beobachtungsprozesses auf Basis von expliziten Ereignissen dargestellt, die im weiteren Beispiel verwendet wird. Bei der dargestellten Integration in den Gesamtprozess handelt es sich abermals um den bereits in Abbildung 5-10 dargestellten Mechanismus einer Rückkopplung. Auf eine detaillierte Beschreibung wird an dieser Stelle verzichtet und stattdessen auf Abschnitt 5.2.2 verwiesen.

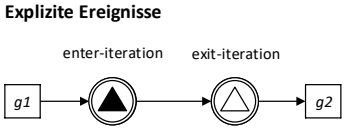


Abbildung 5-14: *Explizite Ereignisse ohne Anpassung zur Auslösung eines AC4BPM*

Kopfgesteuerte Iteration

In Abbildung 5-15 ist ein Beispiel eines *AC4BPM* gezeigt, der durch die Variante der expliziten Ereignisse ausgelöst werden kann und der die Umsetzung einer kopfgesteuerten Iteration beschreibt. Zum besseren Verständnis sind die Elemente der Anpassungs- und Anwendungslogik abermals in den zuvor eingeführten Farben *Grün* und *Blau* hinterlegt. So ist ersichtlich, dass die Anpassungslogik des Aspekts *Iteration* hinsichtlich der kopfgesteuerten Iteration vollständig durch den dargestellten Beobachtungsprozess gestaltet worden ist. Je nach Ergebnis einer Auswertung der dargestellten Bedingung wird der dazugehörige Anpassungsprozess aufgerufen und somit ein iteratives Ausführen der entsprechenden Funktion ermöglicht. Entgegen des Beispiels für den Aspekt *Choice* wird durch das Beenden des Anpassungsprozesses nicht zurück in den Gesamtprozess gesprungen. Stattdessen wird der zugehörige Beobachtungsprozess erneut aufgerufen und die Iteration ist je nach Auswertung der Bedingung in der Lage, fortgesetzt oder beendet zu werden. Ein Abbruch der Iteration löst das Ereignis mit der Bezeichnung *exit-iteration* aus, sodass die Rückkopplung mit dem Gesamtprozess durchgeführt wird und die Ausführung an dortiger Stelle fortgesetzt werden kann.

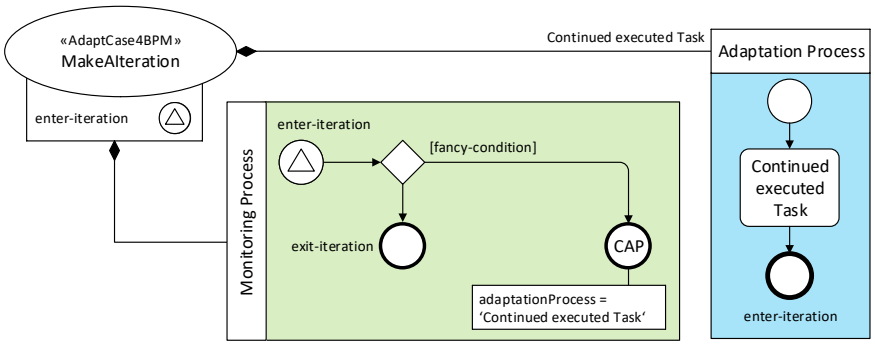


Abbildung 5-15: *Beispiel einer iterativ ausgeführten Funktion für den Aspekt Iteration in ACML4BPM (kopfgesteuert)*

Fußgesteuerte Iteration

Ein Beispiel für die in Abbildung 5-13 dargestellte fußgesteuerte Iteration unter Verwendung von *ACML4BPM* ist in Abbildung 5-16 gezeigt. Zum besseren Verständnis sind die Elemente der Anpassungs- und Anwendungslogik abermals in den zuvor eingeführten Farben *Grün* und *Blau* hinterlegt. In dem Beobachtungsprozess ist die Anpassungslogik des in Abbildung 5-13 eingeführten Beispiels gestaltet worden. Dabei kann der Beobachtungsprozess auf zwei Arten aufgerufen werden. Zum einen kann er extern durch das Ereignis zur Rückkopplung und zum anderen intern über das Ereignis mit der Bezeichnung *evaluate-iterations-choice*, das bei Beendigung des Anpassungsprozesses ausgelöst wird, aufgerufen werden. Hierdurch wird die Funktionsweise einer fußgesteuertern Iteration unterstützt, da die auszuführende Funktion zunächst immer erst ausgeführt wird, bevor die Bedingung der Iteration ausgewertet wird. Die Beendigung der Iteration kann nach Ausführung der enthaltenen Funktion durchgeführt werden. Ferner kann auch eine weitere Iteration gestartet werden.

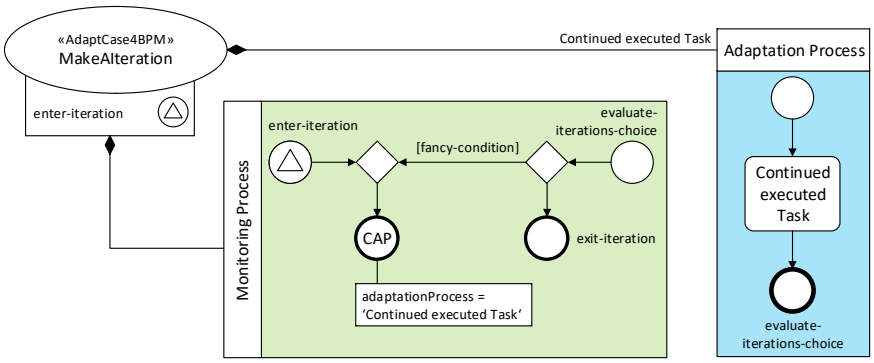


Abbildung 5-16: Beispiel einer iterativ ausgeführten Funktion für den Aspekt Iteration in ACML4BPM (fußgesteuert)

5.2.4 Gestaltung von Cancellation

Der letzte Aspekt *Cancellation* ermöglicht die Gestaltung von vorbestimmten Behandlungen von Abbrüchen eines Tasks bzw. eines Subprozesses. Wie bereits in Abschnitt 5.2.1.6 beschrieben, existieren dabei zwei unterschiedliche Arten. Auf der einen Seite kann der Aspekt *Cancellation* als spezielle Variante des Aspekts *Choice* verstanden werden, wenn z.B. die Ausführung eines Kontrollflusspfades übersprungen wird. Daher spricht man in einem solchen Fall anstelle von *Cancellation* auch von *Skip*. Auf der anderen Seite kann es aber auch sinnvoll sein, einen aktiven Task bzw. Subprozess unter bestimmten Bedingungen abzubrechen. So kann z.B. beim Auftreten von bestimmten Ereignissen eine weitere Bearbeitung im Rahmen des Tasks bzw. des Subprozesses nicht mehr sinnvoll sein. Alternativ kann auch der Fall eines aufgetretenen Fehlers behandelt werden.

Bei der Gestaltung von Prozessen mit Umsetzung des Aspekts *Cancellation* kann ebenfalls die in Kapitel 4 vorgestellte Sprache *ACML4BPM* als Unterstützung eingesetzt werden. Für die Trennung der beiden Logiken ist es zunächst notwendig, dass die jeweils zugehörigen Elemente zunächst identifiziert werden, sodass in einem nachfolgenden Schritt eine getrennte Gestaltung unter Verwendung der Sprache *ACML4BPM* ermöglicht wird. Hierzu ist in Abbildung 5-17 das Ergebnis einer Identifikation von Elementen der Anpassungs- und Anwendungslogik auf Basis des in Abbildung 5-8 eingeführten Beispiels dargestellt.

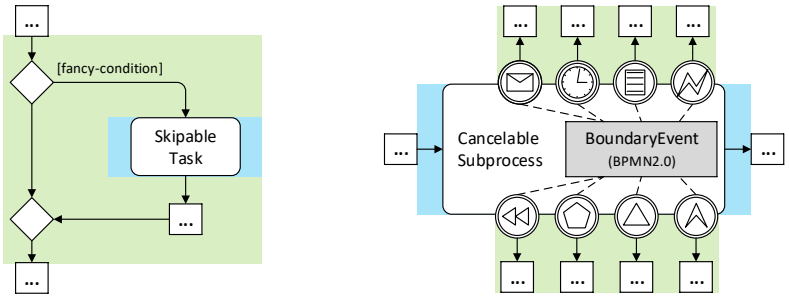


Abbildung 5-17: Identifizierung von Anpassungs- und Anwendungslogik zur Unterstützung des Aspekts Cancellation

Elemente der Anpassungslogik

Dabei sind wesentliche Elemente des Kontrollflusses, die für die Anpassungslogik vorhanden sind, in der Farbe *Grün* hinterlegt dargestellt. Im linken Fall des Aspekts *Skip* handelt es sich hierbei um die Entscheidungspunkte sowie die Bedingung *fancy-condition*. Wohingegen im rechten dargestellten Fall die Anpassungslogik durch den Empfang von Ereignissen dargestellt ist. Ferner kann auch das nachgelagerte Verhalten hinzugezählt werden, das hier durch den aus den Ereignissen ausgehenden Kontrollfluss dargestellt ist.

Elemente der Anwendungslogik

Ferner sind die einzelnen Teile des Kontrollflusses, die die Anwendungslogik darstellen, in der Farbe *Blau* hinterlegt abgebildet. Dabei handelt es sich in den gezeigten Beispielen um den Task mit der Bezeichnung *Skipable Task* bzw. den Subprozess mit der Bezeichnung *Cancelable Subprocess*.

Integration in den Gesamtprozess

Für den im linken Bereich dargestellten Aspekt *Skip* können für den Aspekt *Choice* vorgestellte Inhalte zur Gestaltung genutzt werden (siehe Abschnitt 5.2.2). Daher wird im Folgenden lediglich auf den im rechten Bereich dargestellten Subprozess eingegangen. Im oberen Bereich von Abbildung 5-18 ist dazu die Ausgangslage für den Abbruch des dargestellten Subprozesses gegeben. Tritt das dargestellte Ereignis *Cancel-Subprocess* auf, so terminiert seine Ausführung. Da im Rahmen eines Abbruchs Maßnahmen zur Kompensation gewollt sein können, wird anschließend auf

eine Rückkopplung durch das Ereignis *Fancy-Stuff-Done* gewartet. Für die beiden Punkte zur Erweiterung der dargestellten Kontrollflusspfade *ex1* und *ex2* sind im unteren Bereich Beispiele für die weitere Verfahrensweise gegeben. So ist nach der Rückkopplung bspw. eine Beendigung des Prozesses möglich. Alternativ kann aber auch auf den Hauptpfad des Kontrollflusses zurückgekehrt werden. Die dritte Variante bietet das Fortfahren auf einem alternativen Kontrollflusspfad (hier: *ex1*). Die Sprache *ACML4BPM* kann für den Aspekt *Cancellation* dazu genutzt werden, um Funktionen zur Analyse und Anpassungen bzw. Rückkopplungen zu beschreiben. Auf zwei Beispiele für einen zeitbasierten Abbruch und für einen konditionalen Abbruch wird nachfolgend eingegangen.

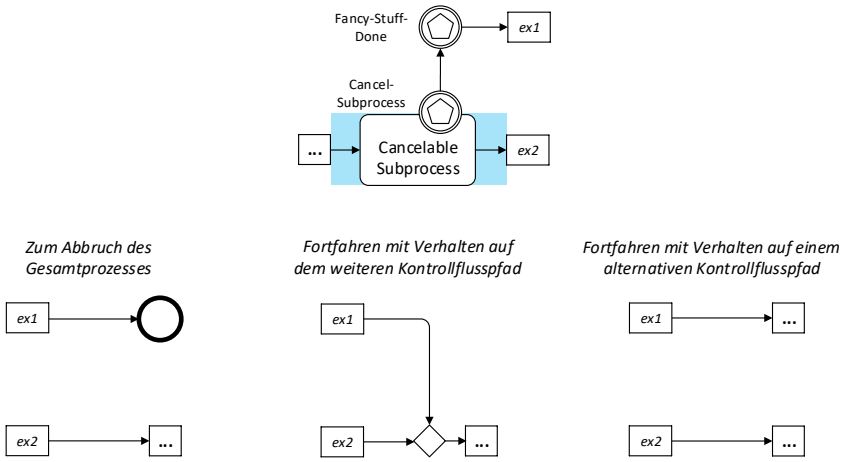


Abbildung 5-18: Explizite Ereignisse zur Integration der Anpassungslogik eines Adapt Case 4 BPM

Cancel-by Timer

Das erste Beispiel beschreibt, wie das Verhalten im Rahmen eines zeitbasierten Abbruchs durch einen *AC4BPM* dargestellt werden kann. Der in Abbildung 5-19 gezeigte *AC4BPM* mit der Bezeichnung *Cancel-by Timer* ist dabei aufrufbar durch ein implizites Ereignis (*onReady*) des Subprozesses *Cancelable Subprocess* (siehe auch Abschnitt 4.3.4.2). Im Rahmen des dargestellten Beobachtungsprozesses wird die Anpassungslogik gestaltet, die hier aus einem Kontrollfluss besteht, in dem nach Ablauf eines Zeitintervalls *after-time-x* ein Anpassungsprozess gestartet wird. Der Anpassungsprozess enthält einen Kontrollfluss, dessen Zweck die Synchronisation mit dem Gesamtprozess ist. So wird der Abbruch des Subprozesses durch das Ereignis *Cancel-Subprocess* ausgelöst. Ferner können weitere Aufgaben, die bei einem Abbruch durchgeführt werden müssen, in den Kontrollfluss des Anpassungsprozesses integriert werden. Dies ist hier konzeptionell durch den Task mit der Bezeichnung *Do-fancy-Stuff* angedeutet.

Ferner ist es möglich, dass die zeitliche Bedingung nicht erfüllt und der Anpassungsprozess somit nicht aufgerufen worden ist. Damit der Beob-

achtungsprozess nicht aktiv bleibt, ist die Rückkopplung mit dem Subprozess möglich. So wird bei der Beendigung der Ausführung des Subprozesses über das implizite Ereignis *onTerminated* die Ausführung des Beobachtungsprozesses abgebrochen.

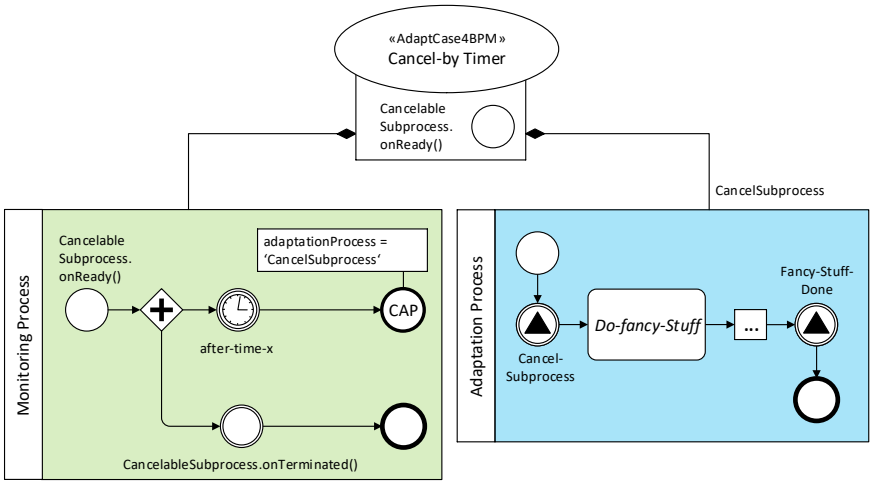


Abbildung 5-19: Beispiel für den Aspekt Cancellation in ACML4BPM (Cancel-by Timer)

Cancel-by Conditional

Das zweite Beispiel beschreibt einen konditionalen Abbruch durch einen *AC4BPM* und ist in Abbildung 5-20 dargestellt. Der dargestellte *AC4BPM* mit der Bezeichnung *Cancel-by Conditional* kann ebenfalls durch das implizite Ereignis *onReady* des Subprozesses *Cancelable Subprocess* aufgerufen werden. Der zugehörige Beobachtungsprozess enthält dabei die Anpassungslogik, in deren Rahmen zunächst eine konzeptionelle Analyse (*Analyze-fancy-Stuff*) durchgeführt und anschließend eine zugehörige Entscheidung (*fancy-condition*) getroffen wird. Je nach Ergebnis der Auswertung der Bedingung wird entweder erneut eine Analyse durchgeführt oder ein zugehöriger Anpassungsprozess aufgerufen.

Der dargestellte Anpassungsprozess erfüllt dabei den gleichen Zweck wie er zuvor auch schon für den *AC4BPM* mit der Bezeichnung *Cancel-by Timer* beschrieben worden ist. Hier wurde jedoch auf zusätzliche Maßnahmen, wie z.B. eine Kompensation, verzichtet. Wenn der Subprozess beendet wird, so kann der Beobachtungsprozess ebenso durch die Rückkopplung über das Ereignis *onTerminated* beendet werden.

5.2.5 Zusammenfassung

Der durch [Sch+08] eingeführte Flexibilitätsaspekt *Flexibility-by Design* kann als ein Entwurfsmuster für die Gestaltung von flexiblen Prozessen

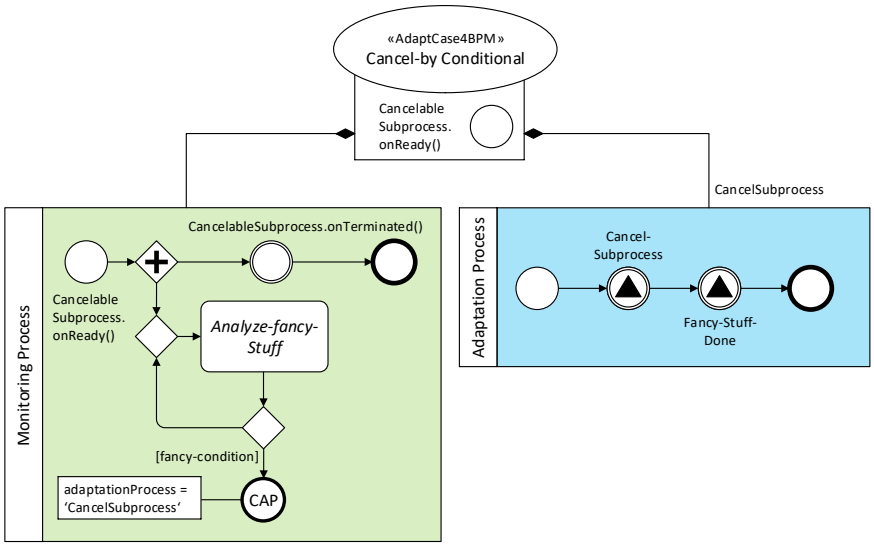


Abbildung 5-20: Beispiel für den Aspekt Cancellation in ACML4BPM (Cancel-by Conditional)

bereits frühzeitig in der Phase *Design & Analyse* betrachtet werden. In den vorherigen Abschnitten wurde zunächst eine Analyse der Aspekte *Choice*, *Iteration*, *Parallelism*, *Interleaving*, *Multiple Instances* und *Cancellation* durchgeführt. Jeder dieser Aspekte stellt dabei einen Teil von *Flexibility-by Design* dar. Dabei wurden Teilaspekte identifiziert, bei denen die Gestaltung von flexiblen Prozessen derartig unterstützt werden kann, dass durch die in Kapitel 4 eingeführte Sprache eine Trennung von Anpassungs- und Anwendungslogik durchgeführt werden kann. Ferner konnte ermittelt werden, dass auch sonstige Anpassungen im Rahmen der restlichen Aspekte möglich sind. Für eine detaillierte Beschreibung wurde dabei auf Abschnitt 5.3 verwiesen. Im Folgenden konnte auf Basis der Aspekte *Choice*, *Iteration* und *Cancellation* charakterisierende Beispiele gegeben werden, die eine Verwendung der in dieser Arbeit entwickelten Sprache *ACML4BPM* in der Gestaltung von Prozessen veranschaulichen. Durch eine konsequente Verwendung der Sprache im Fall von Bedingungen zur Auswahl von alternativen Kontrollflusspfaden konnte gezeigt werden, dass die Gestaltung von flexiblen und anpassbaren Prozessen hinsichtlich einer Trennung von Anpassungs- und Anwendungslogik unterstützt werden kann. Hierdurch wird ermöglicht, die Sicherstellung der Qualität der zu gestaltenden Prozesse hinsichtlich der in Abschnitt 1.3 genannten Anforderungen zu unterstützen.

5.3 Flexibility-by Change

Flexibility-by Change ist ein Flexibilitätsaspekt, der seine Eignung in verschiedenen Einsatzszenarios findet. So können Einzelheiten zu real benötigten Abläufen in Prozessen erst zu einem späteren Zeitpunkt oder in einer nachfolgenden Iteration des *BPM-Lebenszyklus* bekannt sein. Daher können zu diesem Zeitpunkt Anpassungen an bestehenden Prozessen, hier sowohl Prozessmodell als auch die zugehörigen Prozessinstanzen, notwendig sein. Ferner können sich die Anforderungen an den Prozessen innerhalb einer Iteration des *BPM-Lebenszyklus* aber auch ändern, sodass Anpassungen an diesen Prozessen bereits in der aktuellen Iteration notwendig sind. Dieser Umstand ist insbesondere bei langlaufenden Prozessen zu beobachten, bei denen die ursprünglichen Anforderungen aufgrund der Langläufigkeit schnell überholt sein können.

Ein zu dem Flexibilitätsaspekt *Flexibility-by Change* zugehöriges Entwurfsmuster unterstützt Anpassungen von Prozessmodellen und deren Instanzen. Anpassungen von Prozessmodellen werden dabei z.B. durch die in Abschnitt 4.3.3 eingeführten Operationen unterstützt. Als Ergänzung wird im Rahmen von *Flexibility-by Change* insbesondere auch die Überführung von bereits durchgeführten Anpassungen von Prozessmodellen auf einzelne oder alle derzeit aktiven zugehörigen Prozessinstanzen betrachtet. Bei einer solchen Überführung von Anpassungen wird alternativ auch von *Migration* gesprochen. Eine *Migration* kann dabei unter Verwendung einer Strategie durchgeführt werden, die die Verfahrensweise bei einer Anpassung von Prozessinstanzen beschreibt. Eine an [Sch+08] angelehnte Definition des Flexibilitätsaspekts *Flexibility-by Change* wird in Definition 5.3.1 gegeben.

Definition 5.3.1. *(Flexibility-by Change)*

Flexibility-by Change beschreibt die Fähigkeit zur Anpassung von Prozessmodellen und -instanzen während der Ausführungszeit. Dabei steht explizit die Möglichkeit zur Durchführung einer Migration von Anpassungen von den betroffenen Prozessen im Vordergrund, die sowohl Prozessmodelle als auch Prozessinstanzen betreffen können.

Für die Realisierung von *Flexibility-by Change* sind dabei neue Operationen notwendig, die eine Migration von Anpassungen von Prozessmodellen auf deren Instanzen ermöglichen. Einige Beispiele für mögliche Operationen zur Anpassung von Prozessen wurden bereits in Abschnitt 4.3.3 eingeführt, sodass an dieser Stelle lediglich auf Operationen für die Durchführung von Migrationen eingegangen wird.

Eine durch diese Arbeit durchgeführte Interpretation dieser Operationen ist in Abbildung 5-21 dargestellt. Dabei lässt sich das Entwurfsmuster *Flexibility-by Change* in die beiden weiteren Typen *Momentary Change* und *Evolutionary Change* unterteilen, welche zunächst in Abschnitt 5.3.1 kurz beschrieben werden. Aufbauend werden in Abschnitt 5.3.2 zugehörige Strategien für Migrationen vorgestellt. Anschließend wird in Abschnitt 5.3.3 eine konzeptionelle Erweiterung der Sprache *BPMN2.0* beschrieben, die Aspekte von *Flexibility-by Change* berücksichtigt. In Abschnitt 5.3.4 wird eine Reihe von zugehörigen Operationen für die Durchführung von Migrationen eingeführt. Abschließend wird in Abschnitt 5.3.5 eine Zusammenfassung hinsichtlich einer möglichen Verwendung von der Sprache *ACML4BPM* gegeben.

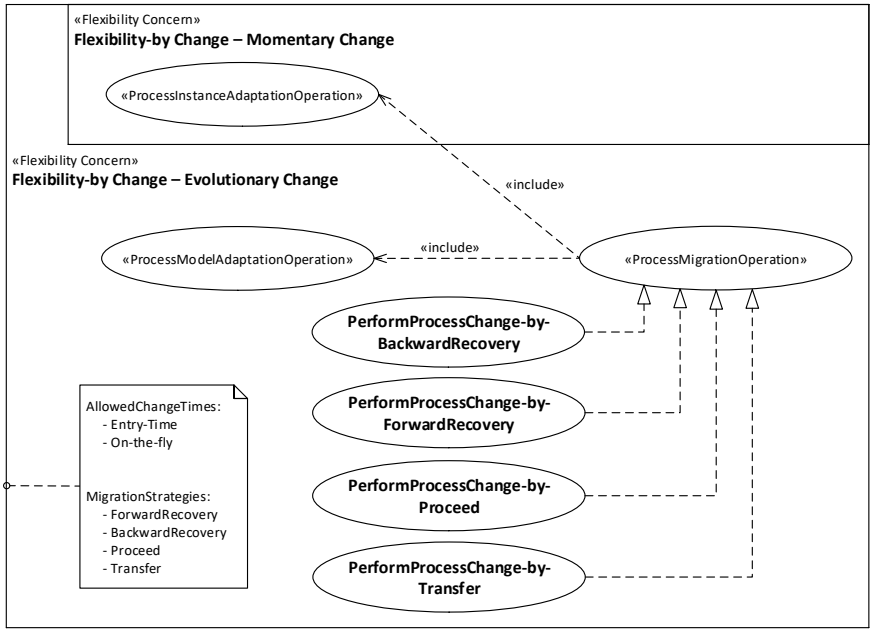


Abbildung 5-21: Gestaltungsaspekte für flexible und anpassbare Prozesse in Hinsicht auf Flexibility-by Change

5.3.1 Gestaltungsaspekte von Flexibility-by Change

Der Flexibilitätsaspekt *Flexibility-by Change* lässt sich in zwei Typen unterscheiden. So wird durch *Schonenberg et. al* [Sch+08] zwischen der momen-

tanen Anpassung (*Momentary Change*) und der evolutionären Anpassung (*Evolutionary Change*) unterschieden. Es können zusätzlich unterschiedliche Zeitpunkte für Anpassungen bestehen, die im Anschluss beschrieben werden.

Typ Momentary Change

Momentane Anpassungen betreffen lediglich Prozessinstanzen (siehe Abschnitt 2.2.3). Der Typ wird *Momentary Change* genannt, da eine Anpassung nur ausgewählte Prozessinstanzen und nicht etwa das zugehörige Prozessmodell betrifft. Werden lediglich Prozessinstanzen ohne das zugehörige Prozessmodell angepasst, kann der Fall eintreten, dass die angepassten Prozessinstanzen möglicherweise nicht mehr konform zu ihrem Prozessmodell sind. Dies kann je nach Auswirkung und Anforderungen an die Anpassung ein hinzunehmender Umstand sein. Sollte dies jedoch nicht gewünscht sein, so bieten sich Anpassungen im Rahmen des Typs *Evolutionary Change* an.

Typ Evolutionary Change

Bei einer Anpassung im Rahmen des Typs *Evolutionary Change* wird in einem ersten Schritt zunächst ein Prozessmodell angepasst. In einem zweiten Schritt werden die durchgeführten Anpassungen in bestehende Prozessinstanzen migriert. Eine diesem Typ zugehörige Anpassung kann sowohl Prozessmodelle als auch die zugehörigen Prozessinstanzen betreffen. Hierbei sind zum einen bestehende Prozessinstanzen gemeint, deren Anpassung durch eine Migration von Anpassungen auf Basis des zugehörigen Prozessmodells durchgeführt wird. Zum anderen sind aber insbesondere auch zukünftige Prozessinstanzen gemeint. Hier kann von impliziten Anpassungen gesprochen werden, da neue Prozessinstanzen auf Basis des angepassten Prozessmodells erstellt werden, auf dem die Anpassung bereits angewendet worden ist. Somit sind die Anpassungen automatisch in zukünftigen Prozessinstanzen enthalten.

Zeitpunkte für Anpassungen

Eine Ausführung von Anpassungen im Rahmen der zuvor aufgeführten Typen *Momentary Change* und *Evolutionary Change* des Flexibilitätsaspekts *Flexibility-by Change* kann problematisch sein, wenn sie zu beliebigen Zeitpunkten vorkommen. So könnten sich z.B. anzupassende Prozesse zum gewählten Zeitpunkt in einer kritischen Phase der Ausführung befinden, sodass ihre Anpassung ein ungewollt hohes Risiko für den zuverlässigen Betrieb einer Anwendung enthalten könnte. Daher kann es notwendig sein, dass zur Verfügung stehende Zeitpunkte, an denen Anpassungen erlaubt sind, explizit beschrieben werden sollten. *Schonenberg et. al* [Sch+08] stellen hierzu die beiden Typen von Zeitpunkten *Entry-Time* und *On-the-fly* vor. Auf eine Erläuterung dieser Typen von Zeitpunkten wird im Folgenden eingegangen.

Typ von Zeitpunkten: Entry-Time

Bei dem Typ *Entry-Time* werden Operationen zur Anpassung nur unmittelbar bei Instanziierung eines Prozesses angewendet. Somit sind in diesem Fall keine weiteren Anpassungen von der Prozessinstanz während der Ausführung vorgesehen. Im Fall von Operationen im Rahmen des Typs *Momentary Change* wird dabei lediglich eine Prozessinstanz angepasst. Im alternativen Fall einer Operation im Rahmen des Typs *Evolutionary Change* werden Anpassungen von Prozessmodellen angewendet. Somit enthalten zukünftig instanziierte Prozesse bereits die durchgeführten Anpassungen. Bereits existierende Prozessinstanzen werden nicht angepasst (siehe z.B. auch Strategie *Proceed* Abschnitt 5.3.2.3).

Typ von Zeitpunkten: On-the-fly

Durch den Typ *On-the-fly* werden sonstige Zeitpunkte beschrieben, die während der Ausführung eines Prozesses vorkommen können. Im Fall von Operationen im Rahmen des Typs *Momentary Change* werden Anpassungen von Prozessinstanzen vorgenommen. Derartige Anpassungen können sinnvoll sein, wenn sie lediglich für eine oder wenige konkrete Situationen im Betrieb in Betracht kommen und somit einmaliger Natur sind. Im Fall einer Operation im Rahmen des Typs *Evolutionary Change* werden Anpassungen sowohl an Prozessmodellen als auch an bestehenden Prozessinstanzen vorgenommen. Die Unterstützung von Zeitpunkten des Typs *On-the-fly* kann dabei herausfordernd sein. So kann es z.B. viele Prozessinstanzen geben, die unmittelbar angepasst werden müssen. Da Prozesse häufig in Abhängigkeit zu anderen Prozessen stehen, kann es hier zu Verzögerungen in der Ausführung kommen. Daher sind geeignete Strategien für Migrationen von Anpassungen von Prozessmodellen notwendig, die geplant und strukturiert mit derartigen Herausforderungen umgehen können.

Im Rahmen von Anpassungen von Prozessen des Typs *Evolutionary Change* können unterschiedliche Strategien für Migrationen verwendet werden. Auf eine grundlegende Übersicht über diese Strategien wird im Folgenden näher eingegangen. Sie bilden dabei die Grundlage für die spätere Definition von den in Abbildung 5-21 dargestellten Operationen zur Unterstützung von Migrationen (siehe Abschnitt 5.3.4). Anpassungen des Typs *Momentary Change* können durch Operationen angeboten werden, die durch Effektorschnittstellen einer Systemkomponente des Typs *BPExecutionComponent* angeboten werden (siehe Abschnitt 4.3.2). Eine Herleitung von möglichen Operationen wurde dabei bereits in Abschnitt 4.3.3 vorgenommen, sodass auf eine weitere Ausführung an dieser Stelle verzichtet werden kann.

5.3.2 Migrationsstrategien

Für die Realisierung von Operationen im Rahmen des Typs *Evolutionary Change* ist der Einsatz von unterschiedlichen Strategien für eine Migration von Anpassungen möglich. Dabei existieren verschiedene Strategien [RR10; Bar+11; Sch+12], die je nach benötigtem Grad an Konformität zwischen Prozessmodell und Prozessinstanz in einem konkreten Anwendungskontext zu wählen sind. In Anlehnung an *Schonenberg et. al* [Sch+08] werden im Folgenden verschiedene Typen von Strategien für derartige Migrationen kurz vorgestellt.

Szenario für Migrationen

In Abbildung 5-22 ist ein grundlegendes Szenario dargestellt, von dem in den nachfolgenden Beschreibungen ausgegangenen wird. Dabei wird auf Basis des Prozessmodells *PM* eine Operation zur Anpassung (*ProcessModelAdaptationOperation*) angewendet (siehe auch Abschnitt 4.3.3). Als Ergebnis der Anwendung dieser Operation wird das Prozessmodell *PM'* erzeugt und dargestellt. Ferner ist auf Basis der Prozessmodelle *PM* und *PM'* jeweils eine Menge von Prozessinstanzen (*PI 1* bis *PI n* bzw. *PI' 1* bis *PI' m*) gezeigt. Die Prozessinstanzen werden als konform (hier: *CompliantTo*) zu ihrem jeweiligen Prozessmodell dargestellt. Je nach gewählter Strategie zur Migration von Anpassungen sind zu spezifischen Zeitpunkten auch Anpassungen von den Prozessinstanzen möglich. Eine Migration dient dabei der Erhaltung der Beziehung *CompliantTo*.

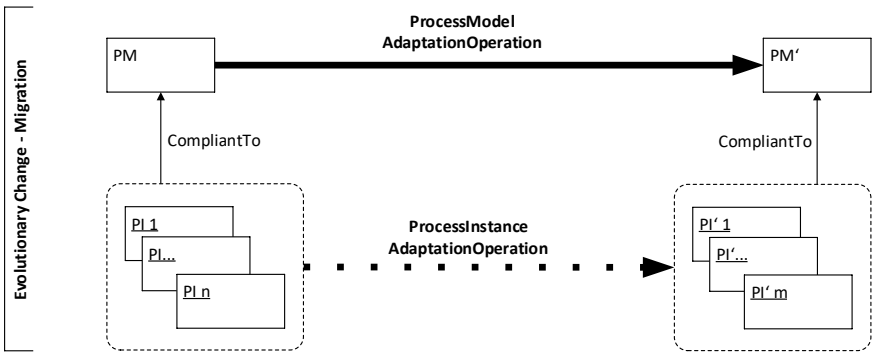


Abbildung 5-22: *Szenario für Migrationen im Rahmen des Typs Evolutionary Change*

Nachfolgend wird für jeden Strategietyp von Migrationen zunächst eine generelle Beschreibung gegeben. Anschließend folgt jeweils eine Beschreibung des Funktionsprinzips in Anlehnung an das in Abbildung 5-22 dargestellte Szenario.

5.3.2.1 Forward Recovery

Im Rahmen von Migrationen des Typs *Forward Recovery* werden bestehende Prozessinstanzen abgebrochen. Anschließend werden keine Anpassungen von Prozessinstanzen durchgeführt. Dies bedeutet für derartige Migrationen, dass die angewendeten Anpassungen am Prozessmodell lediglich in zukünftigen Prozessinstanzen enthalten sein werden. Dies lässt sich dadurch begründen, dass zukünftige Prozessinstanzen auf Basis des angepassten Prozessmodells erstellt sein werden. Eine weitere Behandlung im Rahmen dieses Typs von Migrationen, wie z.B. eine Kompensation, wird nicht unterstützt. Es kann somit davon ausgegangen werden, dass der Erhalt der Beziehung *CompliantTo* bei dem Typ *Forward Recovery* eingehalten wird. Dies lässt sich dadurch begründen, dass neue Prozessinstanzen stets auf Basis des angepassten Prozessmodells erstellt werden und nicht konforme Prozessinstanzen abgebrochen worden sind.

Im oberen Bereich von Abbildung 5-23 ist hierzu eine schematische Darstellung des Funktionsprinzips von Migrationen des Typs *Forward Recovery* dargestellt. So werden die bestehenden Prozessinstanzen (*PI 1* bis *PI n*) auf Basis des Prozessmodells *PM* in *Schritt 1* (*Cancellation*) abgebrochen. In Folge des Abbruchs können in *Schritt 4* (*Create*) neue Prozessinstanzen (*PI′ 1* bis *PI′ m*) auf Basis des angepassten Prozessmodells *PM′* erstellt werden.

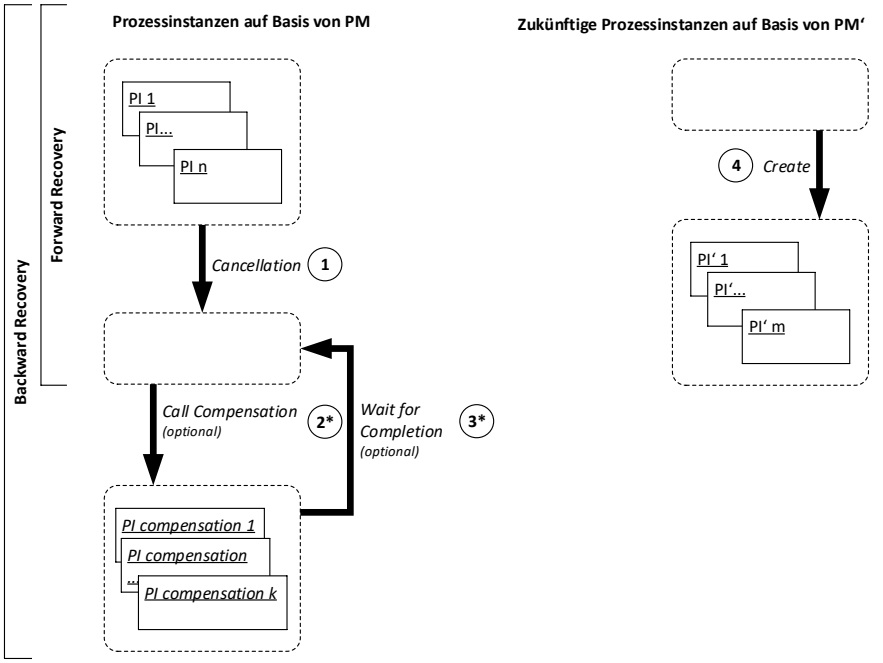


Abbildung 5-23: *Schematische Darstellung der Funktionsprinzipien von Migrationen der Typen Forward Recovery und Backward Recovery*

Die Anzahl an neuen Prozessinstanzen *PI′ ** kann dabei von der Anzahl zuvor abgebrochener Prozessinstanzen abweichen, sofern dies erforderlich ist. Eine Migration des Typs *Forward Recovery* ist mit diesem Schritt abgeschlossen. Die neu erstellten Prozessinstanzen *PI′ ** sind dabei in Beziehung zu *PM′* als konform zu betrachten (siehe Abbildung 5-22).

5.3.2.2 Backward Recovery

Bei Migrationen des Typs *Backward Recovery* sind der erste und der letzte Schritt identisch wie beim Typ *Forward Recovery*. So werden bestehende Prozessinstanzen zunächst abgebrochen und neue Prozessinstanzen auf Basis eines angepassten Prozessmodells erstellt. Bevor neue Prozessinstanzen erstellt werden, können weitere optionale Behandlungen, wie z.B. zum Zweck einer Kompensation, durchgeführt werden. Wie für den Typ *Forward Recovery* ausgeführt, bleibt der Erhalt der Beziehung *CompliantTo* bei dem Typ *Backward Recovery* erhalten, da ebenso neue Prozessinstanzen stets auf Basis des angepassten Prozessmodells erstellt werden.

Eine schematische Darstellung des Funktionsprinzips von Migrationen des Typs *Backward Recovery* ist im unteren Bereich von Abbildung 5-23 dargestellt. Hierbei eingeschlossen sind die bei dem Typ *Forward Recovery* durchgeführten Schritte, in denen zunächst bestehende Prozessinstanzen abgebrochen werden (*Schritt 1*) und am Ende neue Prozessinstanzen erstellt werden (*Schritt 4*).

Berücksichtigung von Kompensationen

Zwischen diesen Schritten ist die Ausführung von optionalen Schritten möglich (*Call Compensation* und *Wait for Completion*). Der optionale *Schritt 2* ermöglicht, weitere Prozessinstanzen auszuführen, die hier als *PI compensation 1* bis *PI compensation k* dargestellt sind. Hierdurch können Maßnahmen zur Kompensation von bereits durchgeführten Aktivitäten für die beendeten Prozessinstanzen *PI 1* bis *PI n* angewendet werden. Dabei kann die Anzahl an Prozessinstanzen *PI compensation ** von der Anzahl an beendeten Prozessinstanzen *PI ** abweichen. Dies lässt sich dadurch erklären, dass nicht für jede der beendeten Prozessinstanzen *PI ** eine Maßnahme zur Kompensation möglich oder gar notwendig ist. Eine Prozessinstanz *PI i* wird dabei durch die Prozessinstanz *PI compensation i* kompensiert. Im nachfolgenden optionalen *Schritt 3* wird auf Beendigung der Ausführung der Prozessinstanzen *PI compensation ** gewartet. In diesem Bezug können zwei Verfahrensweisen sinnvoll sein. Zum einen kann auf die Beendigung der Ausführung aller Prozessinstanzen *PI compensation ** gewartet werden, bevor neue Prozessinstanzen *PI′ ** erstellt werden. Zum anderen kann eine neue Prozessinstanz *PI′ i* auch nach Beendigung der zugehörigen Prozessinstanz zur Kompensation gestartet werden.

Das Warten auf die Beendigung der Ausführung aller Prozessinstanzen mit dem Zweck der Kompensation bietet sich insbesondere in derartigen Fällen an, in denen einen hohe Abhängigkeit zu weiteren Prozessen besteht, sodass unterschiedliches Verhalten im Betrieb vermieden wird. Anderenfalls kann *Schritt 3* ebenfalls als optional betrachtet werden, wenn das Warten für die Instanziierung von zukünftigen Prozessinstanzen auf Basis von *PM'* nicht notwendig ist. Eine Migration des Typs *Backward Recovery* ist mit *Schritt 4* abgeschlossen.

5.3.2.3 Proceed

Soll eine Anpassung nur zukünftige Prozessinstanzen betreffen, so können Migrationen des Typs *Proceed* verwendet werden. Dabei werden aktuell bestehende Prozessinstanzen nicht wie bei den Typen *Forward Recovery* und *Backward Recovery* abgebrochen. Stattdessen ist es vorgesehen, dass die Ausführung von bestehenden Prozessinstanzen regulär beendet wird und neue Prozessinstanzen auf Basis eines angepassten Prozessmodells erstellt werden. Die Strategie des Typs *Proceed* sieht somit keine zusätzliche Verfahrensweise für bereits erstellte und aktuell ausgeführte Prozessinstanzen vor. Der Erhalt der Beziehung *CompliantTo* bei dem Typ *Proceed* bleibt somit ebenso erhalten.

In Abbildung 5-24 ist hierzu eine schematische Darstellung des Funktionsprinzips von Migrationen des Typs *Proceed* dargestellt. So koexistieren sowohl Prozessinstanzen *PI ** als auch Prozessinstanzen *PI' ** zur gleichen Zeit. Sie sind jeweils auf Basis der Prozessmodelle *PM* bzw. *PM'* erstellt worden. Dabei können die beiden Schritte *1a* (*Wait for Regular Completion*) und *1b* (*Create*) parallel durchgeführt werden.

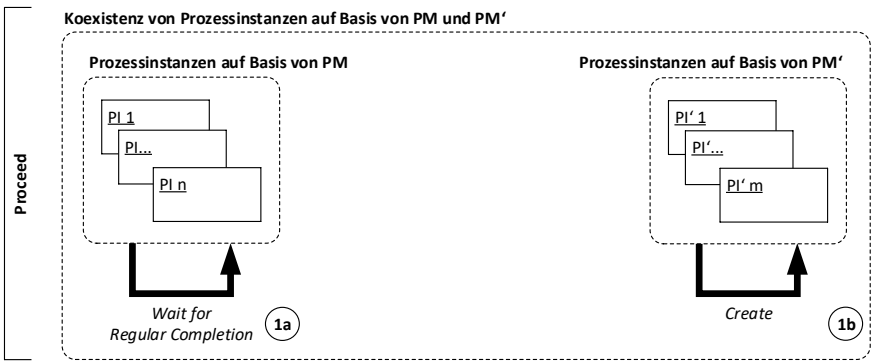


Abbildung 5-24: Schematische Darstellung des Funktionsprinzips von Migrationen des Typs Proceed

Schritt 1a sieht die reguläre Beendigung der Ausführung von Prozessinstanzen $\underline{PI\ *}$ auf Basis von Prozessmodell *PM* vor. In *Schritt 1b* können zeitgleich zusätzliche Prozessinstanzen $\underline{PI'\ *}$ parallel dazu auf Basis des Prozessmodells *PM'* erstellt und ausgeführt werden. Eine Migration des Typs *Proceed* kann als abgeschlossen betrachtet werden, wenn alle Prozessinstanzen $\underline{PI\ *}$ beendet worden sind.

5.3.2.4 Transfer

Im Rahmen von Migrationen des Typs *Transfer* ist es möglich, die Ausführung von bestehenden Prozessinstanzen zu unterbrechen und auf Basis neu erstellter Prozessinstanzen fortzusetzen. Hierzu werden zunächst die internen Zustände einer Prozessinstanz gesichert. Anschließend wird die weitere Ausführung der Prozessinstanz abgebrochen. Ein solcher interner Zustand kann z.B. durch Informationen zu bisher durchgeführten oder aktiven Aktivitäten bzw. Tasks, durch aufgekommene Ereignisse oder durch die Allokation einer Datenquelle gegeben sein.

Die Fortsetzung der Ausführung auf Basis neu erstellter Prozessinstanzen bedingt dabei eine vorherige Zuordnung (engl. *Mapping*) von vorangegangenen gesicherten internen Zuständen zu zugehörigen internen Zuständen der neuen Prozessinstanzen. Derartige Zuordnungen können je nach Umfang oder Art der durchgeführten Anpassung am Prozessmodell nicht möglich sein, da z.B. eine Aktivität im angepassten Prozessmodell nicht mehr vorhanden ist. In solchen Fällen ist eine Lösung dadurch gegeben, dass z.B. die Ausführung dieser Aktivität übersprungen wird (engl. *skip*) und in der korrespondierenden Folgeaktivität fortgesetzt wird. Ferner ist es aber auch möglich, dass eine allokierte Datenquelle nicht mehr benötigt wird, sodass sie freigegeben werden könnte. Wurden gesicherte interne Zustände auf korrespondierende Zustände der neuen Prozessinstanzen zugeordnet, kann die Ausführung fortgesetzt werden und die Migration des Typs *Transfer* ist abgeschlossen.

In Abbildung 5-25 ist hierzu eine schematische Darstellung des Funktionsprinzips von Migrationen des Typs *Transfer* dargestellt. So werden in *Schritt 1* (*Store*) zunächst für die Migration relevante interne Zustände der Prozessinstanzen $\underline{PI\ *}$ gesichert. In *Schritt 2* (*Cancellation*) wird die Ausführung der Prozessinstanzen $\underline{PI\ *}$ abgebrochen. Durch *Schritt 3* (*Create*) wird die Erstellung neuer Prozessinstanzen $\underline{PI'\ *}$ auf Basis des angepassten Prozessmodells *PM'* vorgenommen. Bevor die Fortsetzung einer Ausführung durch die Prozessinstanzen $\underline{PI'\ *}$ stattfinden kann, müssen zuvor die in *Schritt 1* gesicherten internen Zustände auf interne Zustände der Pro-

zessinstanzen PI' * in *Schritt 4* (*Map*) zugeordnet werden. Im Anschluss an diese Zuordnung ist die Migration des Typs *Transfer* abgeschlossen. Die Beziehung *CompliantTo* bleibt auch beim Typ *Transfer* erhalten.

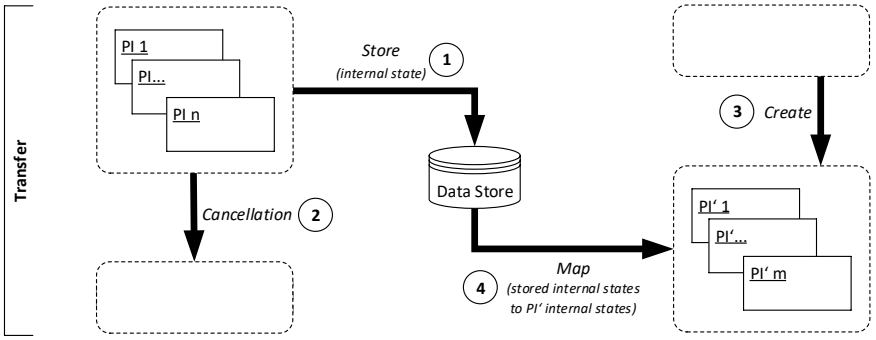


Abbildung 5-25: Schematische Darstellung des Funktionsprinzips von Migrationen des Typs Transfer

In Abbildung 5-26 ist ein Beispiel zur besseren Veranschaulichung einer Zuordnung interner Zustände von Prozessinstanzen mit dem Fokus auf Tasks dargestellt. Dabei wird das durch Abbildung 5-22 eingeführte Szenario hinsichtlich des dargestellten Detailgrades verfeinert.

So ist für die bereits eingeführten Inhalte *PM*, *PM'*, *PI n* und *PI' n* jeweils ein Kontrollfluss mit zwei Tasks dargestellt. In der Darstellung der Prozessinstanzen *PI n* und *PI' n* sind aktive Tasks in der Farbe *Rot* dargestellt. In der Prozessinstanz *PI n* ist somit zum Zeitpunkt des Starts einer Migration vom Typ *Transfer* der Task mit der Bezeichnung *B* aktiv. Eine Migration vom Typ *Transfer* überführt die internen Zustände der Prozessinstanz *PI n* in die internen Zustände der Prozessinstanz *PI' n* in Anlehnung an eine durchgeführte Operation (hier: *ProcessModelAdaptationOperation*).

Die in dem Beispiel aufgezeigten internen Zustände beziehen sich dabei ausschließlich auf die Zustände der Lebenszyklen der Tasks *A*, *B* und *A'* (siehe auch Abschnitt 4.3.4.2). So könnte ein aktiver Task *A* als auch ein aktiver Task *B* der Prozessinstanz *PI n* auf einem aktiven Task *A'* der Prozessinstanz *PI' n* zugeordnet werden. Die Durchführung einer Migration des Typs *Transfer* könnte dabei Bezug auf diese Zuordnung von internen Zuständen nehmen. Das zugehörige Resultat ist im unteren Bereich von Abbildung 5-26 dargestellt. Selbstverständlich lassen sich ebenfalls alternative Zuordnungen in Anlehnung an konkrete Anforderungen erstellen. Hierdurch können Zustände nicht nur von Tasks, sondern auch von weiteren Elementen aus den Perspektiven von Prozessen (siehe Abschnitt 4.3.3) migriert werden.

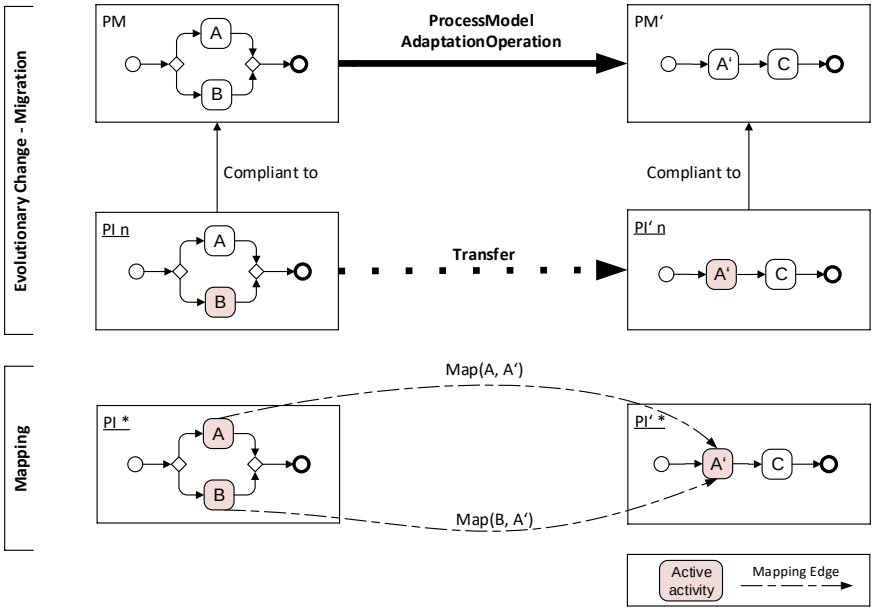


Abbildung 5-26: Schematische Darstellung einer Zuordnung von internen Zuständen zweier Prozessinstanzen

5.3.3 Spracherweiterung für Flexibility-by Change

In den vorherigen Abschnitten wurde auf Details des Flexibilitätsaspekts *Flexibility-by Change* eingegangen. Dabei wurde herausgestellt, dass bei der Anpassung von Prozessen Konzepte benötigt werden, die bisher nicht Teil der Sprache *BPMN2.0* sind und somit nicht in der Gestaltung von flexiblen und anpassbaren Prozessen umgesetzt werden können. Ein Beispiel hierfür stellt die zuletzt vorgestellte Strategie *Transfer* dar. Hier werden im Rahmen des dargestellten Funktionsprinzips Eigenschaften von Prozessinstanzen in Form von internen Zuständen gesichert und anschließend bei der Migration in neuen Prozessinstanzen wiederhergestellt.

Notwendigkeit einer Spracherweiterung

Es fehlen in der Sprache *BPMN2.0* zum einen Sprachelemente, um zwischen den Elementen von Prozessmodellen und Elementen von Prozessinstanzen unterscheiden zu können. Zum anderen kann aber auch die Notwendigkeit zur Gestaltung von weiteren Eigenschaften von Prozessen existieren, die lediglich zur Laufzeit bestehen. Um die Gestaltung von Prozessen in diesem Bezug weiter unterstützen zu können, werden im Folgenden exemplarische Erweiterungen der Sprache *BPMN2.0* vorgestellt, mit denen die zuvor aufgeführte Anforderung erfüllt werden kann. Soll bereits in der Gestaltung von Prozessen beschrieben werden können, welche internen Zustände von Aktivitäten verfügbar und anpassbar sind, so müssen demnach zusätzliche Elemente verfügbar sein, die eine Repräsen-

tation dieser Laufzeiteigenschaften darstellen. Ein Beispiel zur Beschreibung aktiver Taskinstanzen innerhalb einer Prozessinstanz sowie ihrer internen Zustände ist in Abbildung 5-27 dargestellt.

Darstellung von Prozessinstanzen

Durch diese Darstellung sind aktuell aktive Taskinstanzen innerhalb der Prozessinstanz beschreibbar. Die Darstellung einer Repräsentation einer Prozessinstanz *PI* wird in Anlehnung an Elemente des Typs *Pool* der Sprache *BPMN2.0* vorgenommen. In der Prozessinstanz *PI* wird ein Kontrollfluss bestehend aus den drei Taskinstanzen *Task A*, *Task B* und *Task C* gezeigt. Die Liste derzeit aktiver Tasks (*activeTasks*) ist durch ein Kommentarfeld gegeben. Listenelemente, wie z.B. *Task B*, werden in geschweiften Klammern und getrennt durch ein Komma angegeben.

Darstellung von Taskinstanzen

Durch die gezeigte Darstellung können interne Zustände von Taskinstanzen beschrieben werden. Elemente vom Typ *TaskInstanceRepresentation* werden in Anlehnung an Elemente des Typs *Task* der Sprache *BPMN2.0* dargestellt. Jedes dieser Elemente besitzt einen spezifischen Lebenszyklus (*Lifecycle*). Dieser enthält den aktuellen Zustand, der durch das Attribut *activeState* beschrieben wird. Ein Element vom Typ *Lifecycle* wird in Anlehnung an *Klassen* der *UML* dargestellt. Die Zugehörigkeit zu einer Taskinstanz wird in Anlehnung an *Assoziationen* in Form von *Komposition* der *UML* dargestellt.

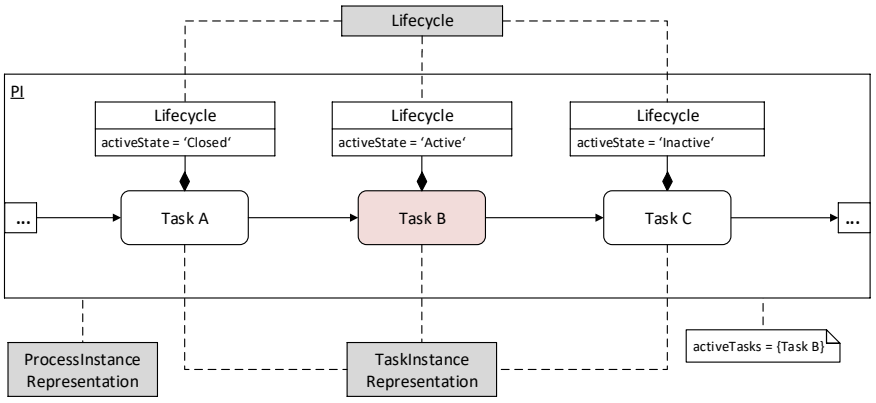


Abbildung 5-27: Darstellung von Elementen der laufzeitspezifischen Erweiterung zur Unterstützung von Flexibility-by Change

Semantik

Der gezeigte Auszug stellt dabei eine Situation in der Ausführung der Prozessinstanz *PI* dar, auf die nachfolgend kurz eingegangen wird. Die Ausführung der Taskinstanz *Task A* ist bereits abgeschlossen, sodass das Attribut *activeState* den Wert *Closed* hat. Derzeit wird *Task B* ausgeführt, weshalb das Attribut *activeState* den Wert *Active* hat. Durch den Wert *Inactive* wird ausgedrückt, dass *Task C* noch nicht ausgeführt werden kann, weil hierfür noch nicht alle Bedingungen erfüllt sind. Derartige Bedingungen sind

in diesem Beispiel durch einen Token gegeben, der nach Beendigung von *Task B* über die dargestellte Assoziation vom Typ *SequenceFlow* zur Aktivierung von *Task C* führt. Prinzipiell ist es möglich, dass sich innerhalb einer Prozessinstanz mehrere Lebenszyklen im Zustand *Active* befinden. In einem solchen Fall wird angenommen, dass die zugehörigen Instanzen von Tasks parallel ausgeführt werden.

Abstrakte Syntax

Die zugehörige abstrakte Syntax der zuvor beschriebenen Darstellung von Prozess- und Taskinstanzen ist in Abbildung 5-28 beschrieben. Dabei ist ein erweitertes Metamodell des in Abschnitt 4.3.3 vorgestellten Auszugs der Sprache *BPMN2.0* dargestellt. Der Zweck der Erweiterung ist dadurch gegeben, dass benötigte Eigenschaften in Hinsicht auf die Laufzeit des Prozesses explizit in der Gestaltung von Prozessen berücksichtigt werden sollen.

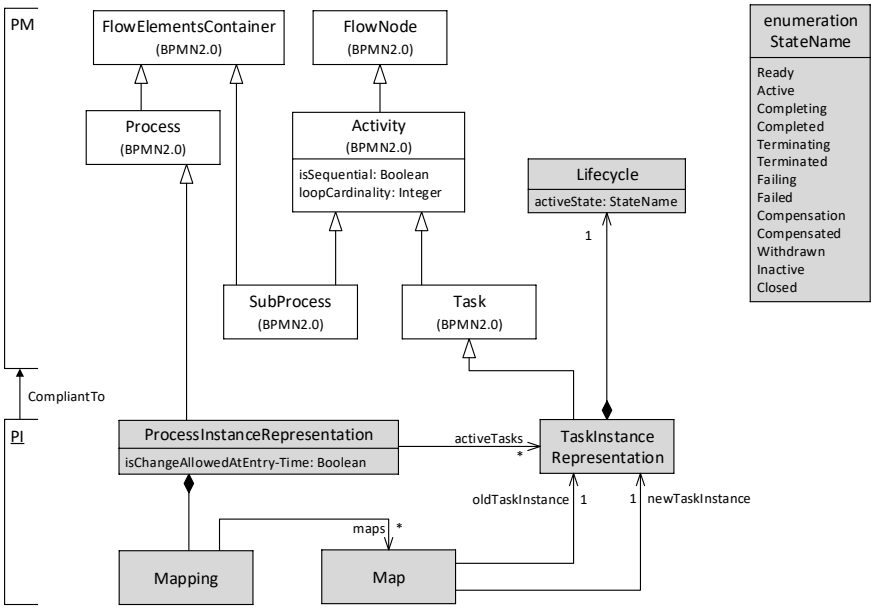


Abbildung 5-28: Auszug einer Erweiterung des Metamodells der Sprache BPMN2.0 zur Unterstützung von Flexibility-by Change

In dem Auszug werden für die zuvor genannten Elemente *Prozessinstanz* und *Taskinstanz* neue Typen von Elementen eingeführt. Die Instanz eines Prozesses wird dabei durch den Typ *ProcessInstanceRepresentation* repräsentiert. Ein Element dieses Typs kann weitere Elemente enthalten, wobei sich – wie zuvor bereits erwähnt – vor allem auf Tasks fokussiert wird. Daher ist ebenso eine Repräsentation von Instanzen der in einer Prozessinstanz vorkommenden Tasks notwendig. Der Typ *TaskInstanceRepresentation* stellt diesen Typ dar. Ferner enthält ein Element dieses Typs zudem ein Element vom Typ *Lifecycle*, welches den aktuellen Zustand des Lebenszy-

klus des Tasks beschreibt. Hierzu wird das Attribut *activeState* vom Typ *StateName* verwendet. Bei dem Typ *StateName* handelt es sich um eine Enumeration mit Literalen aller bereits in Abschnitt 4.3.4 eingeführten Zustände des Lebenszyklus von Aktivitäten. Damit die Verwaltung von aktiven Taskinstanzen vereinfacht werden kann, enthält der Typ *ProcessInstanceRepresentation* darüber hinaus eine Liste mit dem Bezeichner *activeTasks*, die vom Typ *TaskInstanceRepresentation* ist. Hierdurch kann spezifiziert werden, welche Taskinstanzen innerhalb einer Prozessinstanz aktiv sind (siehe auch Abbildung 5-27).

Erweiterung für erlaubte Zeitpunkte

Sollen Anpassungen von Prozessinstanzen nur zu einem bestimmten Zeitpunkt erlaubt sein, so kann dies durch das hier dargestellte Attribut mit der Bezeichnung *isChangeAllowedOnEntry-Time* vom Typ *Boolean* bestimmt werden. Dabei steht der Wert *true* für erlaubte Anpassungen zu Zeitpunkten des Typs *Entry-Time*. Alternativ steht ein nicht spezifizierter Wert bzw. der Wert *false* für Anpassungen zu den Zeitpunkten des Typs *On-the-Fly*.

Erweiterung für Zuordnungen

Ein weiteres benötigtes Sprachelement für Migrationen des Typs *Transfer* ist durch das Konzept des *Mappings* gegeben. Ein *Mapping* kann wie in dem in Abbildung 5-26 dargestellten Beispiel als Tupel bestehend aus zwei Taskinstanzen beschrieben werden. In Abbildung 5-28 ist in diesem Bezug ebenso eine Erweiterung zur Beschreibung des Konzepts *Mapping* dargestellt. Ein Element des Typs *Mapping* beschreibt eine Menge von Zuordnungen verschiedener interner Zustände von Taskinstanzen. Hierzu enthält dieses Element eine Liste mit dem Bezeichner *maps* vom Typ *Map*. Eine einzelne Zuordnung wird durch ein Element des Typs *Map* definiert. So kann eine Zuordnung einer alten Taskinstanz (*oldTaskInstance*) auf eine neue Taskinstanz *(newTaskInstance*) definiert werden.

Die zuvor beschriebene Erweiterung stellt ein Beispiel für die Beschreibung von laufzeitspezifischen Eigenschaften in der Gestaltung von flexiblen und anpassbaren Prozessen dar. Sollen weitere Eigenschaften hinsichtlich der Laufzeit berücksichtigt werden, sind weitere Erweiterungen notwendig, die diese enthalten. Im Rahmen der weiteren Beschreibung der Flexibilitätsaspekte *Flexibility-by Deviation* (siehe Abschnitt 5.4) und *Flexibility-by Underspecification* (siehe Abschnitt 5.5) wird diese Erweiterung ebenso verwendet sowie weitere vorgestellt.

5.3.4 Operationen

Damit die Gestaltung von flexiblen und anpassbaren Prozessen hinsichtlich des Typs *Flexibility-by Evolutionary Change* unterstützt werden kann,

sind Operationen notwendig. Derartige Operationen können als Teil des Verhaltens eines Anpassungsprozesses (siehe Abschnitt 4.2.3) genutzt werden. Die nachfolgend gezeigten Operationen unterstützen dabei die einzelnen in Abschnitt 5.3.2 vorgestellten Funktionsprinzipien von Strategien. Eine Übersicht über die resultierende Menge an Operationen zur Unterstützung der Gestaltung von *Flexibility-by Evolutionary Change* ist in Abbildung 5-29 dargestellt. Dabei lassen sich auf Basis der zuvor beschriebenen Strategien insgesamt vier Operationen benennen.

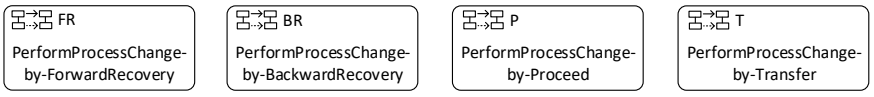


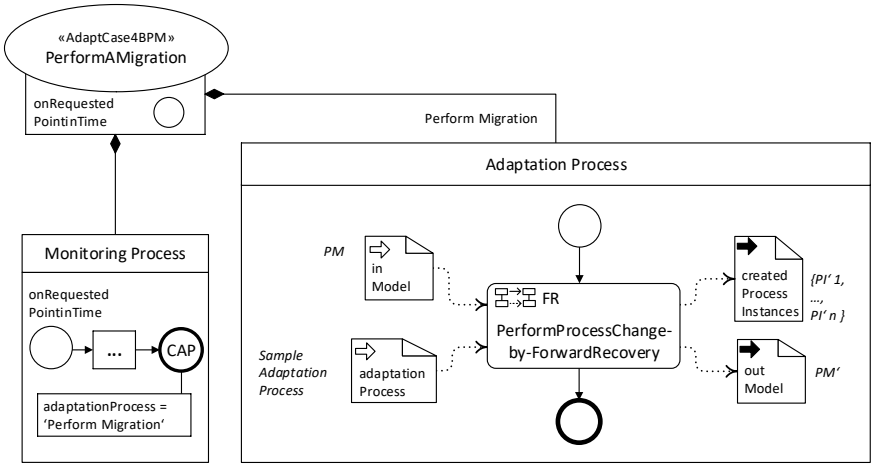
Abbildung 5-29: *Operationen zur Unterstützung von Flexibility-by Evolutionary Change*

Das Konzept für die in Abbildung 5-29 gezeigten Operationen sieht dabei vor, zunächst die Anpassung von Prozessmodellen durchzuführen. Anschließend soll der jeweiligen Strategie folgend mit existierenden Prozessinstanzen bzw. neu zu erstellenden Prozessinstanzen verfahren werden. Ferner steht die Anwendung der Operationen in Abhängigkeit zu erlaubten Typen von Zeitpunkten *Entry-Time* bzw. *On-the-fly*. Jede der Operationen prüft dabei vor der Durchführung ihrer eigentlichen Funktion, ob eine Anwendung hinsichtlich der genannten Typen von Zeitpunkten erlaubt ist. In dem Fall, dass eine Durchführung nicht erlaubt ist, beendet die Operation ihre Ausführung.

Ein Beispiel für eine Verwendung einer Operation ist in Abbildung 5-30 gezeigt. Eine Auslösung des Beobachtungsprozesses ist zu einem bestimmten Ereignis (*onRequestedPointinTime*) angedacht. Dabei wird der Anpassungsprozess (*Perform Migration*) aufgerufen, in dem das Verhalten für die Migration des Typs *Forward Recovery* enthalten ist. Die Parametrisierung der Operation *PerformProcessChange-by-ForwardRecovery* adressiert zunächst das betroffene Prozessmodell *PM*. Nach der Anwendung des durch einen weiteren Anpassungsprozess (*Sample-Adaptation-Process*) beschriebenen Verhaltens für die Anpassung am Prozessmodell *PM* wird das geänderte Prozessmodell *PM'* erzeugt. Das Verhalten zum Abbruch bestehender Prozessinstanzen $\underline{PI}$ und zur Erstellung neuer Prozessinstanzen $\underline{PI'*}$ wird durch die Operation *PerformProcessChange-by-ForwardRecovery* gekapselt.

Die zu jeder Operation zugehörige Implementierung muss dabei auch Eigenschaften eines konkreten IT-Unterstützungssystems, wie z.B. in Form einer Workflow-Engine berücksichtigen, um aktuelle Prozessinstanzen abbrechen und zukünftige erstellen zu können. Eine derartige Berücksichti-

Abbildung 5-30: Beispiel für die Gestaltung einer Migration



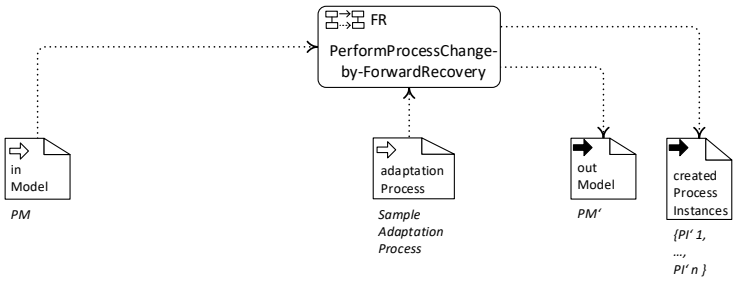
gung ist kein Fokus in dieser Arbeit, sodass sie nur der Vollständigkeit halber genannt ist. Für eine Ausführung jeder einzelnen Operation durch ihre Signatur und der konkreten Syntax wird an dieser Stelle auf die Inhalte in den nachfolgenden Abschnitten 5.3.4.1 und 5.3.4.4 verwiesen.

5.3.4.1 PerformProcessChange-by-ForwardRecovery

Die Operation *PerformProcessChange-by-ForwardRecovery* unterstützt die Durchführung von Migrationen des Typs *ForwardRecovery*. Dabei wird das zuvor durch Abbildung 5-23 dargestellte und bereits beschriebene Funktionsprinzip umgesetzt. Die Signatur und konkrete Syntax der Operation *PerformProcessChange-by-ForwardRecovery* sind in Abbildung 5-31 dargestellt.

Abbildung 5-31: Signatur und konkrete Syntax der Operation PerformProcessChange-by-ForwardRecovery

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	adaptationProcess	*AdaptationProcess*
OUT :	*outModel*	*ProcessModel*
	createdProcessInstances	$Set\langle ProcessInstance\rangle$



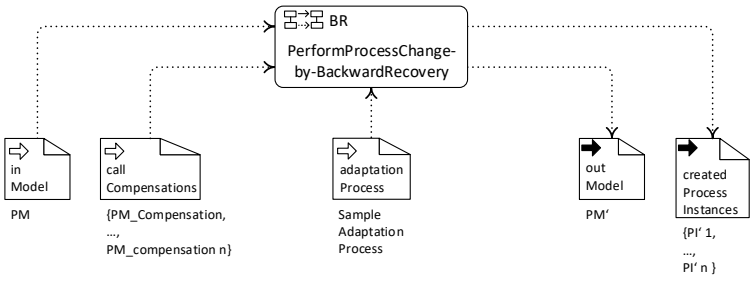
Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*) sowie einen Anpassungsprozess (*adaptationProcess*). Dabei wird das im Rahmen des Anpassungsprozesses beschriebene Verhalten zur Anpassung an dem Prozessmodell *inModel* genutzt. Durch die Verwendung eines Anpassungsprozesses anstelle einer einfachen Operation zur Anpassung, wie z.B. *AddNode* (siehe Anhang A.1.1), können auch komplexere Anpassungen im Rahmen der Migration umgesetzt werden. Die Operation erzeugt die beiden Ausgaben eines geänderten Prozessmodells (*outModel*) sowie einer Menge neu erstellter Prozessinstanzen (*createdProcessInstances*). *Parameter*

5.3.4.2 PerformProcessChange-by-BackwardRecovery

Die Operation *PerformProcessChange-by-BackwardRecovery* unterstützt die Durchführung von Migrationen des Typs *BackwardRecovery*. Dabei wird das zuvor durch Abbildung 5-23 dargestellte und bereits beschriebene Funktionsprinzip umgesetzt. Die Signatur und konkrete Syntax der Operation *PerformProcessChange-by-BackwardRecovery* sind in Abbildung 5-32 dargestellt.

	Parametername	**Parametertyp**
IN :	*inModel* *adaptationProcess*	*ProcessModel* *AdaptationProcess*
IN-Optional :	*callCompensations*	$Set\langle ProcessInstance, ProcessModel, Boolean\rangle$
OUT :	*outModel* *createdProcessInstances*	*ProcessModel* $Set\langle ProcessInstance\rangle$

Abbildung 5-32: Signatur und konkrete Syntax der Operation PerformProcessChange-by-BackwardRecovery



Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*) sowie einen Anpassungsprozess (*adaptationProcess*). Dabei wird das im Rahmen des Anpassungsprozesses beschriebene Verhalten zur Anpassung an dem Prozessmodell *inModel* genutzt. Durch die Verwendung eines Anpassungsprozesses anstelle einer einfachen Operation zur Anpassung, *Parameter*

wie z.B. *AddNode* (siehe Anhang A.1.1), können auch komplexere Anpassungen im Rahmen der Migration umgesetzt werden. Die Operation erzeugt die beiden Ausgaben eines geänderten Prozessmodells (*outModel*) sowie einer Menge neu erstellter Prozessinstanzen (*createdProcessInstances*).

Optionale Parameter

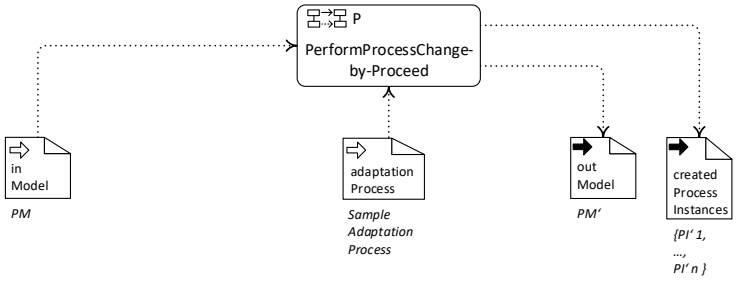
Ferner besitzt die Operation einen optionalen Parameter, durch den das Verhalten der zuvor beschriebenen Schritte *2* (*Call Compensation*) und *3* (*Wait for Completion*) spezifiziert werden kann. So kann durch den Parameter *callCompensations* eine Menge von Tripeln angegeben werden. Ein solches Tripel besteht aus jeweils einer Prozessinstanz, einem Prozessmodell sowie einem booleschen Wert. Dabei beschreibt das Prozessmodell das Verhalten, welches zur Kompensation der Prozessinstanz instanziiert werden soll. Typischerweise handelt es sich bei der hier kompensierten Prozessinstanz um eine der Instanzen, die von der Migration betroffen ist. Der boolesche Wert gibt dabei mit dem Standardwert *true* an, ob auf Beendigung der Prozessinstanz zur Kompensation gewartet werden soll. Soll dies nicht der Fall sein, so kann dies durch den Wert *false* angegeben werden.

5.3.4.3 PerformProcessChange-by-Proceed

Die Operation *PerformProcessChange-by-Proceed* unterstützt die Durchführung von Migrationen des Typs *Proceed*. Dabei wird das zuvor in Abbildung 5-24 dargestellte und bereits beschriebene Funktionsprinzip umgesetzt. Die Signatur und konkrete Syntax der Operation *PerformProcessChange-by-Proceed* sind in Abbildung 5-33 dargestellt.

Abbildung 5-33: Signatur und konkrete Syntax der Operation PerformProcessChange-by-Proceed

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	adaptationProcess	*AdaptationProcess*
	mappings	*Mapping*
	targetURI	*URI*
OUT :	*outModel*	*ProcessModel*
	createdProcessInstances	*Set⟨ProcessInstance⟩*



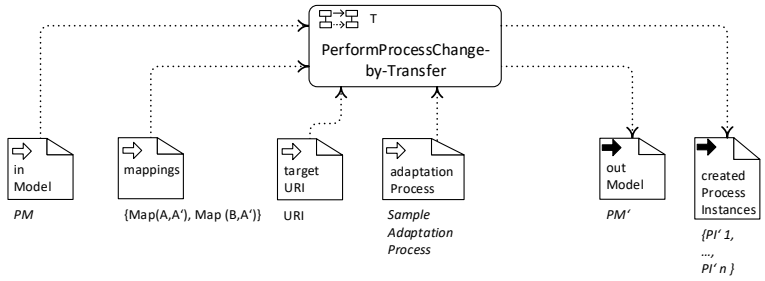
Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*) sowie einen Anpassungsprozess (*adaptationProcess*). Dabei wird das im Rahmen des Anpassungsprozesses beschriebene Verhalten zur Anpassung an dem Prozessmodell *inModel* genutzt. Durch die Verwendung eines Anpassungsprozesses anstelle einer einfachen Operation zur Anpassung, wie z.B. *AddNode* (siehe Anhang A.1.1), können auch komplexere Anpassungen im Rahmen der Migration umgesetzt werden. Die Operation erzeugt die beiden Ausgaben eines geänderten Prozessmodells (*outModel*) sowie einer Menge neu erstellter Prozessinstanzen (*createdProcessInstances*). *Parameter*

5.3.4.4 PerformProcessChange-by-Transfer

Die Operation *PerformProcessChange-by-Transfer* unterstützt die Durchführung von Migrationen des Typs *Transfer*. Dabei wird das zuvor durch Abbildung 5-25 dargestellte und bereits beschriebene Funktionsprinzip umgesetzt. Die Signatur und konkrete Syntax der Operation *PerformProcessChange-by-Transfer* sind in Abbildung 5-34 dargestellt.

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	adaptationProcess	*AdaptationProcess*
	mappings	*Mapping*
	targetURI	*URI*
OUT :	*outModel*	*ProcessModel*
	createdProcessInstances	*Set⟨ProcessInstance⟩*

Abbildung 5-34: Signatur und konkrete Syntax der Operation PerformProcessChange-by-Transfer



Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*) sowie einen Anpassungsprozess (*adaptationProcess*). Dabei wird das im Rahmen des Anpassungsprozesses beschriebene Verhalten zur Anpassung an dem Prozessmodell *inModel* genutzt. Durch die Verwendung eines Anpassungsprozesses anstelle einer einfachen Operation zur Anpassung, wie z.B. *AddNode* (siehe Anhang A.1.1), können auch komplexere Anpassungen im Rahmen der Migration umgesetzt werden. Durch den Parameter *targetURI* kann ein Speicherort für zu erstellende Mappings angegeben *Parameter*

werden. Welches Mapping für die Wiederherstellung verwendet werden soll, kann durch den Parameter *mappings* spezifiziert werden. Die Operation erzeugt die beiden Ausgaben eines geänderten Prozessmodells (*outModel*) sowie einer Menge neu erstellter Prozessinstanzen (*createdProcessInstances*).

5.3.5 Zusammenfassung

In diesem Abschnitt wurde der durch [Sch+08] eingeführte Flexibilitätsaspekt *Flexibility-by Change* durch ein weiteres Entwurfsmuster für die Gestaltung von flexiblen und vor allem anpassbaren Prozessen eingeführt. Dabei wurden zunächst die beiden Untertypen *Momentary Change* und *Evolutionary Change* vorgestellt. Für den Untertyp *Evolutionary Change* konnte festgestellt werden, dass bestimmte funktionale Aspekte bereits durch die in Abschnitt 4.3.3 eingeführten Operationen zur Anpassung von Prozessen beschrieben werden konnten. Dabei fehlte jedoch die Unterstützung von Anpassungen von Prozessmodellen unter Einhaltung der Konformität mit den zugehörigen Prozessinstanzen. Als Lösungskonzept wurden bereits existierende Strategien für Migrationen vorgestellt, die für die weitere Gestaltung in Form von Operationen in Anpassungsprozessen beschrieben worden sind. Zur Unterstützung einer Gestaltung von Prozessen unter Berücksichtigung von Eigenschaften hinsichtlich der Laufzeit von Prozess- und Taskinstanzen, die bisher in der Sprache *BPMN2.0* nicht berücksichtigt werden konnten, wurde zudem eine entsprechende Spracherweiterung vorgestellt. Durch die Komposition dieser zuvor aufgeführten Inhalte ist es nun möglich, Anpassungen durch Beobachtungs- und Anpassungsprozesse zu gestalten, sodass Prozessmodelle als auch deren Instanzen in Abhängigkeit zu einer gewählten Strategie in einer Beziehung zueinanderstehen, die als konform bezeichnet werden kann.

5.4 Flexibility-by Deviation

Ein weiterer Flexibilitätsaspekt ist durch *Flexibility-by Deviation* gegeben. Diesem Entwurfsmuster liegt nach *Schonenberg et. al* [Sch+08] die Motivation zugrunde, dass zu einem spezifischen Zeitpunkt während der Ausführung von Prozessen von dem im Prozessmodell festgelegten Kontrollfluss abgewichen werden kann. Hierdurch erlangt der Prozess die Möglichkeit, flexibel auf Ereignisse seiner Umgebung derartig zu reagieren, dass die von dem Prozess erwartete Ausgabe weiterhin realisiert werden kann.

Eine an [Sch+08] angelehnte Definition des Flexibilitätsaspekts *Flexibility-by Deviation* wird in Definition 5.4.1 gegeben.

Definition 5.4.1. *(Flexibility-by Deviation)*

Flexibility-by Deviation beschreibt die Fähigkeit einer Prozessinstanz, zur Laufzeit von den im Prozessmodell festgelegten Kontrollflusspfaden abzuweichen. Dabei werden explizit keine Änderungen an dem Kontrollfluss durchgeführt, sodass lediglich die Sequenz der Ausführung von Tasks beeinflusst wird.

Flexibility-by Deviation lässt sich nach [Sch+08] für imperative Sprachen durch verschiedene Operationen realisieren. Eine durch diese Arbeit durchgeführte Interpretation dieser Operationen ist in Abbildung 5-35 dargestellt. Auf einzelne Aspekte dieser Operationen wird nachfolgend in Abschnitt 5.4.1 eingegangen. Darauf basierend werden in Abschnitt 5.4.2 Operationen zur Unterstützung des Flexibilitätsaspekts *Flexibility-by Deviation* vorgestellt, sodass die Gestaltung durch die Sprache *ACML4BPM* ermöglicht werden kann. Abschließend wird in Abschnitt 5.4.3 eine Zusammenfassung des vorliegenden Abschnittes gegeben.

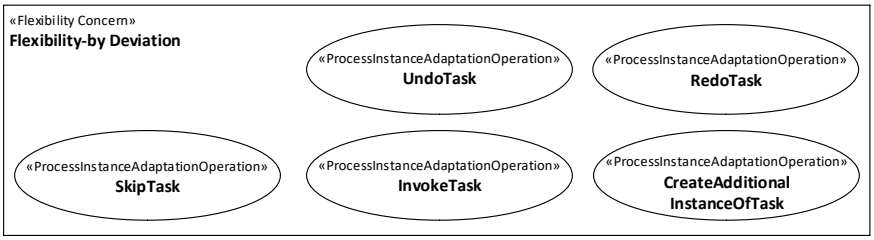


Abbildung 5-35: *Gestaltungsaspekte für flexible und anpassbare Prozesse in Hinsicht auf Flexibility-by Deviation*

5.4.1 Gestaltungsaspekte von Flexibility-by Deviation

In diesem Abschnitt werden einzelne Aspekte des Flexibilitätsaspekts *Flexibility-by Deviation* vorgestellt. Dabei konzentriert sich der Abschnitt insbesondere auf die Beschreibung von Verfahrensweisen, die durch *Schonenberg et. al* [Sch+08] beschrieben worden sind. Das Ziel ist die Durchführung einer Analyse und Beschreibung der generellen Funktionsprinzipien entsprechender Operationen. Hierdurch können nachfolgend Operationen für die Verwendung in der Sprache *ACML4BPM* beschrieben werden, sodass eine weitere Unterstützung in der Gestaltung von flexiblen und anpassbaren Prozessen besteht.

UndoTask Durch die Operation *UndoTask* kann ein aktiver oder bereits abgeschlossener Task zu einem beliebigen Zeitpunkt zurückgesetzt werden. Hierdurch ist es möglich, den Task zu diesem Zeitpunkt erneut auszuführen. Ist die Ausführung des betreffenden Tasks zum Zeitpunkt der Anwendung der Operation *UndoTask* bereits abgeschlossen, sodass weitere Tasks aktiv oder ebenfalls abgeschlossen sind, so werden auch diese zurückgesetzt. Dabei kann es sinnvoll sein, die im Rahmen eines oder einer Menge von Tasks bereits durchgeführten Aufgaben zu kompensieren – wobei dies jedoch nicht immer möglich ist. So kann z.B. durch die Verwendung von Verbrauchsmaterial möglicherweise kein Prozess beschrieben werden, der das eingesetzte Material wieder verfügbar macht. Konkrete Maßnahmen, die im Rahmen einer Kompensation notwendig sind, können durch einen weiteren Prozess beschrieben werden.

RedoTask Die Operation *RedoTask* ermöglicht das erneute Ausführen eines bereits abgeschlossenen Tasks. Im Gegensatz zur Operation *UndoTask* werden aktive und abgeschlossene Tasks nicht zurückgesetzt. Die Anwendung der Operation *RedoTask* kann z.B. sinnvoll sein, wenn durch einen Task Daten erhoben worden sind, die zu einem späteren Zeitpunkt in aktualisierter Form vorliegen. In einem solchen Fall kann durch die Anwendung der Operation *RedoTask* der betreffende Task erneut ausgeführt werden, sodass für spätere Tasks aktualisierte Daten nutzbar sind.

SkipTask Durch die Operation *SkipTask* kann die Ausführung eines derzeit aktiven Tasks übersprungen werden. Die Anwendung dieser Operation kann z.B. sinnvoll sein, wenn andere Tasks im Prozess aufgrund von kontextspezifischen Umständen Vorrang haben oder die weitere Ausführung des betreffenden Task nicht mehr notwendig ist. Im Rahmen der weiteren Ausführung des Prozesses kann es möglich sein, den übersprungenen Task nicht oder zu einem späteren Zeitpunkt erneut auszuführen.

InvokeTask Durch die Anwendung der Operation *InvokeTask* lässt sich ein noch nicht ausgeführter und nicht aktiver Task initiieren. Hierdurch wird auch die Ausführung derzeitig aktiver Tasks auf den Zeitpunkt nach Beendigung der Ausführung des betreffenden Tasks verschoben. Nach Beendigung des durch die Operation *InvokeTask* aufgerufenen Tasks werden alle verschobenen Tasks weiter ausgeführt. Wird der initiierte Task zu einem späteren Zeitpunkt erneut aktiviert, so ist sowohl die erneute Ausführung als auch ein Überspringen des Tasks möglich.

CreateAdditional-InstanceOfTask Durch die Operation *CreateAdditionalInstanceOfTask* ist es möglich, für einen noch nicht aktiven Task zu bestimmen, ob und wie viele seiner Instanzen parallel oder sequentiell ausgeführt werden sollen. Hierdurch ist

es vor Ausführung des Tasks möglich, auf kontextspezifische Gegebenheiten adäquat einzugehen. So kann durch die Anwendung der Operation flexibel auf z.B. variierendes Arbeitsaufkommen Bezug genommen werden, indem mehr oder weniger Instanzen eines Tasks durch Mitarbeiterinnen und Mitarbeiter ausgeführt werden.

Die zuvor beschriebenen Operationen lassen sich durch eine Anpassung an Zuständen des Lebenszyklus einer Aktivität beschreiben. Wie bereits in Abschnitt 5.3.3 beschrieben, lässt sich dies allerdings mittels der Sprache *BPMN2.0* in ihrer jetzigen Form nicht ausdrücken. So existiert z.B. im Gegensatz zur Sprache *UML* [OMG10] keine Möglichkeit zur Beschreibung einer sogenannten *Instance Specification*, mit der Eigenschaften einer Instanz innerhalb eines Systems (hier: *Prozess*) gestaltet werden können. In diesem Bezug lässt sich die bereits in Abschnitt 5.3.3 eingeführte Erweiterung des Metamodells der Sprache *BPMN2.0* wiederverwenden. Sie enthält bereits exemplarische Aspekte zur Unterstützung des Flexibilitätsaspekts *Flexibility-by Deviation* in Bezug zu Instanzen von Prozessen und Tasks. Aufgrund einer Vielzahl möglicher weiterer Erweiterungen wird sich hier auf eine Menge wesentlicher Elemente zur Unterstützung des Flexibilitätsaspekts *Flexibility-by Deviation*, gegeben durch Prozess- und Taskinstanzen, fokussiert. Weitere Aspekte, wie z.B. hinsichtlich des Datenflusses, werden im Rahmen der nachfolgenden Beschreibung nicht behandelt.

5.4.2 Operationen

Der Flexibilitätsaspekt *Flexibility-by Deviation* lässt sich durch spezifische Operationen zur Anpassung von Prozessinstanzen umsetzen. Damit derartige Anpassungen von Prozessinstanzen auch im Rahmen der Gestaltung von Anpassungsprozessen der Sprache *ACML4BPM* (siehe Kapitel 4) verwendet werden können, ist die Definition entsprechender Operationen notwendig. Im Detail werden Operationen für Systemkomponenten des Typs *BPExecutionComponent* (siehe Abschnitt 4.3.1) bzw. deren Effektorschnittstellen vom Typ *BPExecutionEffector* notwendig. Eine Übersicht über die resultierende Menge an Operationen zur Unterstützung der Gestaltung von *Flexibility-by Deviation* ist in Abbildung 5-36 dargestellt. Dabei lassen sich auf Basis der in Abschnitt 5.4.1 beschriebenen Aspekte insgesamt fünf Operationen benennen.

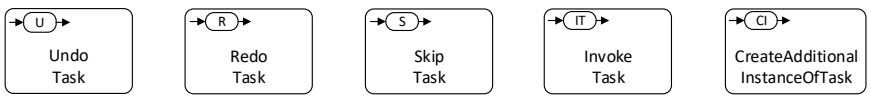
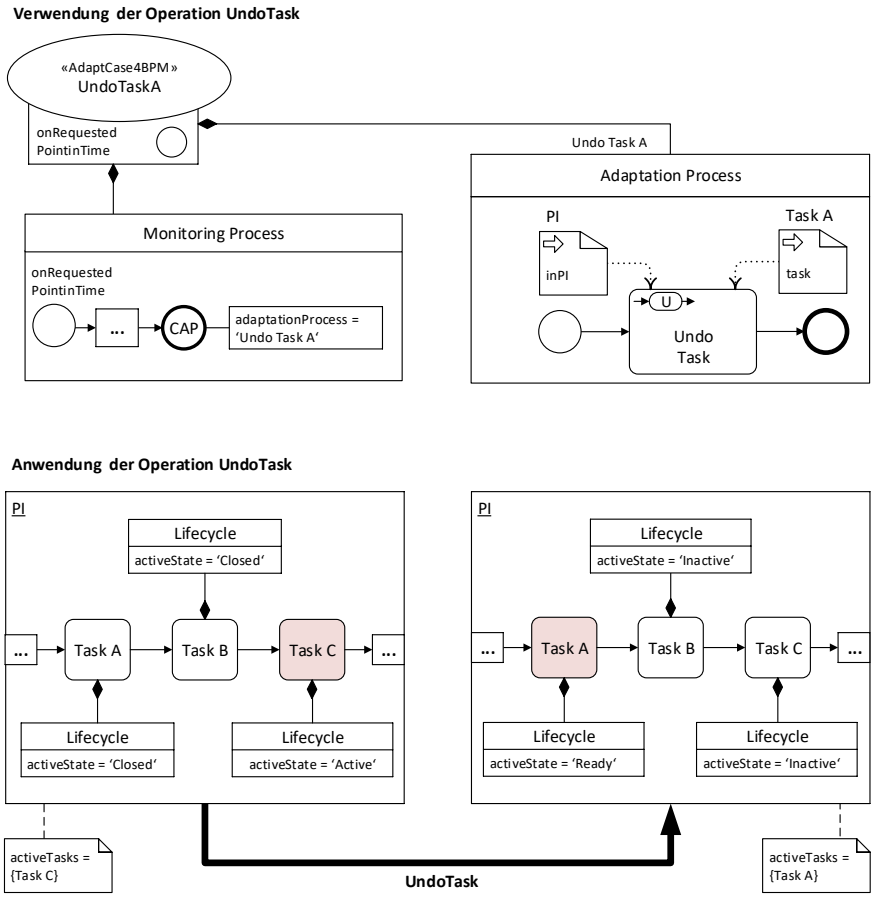


Abbildung 5-36: *Operationen zur Unterstützung von Flexibility-by Deviation*

Ein Beispiel für eine Verwendung dieser Operationen ist im oberen Teil in Abbildung 5-37 gezeigt. Dabei wird die Operation *UndoTask* zum Zurücksetzen von *Task A* eingesetzt. Eine Auslösung des Beobachtungsprozesses ist zu einem bestimmten Ereignis (*onRequestedPointinTime*) angedacht. Dabei wird der Anpassungsprozess (*Undo Task A*) aufgerufen, in dem die Operation *UndoTask* für die Anpassung der Prozessinstanz *PI* verwendet wird. Die Parametrisierung der Operation *UndoTask* adressiert die Prozessinstanz *PI* und das in ihr enthaltene Element *Task A*.

Im unteren Bereich von Abbildung 5-37 wird auf der linken Seite die Ausgangssituation und auf der rechten Seite das Resultat der Anwendung gezeigt. In dem Auszug des Modells für die Prozessinstanz *PI* werden die Taskinstanzen *Task A*, *Task B* und *Task C* dargestellt. In der beschriebenen Situation wurden *Task A* und *Task B* bereits ausgeführt, sodass derzeit die Taskinstanz *Task C* aktiv ist.

Abbildung 5-37: Beispielhafte Verwendung der Operation UndoTask



Das Ergebnis einer Anwendung der Operation *UndoTask* auf die beschriebene Prozessinstanz *PI* ist im rechten Bereich der Abbildung 5-37 dargestellt. So wurden entlang des abgeschlossenen Kontrollflusspfades ausgehend von dem ursprünglich aktiven *Task C* bis zum betreffen *Task A* alle Attribute *activeState* der Lebenszyklen auf den Wert *Inactive* gesetzt. Das Attribut *activeState* im Lebenszykluselement des betreffenden Tasks hat den Wert *Ready*. Das Attribut *activeTasks* der Prozessinstanz *PI* ist auf *Task A* gesetzt. Durch die beschriebene Anwendung der Operation *UndoTask* sind alle aktiven und bereits ausgeführten Taskinstanzen zurückgesetzt worden. Das Resultat ist ein zurückgesetzter *Task A*. Ferner wurden weitere auf dem Kontrollflusspfad bereits terminierte Tasks ebenso zurückgesetzt.

Für eine Ausführung jeder einzelnen Operation durch ihre Signatur, der konkreten Syntax und der Beispiele ihrer Anwendung wird an dieser Stelle auf die Inhalte in den nachfolgenden Abschnitten 5.4.2.1 und 5.4.2.5 verwiesen.

5.4.2.1 UndoTask

Die Anwendung einer Operation vom Typ *UndoTask* setzt den Zustand des Lebenszyklus eines betreffenden Tasks auf den Wert *Ready*. Hierdurch ist es möglich, den betreffenden Task erneut auszuführen. Dabei werden rückwirkend die Zustände der Lebenszyklen von den derzeit ausgeführten und abgeschlossenen Tasks auf den Wert *Inactive* gesetzt. Dies betrifft alle Tasks, die auf dem Kontrollflusspfad nach dem betreffenden Task vorkommen. Ferner wird zwischen zwei Mechanismen unterschieden. Zum einen kann es sinnvoll sein, zusätzlich einen Prozess zur Rückabwicklung (engl. *Compensation*) aufzurufen. Zum anderen kann es jedoch auch sein, dass dies nicht notwendig oder gar möglich ist. Die Signatur und konkrete Syntax der Operation *UndoTask* sind in Abbildung 5-38 angegeben.

Parameter

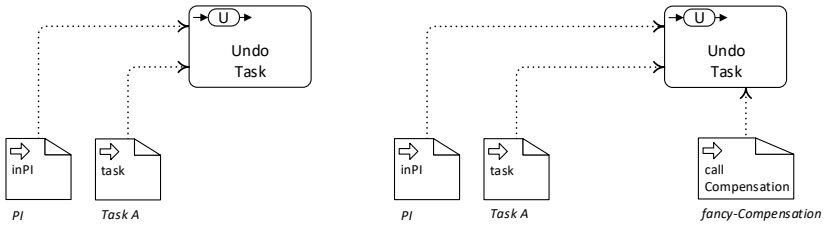
Die Operation erwartet als Eingabe eine Prozessinstanz (*inPI*), auf die die Anpassung ausgeführt werden soll. Dabei wird durch die Anwendung der Operation keine explizite Ausgabe erzeugt – hier durch die Kennzeichnung als *VOID* dargestellt. Ein weiterer Parameter der Operation ist durch die betreffende Taskinstanz (*task*) gegeben.

Optionale Parameter

Sollen vor der Zurücksetzung eines Zustands des Lebenszyklus der Taskinstanz *task* weitere Maßnahmen ausgeführt werden, wie z.B. Kompensationen, ist die Angabe eines weiteren Parameters notwendig. So kann durch die Angabe des Parameters *callCompensation* vom Typ *Process* ein Prozess angegeben werden, der spezifisches Verhalten für den genannten Zweck enthalten kann.

Abbildung 5-38: Signatur und konkrete Syntax der Operation UndoTask

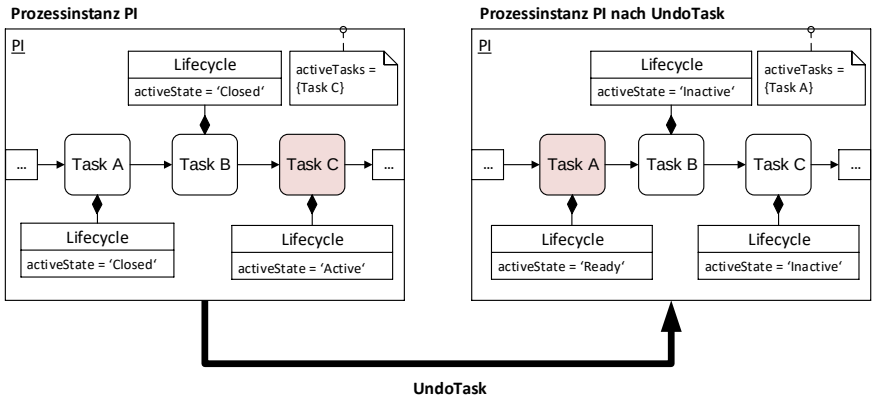
	Parametername	Parametertyp
IN :	*inPI* *task*	*ProcessInstanceRepresentation* *TaskInstanceRepresentation*
IN-Optional :	*callCompensation*	*Process*
OUT :	–	*VOID*



Beispiel einer Anwendung der Operation UndoTask

Eine Anwendung der im linken Bereich von Abbildung 5-38 spezifizierten Operation ist in Abbildung 5-39 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang ein Auszug einer Prozessinstanz *PI* dargestellt. Sie besteht aus den drei Taskinstanzen *Task A*, *Task B* und *Task C*, die durch Assoziationen vom Typ *SequenceFlow* zu einer Sequenz miteinander verbunden sind. Dabei wurden *Task A* und *Task B* bereits ausgeführt, sodass derzeit die Taskinstanz *Task C* aktiv ist.

Abbildung 5-39: Beispielhafte Anwendung der Operation UndoTask



Das Ergebnis einer Anwendung der Operation *UndoTask* auf die beschriebene Prozessinstanz *PI* ohne eine Parametrisierung zur Kompensation ist im rechten Bereich der Abbildung 5-39 dargestellt. So wurden entlang des abgeschlossenen Kontrollflusspfades ausgehend von dem ursprünglich aktiven *Task C* bis zum betreffen *Task A* alle Attribute *activeState* der Lebenszyklen auf den Wert *Inactive* gesetzt. *Task C* befindet sich dabei im Zustand *Ready* und ist der aktuell aktive Task innerhalb der Prozessinstanz

PI. Durch die beschriebene Anwendung der Operation *UndoTask* sind alle aktiven und bereits ausgeführten Taskinstanzen auf dem Kontrollflusspfad zwischen *Task C* und *Task A* zurückgesetzt worden. Hierdurch ist die weitere Ausführung ab dem betreffenden Task, hier gegeben durch *Task A*, möglich.

Das generelle Funktionsprinzip einer Anwendung der Operation *UndoTask* mit der Parametrisierung zur Kompensation wird in Abbildung 5-40 dargestellt. Eine Anwendung der Operation besteht aus insgesamt drei Teilschritten. Im Rahmen des ersten Teilschritts *ResetTask* werden Tasks – wie zuvor beschrieben – zunächst zurückgesetzt. Damit die Ausführung des im Beispiel vorhandenen Tasks *Task A* nach Anwendung der Maßnahme zur Kompensation fortgesetzt werden kann, wird der Zustand seines Lebenszyklus jedoch auf den Wert *Inactive* gesetzt. Damit wird sichergestellt, dass eine Ausführung vorerst unterbrochen ist. Der zweite Teilschritt *CallCompensation* ruft einen Prozess auf, in dem die Maßnahme zur Kompensation beschrieben ist. Nach Beendigung dieses Prozesses wird der dritte Teilschritt *ReturnControl* durchgeführt, indem der Zustand des Lebenszyklus des betreffenden Tasks *Task A* auf den Wert *Ready* gesetzt wird. Hierdurch ist eine weitere Ausführung der Prozessinstanz *PI* möglich.

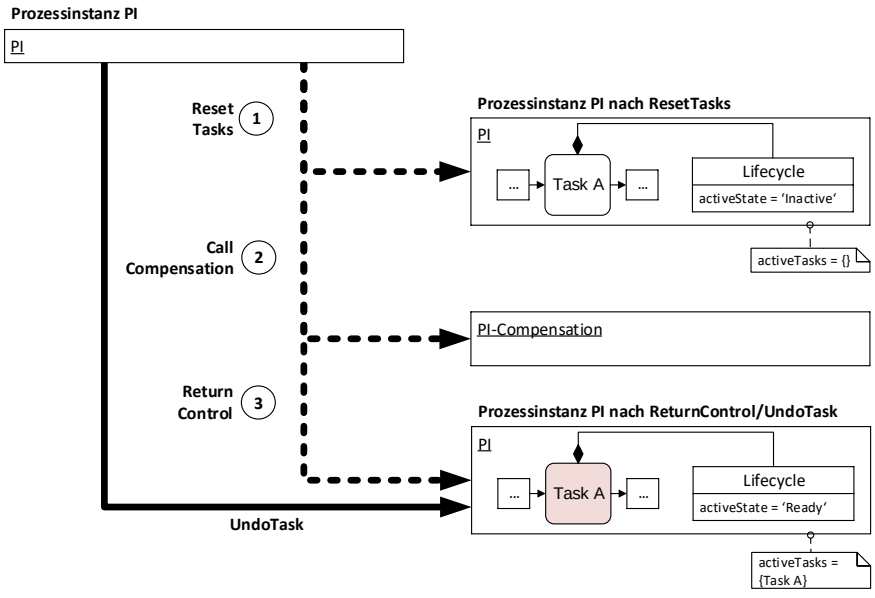


Abbildung 5-40: *Funktionsprinzip einer Anwendung der Operation UndoTask mit Kompensation*

Neben der zuvor beschriebenen Möglichkeit zur Realisierung der Operation *UndoTask* mit der Parametrisierung zur Kompensation wäre auch eine Abweichung von der durch [Sch+08] beschriebenen Funktionsweise

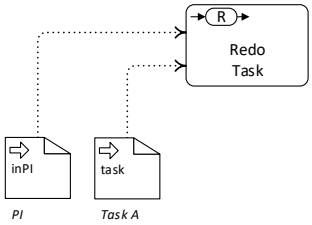
möglich. So könnte auf Basis der in Abschnitt 4.3.3 beschriebenen Operationen zur Anpassung von Prozessen Maßnahmen zur Kompensation in Form einzelner Tasks in den bestehenden Kontrollfluss eingefügt werden. Das Einfügen von Tasks zu diesem Zweck könnte im Kontrollflusspfad vor dem betreffenden Task durchgeführt werden. Hierdurch wäre es möglich, zunächst Tasks zur Kompensation auszuführen, bevor der Kontrollfluss automatisch den betreffenden *Task A* erneut aktiviert. Durch dieses Vorgehen würde allerdings der Kontrollfluss geändert werden. Dies ist aus der Perspektive der Reihenfolge der zu tätigenden Aufgaben jedoch dasselbe Ergebnis mit möglicherweise ungewollten Anpassungen an der Prozessinstanz.

5.4.2.2 RedoTask

Die Anwendung einer Operation vom Typ *RedoTask* ermöglicht die Ausführung eines bereits abgeschlossenen Tasks. Die erneute Ausführung bezieht sich in der hier beschriebenen Realisierung auf eine Kopie des betreffenden Tasks, welche nach Anwendung der Operation zu einem beliebigen zukünftigen Zeitpunkt durchgeführt werden kann. Im Gegensatz zur Operation *UndoTask* werden weitere bereits abgeschlossene oder derzeit ausgeführte Tasks nicht zurückgesetzt und demnach nicht erneut ausgeführt. Ferner werden ebenso keine Maßnahmen zur Rückabwicklung spezifiziert. Die Signatur und konkrete Syntax der Operation *RedoTask* sind in Abbildung 5-41 angegeben.

Abbildung 5-41: *Signatur und konkrete Syntax der Operation RedoTask*

	Parametername	**Parametertyp**
IN :	*inPI*	*ProcessInstanceRepresentation*
	task	*TaskInstanceRepresentation*
OUT :	–	*VOID*



Parameter

Die Operation erwartet als Eingabe eine Prozessinstanz (*inPI*), auf die die Anpassung ausgeführt werden soll. Dabei wird durch die Anwendung der Operation keine explizite Ausgabe erzeugt – hier durch die Kennzeichnung als *VOID* dargestellt. Ein weiterer Parameter der Operation ist durch die betreffende Taskinstanz (*task*) gegeben.

Beispiel einer Anwendung der Operation RedoTask

Eine Anwendung der in Abbildung 5-41 spezifizierten Operation ist in Abbildung 5-42 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang ein Auszug einer Prozessinstanz *PI* dargestellt. Sie besteht aus den beiden Taskinstanzen *Task A* und *Task B*, die durch Assoziationen vom Typ *SequenceFlow* zu einer Sequenz miteinander verbunden sind. Dabei wurde *Task A* bereits ausgeführt, sodass derzeit die Taskinstanz *Task B* aktiv ist.

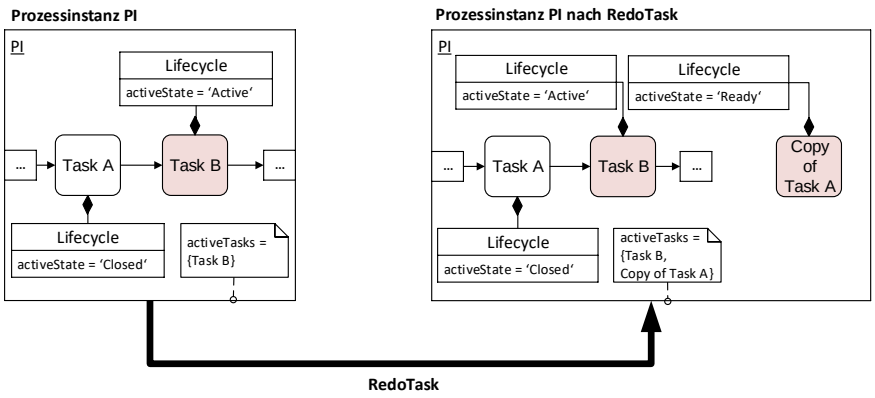


Abbildung 5-42: Beispielhafte Anwendung der Operation RedoTask

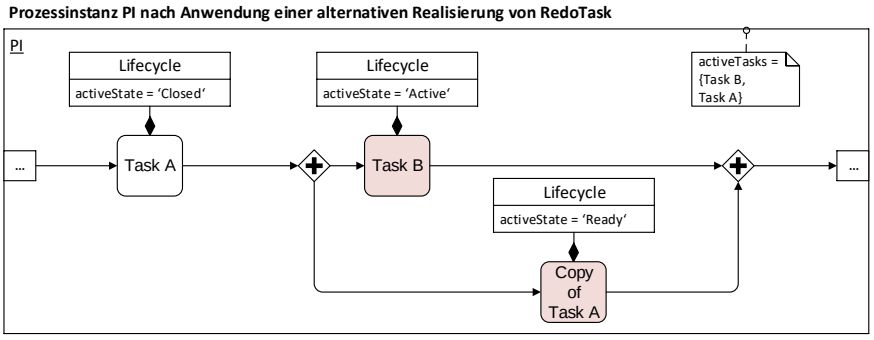
Das Ergebnis einer Anwendung der Operation *RedoTask* auf die beschriebene Prozessinstanz *PI* ist im rechten Bereich der Abbildung 5-42 dargestellt. In dem BPD ist eine neue Taskinstanz mit der Bezeichnung *Copy of Task A* hinzugefügt worden, die nicht mit dem existierenden Kontrollfluss verbunden worden ist. Der Lebenszyklus der Taskinstanz befindet sich im Zustand *Ready*, sodass die Taskinstanz zu einem beliebigen zukünftigen Zeitpunkt ausgeführt werden kann.

Durch die beschriebene Anwendung der Operation *RedoTask* wurde der bestehende Kontrollfluss nicht explizit geändert. Dies kann Nachteile haben, wenn Prozessinstanzen im Rahmen der Verbesserung von Prozessen analysiert werden. Hier können frei integrierte Taskinstanzen womöglich nachträglich nicht mehr richtig zugeordnet werden. Eine weitere Möglichkeit zur Realisierung der Operation *RedoTask* ohne diesen Effekt ist dadurch gegeben, dass die Kopie der Taskinstanz *Task A* in den bestehenden Kontrollfluss integriert wird. Diese Variante weicht allerdings von der durch *Schonenberg et al.* [Sch+08] gegebenen Definition ab.

Der Vollständigkeit halber ist in Abbildung 5-43 das Ergebnis dieser Realisierung anhand eines BPD abgebildet. Dabei wurde die Kopie der Taskinstanz *Task A* in den bestehenden Kontrollfluss so integriert, dass die Ausführung von *Copy of Task A* parallel zur Ausführung der Taskinstanz *Task*

B durchgeführt werden kann. Durch fest in den Kontrollfluss integrierte Taskinstanzen kann das zuvor genannte Problem unter Vernachlässigung der von *Schonenberg et al.* [Sch+08] gegeben Definition umgangen werden.

Abbildung 5-43: Ergebnis für eine alternative Realisierung der Operation RedoTask

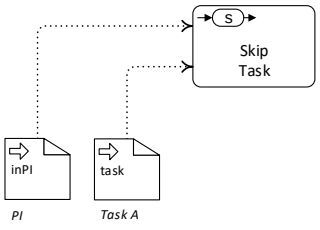


5.4.2.3 SkipTask

Die Anwendung einer Operation vom Typ *SkipTask* ermöglicht das Überspringen einer derzeit ausgeführten Taskinstanz. Dabei wird der Zustand des Lebenszyklus des betreffenden Tasks auf den Wert *Closed* gesetzt. Die Operation kann solange auf einen Task angewendet werden, wie sein Lebenszyklus sich nicht in den Zuständen *Closed* oder *Inactive* befindet. Die Signatur und konkrete Syntax der Operation *SkipTask* sind in Abbildung 5-44 angegeben.

Abbildung 5-44: Signatur und konkrete Syntax der Operation SkipTask

	Parametername	**Parametertyp**
IN :	*inPI*	*ProcessInstanceRepresentation*
	task	*TaskInstanceRepresentation*
OUT :	–	*VOID*



Parameter Die Operation erwartet als Eingabe eine Prozessinstanz (*inPI*), auf die die Anpassung ausgeführt werden soll. Dabei wird durch die Anwendung der Operation keine explizite Ausgabe erzeugt – hier durch die Kennzeichnung als *(VOID)* dargestellt. Ein weiterer Parameter der Operation ist durch den betreffenden Task (*task*) gegeben.

Beispiel einer Anwendung der Operation SkipTask

Eine Anwendung der in Abbildung 5-44 spezifizierten Operation ist in Abbildung 5-45 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. In dem BPD wird eine Prozessinstanz *PI* dargestellt. Sie besteht aus den beiden Taskinstanzen *Task A* und *Task B*, die durch Assoziationen vom Typ *SequenceFlow* zu einer Sequenz miteinander verbunden sind. Dabei ist *Task A* der derzeit aktive Task.

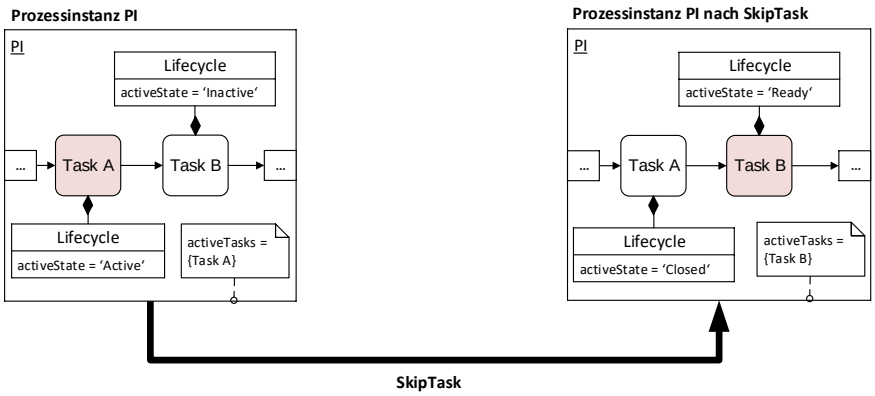


Abbildung 5-45: *Beispielhafte Anwendung der Operation SkipTask*

Das Ergebnis einer Anwendung der Operation *SkipTask* auf die beschriebene Prozessinstanz *PI* ist im rechten Bereich der Abbildung 5-45 dargestellt. So wurde zunächst die Taskinstanz *Task A* beendet und die Taskinstanz *Task B* aktiviert. Das Attribut *activeTasks* des Containerelements *C* ist auf den Wert *{Task B}* gesetzt. Hierdurch wurde die weitere Ausführung von *Task A* übersprungen und die Sequenz der Ausführung fährt mit *Task B* fort.

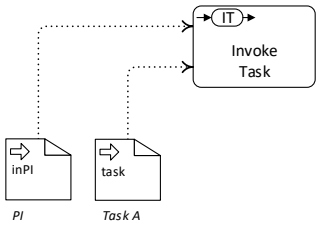
5.4.2.4 InvokeTask

Die Anwendung einer Operation vom Typ *InvokeTask* setzt den Zustand des Lebenszyklus eines noch nicht ausgeführten oder noch nicht aktiven Tasks auf den Wert *Ready*, sodass er im Anschluss ausgeführt wird. Hierdurch ist es möglich, die Ausführung des betreffenden Task vorzuziehen. Dabei wird die Ausführung derzeit aktiver Tasks unterbrochen und die Zustände ihrer Lebenszyklen auf den Wert *Inactive* gesetzt. Nach Beendigung des vorgezogenen Tasks werden die Zustände der Lebenszyklen von unterbrochenen Tasks auf den Wert *Ready* gesetzt. Ausgehende Assoziationen aktivieren keine nachfolgenden Tasks. Dadurch können die unterbrochenen Tasks erneut ausgeführt werden. Wird im weiteren Verlauf der Ausführung des Prozesses der zuvor vorgezogene Task erneut aktiviert, so ist eine erneute Ausführung möglich. Alternativ kann aber auch

durch die zuvor beschriebene Operation *SkipTask* (siehe Abschnitt 5.4.2.3) die Ausführung übersprungen werden. Die Signatur und konkrete Syntax der Operation *InvokeTask* sind in Abbildung 5-46 angegeben.

Abbildung 5-46: Signatur und konkrete Syntax der Operation InvokeTask

	Parametername	**Parametertyp**
IN :	*inPI*	*ProcessInstanceRepresentation*
	task	*TaskInstanceRepresentation*
OUT :	–	*VOID*



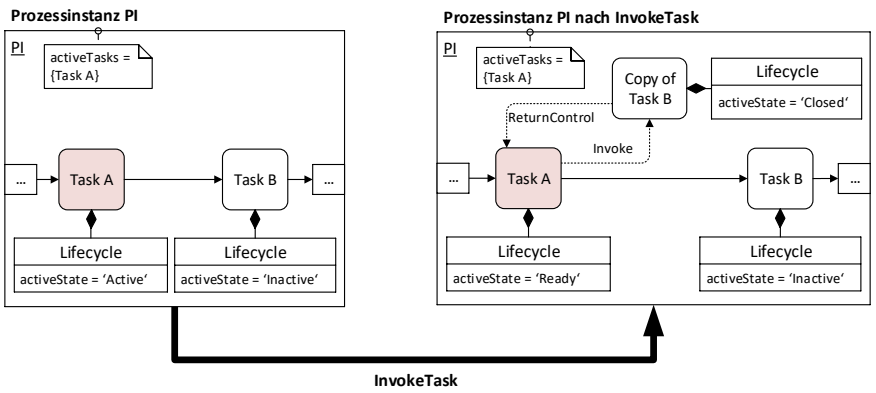
Parameter

Die Operation erwartet als Eingabe eine Prozessinstanz (*inPI*), auf die die Anpassung ausgeführt werden soll. Dabei wird durch die Anwendung der Operation keine explizite Ausgabe erzeugt – hier durch die Kennzeichnung als *(VOID)* dargestellt. Ein weiterer Parameter der Operation ist durch den betroffenen Task (*task*) gegeben.

Beispiel einer Anwendung der Operation InvokeTask

Eine Anwendung der in Abbildung 5-46 spezifizierten Operation ist in Abbildung 5-47 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. In dem BPD wird eine Prozessinstanz *PI* dargestellt. Sie besteht aus den beiden Taskinstanzen *Task A* und *Task B*, die durch Assoziationen vom Typ *SequenceFlow* zu einer Sequenz miteinander verbunden sind. Dabei ist *Task A* der derzeit aktive Task.

Abbildung 5-47: Beispielhafte Anwendung der Operation InvokeTask



Das Ergebnis einer Anwendung der Operation *InvokeTask* auf die beschriebene Prozessinstanz PI ist im rechten Bereich der Abbildung 5-47 dargestellt. Dabei wurde eine Kopie eines vorgezogenen Tasks mit der Bezeichnung *Copy of Task B* eingefügt. Bei den Assoziationen mit der Beschriftung *Invoke* und *ReturnControl* handelt es sich um implizite Kontrollflüsse, die hier lediglich zum besseren Verständnis dargestellt sind und sonst keine Darstellungsform haben. Die Ausführung der vorgezogenen Taskinstanz wird hier als bereits abgeschlossen dargestellt. Ferner wurde die Prozessinstanz aktiviert, deren Ausführung durch das Vorziehen unterbrochen worden ist.

Zum besseren Verständnis ist das Funktionsprinzip der Operation *SkipTask* in Abbildung 5-48 dargestellt. So teilt sich eine Anwendung der Operation *InvokeTask* in die beiden Teilschritte *Invoke* und *ReturnControl* auf. Durch den Teilschritt *Invoke* wird der Prozessinstanz PI eine Kopie des vorgezogenen Tasks hinzugefügt. Ferner wechseln die Lebenszyklen aller aktiven Tasks in den Zustand *Inactive*. Der Lebenszyklus des vorgezogenen Tasks *Copy of Task B* befindet sich im Zustand *Ready*. Demnach ist das Attribut *activeTasks* der Prozessinstanz auf den Wert *Copy of Task B* gesetzt, wodurch er als nächstes ausgeführt werden kann. Nach Beendigung des Tasks mit der Bezeichnung *Copy of Task B* wird der zweite Teilschritt *ReturnControl* durchgeführt. In diesem Rahmen werden alle Lebenszyklen von zuvor aktiv gewesenen Tasks in den Zustand *Ready* gesetzt. Anschließend kann die vorherige Ausführung fortgesetzt werden.

Die zuvor beschriebene Realisierung der Operation *InvokeTask* arbeitet mit Hilfe von impliziten Kontrollflüssen zwischen derzeit aktiven Tasks und dem vorgezogenen Task. Ein impliziter Kontrollfluss beschreibt dabei die Sequenz einer Ausführung von Tasks ohne das explizite Hinzufügen von z.B. Assoziationen vom Typ *SequenceFlow*. Die Kontrolle über den impliziten Kontrollfluss liegt hier bei der Operation *InvokeTask*, die – wie in Abbildung 5-48 dargestellt – bis zur Aktivierung zuvor deaktivierter Tasks aktiv bleibt. Auf diese Weise kann der durch *Schonenberg et al.* gegebenen Beschreibung des Flexibilitätsaspekts *Flexibility-by Deviation* Rechnung getragen werden, indem der bestehende Kontrollfluss nicht explizit angepasst wird.

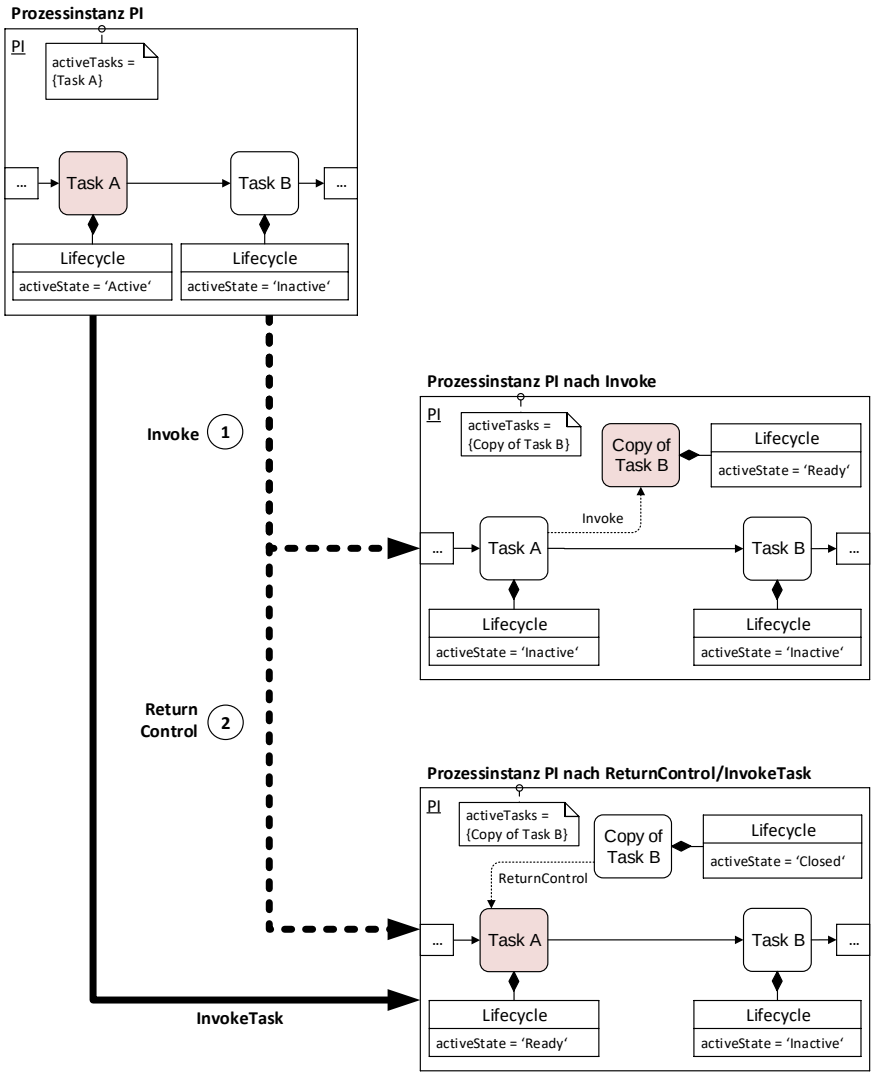


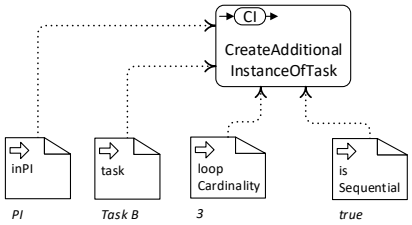
Abbildung 5-48: Funktionsprinzip einer Anwendung der Operation InvokeTask

5.4.2.5 CreateAdditionalInstanceOfTask

Die Anwendung einer Operation vom Typ *CreateAdditionalInstanceOfTask* ermöglicht die Anpassung der Attribute *isSequential* und *loopCardinality* einer Taskinstanz vor ihrer Ausführung. Durch das Attribut *loopCardinality* kann die Anzahl von Taskinstanzen bestimmt werden, die ausgeführt werden sollen. Ferner kann durch das Attribut *isSequential* bestimmt werden, ob die Taskinstanzen sequentiell oder aber parallel ausgeführt werden sollen. Die Signatur und konkrete Syntax der Operation *CreateAdditionalInstanceOfTask* sind in Abbildung 5-49 angegeben.

	Parametername	Parametertyp
IN :	*inPI* *task*	*ProcessInstanceRepresentation* *TaskInstanceRepresentation*
IN-Optional :	*loopCardinality* *isSequential*	*Integer* *Boolean*
OUT :	–	*VOID*

Abbildung 5-49: Signatur und konkrete Syntax der Operation CreateAdditionalInstance-OfTask



Parameter

Die Operation erwartet als Eingabe eine Prozessinstanz (*inPI*), auf die die Anpassung ausgeführt werden soll. Dabei wird durch die Anwendung der Operation keine explizite Ausgabe erzeugt – hier durch die Kennzeichnung als *(VOID)* dargestellt. Ein weiterer Parameter der Operation ist durch den betroffenen Task (*task*) gegeben.

Optionale Parameter

Wie hoch die Anzahl der auszuführenden Taskinstanzen sein soll, kann durch den Parameter *loopCardinality* vom Typ *Integer* angegeben werden. Dabei können lediglich positive Werte einschließlich 0 angegeben werden. Durch den letzten hier aufgeführten Parameter *isSequential* vom Typ *Boolean* lässt sich eine sequentielle oder parallele Ausführung der Taskinstanzen angeben. Der Wert *true* steht dabei für sequentielle und der Wert *false* für eine parallele Ausführung.

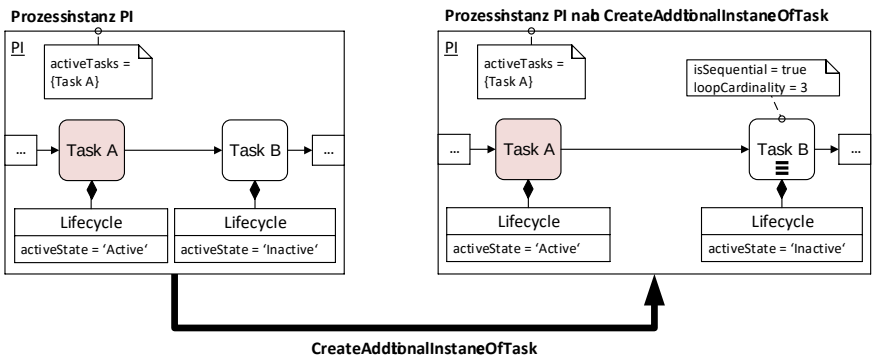


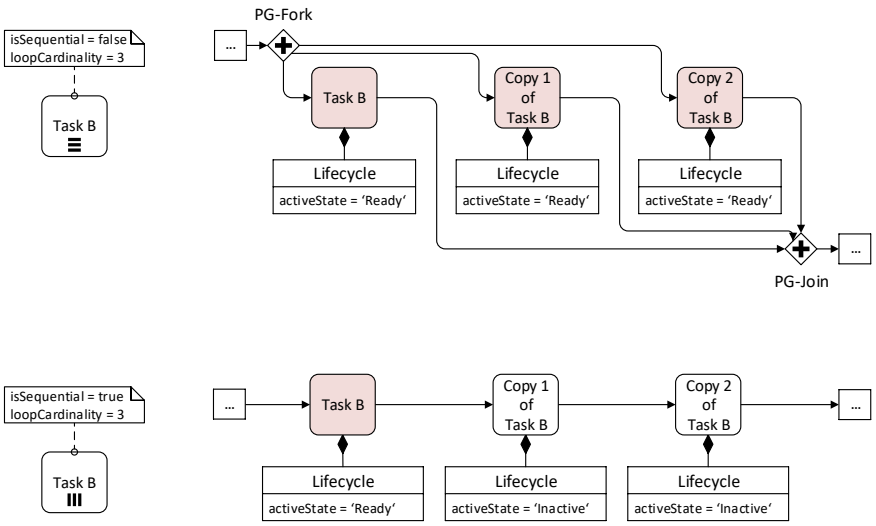
Abbildung 5-50: Beispielhafte Anwendung der Operation CreateAdditionalInstanceOfTask

Beispiel einer Anwendung der Operation CreateAdditionalInstanceOfTask

Eine Anwendung der in Abbildung 5-49 spezifizierten Operation ist in Abbildung 5-50 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. In dem BPD wird eine Prozessinstanz *PI* dargestellt. Sie besteht aus den beiden Taskinstanzen *Task A* und *Task B*, die durch Assoziationen vom Typ *SequenceFlow* zu einer Sequenz verbunden sind. Dabei ist *Task A* der derzeit aktive Task.

Das Ergebnis einer Anwendung der Operation *CreateAdditionalInstanceOfTask* auf die beschriebene Prozessinstanz *PI* ist im rechten Bereich der Abbildung 5-50 dargestellt. So wurden für *Task B* die beiden Attribute *isSequential* mit dem Wert *true* und *loopCardinality* mit dem Wert *5* gesetzt. Der betreffende Task erhält zudem das aus der Sprache *BPMN2.0* stammende Symbol für die sequentielle Ausführung mehrerer Instanzen des Tasks (siehe Abschnitt 2.3.4). Die zuvor beschriebene Realisierung der Operation *CreateAdditionalInstanceOfTask* erzeugt bei Aktivierung des angepassten Tasks eine der in Abbildung 5-51 dargestellten Ausführungssequenzen von *Task B*.

Abbildung 5-51: Darstellung von Ausführungssequenzen von Task B



Im oberen Teil der Abbildung wird eine Ausführungssequenz dargestellt, die eine parallele Ausführung des *Task B* und seiner beiden Kopien beschreibt. Dabei sind zum Zeitpunkt der Betrachtung ausgehend von dem Gateway mit der Bezeichnung *PG-Fork* alle Tasks aktiviert worden, sodass sich ihre Lebenszyklen im Zustand *Ready* befinden. Es ist möglich, dass sich die Lebenszyklen zu einem Zeitpunkt im weiteren Verlauf der Ausführung in unterschiedlichen Zuständen befinden. Eine Synchronisation wird durch das am Ende der Ausführungssequenz befindliche parallele

Gateway mit der Bezeichnung *PG-Join* durchgeführt. Im unteren Teil der Abbildung wird eine Ausführungssequenz dargestellt, die eine sequentielle Ausführung des *Task B* und seiner beiden weiteren Kopien beschreibt. Dabei ist zum Zeitpunkt der Betrachtung der erste Task mit der Bezeichnung *Task B* aktiv. Seine beiden weiteren Kopien sind derzeit inaktiv und werden im Verlauf der dargestellten Sequenz nacheinander aktiviert.

5.4.3 Zusammenfassung

In den vorherigen Abschnitten wurden auf Basis der durch [Sch+08] gegebenen Beschreibung von *Flexibility-by Deviation* verschiedene Operationen zur Realisierung des Flexibilitätsaspekts gegeben. Hier wurde die bereits für den Flexibilitätsaspekt *Flexibility-by Change* eingeführte Erweiterung (siehe Abschnitt 5.3.3) wiederverwendet. Basierend auf dieser Erweiterung wurden im Anschluss fünf Operationen beschrieben, die die Anpassung von Prozess- und Taskinstanzen hinsichtlich des Flexibilitätsaspekts *Flexibility-by Deviation* ermöglichen. Die Operationen lassen sich in der Gestaltung von Anpassungen im Rahmen von Beobachtungs- bzw. Anpassungsprozessen einsetzen, sodass sie einen wichtigen Beitrag für die Umsetzung von flexiblen und anpassbaren Prozessen darstellen.

5.5 Flexibility-by Underspecification

Im Rahmen der Flexibilisierung von Prozessen kann es möglich sein, dass in einer frühen Phase einer Iteration des *BPM-Lebenszyklus* Anforderungen hinsichtlich zu realisierender Funktionen in einem Prozess entweder noch nicht bekannt oder aber hochgradig variabel sein können. Eine benötigte Funktion kann dann entweder nicht beschrieben werden oder ein Prozess kann viele alternative Kontrollflusspfade enthalten. Hierdurch können die weitere Entwicklung sowie die Wartung eines Prozesses komplex sein.

In einem solchen Fall bietet sich für die Gestaltung von flexiblen und anpassbaren Prozessen die Verwendung des Flexibilitätsaspekts *Flexibility-by Underspecification* an. Bei dem Flexibilitätsaspekt *Flexibility-by Underspecification* wird die Auswahl einer zu erbringenden Funktion auf einen späteren Zeitpunkt im *BPM-Lebenszyklus* verschoben. So könnte eine Funktion z.B. erst im Rahmen der Phase *Konfiguration* oder der Phase *Ausführung* auf Basis der dann verfügbaren Anforderungen gewählt werden. Hierdurch ist der Zeitpunkt für die Auswahl einer Funktion entsprechend verschoben worden. Eine an [Sch+08] orientierte Definition des Flexibilitätsaspekts *Flexibility-by Underspecification* wird in Definition 5.5.1 gegeben.

Definition 5.5.1. *(Flexibility-by Underspecification)*

Flexibility-by Underspecification beschreibt die Fähigkeit zur Ausführung eines unvollständigen Prozesses zur Ausführungszeit. Dabei können in Anlehnung an die Auswertungen von Bedingungen an spezifischen Punkten des Kontrollflusses – sogenannten Platzhaltern – noch nicht enthaltene Funktionen beschrieben bzw. gewählt werden, um dann im weiteren Verlauf der Ausführung ausgeführt zu werden.

Flexibility-by Underspecification lässt sich nach [Sch+08] in die zwei weiteren Typen von Flexibilitätsaspekten *Late Selection* und *Late Modeling* unterteilen. Eine Übersicht über die beiden Typen des Flexibilitätsaspekts *Flexibility-by Underspecification* und ihre spezifischen Operationen sowie Eigenschaften ist in Abbildung 5-52 gegeben. Nachfolgend werden in Abschnitt 5.5.1 zunächst die beiden Typen *Late Selection* und *Late Modeling* detaillierter vorgestellt. Dabei wird insbesondere Bezug zu den in Abbildung 5-52 gezeigten Operationen und Eigenschaften genommen. Basierend auf dieser Beschreibung wird in Abschnitt 5.5.2 eine weitere Erweiterung der Sprache *BPMN2.0* beschrieben, die spezifische Elemente zur Unterstützung des Flexibilitätsaspekts *Flexibility-by Underspecification* enthält. In Abschnitt 5.5.3 werden die Operationen zur Unterstützung der Gestaltung des Flexibilitätsaspekts unter Verwendung der Sprache *ACML4BPM* vorgestellt. Abschließend wird in Abschnitt 5.5.4 eine Zusammenfassung der vorgestellten Inhalte gegeben.

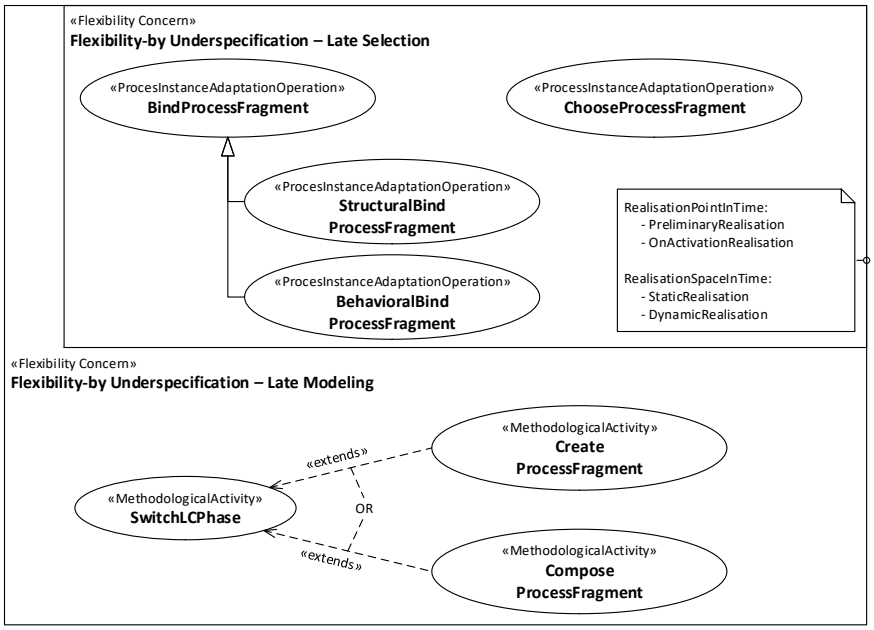


Abbildung 5-52: Gestaltungsaspekte für flexible und anpassbare Prozesse in Hinsicht auf Flexibility-by Underspecification

5.5.1 Gestaltungsaspekte von Flexibility-by Underspecification

In diesem Abschnitt werden verschiedene Aspekte hinsichtlich der Gestaltung des Flexibilitätsaspekts *Flexibility-by Underspecification* beschrieben. Im Detail wird hierbei auf den Typ *Late Selection* und auf den Typ *Late Modeling* eingegangen. Abschließend wird auf die in Abbildung 5-52 dargestellten Eigenschaften der beiden zuvor genannten Typen Bezug genommen.

Elemente zur Gestaltung

Für die beiden Typen *Late Selection* und *Late Modeling* werden die beiden weiteren Elemente Platzhalter und Prozessfragment im Rahmen des Kontrollflusses eines Basisprozesses verwendet. Bei einem Platzhalter handelt es sich um ein Element, dessen Inhalt vor Ausführung des Prozesses unbekannt ist und erst im Lauf der Ausführung durch den Inhalt eines zuvor gestalteten Prozessfragments vervollständigt wird. Diese Vervollständigung ist dabei an die Auswertung einer Bedingung geknüpft, durch die aus einer Menge von verschiedenen Prozessfragmenten gewählt werden kann. Ein Prozessfragment kann als ein spezieller Typ eines Prozesses verstanden werden, der sowohl ein definiertes Start- als auch ein Endsymbol besitzt und eine oder mehrere bestimmte Funktionen unterstützt. Dabei unterscheidet sich ein Prozessfragment von einem Prozess vornehmlich durch seine Verwendungsweise. So wird es an einer spezifischen Stelle in einem Prozess eingesetzt und nicht ohne einen übergeordneten Prozess verwendet. Ein Prozessfragment ist somit in Hinsicht auf den Flexibilitätsaspekt *Flexibility-by Underspecification* eine mögliche Funktion, die an der Stelle eines Platzhalters eingesetzt werden kann.

Typ Late Selection

Bei dem Typ *Late Selection* wird die Auswahl einer benötigten Funktion auf einen späteren Zeitpunkt im *BPM-Lebenszyklus* verschoben. So wird an einem spezifischen Zeitpunkt im Verlauf eines Prozesses eine Auswertung von Bedingungen vorgenommen, deren Ergebnis die Auswahl einer Funktion bzw. deren Realisierung in Form eines Prozessfragments ist. Für die Realisierung von *Late Selection* ist neben den zuvor eingeführten Elementen Platzhalter, Prozessfragment und Bedingung zur Auswahl eines Prozessfragments ein Basisprozess notwendig. Dabei enthält der Kontrollfluss eines Basisprozesses mindestens einen Platzhalter. Ferner werden im Rahmen einer Bedingung und Auswahl eines Prozessfragments die beiden Typen von Operationen *ChooseProcessFragment* und *BindProcessFragment* benötigt, welche nachfolgend beschrieben werden.

ChooseProcessFragment

Durch eine Operation vom Typ *ChooseProcessFragment* kann zu einem spezifischen Zeitpunkt eine Auswahl von benötigten Funktionen getrof-

fen werden. Dabei werden Bedingungen ausgewertet, deren Ergebnis die Auswahl eines oder mehrerer Funktionen in Form von Prozessfragmenten ist. Die durch Operationen des Typs *ChooseProcessFragment* getroffene Auswahl kann innerhalb des Flexibilitätsaspekts *Late Selection* durch Beobachtungsprozesse gestaltet werden.

BindProcessFragment

Die Vervollständigung eines Platzhalters durch ein Prozessfragment wird auch Bindung genannt. Dabei wird im Anschluss an die Auswahl einer benötigten Funktion (*ChooseProcessFragment*) eine Bindung durch Operationen des Typs *BindProcessFragment* durchgeführt. Derartige Operationen lassen sich im Rahmen der Gestaltung von Anpassungsprozessen zur Integration von benötigten Funktionen einsetzen. *Murguzur et. al* [Mur+13] unterscheiden darüber hinaus verschiedene Funktionsprinzipien von Bindungen. So werden verhaltensbasierte und strukturbasierte Verfahren unterschieden. Diese werden in Abbildung 5-52 durch die beiden Typen von Operationen *BehavioralBindProcessFragment* bzw. *StructuralBindProcessFragment* dargestellt und im Folgenden beschrieben.

BehavioralBindProcessFragment

Durch Operationen des Typs *BehavioralBindProcessFragment* werden verhaltensbasierte Verfahren zur Bindung von Funktionen umgesetzt. Hierbei wird nach der Auswertung einer Bedingung (*ChooseProcessFragment*) ein Prozess in der Rolle eines Prozessfragments an der Stelle eines Platzhalters im Kontrollfluss aufgerufen. Nach Beendigung des aufgerufenen Prozesses wird die Ausführung des Basisprozesses durch den vom Platzhalter ausgehenden Kontrollfluss fortgesetzt. Parallele Ausführungspfade bleiben während der Ausführung des aufgerufenen Prozesses aktiv. Bei diesem Verfahren wird das durch den gebundenen Prozess beschriebene Verhalten also nicht an der Stelle eines Platzhalters integriert. Stattdessen wird an der Stelle des Platzhalters der gebundene Prozess aufgerufen und auf die Beendigung seiner Ausführung gewartet.

StructuralBindProcessFragment

Die strukturbasierte Operation *StructuralBindProcessFragment* fügt nach Auswertung einer Bedingung (*ChooseProcessFragment*) an der Stelle des Platzhalters ein Prozessfragment in den Kontrollfluss des Basisprozesses ein. Hierdurch wird der in dem Prozess beschriebene Kontrollfluss so strukturell angepasst, dass das Prozessfragment an der Stelle des Platzhalters in den bestehenden Kontrollfluss integriert wird. Sofern ein Platzhalter als ein Containerelement umgesetzt worden ist, kann eine alternative Realisierung der Operation *StructuralBindProcessFragment* durch die Einbettung des Prozessfragments in den Platzhalter vorgenommen werden (siehe auch Anhang A.3).

Existierende Verfahren in Bezug zum SoC

Für den Typ *Late Selection* existieren bereits Arbeiten, die entweder das verhaltensbasierte oder das strukturbasierte Verfahren verwenden. So wurden Umsetzungen des strukturbasierten Verfahrens bspw. durch [Ada+06; DZK11] und des verhaltensbasierten Verfahrens in [Can+08; CDM09; Ard+11] vorgestellt. In den genannten Arbeiten wurde jedoch keine Trennung der Anpassungs- von der Anwendungslogik in einer frühen Phase der Gestaltung der beteiligten Prozesse vorgenommen. Daher gilt es zu untersuchen, inwiefern die Realisierung der Gestaltung des Flexibilitätsaspekts *Late Selection* durch die in dieser Arbeit vorgestellte Sprache *ACML4BPM* unterstützt werden kann und welche Erweiterungen im Rahmen der Sprache *BPMN2.0* sowie der in Abschnitt 4.3.3 eingeführten Operationen notwendig sind.

Gestaltung von Late Selection durch Beobachtungs- und Anpassungsprozesse

Im Rahmen von Operationen des Typs *ChooseProcessFragment* werden Bedingungen ausgewertet. Basierend auf den Ergebnissen dieser Auswertung kann eine mögliche Einbindung von benötigten Funktionen durchgeführt werden. Dieser Vorgang lässt sich durch das in Abschnitt 4.2.2 vorgestellte Konzept des Beobachtungsprozesses beschreiben. Dabei enthält der Beobachtungsprozess das Verhalten zur Auswertung von spezifischen Bedingungen. Eine Beendigung des Beobachtungsprozesses kann einen Anpassungsprozess aufrufen, in dem das Verhalten zur Bindung von Prozessfragmenten beschrieben wird. Das Verhalten zur Bindung von Prozessfragmenten kann dabei durch entsprechende Operationen zur Anpassung von Prozessen (siehe Abschnitt 4.3.3) unterstützt werden.

Typ Late Modeling

Der Typ *Late Modeling* kann – wie in Abbildung 5-52 dargestellt – als Ergänzung zu dem Typ *Late Selection* verstanden werden. So kann es in der Phase *Ausführung* der Fall sein, dass benötigte Funktionen noch nicht in Form von bestehenden Prozessfragmenten vorliegen. Dann kann es sinnvoll sein, dass die benötigten Funktionen neu gestaltet oder auf Basis bestehender Prozessfragmente komponiert werden. Hierbei können neue Prozessfragmente entstehen, die durch die Auswertung von Bedingungen (*ChooseProcessFragment*) und durch die Anwendung von Operationen zur Bindung (*BindProcessFragment*) ausgeführt werden können. Für die Gestaltung neuer Prozessfragmente ist zunächst ein weiterer Typ von Operationen notwendig, der in Abbildung 5-52 in Form von *SwitchLCPhase* dargestellt ist und nachfolgend beschrieben wird.

SwitchLCPhase

Bei der Operation des Typs *SwitchLCPhase* handelt es sich um ein methodisches Konstrukt. Es steht symbolisch für einen Wechsel aus der Phase *Ausführung* entweder in die Phase *Design & Analyse* oder in die Phase *Konfiguration* im *BPM-Lebenszyklus* (siehe Abschnitt 2.2.2). Derartige Vorgehens-

weisen zur Anpassung von Entwicklungs- und Lebenszyklen von Softwaresystemen wurden bereits in [Faz16] vorgestellt. Durch die Anwendung einer Operation des Typs *SwitchLCPhase* soll an dieser Stelle verdeutlicht werden, dass entweder eine Neugestaltung oder alternativ eine Komposition von Prozessfragmenten durchgeführt werden kann.

Im Rahmen einer Neugestaltung (*Create*) von Prozessfragmenten wird hierzu in die Phase *Design & Analyse* gewechselt, in der neue Prozessfragmente gestaltet werden können. Durch eine Komposition (*Compose*) wird hierzu in die Phase *Konfiguration* gewechselt, in der neue Prozessfragmente auf Basis bestehender Prozessfragmente komponiert werden können. Durch die Anwendung von Operationen des Typs *SwitchLCPhase* wird die Ausführung des Basisprozesses unterbrochen. Nach Beendigung einer Neugestaltung oder Komposition ist es möglich, den Basisprozess weiter auszuführen. Werden nachgelagert Operationen der Typen *ChooseProcessFragment* und *BindProcessFragment* angewendet, können hierbei neu erstellte oder komponierte Prozessfragmente berücksichtigt werden. Ferner ist für die Neugestaltung oder Komposition von Prozessfragmenten im Rahmen der Anwendung von Operationen des Typs *SwitchLCPhase* weitere Typen von Operationen notwendig. Zu diesem Zweck werden nachfolgend die beiden Typen von Operationen *CreateProcessFragment* und *ComposeProcessFragment* eingeführt und beschrieben.

CreateProcessFragment

Im Rahmen einer Neugestaltung können benötigte Funktionen in Form von Prozessfragmenten beschrieben und umgesetzt werden, sodass sie im Anschluss zum Zweck der Bindung (*BindProcessFragment*) zur Verfügung stehen. Durch Operationen des Typs *CreateProcessFragment* werden methodische Aktivitäten zur Neugestaltung von Prozessfragment durch Experten durchgeführt. In [Sch+08] wird hierzu genannt, dass dieser Prozess der Neugestaltung lediglich durch Experten der Anwendung ausgeführt werden sollte. Dies lässt sich u.a. dadurch begründen, dass i.d.R. nur Experten einer Anwendung in der Lage sind, zu erkennen, dass und vor allem welche weiteren Funktionen in Form von Prozessfragmenten benötigt werden.

ComposeProcessFragment

Ein weiterer Typ von Operationen (*ComposeProcessFragment*) unterstützt die Komposition von benötigten Funktionen auf Basis existierender Prozessfragmente. Eine Komposition von Funktionen stellt in diesem Bezug einen Vorgang dar, der aus mindestens zwei bestehenden Funktionen eine weitere neue Funktion erstellt. Dabei wird durch die neue Funktion eine Ausgabe erzeugt, welche ohne die durchgeführte Komposition so nicht möglich gewesen wäre. Eine einfache Komposition kann z.B. aus der Se-

quenz einer Ausführung von zwei Funktionen bestehen. Ferner handelt es sich bei dem Typ von Operationen *ComposeProcessFragment* um methodische Aktivitäten zur Komposition von Prozessfragmenten, die durch Experten durchgeführt werden. Neue Prozessfragmente, die durch die Anwendung von Operationen der Typen *CreateProcessFragment* oder *ComposeProcessFragment* erstellt worden sind, können bei nachfolgenden Anwendungen der Operationen vom Typ *BindProcessFragment* gebunden werden. Dadurch können zuvor nicht vorhandene Funktionen im Rahmen der Ausführung des Basisprozesses genutzt werden, wodurch zudem auch auf spezifische Umstände in einer Umgebung eingegangen werden kann.

5.5.1.1 Eigenschaften von Late Selection und Late Modeling

Für eine mögliche Realisierung der zuvor ausgeführten Typen des Flexibilitätsaspekts *Flexibility-by Underspecification* existieren – wie in Abbildung 5-52 gezeigt – weitere Eigenschaften. So kann es zum einen unterschiedliche Zeitpunkte (*RealisationPointInTime*) für eine Bindung von Prozessfragmenten geben. Zum anderen existieren verschiedene Typen von Zeitdauern (*RealisationSpaceInTime*), für die eine Bindung von Prozessfragmenten bestehen soll. Eine für die vorliegende Arbeit angepasste Interpretation dieser Eigenschaften wird im Folgenden näher beschrieben.

In Tabelle 5-2 ist eine Übersicht über Zeitpunkte einer Bindung in Relation zu relevanten Phasen und betroffenen Artefakten zusammengefasst. Eine Bindung von Prozessfragmenten kann zu unterschiedlichen Zeitpunkten im *BPM-Lebenszyklus* durchgeführt werden. So existieren die beiden Typen von Zeitpunkten *PreliminaryRealisation* und *OnActivationRealisation*, an denen ein ausgewähltes Prozessfragment gebunden werden kann. Bei beiden Typen von Zeitpunkten können spezifische Ereignisse auftreten, auf deren Basis eine Auswahl (*ChooseProcessFragment*) und eine Bindung (*BindProcessFragment*) eines geeigneten Prozessfragments getroffen bzw. durchgeführt werden kann.

Zeitpunkt	**Phase Konfiguration**	**Phase Ausführung**
Preliminiary Realisation	Prozessmodell	Prozessmodell Prozessinstanz
OnActivation Realisation	–	Prozessinstanz

Tabelle 5-2: *Typen von Zeitpunkten und betroffene Artefakte entlang relevanter Phasen des BPM-Lebenszyklus*

PreliminaryRealisation

Bei Zeitpunkten des Typs *PreliminaryRealisation* wird eine Bindung vor der Aktivierung eines Platzhalters durchgeführt. Beispiele für konkrete Zeitpunkte im *BPM-Lebenszyklus* sind z.B. durch die Phase *Konfiguration*, in der Prozesse konfiguriert werden, oder zu ihrer Instanziierung in der Phase *Ausführung* gegeben. Selbstverständlich bestehen hier auch Zeitpunkte während der Ausführung eines Prozesses, aber vor Aktivierung eines in seinem Kontrollfluss enthaltenen Platzhalters.

OnActivationRealisation

Der Typ von Zeitpunkten *OnActivationRealisation* stellt den Zeitpunkt der Bindung eines Prozessfragments bei Aktivierung des Platzhalters dar. Somit wird die Bindung unmittelbar an der Stelle im Kontrollfluss durchgeführt, die flexibel durch die Auswahl von Prozessfragmenten gestaltet werden soll. Durch den Typ *OnActivationRealisation* kann ein besonders hoher Grad an Flexibilität gewährleistet werden, da die Bindung zu dem spät möglichsten Zeitpunkt durchgeführt werden kann.

Ferner existiert eine weitere Eigenschaft, durch die der Typ einer Realisierung hinsichtlich einer Mehrfachanwendung eines Platzhalters bzw. seiner gebundenen Prozessfragmente beschrieben werden kann. So wird zwischen den beiden Typen *StaticRealisation* und *DynamicRealisation* unterschieden, welche zur Übersicht in Tabelle 5-3 in Relation zu Phasen und betroffenen Artefakten dargestellt sind.

Tabelle 5-3: *Typen von Zeitdauern und betroffene Artefakte entlang relevanter Phasen des BPM-Lebenszyklus*

Zeitdauer	**Phase Konfiguration**	**Phase Ausführung**
Static Realisation	Prozessmodell	Prozessmodell Prozessinstanz[1]
Dynamic Realisation	Prozessmodell	Prozessmodell Prozessinstanz

StaticRealisation

Bei dem Typ *StaticRealisation* wird ein einmal gebundenes Prozessfragment auch für die weitere Ausführung des Basisprozesses an der Stelle des Platzhalters gebunden sein. Eine Bindung im Rahmen des Typs *StaticRealisation* kann also als permanent betrachtet werden (siehe auch Abschnitt 5.3), da sie nachträglich nicht mehr veränderbar ist. So könnte z.B. eine Bindung bereits in der Phase *Konfiguration* vorgenommen worden sein, die auch aufgrund aktualisierter Anforderungen in der Phase *Ausführung* Bestand haben soll. Eine Mehrfachanwendung eines Platzhalters ist somit nicht vorgesehen.

DynamicRealisation

Der Typ *DynamicRealisation* hingegen ermöglicht, ein Prozessfragment zunächst zu binden und zu einem späteren Zeitpunkt eine erneute Bindung eines anderen Prozessfragments vorzunehmen. Dies kann notwendig sein,

[1] Bindung ist nur möglich, wenn nicht bereits in der Phase *Konfiguration* getätigt worden ist.

wenn sich an geänderte Bedingungen angepasst werden muss. So könnten z.B. aktualisierte Bedingungen in einer Umgebung vorherrschen, die die weitere Ausführung der Prozesse behindern oder die erfolgreiche Beendigung unmöglich machen. Durch den Typ *DynamicRealisation* können also dynamische Bindungen durchgeführt werden. Im Rahmen des Typs *DynamicRealisation* wäre so z.B. eine Bindung eines vorläufigen Prozessfragments in der Phase *Konfiguration* möglich. Dieses vorläufige Prozessfragment kann in der Phase *Ausführung* durch ein anderes Prozessfragment ersetzt werden. Eine Mehrfachanwendung eines Platzhalters ist somit vorgesehen.

Für die beiden Typen *Late Selection* und *Late Modeling* des Flexibilitätsaspekts *Flexibility-by Underspecification* sind Sprachelemente notwendig, die sich jedoch mittels der Sprache *BPMN2.0* in ihrer ursprünglichen Form nicht ausdrücken lassen. Daher ist eine Erweiterung des Metamodells notwendig, mit der relevante Aspekte zur Unterstützung des Flexibilitätsaspekts *Flexibility-by Underspecification* beschrieben werden können. Hierzu wird im nachfolgenden Abschnitt 5.5.2 zunächst eine Erweiterung des Metamodells und anschließend in Abschnitt 5.5.3 eine detailliertere Beschreibung der zuvor aufgeführten Operationen zur Unterstützung der Gestaltung des Flexibilitätsaspekts *Flexibility-by Underspecification* durch die Sprache *ACML4BPM* gegeben.

5.5.2 Spracherweiterung für Flexibility-by Underspecification

Für den Flexibilitätsaspekt *Flexibility-by Underspecification* sowie für die enthaltenen Typen *Late Selection* und *Late Modeling* werden im Rahmen der Gestaltung von flexiblen und anpassbaren Prozessen weitere Sprachelemente benötigt. Da sich die beiden Untertypen *Late Selection* und *Late Modeling* sowohl auf Anpassungen von Prozessmodellen als auch auf deren Instanzen beziehen können, werden zur Unterscheidung dieser beiden Arten von Modell- und Prozesselementen abermals verschiedene Typen benötigt (siehe auch Abschnitt 5.3 bzw. Abschnitt 5.4). Hierzu wird eine entsprechende konzeptionelle Spracherweiterung in Anlehnung an den in Abschnitt 4.3.3 vorgestellten Auszug der Sprache *BPMN2.0* vorgestellt.

In Abbildung 5-53 ist die konkrete Syntax der Elemente *Placeholder*, *ProcessFragment* sowie der zugehörigen Start- bzw. Endsymbole *ProcessFragmentStartEvent* und *ProcessFragmentEndEvent* anhand von Auszügen der beiden Prozessmodelle *m* und *n* beschrieben. Auf Details wird nachfolgend eingegangen.

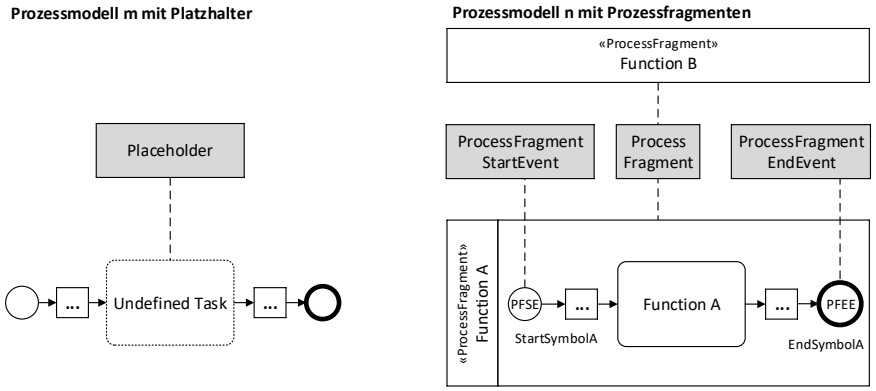


Abbildung 5-53: Konkrete Syntax für Platzhalter, Prozessfragmente sowie Start- und Endsymbole

Platzhalter

Im linken Bereich ist ein Auszug des Prozessmodells *m* in Form eines Kontrollflusses dargestellt. Dieser enthält ein Element vom Typ *Placeholder* mit der Bezeichnung *Undefined Task*. Elemente des Typs *Placeholder* werden in Anlehnung an Elemente des Typs *SubProcess* in der Sprache *BPMN2.0* dargestellt. Dabei wird zur Kennzeichnung dieses speziellen Typs eines Subprozesses auf ein sonst übliches eigenes Symbol, wie z.B. bei *Manual Tasks*, *Human Tasks* oder *Service Tasks* (siehe Abschnitt 2.3.4), verzichtet. Stattdessen wird der Rahmen des Elements gepunktet dargestellt. Hierdurch soll verdeutlicht werden, dass es sich um einen Platzhalter handelt, dessen Inhalt zunächst undefiniert bleibt und zu einem späteren Zeitpunkt durch den Inhalt eines Prozessfragments vervollständigt wird.

Prozessfragment

Im rechten Bereich wird ein Auszug des Prozessmodells *n* mit zwei Prozessfragmenten zur Realisierung der jeweiligen Funktionen mit den Bezeichnungen *Function A* und *Function B* dargestellt. Für die Darstellung eines Prozesses in der Sprache *BPMN2.0* existiert keine explizite grafische Darstellung. Dies kann als sinnvoll betrachtet werden, da der Prozess mit den in ihm enthaltenen weiteren Elementen dargestellt wird. In der hier vorgestellten Lösung wird jedoch die Möglichkeit zur Abgrenzung von regulären Prozessen und Elementen des Typs *ProcessFragment* gegeben. So wird für Prozessfragmente eine Darstellungsweise in Anlehnung an Elemente vom Typ *Pool* der Sprache *BPMN2.0* gewählt. Ein Element des Typs *ProcessFragment* wird demnach als *Pool* dargestellt; dabei kann ein solches Prozessfragment sowohl aufgeklappt als auch zugeklappt dargestellt werden. Ein Beispiel für ein zugeklapptes Prozessfragment ist in Prozessmodell *n* durch das Element mit der Bezeichnung *Function B* gegeben. Ein aufgeklapptes Prozessfragment ist durch das Element mit der Bezeichnung *Funktion A* dargestellt. Beide Darstellungsweisen enthalten zum Zweck der Kennzeichnung den Namen des Typs in Guillemets geklammert.

Startsymbol

Im Rahmen des Prozessfragments mit der Bezeichnung *Function A* wird ein Auszug eines Kontrollflusses dargestellt. Jedes Prozessfragment hat genau ein Startsymbol vom Typ *ProcessFragmentStartEvent*. Ein solches Startsymbol wird in Anlehnung an Startereignisse der Sprache *BPMN2.0* dargestellt. Zur Unterscheidung von Elementen des Typs *ProcessFragmentStartEvent* und *StartEvent* wird hier jedoch die Abkürzung des Typs *PFSE* als Label dargestellt.

Endsymbol

Das Ende eines Prozessfragments wird durch ein Endsymbol vom Typ *ProcessFragmentEndEvent* gekennzeichnet. Die Darstellung dieses Typs wird in Anlehnung an Endereignisse der Sprache *BPMN2.0* umgesetzt. Zur Unterscheidung von Elementen des Typs *ProcessFragmentEndEvent* und *EndEvent* wird hier jedoch die Abkürzung des Typs *PFEE* als Label dargestellt.

Für die zuvor beschriebenen Elemente sollen zudem auch Repräsentationen ihrer Instanzen verfügbar sein. Dies betrifft insbesondere Instanzen von Platzhaltern und Prozessfragmenten. In Abbildung 5-54 ist die konkrete Syntax dieser Elemente gezeigt. Dabei werden verschiedene Beispiele anhand der Auszüge der Prozessinstanzen *PI*, *PI2* und *PFI* dargestellt und nachfolgend beschrieben.

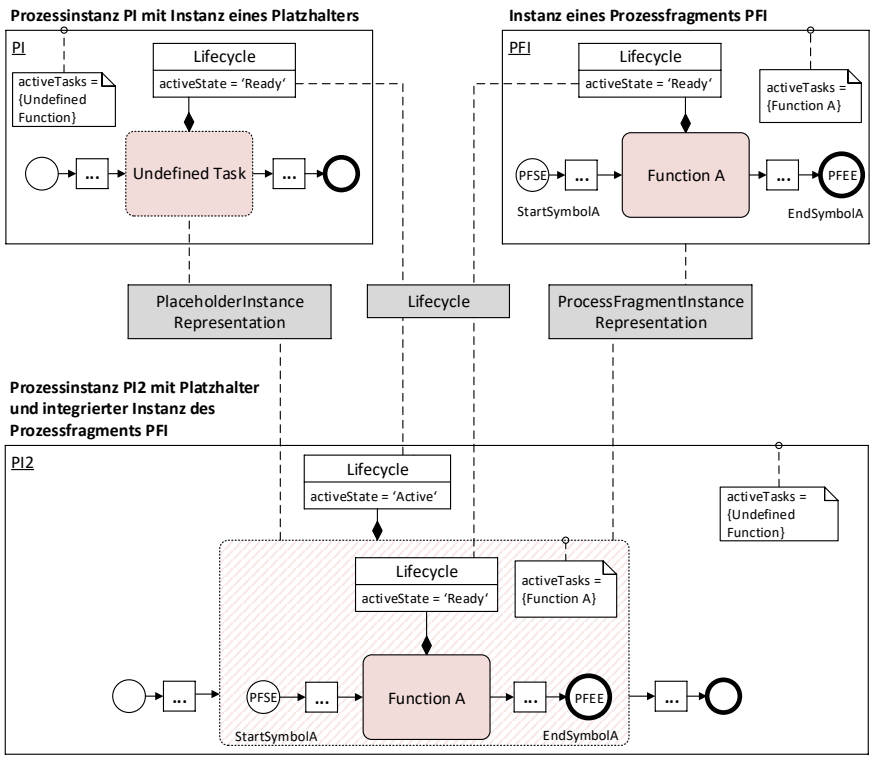


Abbildung 5-54: Konkrete Syntax für Instanzen von Platzhaltern und Prozessfragmenten

Instanz eines Platzhalters

Die Repräsentation einer Instanz eines Platzhalters wird in Abbildung 5-54 im Kontext einer Prozessinstanz *PI* dargestellt. Auf eine Beschreibung der konkreten Syntax von Prozessinstanzen wird an dieser Stelle verzichtet und stattdessen auf Abschnitt 5.3.3 verwiesen. Die Prozessinstanz *PI* enthält einen Kontrollfluss, in dem ein Platzhalter mit der Bezeichnung *Undefined Task* enthalten ist. Die Repräsentation einer Instanz eines Platzhalters wird in Anlehnung an *Tasks* der Sprache *BPMN2.0* dargestellt. Dabei wird der Rahmen eines Platzhalters jedoch – wie in Abbildung 5-54 abgebildet – gestrichelt gezeichnet.

Instanz eines Prozessfragments

Bei einer Instanz eines Prozessfragments handelt es sich um einen Untertyp einer Instanz eines Prozesses (siehe auch Abbildung 5-55). Die Darstellungsweise wird daher zur Vereinfachung an dieser Stelle übernommen. Die in Abbildung 5-54 dargestellte Instanz eines Prozessfragments *PFI* enthält einen Kontrollfluss mit spezifischen Start- (*PFSE*) und Endsymbolen (*PFEE*) sowie einer beispielhaften Instanz eines Tasks mit der Bezeichnung *Function A*. Aktive Instanzen von Tasks können analog zu Prozessinstanzen (siehe Abschnitt 5.3.3) über das Attribut mit der Bezeichnung *activeTasks* angegeben werden.

Instanz eines Platzhalters mit integrierter Instanz eines Prozessfragments

Neben der getrennten Darstellung von Instanzen von Platzhaltern und Prozessfragmenten ist es ebenso möglich, die Instanz eines Prozessfragments integriert in einer Instanz eines Platzhalters darzustellen. Hierzu ist im unteren Bereich von Abbildung 5-54 die Prozessinstanz *PI2* angegeben. Dabei wird die Instanz des Platzhalters mit der Bezeichnung *Undefined Task* mit einer in ihr integrierten Instanz des Prozessfragments *PFI* gezeigt. Die Darstellungsweise eines integrierten Prozessfragments beschreibt dabei eine Situation, in der zuvor eine strukturbasierte Bindung (*StructuralBindProcessFragment*) stattgefunden hat. Eine verhaltensbasierte Bindung (*BehavioralBindProcessFragment*) besitzt in diesem Ansatz keine eigene grafische Darstellung. Bei dieser Darstellungsweise ist anzumerken, dass sowohl für die übergeordnete Prozessinstanz *PI2* als auch für die Instanz eines Prozessfragments *PFI* jeweils die Attribute mit der Bezeichnung *activeTasks* zur Kenneichung von derzeit aktiven Instanzen von Platzhaltern und Tasks verwendet werden kann.

Für die zuvor beschriebenen Elemente wird nachfolgend die abstrakte Syntax vorgestellt. In Abbildung 5-55 wird hierzu ein Auszug eines Metamodells dargestellt. Dabei werden die für die Gestaltung des Flexibilitätsaspekts *Flexibility-by Underspecification* benötigten Typen *ProcessFragment*, *Placeholder*, *ProcessFragmentInstanceRepresentation* und *PlaceholderInstanceRepresentation* eingeführt.

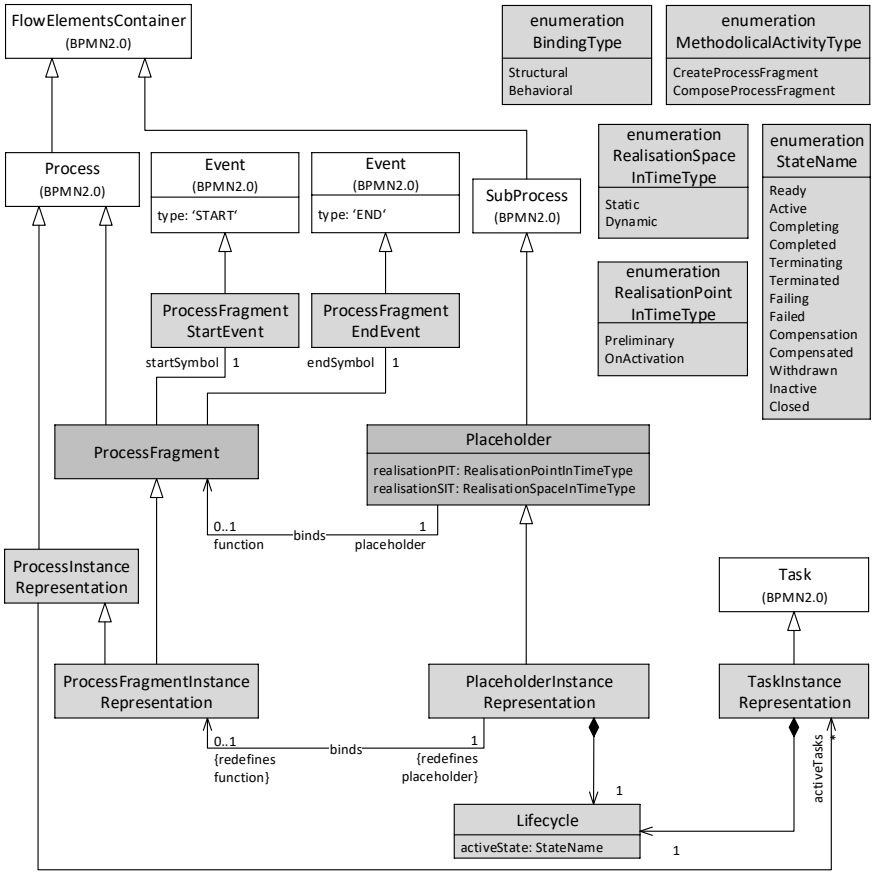


Abbildung 5-55: Auszug einer Erweiterung des Metamodells der BPMN2.0 zur Unterstützung von Flexibility-by Underspecification

Placeholder

Ein Task, dessen Inhalt zum Zeitpunkt der Gestaltung undefiniert bleibt, wird hier von dem Typ *Placeholder* dargestellt. Dabei erbt in der hier vorgestellten Lösung ein Element vom Typ *Placeholder* von dem Typ *SubProcess*. Hierdurch werden Eigenschaften übernommen, sodass es sich bei dem Typ *Placeholder* um ein Containerelement handelt. Sollen Elemente dieses Typs angepasst werden, können daher auch die in Anhang A.3 beschriebenen Operationen zur Anpassung von Containerelementen verwendet werden. Zeitliche Eigenschaften eines Platzhalters können in dem Typ *Placeholder* über die beiden Attribute beschrieben werden. Mit dem Attribut *realisationPIT* kann angegeben werden, welcher Zeitpunkt für eine Bindung vorgesehen ist. Wohingegen mit dem Attribut *realisationSIT* angegeben werden kann, für welche Zeitdauer eine Bindung gelten soll.

ProcessFragment

Eine zu bindende Funktion wird durch den Typ *ProcessFragment* dargestellt. Um sowohl struktur- als auch verhaltensbasierte Bindungen unterstützen zu können, wird in dem hier dargestellten Konzept der Typ *Process-*

Fragment als von dem Typ *Process* erbend dargestellt. Auf Details wird an dieser Stelle verzichtet und stattdessen auf Abschnitt 5.5.3 verwiesen. Ferner referenziert der Typ *Placeholder* sowohl auf ein spezifisches Startsymbol (*ProcessFragmentStartEvent*) als auch auf ein spezifisches Endsymbol (*ProcessFragmentEndEvent*), die beide jeweils von Start- bzw. Ereignissen (*Event*) der Sprache *BPMN2.0* erben. Durch die in dieser Arbeit beschriebene Lösung werden in der Gestaltung von Prozessfragmenten lediglich solche Prozessfragmente unterstützt, die jeweils über genau ein Start- und Endsymbol verfügen. Durch die beschriebene Vererbung und Einführung spezifischer Start- und Endsymbole auf Basis üblicher Typen der Sprache *BPMN2.0* kann sowohl eine struktur- als auch eine verhaltensbasierte Bindung beschrieben werden (siehe Abschnitt 5.5.3). Die Assoziation zwischen den beiden Typen *Placeholder* und *ProcessFragment* beschreibt, dass ein Element vom Typ *Placeholder* ein Element vom Typ *ProcessFragment* in der Rolle einer zu realisierenden Funktion bindet.

Für die zuvor eingeführten Typen *ProcessFragment* und *Placeholder* werden zudem Repräsentationen ihrer Instanzen benötigt. Hierdurch können sowohl Anpassungen von Prozessmodellen als auch deren Instanzen gestalten werden. Nachfolgend wird auf die Repräsentation von Instanzen der zuvor genannten Elemente eingegangen.

PlaceholderInstanceRepresentation

Die Instanz eines Platzhalters wird durch den Typ *PlaceholderInstanceRepresentation* repräsentiert. Ein Element dieses Typs kann weitere Elemente enthalten, wobei sich in Anlehnung an Abschnitt 5.3 auf Tasks in Form von Elementen des Typs *TaskInstanceRepresentation* beschränkt wird. Ferner enthält ein Element des Typs *PlaceholderInstanceRepresentation* ein Element vom Typ *Lifecycle*, welches den aktuellen Zustand des Lebenszyklus des Platzhalters beschreibt. Hierzu wird das Attribut *activeState* vom Typ *StateName* verwendet. Bei dem Typ *StateName* handelt es sich um eine Enumeration mit Literalen aller bereits in Abschnitt 4.3.4.2 eingeführten Zustände des Lebenszyklus von Aktivitäten.

ProcessFragmentInstanceRepresentation

Die Instanz eines Prozessfragments wird in der hier vorgestellten Lösung durch den Typ *ProcessFragmentInstanceRepresentation* repräsentiert. Sie erbt sowohl alle Eigenschaften vom Typ *ProcessFragment* als auch vom Typ *ProcessInstanceRepresentation*. Durch die zuletzt genannte Vererbung kann die Verwaltung von aktiven Taskinstanzen in einer Instanz eines Prozessfragments vereinfacht werden, da die Liste mit dem Bezeichner *activeTasks*, die vom Typ *TaskInstanceRepresentation* ist, als Eigenschaft vorliegt.

Die Assoziation (*binds*) zwischen den beiden Typen *PlaceholderInstance* und *ProcessFragmentInstance* beschreibt, dass ein Element vom Typ *PlaceholderInstance* ein Element vom Typ *ProcessFragmentInstance* in der Rolle einer zu realisierenden Funktion bindet. Dabei werden die beiden Rollen *function* und *placeholder* derartig redefiniert, dass Elemente zur Repräsentation von Instanzen auch nur solche referenzieren können.

Lebenszyklus für Platzhalter

Für die zuvor beschriebenen Elemente der Typen *PlaceholderInstanceRepresentation* und *TaskInstanceRepresentation* kann durch den jeweiligen spezifischen Lebenszyklus (*Lifecycle*) beschrieben werden, in welchen aktuellen Zustand sich das Element befindet. Ein Element vom Typ *Lifecycle* wird in Anlehnung an *UML-Klassen* [OMG10] dargestellt. Die Zugehörigkeit zu einer Instanz eines Tasks oder Platzhalters wird in Anlehnung an *UML-Assoziationen* in Form der *Komposition* [OMG10] dargestellt.

Ein Beispiel für einen spezifischen Lebenszyklus zur Unterstützung des Flexibilitätsaspekts *Flexibility-by Underspecification* ist in Abbildung 5-56 gezeigt. So wird der in Abschnitt 4.3.4.2 eingeführte Lebenszyklus um einen weiteren Zustand (*Binding*) erweitert. Der Zustand *Binding* wird aktiv, wenn ein Token den Platzhalter erreicht. Sobald eine Bindung durchgeführt worden ist, wird der Zustand verlassen und der Lebenszyklus folgt dem bereits zuvor beschriebenen Ablauf. Damit Ereignisse des Lebenszyklus eines Platzhalters zur Auslösung eines *AC4BPM* genutzt werden können, kann, wie für implizite Ereignisse gezeigt worden ist, eine Transformation der Zustände in Ereignisse durchgeführt werden (siehe Abschnitt 4.3.4.2). Als Ergebnis einer solchen Transformation ergibt sich das neue implizite Ereignis *onBindingDone*. Das Ereignis wird im weiteren Verlauf des Abschnitts dazu verwendet, eine Rückkopplung mit dem Platzhalter im Anschluss an eine durchgeführte Bindung zu ermöglichen. Selbstverständlich sind je nach Anforderungen auch alternative Lebenszyklen denkbar, in denen bspw. weitere Ereignisse für eine Rückkopplung eingesetzt werden können.

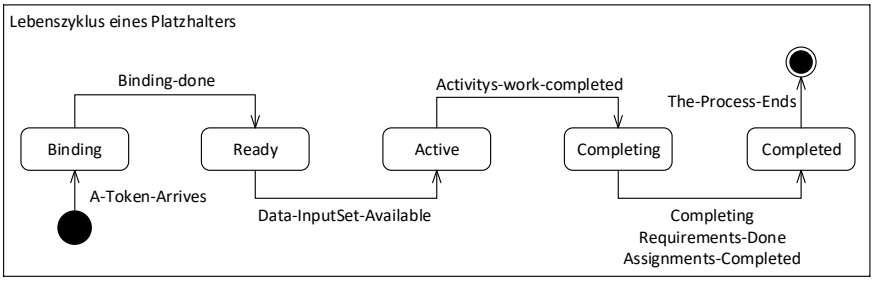


Abbildung 5-56: Beispielhafte Darstellung von Elementen der Erweiterung in Hinsicht auf die Unterstützung von Flexibility-by Underspecification

5.5.3 Operationen

Für die Realisierung der beiden Typen *Late Selection* und *Late Modeling* des Flexibilitätsaspekts *Flexibility-by Underspecification* unter Verwendung der Sprache *ACML4BPM* sind neben den im Abschnitt 5.5.2 vorgestellten Spracherweiterungen auch zugehörige Operationen notwendig. Derartige Operationen werden in diesem Abschnitt zum Zweck der Verwendung in der Gestaltung von Beobachtungs- und Anpassungsprozessen auf Basis der in Abschnitt 5.5.1 eingeführten Funktionsprinzipien beschrieben. In diesem Bezug wird zunächst in Abschnitt 5.5.3.1 eine Operation zur Bindung von Prozessfragmenten vorgestellt. Nachgelagert wird in Abschnitt 5.5.3.2 ein Beispiel für die Gestaltung einer Operation vom Typ *ChooseProcessFragment* unter Verwendung eines Beobachtungsprozesses gegeben. Abschließend wird in Abschnitt 5.5.3.3 eine Operation vom Typ *SwitchLCPhase* vorgestellt. Für jede der zuvor genannten Operationen wird zudem die operationale Semantik in Anlehnung an ein Beispiel beschrieben.

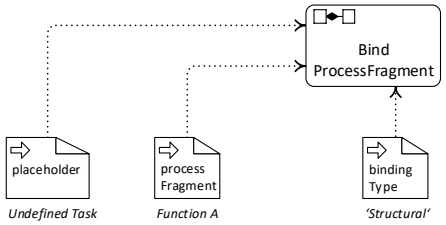
5.5.3.1 BindProcessFragment

Durch eine Anwendung einer Operation vom Typ *BindProcessFragment* lässt sich ein Prozessfragment an der Stelle eines Platzhalters in dem Kontrollfluss eines Prozesses binden. Hierdurch ist es möglich, die Entscheidung für die Verwendung einer konkreten Funktion in eine spätere Phase des *BPM-Lebenszyklus* zu verlegen – hier die Phase *Ausführung*. Ferner wird dabei zwischen zwei Mechanismen unterschieden. Zum einen können Prozessfragmente strukturbasiert in einen Platzhalter eingefügt werden. Zum anderen können Prozessfragmente aber auch verhaltensbasiert gebunden werden. Bei dem verhaltensbasierten Mechanismus ist keine strukturelle Anpassung wie bei dem strukturbasierten Mechanismus in Form einer Integration der benötigten Funktion im Platzhalter notwendig. Die Signatur und konkrete Syntax der Operation *BindProcessFragment* sind in Abbildung 5-57 angegeben.

Parameter Die Operation *BindProcessFragment* erwartet als Eingabe eine Instanz eines Platzhalters (*placeholder*), an dessen Stelle die Instanz eines Prozessfragments gebunden werden soll. Ein weiterer Parameter der Operation ist durch die betreffende Instanz des zu bindenden Prozessfragments (*processFragment*) gegeben. Welcher der beiden in Abschnitt 5.5.1 eingeführten

	Parametername	Parametertyp
IN :	*placeholder* *processFragment* *bindingType*	*PlaceholderInstanceRepresentation* *ProcessFragmentInstanceRepresentation* *BindingType*
OUT :	–	*VOID*

Abbildung 5-57: *Signatur und konkrete Syntax der Operation BindProcessFragment*



Mechanismen zur Bindung verwendet werden soll, kann durch das Attribut *bindingType* angegeben werden. Gültige Werte sind durch die in Abbildung 5-55 dargestellte Enumeration *BindingType* gegeben. So wird durch das Literal *Structural* angegeben, dass eine strukturbasierte Bindung durchgeführt werden soll. Alternativ wird durch das Literal *Behavioral* angegeben, dass eine verhaltensbasierte Bindung durchgeführt wird. Die Anwendung der Operation erzeugt keine explizite Ausgabe – hier durch die Kennzeichnung als *VOID* dargestellt.

Im Folgenden wird auf die operationale Semantik der Anwendung der Operation *BindProcessFragment* eingegangen. Zunächst erfolgt eine Beschreibung einer Parametrisierung für die strukturbasierte Bindung und anschließend für die verhaltensbasierte Bindung. Dabei wird bei beiden Funktionsprinzipien zunächst geprüft, ob eine Anwendung der Operation hinsichtlich eines gegebenen Zeitpunkts und einer Zeitdauer erlaubt ist. So kann nach den in Abschnitt 5.5.1.1 eingeführten Typen von Zeitpunkten *PreliminaryRealisation* und *OnActivationRealisation* die Anwendung der Operation möglicherweise nicht erlaubt sein. Gleiches gilt für eine Situation, in der eine Bindung nicht erneut durchgeführt werden darf (*StaticRealisation*). Wurden die zugehörigen Attribute eines Platzhalters entsprechend gesetzt, führt die Operation keine der im Folgenden beschriebenen Schritte aus.

Strukturbasierte Bindung von Prozessfragmenten Das Funktionsprinzip einer Anwendung der Operation *BindProcessFragment* mit der Parametrisierung für eine strukturbasierte Bindung wird in Abbildung 5-58 dargestellt. Eine Anwendung der Operation besteht demnach aus den zwei Teilschritten *IntegrateProcessFragment* und *ReturnControl*.

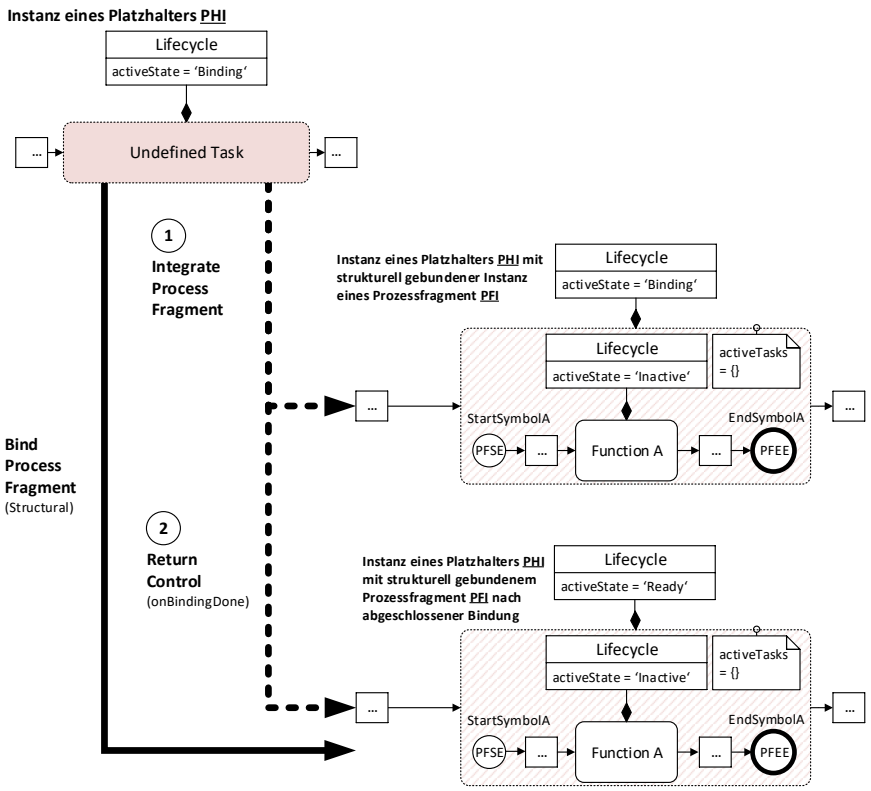


Abbildung 5-58: Beispielhafte Anwendung der Operation BindProcessFragment (Structural)

So wird ausgehend von einer Instanz eines Platzhalters, die sich in dem Zustand *Binding* ihres Lebenszyklus befindet, der erste Teilschritt *IntegrateProcessFragment* durchgeführt. Das Ergebnis des ersten Teilschrittes ist eine strukturelle Integration einer gewählten Instanz eines Prozessfragments *PFI* in die Instanz des Platzhalters *PHI*. Die in dem integrierten Prozessfragment enthaltenen Tasks zur Realisierung der gewählten Funktion bleiben vorerst, wie dargestellt, noch inaktiv.

Der zweite Teilschritt *ReturnControl* beendet den Vorgang der Bindung. Dabei wird in den Zustand *Ready* des Lebenszyklus der Instanz des Platzhalters *PHI* gewechselt. Siehe in diesem Bezug auch das in Abbildung 5-56 dargestellte Beispiel eines Lebenszyklus mit Unterstützung von *Flexibility-by Underspecification*. Nachfolgend ist die Ausführung des in *PHI* beschrieben Verhaltens möglich. Dabei können die einzelnen in Abschnitt 4.3.4.2 beschriebenen Zustände des Lebenszyklus des Platzhalters durchlaufen werden. Bei diesem Vorgang wird das in *PHI* gebundene Prozessfragment *PFI* als Inhalt des Subprozesses vom Typ *Placeholder* aktiviert und ausgeführt.

Verhaltensbasierte Bindung von Prozessfragmenten Für die Anwendung der Operation *BindProcessFragment* mit der Parametrisierung zur verhaltensbasierten Bindung wird das generelle Funktionsprinzip in Abbildung 5-59 dargestellt. Eine Anwendung der Operation besteht demnach aus den drei Teilschritten *PassingControl*, *ExecuteFunction* und *ReturnControl*.

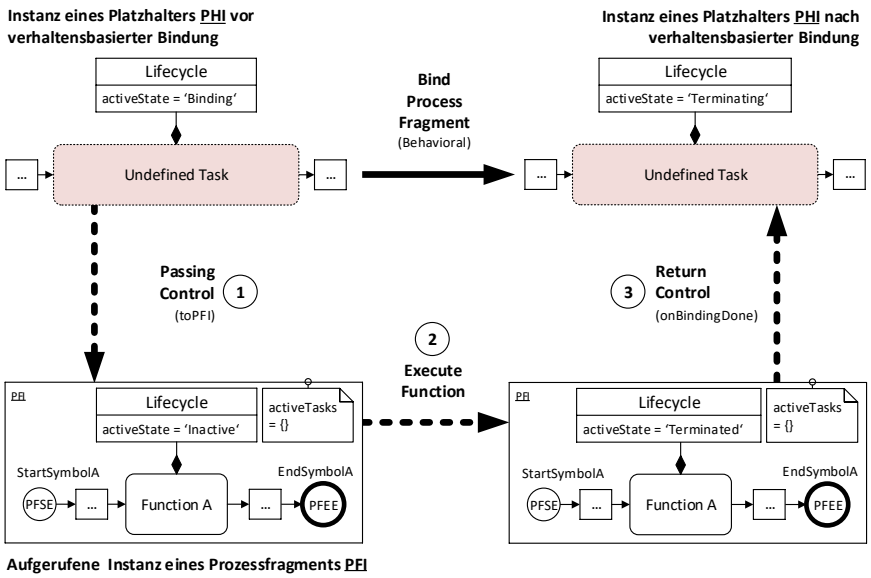


Abbildung 5-59: *Beispielhafte Anwendung der Operation BindProcessFragment (Behavioral)*

Bei der verhaltensbasierten Bindung wird ein Prozessfragment nicht wie bei der strukturbasierten Bindung in den Platzhalter integriert. Stattdessen wird an der Stelle der Instanz eines Platzhalters *PHI* eine Funktion in Form einer Instanz eines Prozessfragments *PFI* als eigenständiger Prozess aufgerufen und ausgeführt. Der Aufruf einer solchen Instanz eines Prozessfragments *PFI* ist daher in Abbildung 5-59 als der erste Teilschritt *PassingControl* dargestellt.

Die Ausführung des gebundenen Prozessfragments *PFI* stellt den zweiten Teilschritt *ExecuteFunction* im Rahmen einer Anwendung der Operation *BindProcessFragment* dar. Während dieses Teilschrittes bleibt der Kontrollflusspfad des Basisprozesses, von dem die Bindung ausgegangen ist, inaktiv. Parallele Kontrollflusspfade bleiben jedoch aktiv.

Nachdem die Ausführung der Instanz eines Prozessfragments *PFI* abgeschlossen ist, wird der dritte und letzte Teilschritt *ReturnControl* durchgeführt. So wird die Instanz des ausgehenden Platzhalters in den Zustand *Terminating* seines Lebenszyklus versetzt. Hierdurch kann seine

Ausführung abgeschlossen werden. Bei einer verhaltensbasierten Bindung werden somit die einzelnen Zustände des Lebenszyklus eines Platzhalters zwischen *Binding* und *Terminating* übersprungen. Für dieses Vorgehen wurde sich entschieden, da die eigentliche Ausführung einer benötigten Funktion im Gegensatz zu dem strukturbasierten Mechanismus an einen aufzurufenden Prozess delegiert wird. Die dazwischenliegenden Zustände des Lebenszyklus der Instanz eines Platzhalters *PHI* werden somit nicht benötigt.

Die Anwendung der Operation *BindProcessFragment* zur verhaltensbasierten Bindung beinhaltet in ihrer zuvor beschriebenen Form explizit auch die Ausführung der aufgerufenen Instanz eines Prozessfragments. Dies steht im Kontrast zu der Funktionsweise der Operation im Rahmen einer strukturbasierten Bindung, da hier die integrierte Instanz eines Prozessfragments erst nach Beendigung ausgeführt wird. Es wurde sich für die vorliegende Variante entschieden, da die weitere Ausführung des ausgehenden Kontrollflusses stets nach dem Schritt *ReturnControl* durchgeführt wird. Dies schließt den Zeitrahmen der Ausführung einer aufgerufenen Instanz eines Prozessfragments im Rahmen der verhaltensbasierten Bindung ein.

5.5.3.2 ChooseProcessFragment

Im Rahmen von Operationen des Typs *ChooseProcessFragment* wird eine Auswahl von zu bindenden Funktionen getroffen. Eine derartige Funktion steht in diesem Zusammenhang in Form eines Prozessfragments zur Verfügung. Dabei kann im Rahmen der zuvor genannten Auswahl eine Auswertung von Bedingungen durchgeführt werden, durch die die Flexibilität eines Prozesses gesteigert werden kann (siehe auch Abschnitt 5.2.2). Im Rahmen derartiger Auswertungen von Bedingungen können z.B. verschiedene Eigenschaften in Hinsicht auf die Laufzeit einer konkreten Umgebung enthalten sein. Es kann sich in diesem Bezug auch um komplexe Prozesse zur Entscheidung handeln, sodass neben einer reinen Auswertung von Bedingungen auch weitere Analyseschritte notwendig sein können.

Auswahl von Prozessfragmenten im Rahmen von Beobachtungsprozessen

Anstelle der Definition eines eigenen Sprachelements für die Realisierung der Operation *ChooseProcessFragment* bietet sich die Verwendung des Beobachtungsprozesses (*MonitoringProcess*) an (siehe Abschnitt 4.2.2). Dies lässt sich dadurch begründen, dass durch einen Beobachtungsprozess spezifisches Verhalten sowohl für die Analyse als auch für darauf basierende Entscheidungen beschrieben werden kann. Anstelle einer Spezifikation

der Operation *ChooseProcessFragment* wird im Folgenden ein Beispiel zur Beschreibung von Bedingungen unter Verwendung eines Beobachtungsprozesses gegeben. Diese Beschreibung ist dabei eng an die bereits in Abschnitt 5.2.2 beschriebene Verwendung eines Beobachtungsprozesses zur Gestaltung des Flexibilitätsaspekts *Flexibility-by Design* angelehnt.

In Abbildung 5-60 ist ein *AC4BPM* mit der Bezeichnung *ACaseToBindA-ProcessFragment* zum Zweck der Auswahl und Bindung eines Prozessfragments dargestellt. Der *AC4BPM* besteht dabei aus einem Beobachtungsprozess (*Monitoring Process*) und zwei Anpassungsprozessen (*Adaptation Process*). Dabei wird durch den dargestellten Beobachtungsprozess eine Auswahl von einer der beiden durch einen Anpassungsprozess beschriebenen Bindungen eines Prozessfragments beschrieben. Das in den Beobachtungs- und Anpassungsprozessen dargestellte Verhalten wird im Folgenden näher beschrieben.

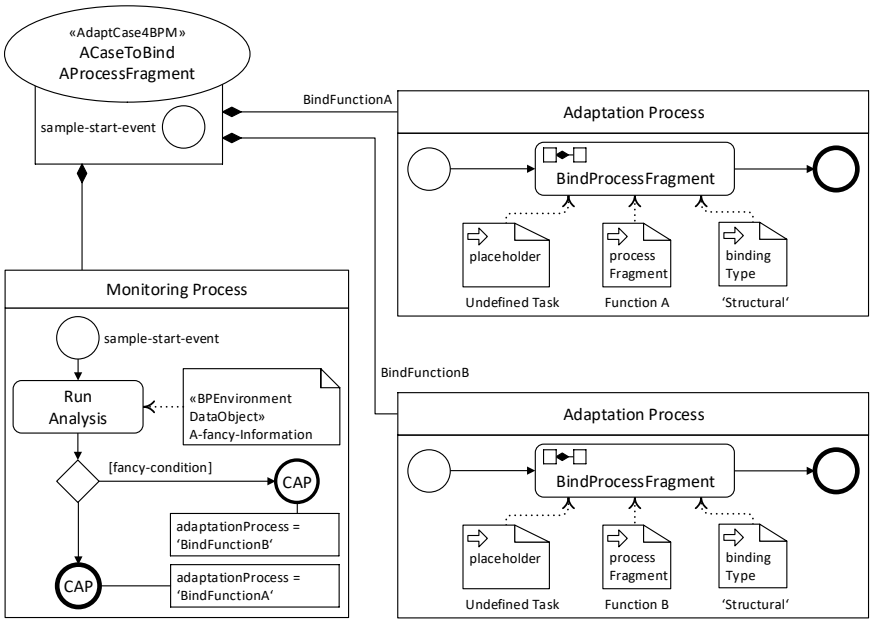


Abbildung 5-60: *Verwendung eines Beobachtungsprozesses zur Gestaltung einer Auswahl eines Prozessfragments*

Der dargestellte Beobachtungsprozess (*Monitoring Process*) enthält das Verhalten zur Analyse und zur Auswahl einer konkreten Bindung eines geeigneten Prozessfragments. Dabei wird der Beobachtungsprozess durch das Aufkommen des dargestellten Startereignisses mit der Bezeichnung *sample-start-event* gestartet. Nachfolgend wird basierend auf Informationen aus dem Kontext des Prozesses, hier dargestellt durch das Datenobjekt mit der Bezeichnung *A-fancy-Information*, eine Analyse durchgeführt. Für eine Beschreibung von Datenobjekten im Rahmen der Sprache

Beobachtungsprozess für die Auswahl eines Prozessfragments

ACML4BPM wird an dieser Stelle auf Abschnitt 4.3.2 verwiesen. Die Analyse ist hier beispielhaft durch den Task mit der Bezeichnung *Run Analysis* dargestellt. Basierend auf den Ergebnissen der Analyse wird nachfolgend die Bedingung *fancy-condition* ausgewertet und je nach Ergebnis entweder der Anpassungsprozess mit der Bezeichnung *BindFunctionB* oder mit der Bezeichnung *BindFunctionA* aufgerufen. Die Ausführung des Beobachtungsprozesses ist mit dem Aufruf eines der beiden genannten Anpassungsprozesse abgeschlossen.

Anpassungsprozess zur Bindung eines Prozessfragments

Im rechten Bereich von Abbildung 5-60 werden zwei durch den zuvor beschriebenen Beobachtungsprozess aufrufbare Anpassungsprozesse dargestellt. Dabei wird im Rahmen des in ihnen enthaltenen Verhaltens die in Abschnitt 5.5.3.1 eingeführte Operation *BindProcessFragment* zur Bindung von Prozessfragmenten wiederverwendet. So beschreiben beide Anpassungsprozesse eine strukturbasierte Bindung (*Structural*) der Prozessfragmente *Function A* bzw. *Function B* in den Platzhalter *Undefined Task*. Eine Bindung der genannten Prozessfragmente ist mit der Beendigung einer der beiden Anpassungsprozesse abgeschlossen.

5.5.3.3 SwitchLCPhase

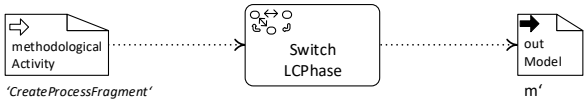
Durch eine Anwendung einer Operation vom Typ *SwitchLCPhase* wird, aus einer methodischen Perspektive heraus, von der Phase *Ausführung* in eine andere Phase des *BPM-Lebenszyklus* gewechselt. Hierdurch ist es möglich, bestimmte methodische Aktivitäten der anderen Phasen auszuführen. In dieser Arbeit werden dabei zwei unterschiedliche methodische Aktivitäten im Rahmen des *BPM-Lebenszyklus* betrachtet, die gemäß der in Abschnitt 5.5.1 gegebenen Beschreibung des Flexibilitätsaspekts *Late Modeling* typischerweise unterstützt werden sollten. Dabei handelt es sich zum einen um die Neugestaltung und zum anderen um die Komposition von Prozessfragmenten. So wird bei einem Wechsel in die Phase *Design & Analyse* eine Neugestaltung von Prozessfragmenten ermöglicht. Die hier im Folgenden näher beschriebene Operation vom Typ *SwitchLCPhase* bietet die Funktionalität der beiden zuvor beschriebenen Typen von Operationen *CreateProcessFragment* und *ComposeProcessFragment*. Die Signatur und konkrete Syntax der Operation *SwitchLCPhase* sind in Abbildung 5-61 angegeben.

Parameter

Die Operation *SwitchLCPhase* erwartet als Eingabe die methodische Aktivität (*methodologicalActivity*), die bei der Anwendung der Operation ausgeführt werden soll. Die Ausgabe einer Anwendung der Operation ist ein Prozessmodell (*outModel*), welches ein neu erstelltes oder komponiertes

	Parametername	Parametertyp
IN :	*methodologicalActivity*	*MethodologicalActivityType*
OUT :	*outModel*	*ProcessModel*

Abbildung 5-61: *Signatur und konkrete Syntax der Operation SwitchLCPhase*



Prozessfragment enthält. Gültige Werte für den Parameter *methodological-Activity* sind durch die in Abbildung 5-55 dargestellte Enumeration vom Typ *MethodologicalActivityType* gegeben. So kann durch das Literal *Create-ProcessFragment* angegeben werden, dass eine Neugestaltung eines Prozessfragments durchgeführt werden soll. Alternativ wird durch das Literal *ComposeProcessFragment* angegeben, dass eine Komposition eines neuen Prozessfragments auf Basis bestehender Prozessfragmente durchgeführt werden soll.

Im Folgenden wird auf die operationale Semantik der Anwendung der Operation *SwitchLCPhase* eingegangen. Zunächst erfolgt eine solche Beschreibung für eine Parametrisierung für die Neugestaltung und anschließend für die Komposition von neuen Prozessfragmenten.

Neugestaltung von Prozessfragmenten Das generelle Funktionsprinzip einer Anwendung der Operation *SwitchLCPhase* mit der Parametrisierung zur Neugestaltung von Prozessfragmenten wird in Abbildung 5-62 dargestellt. Dabei werden im Rahmen der Anwendung der Operation die beiden Teilschritte *SwitchToDaA* und *SwitchToEnactment* durchgeführt, die im Folgenden näher beschrieben werden.

SwitchToDaA

Im Rahmen des ersten Teilschrittes *SwitchToDaA* wird die Neugestaltung von Prozessfragmenten durch einen Wechsel von der Phase *Ausführung* in die Phase *Design & Analyse* ermöglicht. Anschließend können beginnend in der Phase *Design & Analyse* verschiedenste methodische Aktivitäten (*MethodologicalActivity*) ausgeführt werden, die für eine Neugestaltung notwendig sind. Für eine Übersicht über typische Aktivitäten in dieser Phase wird auf die Beschreibung des *BPM-Lebenszyklus* in Abschnitt 2.2.2 verwiesen. Zur besseren Veranschaulichung des Funktionsprinzips ist hier ein Prozess mit einer exemplarischen Auswahl von methodischen Aktivitäten dargestellt. So wird hier stellvertretend ein Prozess dargestellt, der methodische Aktivitäten sowohl aus der Phase *Design & Analyse* als auch aus der Phase *Konfiguration* einschließt. Auf eine Beschreibung dieser Aktivitäten wird nachfolgend Bezug genommen.

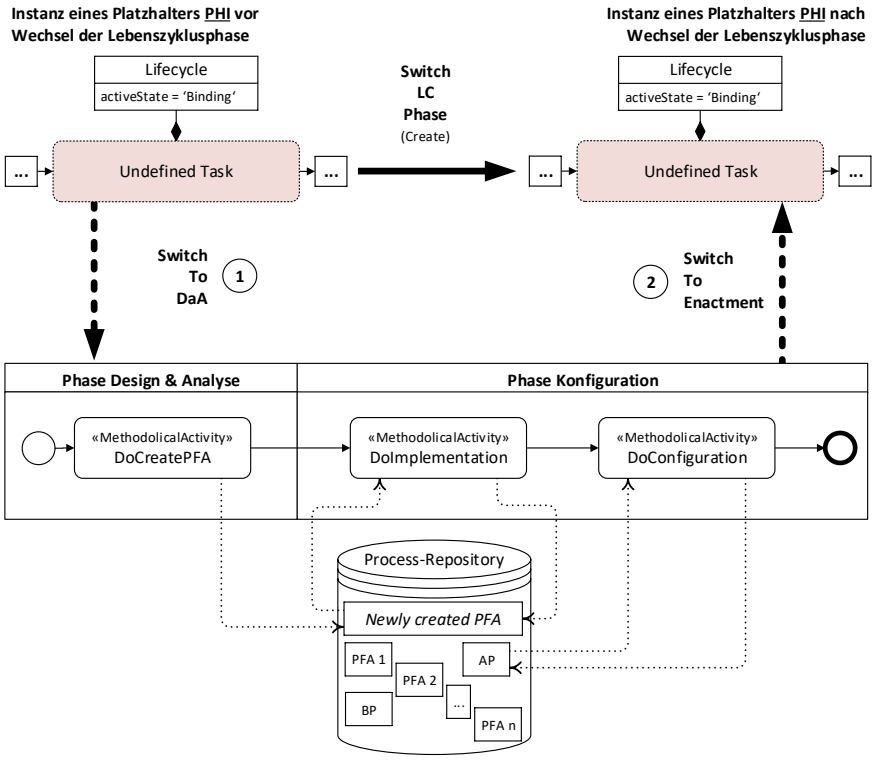


Abbildung 5-62: Beispielhafte Anwendung der Operation SwitchLCPhase (Create)

Das Erstellen eines neuen Prozessfragments wird durch den Task *DoCreatePFA* dargestellt. Ein Prozessfragment wird typischerweise im Anschluss an seine Erstellung in einem Datenspeicher (*Process-Repository*) zusammen mit zuvor erstellten Prozessfragmenten (*PFA 1, PFA 2, ..., PFA n*) abgelegt. Dabei kann ein solcher Datenspeicher aber auch, wie hier dargestellt, für eine Ablage aller Prozesse einschließlich der Beobachtungs- (*BP*) und Anpassungsprozesse (*AP*) verwendet werden. Auf Basis dieser im Datenspeicher abgelegten Prozessmodelle ist eine Ausführungsumgebung in der Lage, zugehörige Instanzen zu bilden (siehe Abschnitt 2.2).

Sollen neu erstellte Prozessfragmente auch für zukünftige Bindungen zur Verfügung stehen, so können weitere methodische Aktivitäten notwendig sein. Daher wird hier im Rahmen der Phase *Konfiguration* beispielhaft die methodische Aktivität *DoImplementation* aufgeführt. *DoImplementation* umfasst dabei Tätigkeiten hinsichtlich benötigter Implementierungen des Prozessfragments, sodass es in zukünftigen Bindungen verwendet werden kann. Ferner ist für die Verwendung im Rahmen einer Bindung auch die Anpassung von Beobachtungs- und Anpassungsprozessen notwendig. Dies lässt sich damit begründen, dass derartige bestehende Prozesse entsprechend der Verfügbarkeit des neu erstellten Prozessfragments erweitert bzw. selbst angepasst werden müssen. Daher ist in Abbil-

dung 5-62 die weitere methodische Aktivität *DoConfiguration* aufgeführt, in der eine Konfiguration von bestehenden Prozessen – hier als Beispiel ein Anpassungsprozess – vorgenommen werden kann.

SwitchToEnactment

Ist der hier dargestellte stellvertretende Prozess zur Neugestaltung eines Prozessfragments nach der Aktivität *DoConfiguration* abgeschlossen, wird der zweite Teilschritt *SwitchToEnactment* durchgeführt. Die Anwendung der Operation *SwitchLCPhase* ist damit abgeschlossen. Man beachte, dass hinsichtlich der Instanz *PHI* keine Anpassung vorgenommen wurde, wie sie z.B. an ihrem aktuellen Zustand des Lebenszyklus möglich wäre. Dies lässt sich damit begründen, dass eine Anwendung der Operation *SwitchLCPhase* unter der aktuell betrachteten Parametrisierung lediglich ein neues Prozessfragment sowohl erstellt als auch die zugehörige Implementierung und Konfiguration vornimmt. Eine Bindung oder Ausführung des neu zur Verfügung gestellten Verhaltens kann im Anschluss durchgeführt werden.

Komposition von Prozessfragmenten Das generelle Funktionsprinzip einer Anwendung der Operation *SwitchLCPhase* mit der Parametrisierung zur Komposition von Prozessfragmenten wird in Abbildung 5-64 dargestellt. Dabei werden im Rahmen der Anwendung der Operation die beiden Teilschritte *SwitchToConfiguration* und *SwitchToEnactment* durchgeführt, die im Folgenden näher beschrieben werden.

SwitchToConfiguration

Im Rahmen des ersten Teilschrittes *SwitchToConfiguration* wird die Komposition eines Prozessfragments auf Basis bestehender Prozessfragmente ermöglicht. Hierzu ist zuvor ein Wechsel von der Phase *Ausführung* in die Phase *Konfiguration* notwendig. Anschließend können beginnend in der Phase *Konfiguration* verschiedenste methodische Aktivitäten (*MethodologicalActivity*) ausgeführt werden, die für eine Komposition notwendig sind. Für eine Übersicht von typischen Aktivitäten in dieser Phase wird auf die Beschreibung des *BPM-Lebenszyklus* in Abschnitt 2.2.2 verwiesen. Zur besseren Veranschaulichung des Funktionsprinzips ist hier ein Prozess mit einer exemplarischen Auswahl von methodischen Aktivitäten dargestellt. Auf eine Beschreibung dieser Aktivitäten wird nachfolgend Bezug genommen.

So wird die Komposition eines neuen Prozessfragments durch den Task *DoComposePFA* dargestellt. Als Eingabe dienen dabei bestehende Prozessfragmente (*PFA 1* und *PFA 2*), die bereits im Datenspeicher (*Process-Repository*) abgelegt waren. Für eine Verwendung des neu komponierten Prozessfragments kann, wie hier dargestellt, ebenfalls die Ausführung

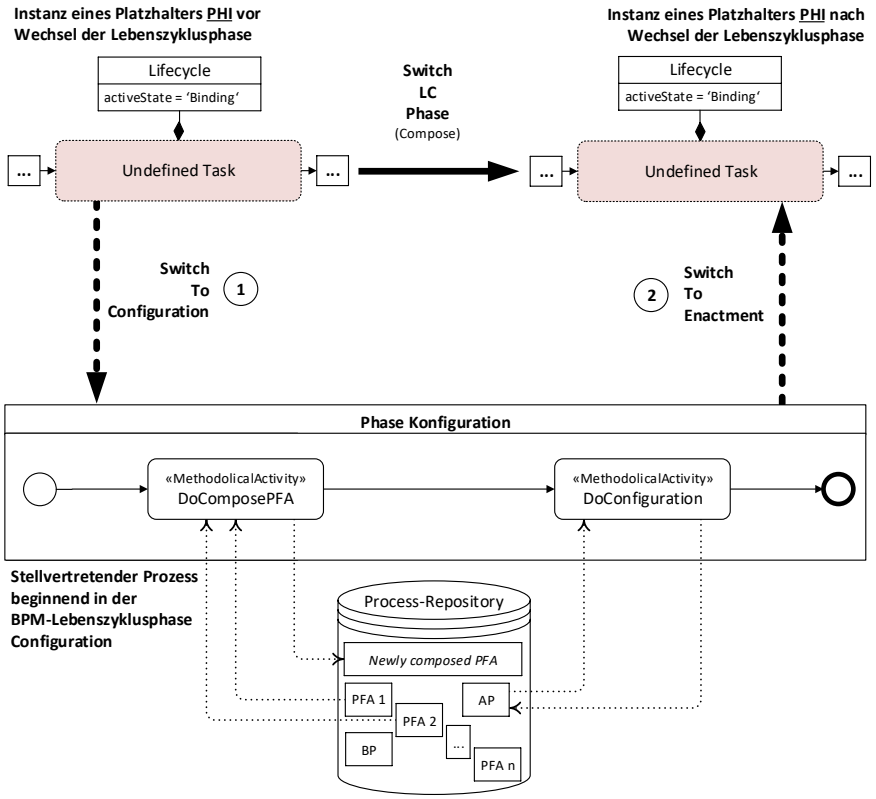


Abbildung 5-63: Beispielhafte Anwendung der Operation Switch-LCPhase (Compose)

weiterer methodischer Aktivitäten notwendig sein. So wird hier beispielhaft, wie zuvor bei der Parametrisierung für die Neugestaltung, ebenfalls die methodische Aktivität *DoConfiguration* mit aufgeführt.

SwitchToEnactment

Ist der hier dargestellte stellvertretende Prozess zur Komposition eines Prozessfragments nach der Aktivität *DoConfiguration* abgeschlossen, wird der zweite Teilschritt *SwitchToEnactment* durchgeführt. Die Anwendung der Operation *SwitchLCPhase* ist damit abgeschlossen. Man beachte, dass auch hier hinsichtlich der Instanz *PHI* keine Anpassung vorgenommen wurde. Dies lässt sich damit begründen, dass eine Anwendung der Operation *SwitchLCPhase* unter der aktuell betrachteten Parametrisierung lediglich ein neues Prozessfragment aus bestehenden Prozessfragmenten komponiert und die zugehörige Konfiguration vornimmt. Eine Bindung oder Ausführung des neu zur Verfügung gestellten Verhaltens kann im Anschluss durchgeführt werden.

Beispiel für eine Komposition eines Prozessfragments

Eine Komposition eines Prozessfragments auf Basis bestehender Prozessfragmente kann auch als eine Form einer Anpassung eines leeren Prozessmodells verstanden werden. Daher wird nachfolgend ein detailliertes Beispiel für eine Komposition unter Verwendung der in Abschnitt 4.3.3 vor-

gestellten Operationen gegeben (siehe Abbildung 5-64). Das Beispiel zeigt die Komposition des Prozessfragments *Newly composed PFA* auf Basis der beiden Prozessfragmente *Function A* und *Function B*. Dabei handelt es sich um eine einfache Sequenzierung, sodass zunächst das Verhalten des Prozessfragments *Function A* und anschließend das Verhalten des Prozessfragments *Function B* vorkommen soll. Die hier dargestellte Komposition besteht aus insgesamt vier Schritten, in denen eine Reihe von Operationen zur Anpassung (siehe Abschnitt 4.3.3) verwendet werden. Die Operationen sind hier in ihrer textuellen Notation dargestellt. Die vier Schritte *PopulateModel*, *Connect*, *ClearUp* und *AdaptToContext* werden nachfolgend detailliert beschrieben.

Im ersten Schritt *PopulateModel* werden die Elemente aus den Prozessfragmenten *Function A* und *Function B* in das Prozessfragment *Newly Composed PFA* verschoben. Für diesen Vorgang werden zwei Anwendungen der Operation *ModifyPositionOfNodesInContainer* unter der Parametrisierung des Kopierens verwendet (siehe Anhang A.3.5). Eine Anpassung der Prozessfragmente *Function A* und *Function B* wird im weiteren Verlauf somit nicht vorgenommen. Im zweiten Schritt *Connect* werden die kopierten Elemente innerhalb des Prozessfragments *Newly composed PFA* miteinander verbunden, sodass ein zusammenhängender Kontrollfluss entsteht. Für diesen Vorgang wird eine Anwendung der Operation *AddEdge* verwendet (siehe Anhang A.2.1). Dabei wird so vorgegangen, dass ein neues Kantenelement (*e3*) vom Typ *SequenceFlow* hinzugefügt wird. Das Kantenelement *e3* wird derartig mit bestehenden Elementen verbunden, dass es ausgehend vom Knotenelement *LastNode* hin zum Knotenelement *FirstNode* verläuft. Durch das Hinzufügen des Kantenelements *e3* werden die beiden ursprünglich getrennten Kontrollflüsse miteinander verbunden. Die bestehenden Start- und Endsymbole mit den Bezeichnungen *StartSymbolB* und *EndSymbolA* sowie ihre ein- bzw. ausgehenden Kantenelemente *e1* und *e2* werden nun nicht mehr benötigt. Daher werden sie im dritten Schritt *ClearUp* durch zwei Anwendungen der Operation *RemoveNode* entfernt (siehe Anhang A.1.2). Hier kann die Funktionsweise der Operation *RemoveNode* zur Vermeidung weiterer Anwendungen von Operationen zur Anpassung genutzt werden, da auch die nicht mehr benötigten ein- und ausgehenden Kantenelemente *e1* und *e2* entfernt werden. Im letzten Schritt *AdaptToContext* werden im Bedarfsfall Eigenschaften von Elementen an ihren neuen Kontext angepasst. In dem hier dargestellten Beispiel wird daher die Benennung des Start- bzw. des Endsymbols des Prozessfragments in Anlehnung an seinen Namen hin zu *StartSymbolNcPFA* bzw. *EndSymbolNcPFA* angepasst.

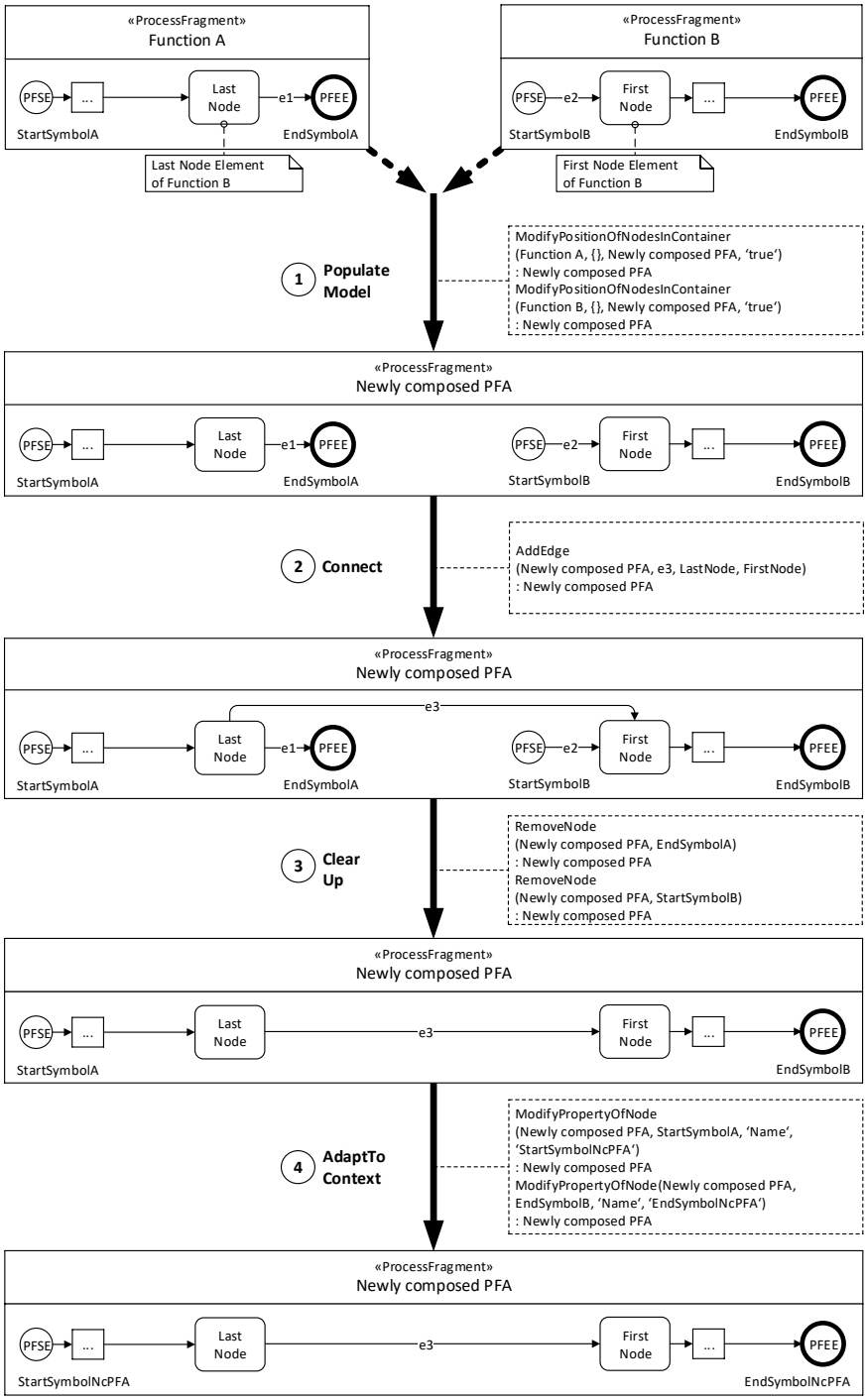


Abbildung 5-64: Beispiel für die Komposition eines Prozessfragments

Die zuvor beschriebene Komposition zur Erstellung des Prozessfragments *Newly composed PFA* stellt nur ein mögliches Beispiel für die Komposition neuer Prozessfragmente auf Basis bestehender Prozessfragmente dar. So lassen sich nicht nur einfache Sequenzierungen beschreiben; stattdessen sind auch komplexere Kompositionen möglich, bei denen z.B. auch neue Elemente hinzugefügt oder bestehende entfernt werden können.

5.5.4 Zusammenfassung

In diesem Abschnitt wurde das vierte und letzte Entwurfsmuster in Anlehnung an den Flexibilitätsaspekt *Flexibility-by Underspecification* vorgestellt. Dabei wurden zunächst die beiden Untertypen *Late Selection* und *Late Modeling* detailliert vorgestellt. Im weiteren Verlauf des Abschnittes wurde eine Spracherweiterung zur Gestaltung von Prozessfragmenten und zur Unterstützung des Flexibilitätsaspekts beschrieben. Bei der Spracherweiterung wird auf bereits vorgestellte Konzepte zur Beschreibung von Task- und Prozessinstanzen zurückgegriffen (siehe Abschnitt 5.3.3). Unter Verwendung dieser Spracherweiterung konnten anschließend Operationen beschrieben werden, die die wesentlichen Funktionsprinzipien von *Late Selection* und *Late Modeling* umsetzen. Der in diesem Abschnitt vorgestellte Gesamtansatz bietet hierdurch die Möglichkeit einer erweiterten Unterstützung in der Gestaltung von flexiblen und anpassbaren Prozessen.

5.6 Zusammenfassung

In den vorherigen Abschnitten wurden verschiedene Flexibilitätsaspekte aus der Domäne *BPM* vorgestellt. Jeder dieser Flexibilitätsaspekte wurde dabei zunächst analysiert. In einem weiteren Schritt wurde eine mögliche Unterstützung bei der Gestaltung von flexiblen und anpassbaren Prozessen durch die Sprache *ACML4BPM* beschrieben. Für diese Unterstützung waren je nach Flexibilitätsaspekt verschiedene Erweiterungen notwendig. So wurde bspw. eine methodisch neuartige Gestaltung für den Flexibilitätsaspekt *Flexibility-by Design* (siehe Abschnitt 5.2) vorgestellt. Für weitere Aspekte von Flexibilität, wie etwa *Flexibility-by Change*, *Flexibility-by Deviation* und *Flexibility-by Underspecification*, wurden Konzepte zur Gestaltung unter Verwendung der Sprache *ACML4BPM* beschrieben. Als Resultat in Form der Gesamtheit dieser Konzepte stehen Entwurfsmuster für flexible und anpassbare Prozesse zur Verfügung, die für die Entwicklung

von prozesszentrischen Softwaresystemen als eine Art *Best-Practice* eingesetzt werden können. In diesem Abschnitt wird nachfolgend auf die in Abschnitt 5.1 eingeführten Fragestellungen eingegangen. Hierzu ist in Tabelle 5-4 eine Auflistung von Zielen sowie deren Erfüllung in Anlehnung an diese Fragestellungen dargestellt.

Tabelle 5-4: Übersicht über gesetzte Ziele und deren Erfüllung für die Musterbasierte Unterstützung in der Gestaltung von flexiblen und anpassbaren Prozessen

Fragestellung	Ziel	Erfüllung
1	Identifikation von relevanten Flexibilitätsaspekten in Prozessen der Domäne *BPM*	✓
2	Erarbeitung von Entwurfsmustern zur Beschreibung von Flexibilitätsaspekten unter Verwendung der Sprachen *ACML4BPM* und *BPMN2.0*	✓
3	Beschreibung von Spracherweiterungen zur Unterstützung bei der Gestaltung von Flexibilitätsaspekten unter Verwendung der Sprache *ACML4BPM*	✓
4	Beschreibung von Erweiterungen von Methoden zur Gestaltung	✓

Identifikation von Flexibilitätsaspekten

In der Domäne *BPM* existieren zahlreiche Arbeiten, die sich mit Flexibilität von Prozessen beschäftigen. Dabei wurde für diese Arbeit initial die Anforderung gesetzt, nicht lediglich einen einzelnen Flexibilitätsaspekt zu unterstützen. Stattdessen sollte eine Menge von klassischen Typen von Flexibilitätsaspekten unterstützt werden, sodass die daraus folgende Menge von Entwurfsmustern als Beispiel für *Best-Practice* verwendet werden kann (siehe Abschnitt 1.3). Hierbei bot sich die Anlehnung an verschiedene Taxonomien für Typen von Flexibilität in Prozessen an, wie sie z.B. durch [Sch+08], [RW12] oder [RSS06] gegeben sind. Aufgrund einer hohen Übereinstimmung gegebener Taxonomien wurde sich für eine Arbeit [Sch+08] entschieden, die auf einem vergleichbaren Abstraktionsniveau wie die vorliegende Lösung beschrieben worden ist.

Entwurfsmuster

Die in den vorherigen Abschnitten vorgestellten Entwurfsmuster von Flexibilitätsaspekten basieren auf *Schonenberg et. al* [Sch+08]. So werden in der vorliegenden Arbeit insgesamt vier Entwurfsmuster vorgestellt, die hinsichtlich *Schonenberg et. al* eine abgeschlossene Menge von Flexibilitätsaspekten darstellen. Durch das zuerst in Abschnitt 5.2 vorgestellte Entwurfsmuster *Flexibility-by Design* ist die Gestaltung von Prozessen unmittelbar selbst betroffen. So wird die Gestaltung der drei Aspekte *Choice*, *Iteration* und *Cancellation* unter Verwendung der Sprache *ACML4BPM* (siehe Kapitel 4) beschrieben. Anschließend folgt die Vorstellung des Entwurfsmusters für den Flexibilitätsaspekt *Flexibility-by Change* in Abschnitt 5.3. Bei diesem Entwurfsmuster wird die Anpassung von Prozes-

sen hinsichtlich ihrer Modelle und deren Instanzen fokussiert. Dabei wurden diverse Strategien für die Migration von Anpassungen an Modellen hin zu deren Instanzen vorgestellt. Das zugehörige Entwurfsmuster enthält in diesem Bezug eine Reihe von Operationen sowie eine konzeptionelle Erweiterung der Sprache *BPMN2.0*. Das Entwurfsmuster des Flexibilitätsaspekts *Flexibility-by Deviation* wird in Abschnitt 5.4 beschrieben. Es greift auf die konzeptionelle Erweiterung der Sprache *BPMN2.0* für das Entwurfsmuster des Flexibilitätsaspekts *Flexibility-by Change* zurück. Ferner werden spezifische Operationen vorgestellt, mit denen typische Verfahrensweisen des Flexibilitätsaspekts für die Abweichung von vordefinierten Kontrollflüssen abgebildet werden können. Das letzte Entwurfsmuster für den Flexibilitätsaspekt *Flexibility-by Underspecification* (siehe Abschnitt 5.5) stellt eine in der Literatur häufig vorkommende Art von Flexibilität dar. Die Beschreibung des Entwurfsmusters enthält neben einer weiteren konzeptionellen Erweiterung der Sprache *BPMN2.0* auch diverse Operationen zur Unterstützung des Flexibilitätsaspekts.

Spracherweiterungen

Für die in diesem Kapitel vorgestellten Entwurfsmuster waren diverse Erweiterungen der Sprache *ACML4BPM* notwendig. Beispiele hierfür waren die Entwurfsmuster der Flexibilitätsaspekte *Flexibility-by Change*, *Flexibility-by Deviation* und *Flexibility-by Underspecification*. Derartige Erweiterung sind gegeben durch Sprachelemente zur Beschreibung von grundlegenden Konzepten eines Flexibilitätsaspekts. Aufbauend konnten jeweils Operationen eingeführt werden, die typische Verfahrensweisen eines jeden betrachteten Flexibilitätsaspekt umsetzen.

Erweiterung von Methoden

Für die Gestaltung von flexiblen und anpassbaren Prozessen sind geeignete Methoden notwendig, die sich in Teilen auch auf die Verwendung eines Flexibilitätsaspekts beziehen. Beispiele hierfür sind durch *Late Modeling* oder durch *Flexibility-by Design* gegeben. In den vorgestellten Lösungen wird je nach Notwendigkeit auch Bezug auf methodische Besonderheiten bei der Gestaltung von Flexibilitätsaspekten genommen. Eine Verortung von grundlegenden methodischen Aktivitäten in einer domänenspezifischen Methode wird in einem weiteren Lösungsteil (siehe Kapitel 6) vorgenommen.

Kapitel **6**

Adaptivity Engineering für flexible und anpassbare Prozesse

Die Gestaltung von flexiblen und anpassbaren Prozessen sollte neben einer adäquaten Sprache (siehe Kapitel 4) sowie zugehörigen Entwurfsmustern (siehe Kapitel 5) auch durch einen geeigneten methodischen Rahmen unterstützt werden. Dieses Kapitel stellt daher eine domänenspezifische Erweiterung des *Adaptivity Engineering* in Form der Methode *Adapt Cases 4 BPM* vor. Hierdurch soll die angedachte Verwendung von bisher vorgestellten Lösungsteilen in Form der Sprache *ACML4BPM* sowie von den Entwurfsmustern verdeutlicht werden. Eine Übersicht über die nachfolgenden Inhalte ist in Abbildung 6-1 dargestellt.

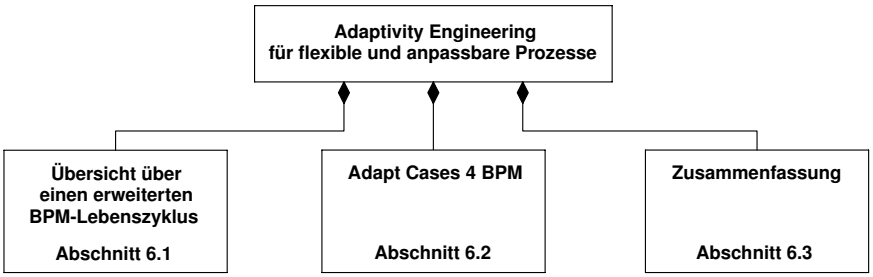


Abbildung 6-1: Übersicht über das Adaptivity Engineering für flexible und anpassbare Prozesse

Im Folgenden wird auf die in Abbildung 6-1 abgebildeten Inhalte des *Adaptivity Engineering* für flexible und anpassbare Prozesse eingegangen. Da im *BPM* typischerweise Modelle des Lebenszyklus von Prozessen (siehe auch Abschnitt 2.2.2) eingesetzt werden, wird die Methode *Adapt Cases 4 BPM* als eine Erweiterung eines *BPM-Lebenszyklus* beschrieben. Eine Übersicht über die Einbettung der Methode *Adapt Cases 4 BPM* in diesen *BPM-Lebenszyklus* wird zunächst in Abschnitt 6.1 gegeben. Im An-

schluss folgt in Abschnitt 6.2 eine detaillierte Beschreibung relevanter Aktivitäten und Artefakte sowie deren Relation entlang eines domänenspezifischen methodischen Rahmens in Form der Methode *Adapt Cases 4 BPM*. Das Kapitel schließt in Abschnitt 6.3 mit einer kritischen Zusammenfassung der in diesem Kapitel vorgestellten Inhalte ab.

6.1 Übersicht über einen erweiterten BPM-Lebenszyklus

In diesem Abschnitt wird die Integration von relevanten Aktivitäten und Artefakten des *Adaptivity Engineering* für flexible und anpassbare Prozesse in den *BPM-Lebenszyklus* nach *Weske* [Wes12] beschrieben. Die Darstellung eines erweiterten *BPM-Lebenszyklus* ist in Abbildung 6-2 dargestellt.

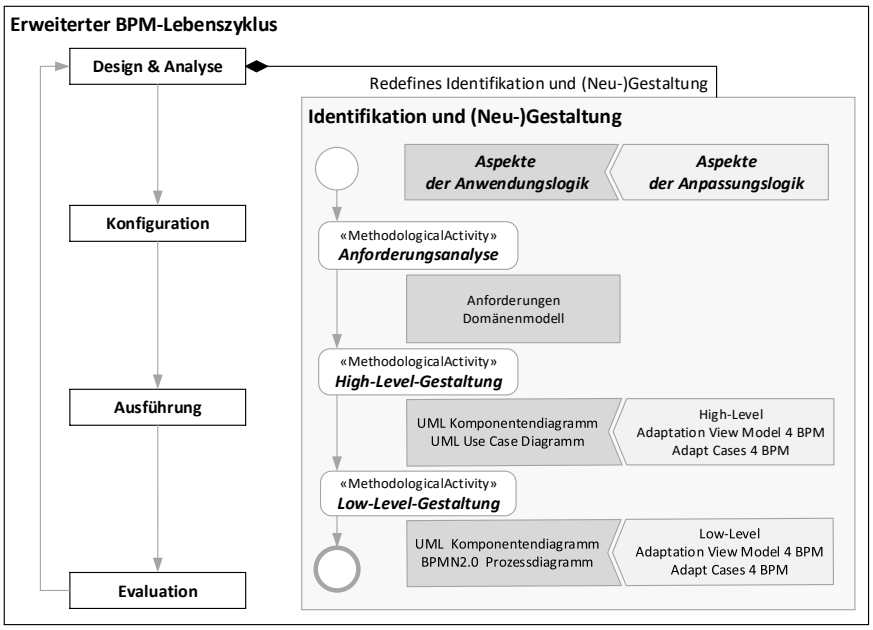


Abbildung 6-2: Schematische Darstellung des erweiterten BPM-Lebenszyklus

Im linken Bereich sind die bereits in Abschnitt 2.2.2 vorgestellten Phasen *Design & Analyse*, *Konfiguration*, *Ausführung* sowie *Evaluation* des *BPM-Lebenszyklus* dargestellt. Da durch diese Arbeit die Gestaltung von Prozessen fokussiert worden ist, betrifft die im rechten Bereich dargestellte Erweiterung die Phase *Design & Analyse*. Das *Adaptivity Engineering* kann dabei auch für weitere Phasen relevant sein, da erstellte Artefakte auch Einfluss auf weitere methodische Aktivitäten und auf deren übliche Artefakte haben können. Als Beispiel kann die Fragestellung hinsichtlich ei-

ner gemeinsamen Ausführung von Aspekten der Anpassungs- und Anwendungslogik eingeführt werden. So könnte die Anforderung bestehen, das *Separation-of-Concerns* nicht nur in der Gestaltung, sondern auch in der Ausführung durchzuführen. Aufgrund des gesetzten Fokus auf die Gestaltung von Prozessen wurden derartige Fragestellungen hinsichtlich weiterer Phasen des *BPM-Lebenszyklus* nicht behandelt. Sie stellen dabei jedoch die Basis für die zukünftige Forschung an entsprechenden Lösungen (siehe Abschnitt 8.2) dar. Dennoch wird auf unterschiedliche Zwecke für die Verwendung der Sprache *ACML4BPM* und der zugehörigen Entwurfsmuster in Form einer Ergänzung in Abschnitt 6.2.4 näher eingegangen.

Die dargestellte Erweiterung stellt ein mögliches Beispiel für eine methodische Integration notwendiger Aktivitäten und Artefakte des *Adaptivity Engineering* dar. So können je nach Anforderungen an einen methodischen Rahmen auch Alternativen, wie z.B. der Lebenszyklus nach *Dumas* [Dum+18], gewählt werden. Dabei ist jedoch jeweils individuell zu prüfen, wie sich spezifische methodische Aktivitäten übertragen lassen. So ist es z.B. möglich, dass sich die drei dargestellten Aktivitäten auf mehrere Phasen eines anderen Lebenszyklus verteilen. Es wird angenommen, dass eine Übertragbarkeit aufgrund generischer Eigenschaften der nachfolgenden Beschreibung und Darstellung von Abhängigkeiten jedoch realisierbar ist.

Bei der Integration von relevanten Aktivitäten und deren Artefakten des *Adaptivity Engineering* in den *BPM-Lebenszyklus* wurde sich für eine Redefinition der Aktivität *Identifikation und (Neu-)Gestaltung* entschieden. Die Integration folgt in weiten Teilen dem durch *Luckey* [Luc13] beschriebenen Vorgehen (siehe auch Abschnitt 2.4.4) und wurde an spezifischen Stellen hinsichtlich üblicher Artefakte des *BPM* angepasst. Insgesamt umfasst dieses Vorgehen die drei methodischen Aktivitäten *Anforderungsanalyse*, *High-Level-Gestaltung* und *Low-Level-Gestaltung*.

Im Rahmen jeder Aktivität werden Artefakte erzeugt, die in der Gestaltung von flexiblen und anpassbaren Prozessen benötigt werden. Das *Adaptivity Engineering* fokussiert dabei die Trennung von Aspekten der Anpassungslogik und Aspekten der Anwendungslogik (*Separation-of-Concerns*). Die jeweils zugehörigen Artefakte werden in den Farben *Dunkelgrau* hinsichtlich der Aspekte der Anwendungslogik und in *Hellgrau* hinsichtlich der Aspekte der Anpassungslogik dargestellt.

Nachfolgend wird in Abschnitt 6.2 die Methode *Adapt Case 4 BPM* in Form einer Detaillierung der Redefinition der Aktivität *Identifikation und (Neu-)Gestaltung* beschrieben.

6.2 Adapt Cases 4 BPM

Die Durchführung eines *Adaptivity Engineering* für flexible und anpassbare Prozesse ist durch die Methode *Adapt Cases 4 BPM* möglich. Dabei liegt der Fokus dieser Methode auf der getrennten Gestaltung von Aspekten der Anpassungslogik und Aspekten der Anwendungslogik. Hierzu wurde zuvor in Abschnitt 6.1 die Integration von spezifischen Aktivitäten und Artefakten eingeführt. Nachfolgend wird eine Detaillierung der Aktivität *Identifikation und (Neu-)Gestaltung* in Anlehnung an Abbildung 6-3 beschrieben.

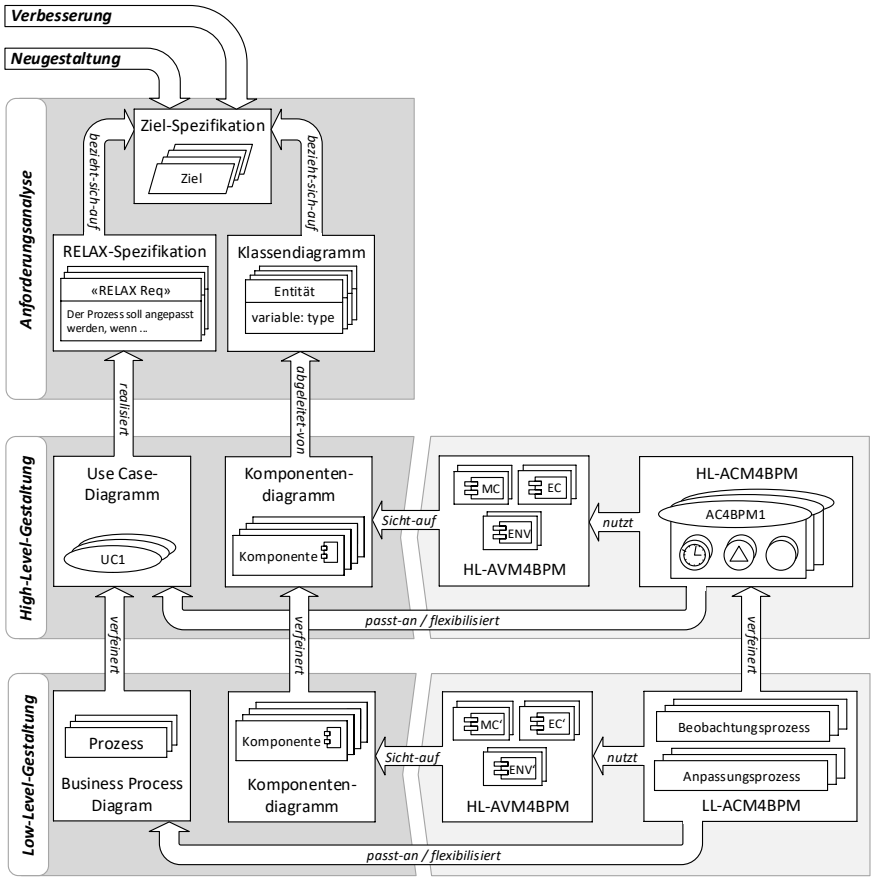


Abbildung 6-3: Detaillierung der Aktivität Identifikation und (Neu-)Gestaltung des Adaptivity Engineering

Im weiteren Fokus liegen dabei die in Abbildung 6-3 dargestellten Abhängigkeiten zwischen spezifischen Artefakten der Aktivitäten *Anforderungsanalyse*, *High-Level-Gestaltung* und *Low-Level-Gestaltung*. So wird nachfolgend zunächst auf die *Anforderungsanalyse* in Abschnitt 6.2.1 eingegangen. Es folgt die Beschreibung der *High-Level-Gestaltung* in Abschnitt 6.2.2 und der *Low-Level-Gestaltung* in Abschnitt 6.2.3. Dabei können je nach Anfor-

derungen an das methodische Vorgehen und je nach gewählten Techniken zur Gestaltung auch andere Artefakte vorkommen. Die Beschreibungen sind daher als ein exemplarischer Verlauf zu betrachten.

6.2.1 Anforderungsanalyse

Die erste Aktivität der Methode *Adapt Cases 4 BPM* in Form der Anforderungsanalyse wird klassischerweise durch zwei unterschiedliche Treiber angestoßen. Da es sich um einen Lebenszyklus von Prozessen handelt, ist es möglich, dass es sich um die erste Iteration oder um eine der nachfolgenden Iterationen handelt. Man spricht hierbei von einer *Neugestaltung* oder von einer *Verbesserung* von Prozessen.

Die Neugestaltung adressiert sowohl existierende – aber bisher nicht durch das *BPM* erfasste – als auch zukünftige Prozesse. So können in Organisationen eine Vielzahl von gelebten oder kulturell belegten Prozessen existieren, deren systematische Dokumentation oder IT-Unterstützung bisher nicht im Fokus eines praktisch durchgeführten *BPM* lagen. Sollen derartige Prozesse erstmalig durch das *BPM* verwaltet werden, handelt es sich um die erste Iteration ihres Lebenszyklus. Beide Arten von Prozessen werden folglich neu gestaltet.

Bereits existierende Prozesse unterliegen in Organisationen einer kontinuierlichen Verbesserung, in der der Prozess iterativ verbessert wird. Man spricht bei diesem Vorgehen auch von einem *kontinuierlichen Verbesserungsprozess (KVP)*. Eine Iteration eines Lebenszyklus stellt dabei einen natürlichen Ablauf zur Verbesserung von Prozessen dar. Dabei können Verbesserungen eines Prozesses durch geänderte organisationale Ziele bewirkt werden oder auf Analysen auf Basis von Prozesshistorien beruhen. Die Identifikation von geänderten Zielen oder die durchgeführten Analysen, wie z.B. das *Process Mining* oder das *Conformance Checking* [Aal16], können als Teil der Phase *Evaluation* verstanden werden. Die Artefakte der Phase *Evaluation* können anschließend in einem nachfolgenden Durchlauf des *BPM-Lebenszyklus* verwendet werden.

In beiden Varianten von Treibern steht das Ziel auf der strategischen Ebene der Betrachtung im Vordergrund. Ziele können dazu verwendet werden, um in einer frühen Phase der Gestaltung die grundlegende Motivation für Prozesse auf einer strategischen Ebene zu beschreiben (siehe Abschnitt 2.2.1). Je nach organisationaler Einbettung können Ziele aus unterschiedlichen Arbeitseinheiten stammen. Dabei entsteht häufig die Herausforderung für die Durchführung eines Zielabgleichs zwischen unter-

schiedlichen Arbeitseinheiten. Hier können Konflikte entstehen, die sich methodisch durch ein geeignetes *Business IT-Alignment* [GTG15] auflösen lassen. Arbeiten, die sich bspw. mit der zielorientierten Gestaltung von Prozessen auseinandergesetzt haben und im Rahmen des *Business IT-Alignment* eingesetzt werden können, wurden durch [DP10; Poe+13; Nag15] vorgestellt. Wurden Ziele abgeglichen, können sie zudem als Grundlage für die Beschreibung von natürlichsprachlichen Anforderungen und von Domänenmodellen eingesetzt werden.

Anforderungen in natürlicher Sprache werden heutzutage in vielen unterschiedlichen Methoden eingesetzt. Dabei birgt der Einsatz von natürlicher Sprache oftmals die Gefahr von Fehlinterpretationen oder einer zu umfangreichen Größe der Anforderungsspezifikation. Hier können Methoden und Techniken eingesetzt werden, die die Kontrolle hinsichtlich der eingesetzten Sprache fokussieren. Eine solche Sprache kann dann auch als *kontrollierte natürliche Sprache* (engl. *CNL*) bezeichnet werden. Eine *CNL* stellt dabei eine Untermenge der natürlichen Sprache dar, die sich durch Restriktionen und durch die Einführung fester Begrifflichkeiten auszeichnet. Hierdurch können derartige Sprachen auch maschinenlesbar sein, sodass sie vorteilhaft in der modellgetriebenen Entwicklung im Rahmen einer Automatisierung eingesetzt werden können.

Beispiele für kontrollierte Sprachen sind in der Literatur durch *ACE* [FKK08] oder *CPL* [Cla+09] gegeben. Dabei ist anzumerken, dass das *Adaptivity Engineering* insbesondere solche Funktionen fokussiert, die die Anpassungsfähigkeit eines Systems betreffen. Für die Gestaltung von Anforderungen derartiger Funktionen sind Elemente einer Sprache notwendig, die eine gewisse Unsicherheit (engl. *uncertainty*) berücksichtigen. Eine derartige Sprache wird auch bereits für den vorherigen Stand des *Adaptivity Engineering* durch *Luckey* [Luc13] empfohlen und ist durch *RELAX* [Whi+09] gegeben.

Eine erste Gestaltung von wesentlichen Konzepten und Eigenschaften von Prozessen kann durch Domänenmodelle beschrieben werden. Domänenmodelle enthalten relevante Konzepte bzw. Begrifflichkeiten sowie Eigenschaften und Relationen zueinander. Hierfür können je nach späterer Weiterverwendung unterschiedliche Techniken eingesetzt werden. Beispiele sind durch geordnete natürlichsprachliche Texte in Form von Glossaren oder durch Diagramme in Form von *UML Klassendiagrammen* oder *Mind-Maps* gegeben. Domänenmodelle können anschließend die Basis für die Ableitung von ersten strukturellen Informationen über das System und seine Umgebung bilden.

6.2.2 High-Level-Gestaltung

Auf Basis der zuvor erstellten Anforderungen und Domänenmodelle kann in dem nachfolgenden Schritt der *High-Level-Gestaltung* eine frühe Version sowohl vom Verhalten als auch von der Struktur des Systems und seiner Umgebung beschrieben werden. Dabei unterscheidet das *Adaptivity Engineering* klassischerweise zwischen den Beschreibungen von Aspekten der Anpassungs- und der Anwendungslogik. Ein Beispiel für den Schritt der *High-Level-Gestaltung* wird in Abschnitt 7.1 im Zuge der Evaluation durch das beschriebene Szenario gegeben.

Aspekte der Anwendung

Im Rahmen der *High-Level-Gestaltung* wird das Verhalten der Anwendungslogik eines Systems und seiner Umgebung in Form von *UML Use Cases* beschrieben. Jeder gestaltete *Use Case* stellt dabei eine Funktion dar, durch die eine (Teil-)Anforderung an das Verhalten erfüllt bzw. ein (Teil-)Ziel erreicht werden kann. Dementsprechend kann es notwendig sein, eine Reihe von Funktionen zu gestalten, bei der erst deren Gesamtheit eine Anforderung erfüllt.

In der Gestaltung von Funktionen können dabei bereits Eigenschaften berücksichtigt werden, die in einer ersten Strukturierung des Systems vorkommen. Hierzu können z.B. *UML Komponentendiagramme* verwendet werden. Dabei werden erste Eigenschaften berücksichtigt, die auch bereits in der Beschreibung der Domänenmodelle vorgekommen sind. Durch Komponenten kann darüber hinaus auch die Architektur des Systems und seiner Umgebung beschrieben werden. Oftmals werden dabei schon in diesem Schritt vorerst einzelne Funktionen auf Komponenten verteilt.

Aspekte der Anpassung

In enger Anlehnung an das bisher beschriebene Verhalten und die Struktur des Systems werden in einem weiteren Schritt auch Aspekte der Anpassungslogik beschrieben. Das *Adaptivity Engineering* sieht in diesem Schritt die Gestaltung von zwei aufeinander aufbauenden Artefakten vor. Zum einen ist das *Adaptation View Model 4 BPM* (siehe Abschnitt 4.3) und zum anderen das *Adapt Case Model 4 BPM* (siehe Abschnitt 4.2) zu gestalten.

Zuerst bietet sich die Gestaltung des *Adaptation View Model 4 BPM* (*AVM4BPM*) an. Das *AVM4BPM* beschreibt in diesem Schritt eine frühe Sicht auf das Gesamtsystem aus der Perspektive der Flexibilisierung der betroffenen Prozesse. So sollte es bspw. auch bereits Ereignisse beschreiben, die für die Auslösung des Verhaltens für eine mögliche Anpassung verwendet werden können. Derartige Ereignisse sind Teil der Gestaltung von Sensorschnittstellen einer System- oder Umgebungskomponente (siehe Abschnitt 4.3.2). Die im *AVM4BPM* beschriebenen Eigenschaften

können anschließend in der Gestaltung von Funktionen der Anpassungslogik im Rahmen des *ACM4BPM* genutzt werden (siehe Abbildung 6-3).

Das *ACM4BPM* enthält Funktionen zur Anpassung bzw. zur Flexibilisierung. Derartige Funktion werden durch das Konzept des *Adapt Case 4 BPM* (*AC4BPM*) repräsentiert. Ein *AC4BPM* kann sowohl zeit- als auch ereignisbasiert ausgelöst werden. Nicht zeitbasierte Ereignisse sollten dabei bereits durch das *AVM4BPM* eingeführt worden sein.

Bei der zuvor beschriebenen Reihenfolge für die Erstellung des *AVM4BPM* und des *ACM4BPM* handelt es sich um eine Empfehlung. Sie kann vernachlässigt werden, wenn z.B. zunächst losgelöst Aspekte der Anpassungslogik beschrieben werden sollen, die sich erst in späteren Schritten der Verfeinerung auf weitere Eigenschaften des Systems oder seiner Umgebung beziehen. Wurden sowohl Aspekte der Anpassungs- als auch der Anwendungslogik durch *UML Use Cases* bzw. *Adapt Cases 4 BPM* beschrieben, kann in die nächste Aktivität des *Adaptivity Engineering* gewechselt werden.

6.2.3 Low-Level-Gestaltung

Die Aktivität der *Low-Level-Gestaltung* stellt den letzten Schritt im Rahmen des *Adaptivity Engineering* für flexible und anpassbare Prozesse dar. Dabei werden die Verfeinerungen des zuvor beschriebenen Verhaltens und der Struktur des Systems und seiner Umgebung vorgenommen. Dabei wird abermals zwischen Aspekten der Anwendungs- und Anpassungslogik unterschieden. Ein Beispiel für den Schritt der *Low-Level-Gestaltung* wird in Abschnitt 7.1 im Zuge der Evaluation durch das beschriebene Szenario gegeben.

Aspekte der Anwendung

So können durch Prozessdiagramme (BPD) der Sprache *BPMN2.0* sogenannte Basisprozesse (hier: *Prozess*) beschrieben werden. Ein Basisprozess stellt einen Prozess dar, der als Basis für anzuwendende Anpassungen oder Maßnahmen zur Flexibilisierung betrachtet wird. Das in ihm enthaltene Verhalten verfeinert einen oder eine Reihe von zuvor gestalteten Funktionen in Form von *UML Use Cases*. Alternativ kann er aber auch weiteres Verhalten enthalten, welches als Ergänzung zu zuvor beschriebenen Funktionen betrachtet werden kann. Basisprozesse können darüber hinaus auch mit anderen Basisprozessen interagieren. Für die Beschreibung derartiger Interaktionen bieten sich weitere Diagrammtypen der Sprache *BPMN2.0* an, wie z.B. Kollaborations- oder Konversationsdiagramme. Da der in dieser Arbeit beschriebene Ansatz die Gestaltung von Prozessen

durch BPD fokussiert, werden derartige Interaktionen methodisch nicht aktiv berücksichtigt. Durch die enge Anlehnung an die Sprache *BPMN2.0* kann die Verwendung derartiger Diagramme aber als durchführbar betrachtet werden.

Die Struktur des Systems und seiner Umgebung wurde bereits in dem letzten Schritt durch *UML Komponentendiagramme* beschrieben. Da es im Rahmen des *Adaptivity Engineering* jedoch vorkommen kann, das Funktionen ungünstig verteilt oder strukturelle Eigenschaften nicht umfassend genug gestaltet worden sind, kann es notwendig sein, dass zuvor erstellte *UML Komponentendiagramm* anzupassen oder zu verfeinern. Das geänderte bzw. verfeinerte *UML Komponentendiagramm* enthält anschließend eine umfassende Beschreibung von Eigenschaften und eine Verteilung von Funktionen auf Komponenten.

Aspekte der Anwendung

Durch die *Low-Level-Gestaltung* wird die Verfeinerung von Anpassungsfällen (*AC4BPM*) vorgesehen. Ein *AC4BPM* kann durch Beobachtungs- (*Monitoring Process*) und Anpassungsprozesse (*Adaptation Process*) verfeinert werden (siehe Abschnitt 4.2.2 bzw. Abschnitt 4.2.3).

Analog zur *High-Level-Gestaltung* ist auch hier zu empfehlen, zunächst die Gestaltung des *AVM4BPM* vorzunehmen. Falls sich Komponenten der Anwendungslogik durch die jeweilige Verfeinerung bzw. Anpassung geändert haben, bietet sich die Erstellung des *AVM4BPM* als verfeinerte Sicht auf eben diese Komponenten an. Neben den bereits in der *High-Level-Gestaltung* hinzugefügten Ereignissen sollten Sensor- und Effektorschnittstellen auch über Operationen zur Anpassung sowie für einen kontrollierten Zugriff auf Daten zur Verfügung gestellt werden. Dabei profitiert die Methode *Adapt Cases 4 BPM* von dem Umstand, dass bereits diverses domänenspezifisches Wissen in der Sprache *ACML4BPM* integriert ist und an dieser Stelle verwendet werden kann. So ist eine Beschreibung von elementaren Operationen zur Anpassung von Prozessen nicht in jedem Fall notwendig, da sie bereits Bestandteil der Sprache sind. Weitere oder spezielle Operationen zur Anpassung von Prozessen oder ihrer Umgebung können selbstverständlich in Anlehnung an Anforderungen hinzugefügt werden.

Die *Low-Level-Gestaltung* schließt mit der Gestaltung von Beobachtungs- und Anpassungsprozessen ab, die Teil des *AVM4BPM* sind. Diese Prozesse verfeinern die durch einen *Adapt Case 4 BPM* gegebene Funktion durch konkreteres Verhalten. Derartiges Verhalten kann durch reguläre Elemente der Sprache *BPMN2.0*, durch bereits in der Sprache *ACML4BPM* integriertes domänenspezifisches Wissen oder aber durch neu eingeführte

Elemente des *AVM4BPM* gestaltet werden. Dabei kann in jedem dieser Prozesse umfangreiches Verhalten zur Aggregation von Daten und deren Analyse (*Beobachtungsprozesse*) und für die Anpassung von Prozessen oder ihrer Umgebung (*Anpassungsprozesse*) gestaltet werden. Im Rahmen der Gestaltung des Verhaltens können sowohl Eigenschaften des verfeinerten *AVM4BPM* als auch in der Sprache integriertes domänenspezifisches Wissen genutzt werden.

Die Gestaltung von Basis-, Beobachtungs- und Anpassungsprozessen kann auf unterschiedliche Weisen erfolgen. So ist die zuvor beschriebene Reihenfolge nicht zwingend erforderlich. Auch iterative Vorgehensweisen können sich anbieten. Dies ist dadurch zu begründen, dass je nach Anforderungen hinsichtlich des *Separation-of-Concerns* der Nutzer oder Domänenexperte entscheiden muss, welche Teile zu einer der jeweiligen Logiken gehören sollte. So kann eine strikte Trennung des Aspekts *Flexibility-by Design* bspw. zu unübersichtlichen Prozessmodellen führen. Nutzer und Domänenexperten sind daher hinsichtlich ihrer Erfahrung gefragt, in welchem Ausschnitt eines Basisprozesses auf eine strikte Trennung, wie z.B. in Bezug zur Übersichtlichkeit, verzichtet werden kann.

6.2.4 Ergänzung

Flexible und anpassbare Prozesse sind, wie an verschiedenen Stellen der vorliegenden Arbeit bereits beschrieben, nicht alleine durch eine adäquate Gestaltung realisierbar. In Anlehnung an den betrachteten *BPM-Lebenszyklus* sind offensichtlich weitere Techniken notwendig, die die Phase *Konfiguration*, die Phase *Ausführung* aber auch die Phase *Evaluation* betreffen können.

Die Bearbeitung derartiger Techniken lag dabei nicht im Fokus dieser Arbeit. Stattdessen lag die Einführung des *Adaptivity Engineering* in der Domäne *BPM* und insbesondere in Bezug zu Gestaltung von flexiblen und anpassbaren Prozessen im Vordergrund. Dabei ist anzumerken, dass dabei stets die Gestaltung der erste Schritt einer derartigen Einführung ist, welche durch diese Arbeit entsprechend gegeben ist. Hier kann durch zukünftige Forschung (siehe Abschnitt 8.2) auf Konzepte der vorliegenden Arbeit aufgebaut werden.

Ferner kann an dieser Stelle insbesondere auf ausgesuchte Entwurfsmuster Bezug genommen werden, bei denen Abweichungen vom Fokus auf die reine Gestaltung von flexiblen und anpassbaren Prozessen existieren. So wurden in Kapitel 5 verschiedenste Typen von Flexibilität in Prozessen

beschrieben. Dabei werden ebenso grundlegende Eigenschaften anderer Phasen explizit berücksichtigt.

Ein Beispiel ist durch den Flexibilitätsaspekt *Flexibility-by Underspecification* und seinem Untertyp *Late Modeling* gegeben. In *Late Modeling* wird, wie in Abschnitt 5.5.1 bereits beschrieben, zu einem spezifischen Zeitpunkt von der Phase *Ausführung* in die Phase *Design & Analyse* gewechselt. Dabei werden methodische Aktivitäten ausgeführt, die die Komposition oder Erstellung von neuen Funktionen betreffen. Dies stellt auf methodischer Ebene ebenfalls eine Technik für eine praktisch betrachtete Ausführung dar. So wird hier methodisch die Gestaltung von weiteren Funktionen während der Ausführung beschrieben. Für eine detaillierte Beschreibung wird auf Abschnitt 5.5 verwiesen.

Ein weiteres Beispiel ist durch den Flexibilitätsaspekt *Flexibility-by Change* und seinem Untertyp *Evolutionary Change* gegeben. Bei *Evolutionary Change* können Anpassungen von Prozessmodellen auf deren Instanzen migriert werden (siehe Abschnitt 5.3). Ein solches Vorgehen kann dabei zum Zweck der Verbesserung von Prozessen eingesetzt werden. Der Bedarf an Verbesserung kann z.B. in der Phase *Evaluation* ermittelt werden. Bei der anschließenden Gestaltung von Anpassungsprozessen, die sowohl Modelle als auch deren weiterhin existierende Instanzen betreffen, handelt es sich um eine methodische Technik, die über die Phase *Design & Analyse* hinausgeht. Die beschriebenen Strategien zur Migration von Anpassungen und deren Verwendung durch das entsprechende Entwurfsmuster sind somit auch ein Konzept zur Anwendung in weiteren Phasen des *BPM-Lebenszyklus*.

6.3 Zusammenfassung

In diesem Kapitel wurde das *Adaptivity Engineering* für flexible und anpassbare Prozesse vorgestellt. Dabei wird die Methode *Adapt Cases 4 BPM* mit spezifischen Aktivitäten und Artefakten in einen domänenspezifischen methodischen Rahmen integriert. Es wurde sich hierbei an dem durch *Weske* [Wes12] eingeführten *BPM-Lebenszyklus* orientiert. Ferner wurden auch Abhängigkeiten von unterschiedlichen Artefakten entlang typischer Aktivitäten des *Adaptivity Engineering* in Form einer spezifischen *Anforderungsanalyse* sowie der *High-Level-Gestaltung* und der *Low-Level-Gestaltung* gegeben. Die Methode orientiert sich dabei maßgeblich an dem durch *Luckey* [Luc13] beschriebenen Vorgehen. An spezifischen Punkten wurden Artefakte der Domäne *BPM* eingeführt und mit existierenden Artefakten in

Relation gesetzt. Hierdurch können weitere Lösungsteile wie die Sprache *ACML4BPM* (siehe Kapitel 4) sowie die zugehörigen Entwurfsmuster (siehe Kapitel 5) in einem domänenspezifischen methodischen Rahmen eingesetzt werden.

Teil III

Evaluation, Zusammenfassung und Ausblick

Kapitel **7**

Evaluation

In den vorherigen Kapiteln wurden verschiedene Lösungsteile für ein domänenspezifisches *Adaptivity Engineering* in Form des Ansatzes *Adapt Cases 4 BPM* vorgestellt. So wurde die Sprache *ACML4BPM*, eine Reihe von verschiedenen Entwurfsmustern unter Verwendung der entwickelten Sprache und eine zugehörige Methode beschrieben. Die Zielsetzung dieses Kapitels ist die Evaluation dieser drei Bestandteile. Für einzelne Teile der durchgeführten Evaluation ist eine Übersicht in Abbildung 7-1 gegeben. Die Evaluation lässt sich insgesamt in die beiden Ziele *Plausibilisierung* und *Vergleich* unterteilen. Hinsichtlich dieser Ziele werden einzelne Teile der Evaluation nachfolgend kurz beschrieben.

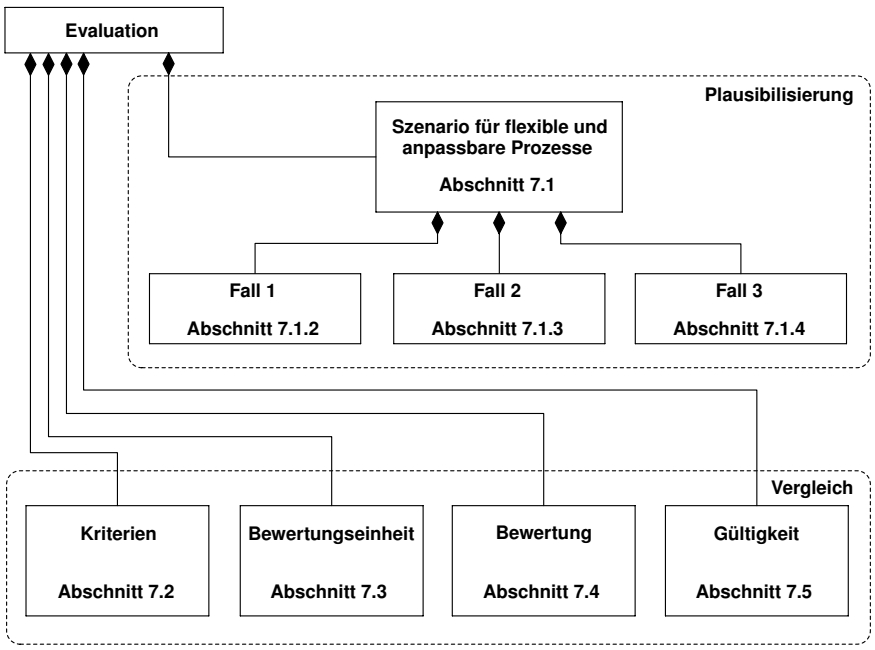


Abbildung 7-1: Übersicht über die Evaluation

Evaluation zum Zweck der Plausibilisierung

Der erste Teil der Evaluation betrifft die Anwendbarkeit der Sprache *ACML4BPM*, die vorgestellten Entwurfsmuster und die zugehörige Methode auf Basis praxisnaher Beispiele. Hiermit wird zum einen das Ziel verfolgt, die zuvor genannte Anwendbarkeit von eingeführten Konzepten zu zeigen. Zum anderen soll auf Basis der Beispiele das Verständnis von erarbeiteten Inhalten so vertieft werden, dass sie als Leitfaden für den Transfer auf Anwendungen in der Praxis eingesetzt werden können. Dabei wird zunächst in Abschnitt 7.1 ein Szenario vorgestellt, für das jeweils unterschiedliche Anforderungen an die Flexibilität beteiligter Prozesse gegeben sind. Nachfolgend wird in Abschnitt 7.1.2 bis Abschnitt 7.1.4 veranschaulicht, wie diese Anforderungen umgesetzt werden können. Dabei werden sowohl die Sprache *ACML4BPM* als auch ausgesuchte Entwurfsmuster eingesetzt. Der Ablauf folgt der Methode *Adapt Cases 4 BPM* für die Gestaltung von flexiblen und anpassbaren Prozessen.

Evaluation zum Zweck des Vergleichs

Der zweite Teil der Evaluation konzentriert sich anschließend auf ausgesuchte Kriterien zur Bewertung der Sprache *ACML4BPM* und die zugehörigen Entwurfsmuster. Dabei werden zwei verschiedene Kataloge von Kriterien für die Evaluation verwendet. Hierdurch wird das Ziel der Durchführung eines Vergleichs mit existierenden Ansätzen verfolgt. In der Bewertung von beiden Katalogen wird an geeigneter Stelle auf die Inhalte des zuvor eingeführten Szenarios zurückgegriffen. Der zweite Teil der Evaluation gliedert sich nachfolgend so, dass zunächst verschiedene Kriterienkataloge in Abschnitt 7.2 vorgestellt werden. Ergänzend werden in Abschnitt 7.3 die Bewertungseinheit sowie grundsätzliche Annahmen für die Bewertung zuvor eingeführter Kriterien beschrieben. Die Ergebnisse der Evaluation werden in Abschnitt 7.4 veranschaulicht. Das Kapitel schließt in Abschnitt 7.5 mit einer kritischen Diskussion hinsichtlich der Gültigkeit der vorgestellten Evaluation ab.

7.1 Szenario für flexible und anpassbare Prozesse

Die in dieser Arbeit beschriebenen Konzepte wurden bisher lediglich in Form von Beispielen zur generellen Verdeutlichung der schematischen Funktionsprinzipien erläutert. Daher wird in diesem Abschnitt ein Szenario beschrieben, in dem die Gestaltung von Flexibilität in Prozessen an praxisorientierten Beispielen veranschaulicht wird. Hierdurch werden zwei wesentliche Ziele verfolgt. Zum einen soll ein beispielhafter Ausschnitt für die Verwendung der erarbeiteten Konzepte dargestellt werden, wodurch ein vereinfachter transdisziplinärer Transfer sowie die Ermöglichung ei-

nes tieferen Verständnisses der vorgestellten Konzepte adressiert werden. Zum anderen dient das Szenario aber auch der Argumentation in Bezug zur Bewertung von Kriterien in weiteren Teilen der Evaluation (siehe Abschnitt 7.2.1 bis Abschnitt 7.2.2). Eine schematische Übersicht über das hier betrachtete Szenario ist in Abbildung 7-2 dargestellt.

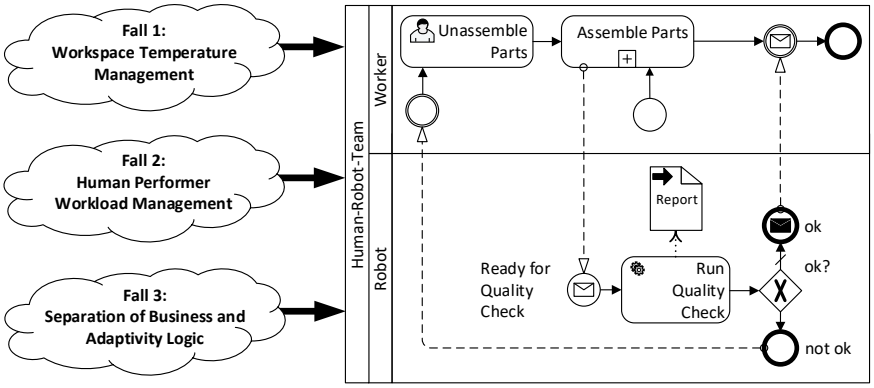


Abbildung 7-2: Schematische Übersicht über das betrachtete Szenario

Jeder der hier aufgeführten Fälle (engl. *Case*) handelt im Kontext eines gemeinsamen Hauptprozesses. Der hier verwendete Hauptprozess *Human-Robot-Team* spielt in einem mittelständischen Unternehmen, in dem trotz zunehmender Automatisierung die Endmontage von Produkten vornehmlich manuell durchgeführt wird. Dabei handelt es sich um eine Beschreibung von Abläufen in einem sogenannten *Shared-Workspace*. Ein *Shared-Workspace* ist eine spezielle Arbeitsumgebung, in der sowohl menschliche als auch technisierte Akteure, wie z.B. ein IT-gestütztes Assistenzsystem oder ein Roboter, kollaborativ tätig sind.

Die in der Arbeitsplatzumgebung notwendigen Funktionen innerhalb der relevanten geschäftlichen Abläufe können dabei durch die menschlichen oder technisierten Akteure exklusiv oder kooperativ durchgeführt werden. Hiervon können z.B. die Assistenz in Arbeitsabläufen, die Aufgabenplanung oder eine damit verbundenen Entscheidungsfindung betroffen sein. Das Ziel des hier dargestellten Hauptprozesses im *Shared-Workspace* ist durch die Montage von Bauteilen eines Produktes und der anschließenden Qualitätssicherung gegeben. Die Montage durch die Rolle *Worker* ist hier durch den Subprozess *Assemble Parts* dargestellt. Sobald die Montage abgeschlossen ist, übernimmt die Rolle *Robot* die Qualitätssicherung (*Run Quality Check*). Der Prozess terminiert, wenn die Qualität in Ordnung ist. Im alternativen Fall übernimmt die Rolle *Worker* die Aufgabe der Demontage (*Unassemble Parts*). Die dargestellte Endmontage findet dabei an Arbeitsstationen im Rahmen des *Shared-Workspace* statt.

Wie zu Beginn dieser Arbeit motiviert, steht in dem betrachteten Szenario dabei insbesondere die Flexibilisierung beteiligter Prozesse im Vordergrund, sodass neben der eigentlichen Funktion des Gesamtsystems – der Fertigung – auch bedarfsorientierte Anpassungen von Eigenschaften der beteiligten Prozesse sowie ihrer Umgebung unterstützt werden. Wesentliche Teile des Szenarios wurden dabei in verschiedenen Publikationen bereits eingeführt [Eng+18; ET18; EST18].

Die Relevanz für die Praxis der im Folgenden betrachteten Fälle des Szenarios wurde auf Basis von Gesprächen mit Experten aus Wissenschaft und Industrie in verschiedenen Workshops im Rahmen des *NRW Fortschrittskollegs „Gestaltung von flexiblen Arbeitswelten"* identifiziert. Nachfolgend werden die einzelnen Fälle *1* bis *3* zunächst vorgestellt. Die nachfolgende Beschreibung der Fälle kann als der erste Schritt der Methode *Adapt Cases 4 BPM* in Form einer *Anforderungsanalyse* betrachtet werden (siehe Abschnitt 6.2.1). So enthalten die Beschreibungen natürlichsprachliche Anforderungen, die durch weitere Schritte der Methode in Form der *High-Level-Gestaltung* und *Low-Level-Gestaltung* anschließend umgesetzt werden.

Fall 1: Workspace Temperature Management (WTM)

Im ersten Fall wird zunächst angenommen, dass aufgrund von rechtlichen Bestimmungen die Temperatur in Arbeitsumgebungen überwacht werden muss. Kommt eine zu hohe Temperatur auf, sollen geeignete Maßnahmen zur Entlastung von menschlichen Akteuren durchgeführt werden. Für den hier betrachteten *Shared-Workspace* soll eine Anpassung am Hauptprozess durchgeführt werden, sobald die Temperatur höher als 24 Grad Celsius beträgt. Als anzuwendende Maßnahme in einem solchen Fall wurde ein Wechsel der aktuell zugewiesenen menschlichen Akteure festgelegt. So soll die Zuweisung eines menschlichen Akteurs alle 20 Minuten geändert werden. Ein zuvor aktiver und anschließend ausgetauschter menschlicher Akteur kann so die Zwischenzeit nutzen, um sich im Pausenraum zu erholen.

Fall 2: Human Performer Workload Management (HPWM)

Im Rahmen des zweiten Falls soll eine weitere Funktion zur Menschenzentrierung des Hauptprozesses unterstützt werden. So soll die physische Arbeitsbelastung eines menschlichen Akteurs auf Basis seiner Herzfrequenz und seiner zuvor bereits erstellten individuellen Arbeitslastprofile beobachtet werden. Tritt die Situation einer Überlastung auf, so wird der Prozess angepasst, indem ein anderer zur Verfügung stehender menschlicher Akteur zugewiesen wird. Neben dieser automatisierten Auslösung einer Anpassung soll aber auch manuell eine Zuweisung eines ablösenden Akteurs angestoßen werden können.

Fall 3: Separation of Business and Adaptivity Logic (SoC)

In *Fall 1* und *Fall 2* sind vor allem potentielle Auslöser aus der Umgebung des Prozesses für eine Anpassung angedacht. Darüber hinaus kann es aber auch sinnvoll sein, bestehende Prozesse hinsichtlich ihrer bisherigen Umsetzung von verschiedenen Aspekten von Flexibilität zu untersuchen und ggf. im Rahmen der Verbesserung von Prozessen anzupassen. Eine derartige Anpassung kann damit als evolutionär hinsichtlich zukünftiger Instanzen des Hauptprozesses betrachtet werden. Der hier betrachtete Hauptprozess enthält dabei eine Schleife, in die gewechselt wird, falls die Qualität des montierten Produktes nicht ausreichend ist. Diese Schleife kann als Aspekt *Iteration* von *Flexibility-by Design* verstanden werden. Auf eine beispielhafte Gestaltung wurde in Abschnitt 5.2.3 eingegangen. Im dritten Szenario wird daher eine Umstrukturierung des bestehenden Hauptprozesses beschrieben, indem eine Trennung der Anwendungs- von der Anpassungslogik vorgenommen wird.

Auslösung des WTM und HPWM

Für die zuvor beschriebenen Fälle 1 und 2 wird ferner angenommen, dass neben einer Gestaltung, in der ein menschlicher Akteur selbst eine Anpassung anstoßen kann, auch insbesondere die automatisierte und regelbasierte Umsetzung gefordert ist. Hierfür kann es auch in der Praxis verschiedene Gründe geben. So kann zum einen mit der Art und dem Umfang der Tätigkeiten, die durch menschliche Akteure auszuführen sind, eine besonders hohe Konzentration gefordert sein. Hier könnten Überforderungen der menschlichen Akteure eintreten, wenn bspw. die Überwachung der kontextuellen Temperatur des *Shared-Workspace* zusätzlich gefordert werden würde. Zum anderen ist es aber auch möglich, dass menschliche Akteure nur eingeschränktes Wissen über rechtliche Regularien hinsichtlich ihrer Arbeitsumgebung verfügen und somit nicht in der Lage sind oder nicht die Befugnisse haben, entsprechende Maßnahmen selbstständig auszuführen.

Die beiden zuvor genannten Gründe werden zusätzlich bestärkt, wenn man in Betracht zieht, dass eine Vielzahl von verschiedenen Umgebungsfaktoren den sicheren und menschenzentrierten Betrieb der Fertigung bedingen können. Die Beobachtung von sich stetig ändernden Umgebungsfaktoren allein auf menschlicher Basis birgt somit die Gefahr der Überforderung menschlicher Akteure und damit die Gefährdung eines sicheren Betriebs. Eine Automatisierung der Beobachtung von sich ändernden Umgebungsfaktoren sowie der anschließenden Auswahl von möglichen Anpassungen der beteiligten Prozesse bzw. ihrer Umgebung kann daher in diesem Bezug als Erleichterung, wenn nicht sogar als einzige Möglichkeit verstanden werden, einer potentiell hohen Anzahl an möglichen Umge-

bungsfaktoren, Entscheidungen und nachfolgenden Anpassungen gerecht zu werden. Dennoch können vereinzelte Interaktionen zwischen menschlichen Akteuren mit ihrer Umgebung als sinnvoll erachtet werden. So können stets unvorhergesehene Ereignisse – auch Ausnahmen genannt – Anpassungen notwendig machen, die im Rahmen der regelbasierten Automatisierung nicht behandelt werden können. Alternativ lassen sich aber auch bestimmte Umgebungsfaktoren nur schwer oder nicht durch bestehende Techniken erfassen, sodass es oftmals leichter ist, den Menschen als Entscheidungsträger eintreten zu lassen, um eine Anpassung manuell auszulösen. Ein Beispiel hierfür ist im *Fall 2* enthalten. Hier können menschliche Akteure auch manuell eine Zuweisung eines ablösenden Akteurs auslösen.

Komplexität des Hauptprozesses

Ein weiterer Hinweis für die betrachteten Fälle muss in Bezug zur enthaltenen Komplexität gegeben werden. So handelt es sich bei dem vorliegenden Hauptprozess um einen auf den ersten Blick simplen Ablauf von nur wenigen Tasks. Man könnte also zunächst annehmen, dass der Prozess *Human-Robot-Team* für eine Evaluation eher ungeeignet scheint. Dem kann auf Basis gesetzter Anforderungen an die erarbeitete Lösung jedoch widersprochen werden.

So beschäftigt sich der vorliegende Ansatz vornehmlich mit der Anforderung des *Separation-of-Concerns* hinsichtlich der Anpassungs- und Anwendungslogik. Dies bedeutet, dass diese einzelnen Aspekte eines Prozesses getrennt gestaltet werden. Das Bindeglied bildet hierbei eine Regel, die ereignisbasiert bzw. zeitgesteuert ausgelöst werden kann. Dabei wird zunächst im Rahmen eines Beobachtungsprozesses eine Analyse von Umgebungsfaktoren vorgenommen. Anschließend wird eine Entscheidung für die Auswahl von vordefinierten Maßnahmen getroffen. Derartige Maßnahmen liegen in Form von Anpassungsprozessen vor. Der Ansatz stellt somit eine Möglichkeit dar, in Anlehnung an das Paradigma *MAPE-K* [KC03], eine Regel in der Form *Wenn-Dann* bzw. *Wenn-Dann-Anders* zu definieren. Dabei wird bereits bei einer Analyse des Ansatzes *Adapt Cases* nach *Luckey* [LE13] deutlich, dass weder Fähigkeiten hinsichtlich der Funktion – wie z.B. fortgeschrittene Analysetechniken zur Beobachtung – noch konkrete Handlungsempfehlungen für durchzuführende Maßnahmen im Fokus stehen. Diese können sowohl in dem Ansatz *Adapt Cases* als auch in dem Ansatz *Adapt Cases 4 BPM* in späteren Phasen der Gestaltung durch etwaige Verfeinerungen hinzugefügt werden.

Der Ansatz soll vielmehr die Gestaltung und Analyse von Prozessen dahingehend unterstützen, dass, wenn eine getrennte Gestaltung der bei-

den Logiken gefordert ist, sie für die Domäne *BPM* und für dort übliche Konzepte durch den Ansatz *Adapt Cases 4 BPM* möglich ist. Die Komplexität des Hauptprozesses *Human-Robot-Team* und der zugehörigen Fälle des Szenarios kann daher als angemessen betrachtet werden, da veranschaulicht wird, wie die Trennung der Anpassungs- und Anwendungslogik durchgeführt werden kann. Ferner wird angenommen, dass ein Übertrag auf Prozesse aus der Praxis dadurch unterstützt werden kann, dass die in den Abschnitten 7.1.2 bis 7.1.4 gegebenen praxisnahen Beispiele als Leitfaden verwendet werden können. Da die in dieser Arbeit vorgestellten Konzepte zur Gestaltung von flexiblen und anpassbaren Prozessen vielfältig sind, können nur ausgesuchte Elemente der Sprache *ACML4BPM* sowie von passenden Entwurfsmustern im Rahmen der Evaluation berücksichtigt werden. Sie stehen dabei stellvertretend als Leitfaden für den Übertrag auf Prozesse aus der Praxis.

Plausibilisierung der Methode Adapt Cases 4 BPM

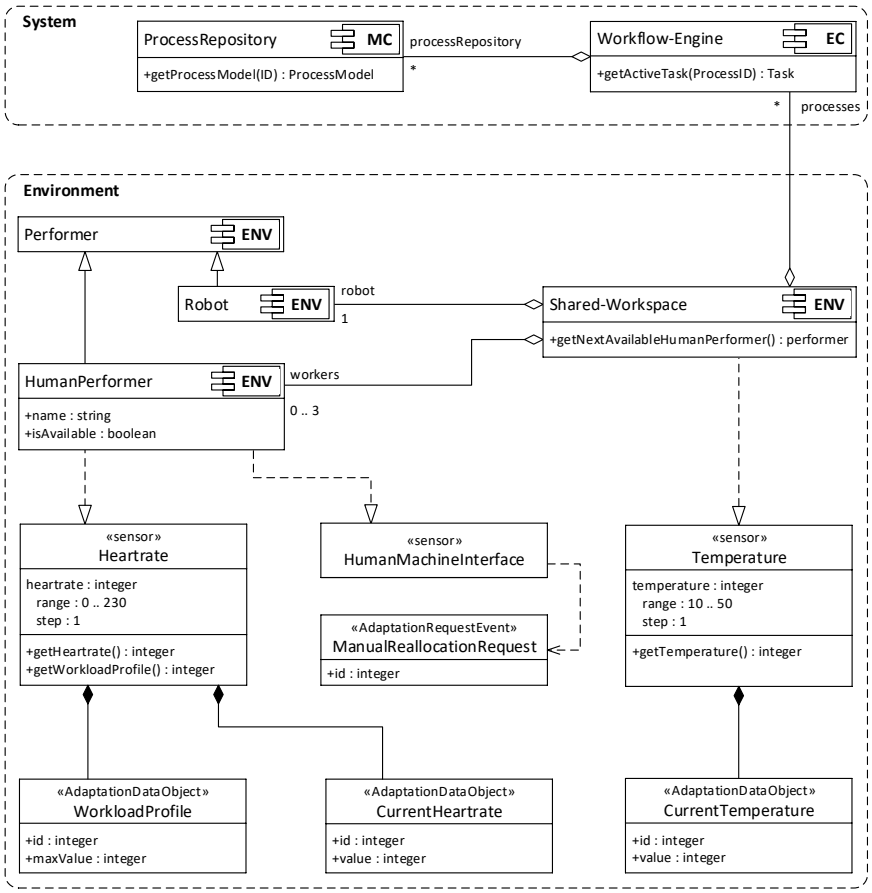
Das Vorgehen folgt der in Kapitel 6 beschriebenen Methode *Adapt Cases 4 BPM*. In Bezug zu den zuvor eingeführten Fällen des Szenarios werden zunächst in Abschnitt 7.1.1 relevante System- und Umgebungskomponenten als Teil eines gemeinsamen *AVM4BPM* (siehe Abschnitt 4.3) beschrieben. Das *AVM4BPM* stellt eine Sicht auf das Gesamtsystem aus der Perspektive der Anpassungslogik dar. Es wird zunächst im Rahmen der *High-Level-Gestaltung* erstellt und im Rahmen der *Low-Level-Gestaltung* verfeinert (siehe Abschnitt 6.1). Auf die Verfeinerung des im Rahmen des Szenarios beschriebenen *AVM4BPM* wird jedoch verzichtet, da es sich hierbei um eine Korrekturmaßnahme in einem realen Vorgehen handelt. In den nachfolgenden Abschnitten 7.1.2 bis 7.1.4 wird auf die jeweiligen Fälle des Szenarios in Form einer spezifischen Gestaltung durch ein *ACM4BPM* eingegangen (siehe Abschnitt 4.2). Für jeden Fall des Szenarios wird dabei das Resultat von sowohl der *High-Level-Gestaltung* als auch der *Low-Level-Gestaltung* von prozessspezifischen Aspekten dargestellt (siehe Kapitel 6).

7.1.1 Die Arbeitsumgebung Human-Robot-Team

Damit flexible und anpassbare Prozesse im Rahmen der Methode *Adapt Cases 4 BPM* beschrieben werden können, ist die Gestaltung von einem für die Arbeitsumgebung *Human-Robot-Team* zugehörigen *AVM4BPM* notwendig (siehe Abschnitt 4.3). Dabei wurde sich an der Beschreibung der einzelnen Fälle des Szenarios orientiert, die sich jeweils auch als natürlichsprachliche Anforderungen auffassen lassen (siehe Abschnitt 6.2.1). In Abbildung 7-3 ist ein Ausschnitt des für die Arbeitsumgebung *Human-Robot-Team* gestalteten *AVM4BPM* dargestellt. Es lässt sich dabei in die beiden

Teile *System* und *Environment* unterteilen, deren Komponenten, Sensoren, Ereignisse und Daten nachfolgend beschrieben werden.

Abbildung 7-3: AVM4BPM für die Arbeitsumgebung Human-Robot-Team



Beschreibung des Systems

In dieser Arbeit stehen Prozesse im Fokus, sodass die zugehörigen Systemkomponenten *ProcessRepository* und *WorkflowEngine* verwendet werden. Durch die Systemkomponente *ProcessRepository* werden alle Modelle von Prozessen gekapselt. Auf Prozessmodelle kann durch die Methode *getProcessModel(ID)* unter Angabe eines eindeutigen Identifizierers (*ID*) zugegriffen werden. Als zweite wesentliche Systemkomponente wird *WorkflowEngine* verwendet. Sie steht für eine Ausführungsumgebung. In ihr werden auf Basis der in der Systemkomponente *ProcessRepository* gekapselten Prozessmodelle benötigte Prozessinstanzen erstellt und ausgeführt.

Beschreibung der Umgebung des Systems

Die Umgebung des Hauptprozesses wird in Abbildung 7-3 durch die Umgebungskomponente *Shared-Workspace* dargestellt. Ferner existieren verschiedene Komponenten aus der Umgebung des Hauptprozesses, die als

Teil des *Shared-Workspace* betrachtet werden müssen. Diese werden durch die Komponenten *Robot* und *HumanPerformer* dargestellt. Bei diesen beiden Komponenten handelt es sich um eine Beschreibung von am Prozess beteiligten Akteuren (hier: *Performer*) Die Umgebungskomponente *Robot* kapselt Inhalte des in der Montage eingesetzten Roboters. Da keine Anpassungen auf Basis dieser Inhalte durchgeführt werden sollen, implementiert diese Komponente keine Schnittstellen und ist daher nur der Vollständigkeit halber dargestellt. Die Umgebungskomponente *HumanPerformer* steht stellvertretend für einen menschlichen Akteur und kapselt für die Anpassung relevante Inhalte. So können verschiedene Informationen wie die aktuelle Verfügbarkeit eines menschlichen Akteurs und sein Name abgerufen werden. Hierfür stehen die Attribute *isAvailable* vom Typ *boolean* bzw. *name* vom Typ *String* zur Verfügung. Insgesamt können im Rahmen des *Shared-Workspace* bis zu drei menschliche Akteure eingesetzt werden.

Sensor für die Temperatur

Die Umgebungskomponente *Shared-Workspace* implementiert eine Sensorschnittstelle (*Temperature*), die den Zugriff auf die aktuelle Temperatur der Arbeitsumgebung sicherstellt. Dabei wird in der Gestaltung der zugehörigen Prozesse ein Zugriff auf diese Eigenschaft durch das Element vom Typ *AdaptationDataObject* mit der Bezeichnung *CurrentTemperature* ermöglicht.

Sensoren für die Herzfrequenz und die manuelle Interaktion

Die Komponente *HumanPerformer* implementiert die beiden Sensorschnittstellen *Heartrate* und *HumanMachineInterface*. Durch den Sensor mit der Bezeichnung *Heartrate* wird der Zugriff auf die aktuelle Herzfrequenz eines menschlichen Akteurs ermöglicht. In der späteren Gestaltung des zugehörigen *ACM4BPM* kann das Element vom Typ *AdaptationDataObject* mit der Bezeichnung *CurrentHeartrate* verwendet werden, um auf aktuelle Daten zuzugreifen. Ferner ist das persönliche Belastungsprofil durch das Datenobjekt mit der Bezeichnung *WorkloadProfile* zugreifbar. Zur Vereinfachung enthält das Belastungsprofil an dieser Stelle lediglich ein Attribut zur Kennzeichnung eines individuellen maximalen Wertes für die Herzfrequenz (*maxValue*). In einer realen Anwendung lassen sich hier selbstverständlich auch komplexere Belastungsprofile beschreiben, deren Verwendung im Rahmen der Evaluation jedoch den Fokus zu sehr verschieben würden. Durch den Sensor mit der Bezeichnung *HumanMachineInterface* wird das Auslösen eines Ereignisses von Typ *AdaptationRequestEvent* mit der Bezeichnung *ManualReallocationRequest* ermöglicht. Dieses Ereignis ist für die Auslösung eines manuellen Wechsels des zugewiesenen menschlichen Akteurs bestimmt.

Das zuvor beschriebene *AVM4BPM* bietet ein mögliches Beispiel für die Gestaltung von relevanten Teilen des betrachteten Systems und seiner Um-

gebung, sodass die Gestaltung von flexiblen und anpassbaren Prozessen durch das Konzept *Adapt Case 4 BPM* in Bezug zu den verschiedenen Fälle des Szenarios ermöglicht werden kann. In den nachfolgenden Abschnitten 7.1.2 bis 7.1.4 wird die Verwendung dieser Elemente im Rahmen von Beobachtungs- und Anpassungsprozessen der einzelnen Fälle des Szenarios gezeigt.

7.1.2 Fall 1: Workspace Temperature Management

In dem ersten Fall soll eine Anpassung an dem Hauptprozess beim Überschreiten einer vordefinierten Temperatur in dem *Shared-Workspace* durchgeführt werden. Hierzu ist in Abbildung 7-4 die Gestaltung des zugehörigen *AC4BPM Workspace Temperature Management (WTM)* dargestellt. Es wurde sich für eine integrierte Darstellung der *High-Level-Gestaltung* und der *Low-Level-Gestaltung* entschieden. Hierdurch soll der Zusammenhang von Elementen aus unterschiedlichen Aktivitäten der Methode *Adapt Cases 4 BPM* verdeutlicht werden. So wird sowohl der *AC4BPM* des *WTM* als auch seine Verfeinerung in Form der zugehörigen Beobachtungs- und Anpassungsprozesse gezeigt.

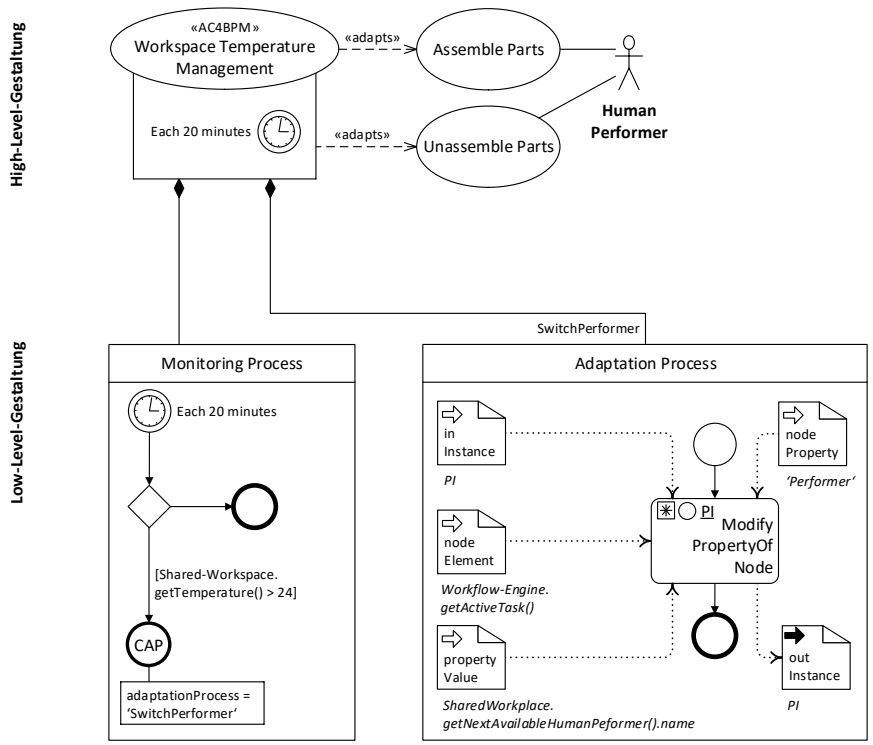


Abbildung 7-4: AC4BPM für das Workspace Temperature Management

High-Level-Gestaltung des WTM

Durch die *High-Level-Gestaltung* wird hier ein *AC4BPM* mit der Bezeichnung *Workspace Temperature Management* beschrieben. Das *WTM* stellt die Funktion zur Anpassung des Hauptprozesses im Rahmen von *Fall 1* dar. Es wurde sich dafür entschieden, dass die Funktion *WTM* zeitbasiert alle *20* Minuten ausgeführt werden soll. Es lassen sich auch alternative Intervalle je nach Anforderung an die Aktualität der Prüfung definieren. Dabei treten im Fall der Notwendigkeit einer Anpassung mögliche Anpassungen an den Funktionen der Hauptprozesse *Assemble Parts* bzw. *Unassemble Parts* statt. Die beiden genannten Funktionen werden dabei durch einen menschlichen Akteur ausgeführt, welcher hier dargestellt ist durch den Akteur *Human Performer*.

Low-Level-Gestaltung des WTM

Durch die *Low-Level-Gestaltung* wird der *AC4BPM* der Funktion *WTM* durch einen Beobachtungs- und Anpassungsprozess verfeinert. Der hier dargestellte Beobachtungsprozess beschreibt dabei das vorgesehene Verhalten für die Prüfung, ob die zulässige maximale Temperatur zu einem Zeitpunkt überschritten ist. Fällt die Prüfung negativ aus, so terminiert der Beobachtungsprozess und es wird keine Anpassung am Hauptprozess ausgeführt. Im positiven Fall terminiert der Beobachtungsprozess ebenfalls. Dabei wird jedoch zuvor der dargestellte Anpassungsprozess mit der Bezeichnung *SwitchPerformer* aufgerufen. Dieser Beobachtungsprozess beschreibt das Verhalten zur Anpassung des Hauptprozesses. Hier wird also ein aktuell zugewiesener menschlicher Akteur durch einen anderen ersetzt. Dabei wird eine der in Abschnitt 4.3.3 vorgestellten Operationen zur Anpassung von Prozessen verwendet. So wird die Operation *ModifyPropertyOfNode* eingesetzt, um das Attribut *Performer* des aktuell aktiven Tasks so zu setzen, dass ein noch nicht zugewiesener menschlicher Akteur zugeordnet wird. Die benötigte Anpassung ist mit dieser Zuweisung anschließend abgeschlossen.

Ergänzung

Das zuvor beschriebene Verhalten des *WTM* setzt die in Abschnitt 7.1 beschriebene Funktionalität in Form ausgesuchter Lösungsbestandteile dieser Arbeit um. So wird zeitgesteuert eine Prüfung der Temperatur durchgeführt und im Bedarfsfall ein Wechsel von zugewiesenen menschlichen Akteuren im *Shared-Workspace* durchgeführt.

Der zuvor beschriebene *AC4BPM* des *WTM* sowie seine Verfeinerungen, hier gegeben durch den Beobachtungs- und Anpassungsprozess, stellen ein erstes kleines aber praxisnahes Beispiel zum Zweck der Plausibilisierung der vorgestellten Lösung dar. Selbstverständlich können weitere Eigenschaften des Systems und seiner Umgebung bei einer Anpassung im Rahmen des *WTM* berücksichtigt werden, sofern sie zuvor dem

AVM4BPM hinzugefügt worden sind. Aufgrund einer gesteigerten Übersichtlichkeit wurde sich jedoch dagegen entschieden, derartige Details hier darzustellen, da sie lediglich eine Wiederholung gleicher grundsätzlicher Mechanismen darstellen. Für weiterführende Beispiele wird auf die nachfolgenden Fälle des Szenarios (siehe Abschnitte 7.1.3 und 7.1.4) verwiesen.

7.1.3 Fall 2: Human Performer Workload Management

Der zweite Fall beschäftigt sich mit Eigenschaften eines menschlichen Akteurs, der im Rahmen des *Shared-Workspace* zu einem Zeitpunkt tätig ist. Dabei soll ein Wechsel von menschlichen Akteuren durchgeführt werden, sobald eine Überbelastung des aktuell zugewiesenen menschlichen Akteurs entweder automatisiert erkannt oder manuell angezeigt wird. Hierzu ist in Abbildung 7-5 erneut eine integrierte Darstellung gegeben, die sowohl die *High-Level-Gestaltung* als auch die *Low-Level-Gestaltung* des erstellten *AC4BPM* für das *Human Performer Workload Management (HPWM)* zeigt.

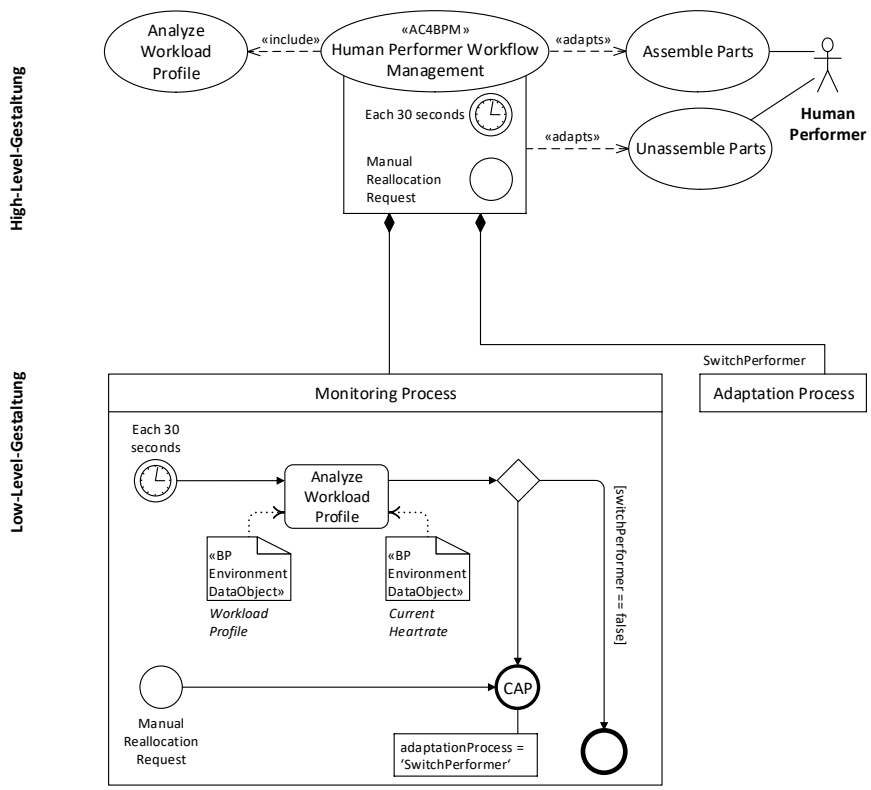


Abbildung 7-5: AC4BPM für das Human Performer Workload Management

High-Level-Gestaltung des HPWM

Durch die *High-Level-Gestaltung* wird hier ein *AC4BPM* mit der Bezeichnung *Human Performer Workspace Management* beschrieben. Das *HPWM* stellt die Funktion zur Anpassung des Hauptprozesses im Rahmen von *Fall 2* dar. Das *HPWM* kann dabei sowohl zeitbasiert als auch ereignisbasiert aufgerufen werden. Der zeitbasierte Aufruf der Funktion *HPWM* wird alle 30 Sekunden durchgeführt. Analog zum *WTM* sind auch hier alternative Intervalle möglich. Ein Aufruf durch das Ereignis mit der Bezeichnung *ManualReallocationRequest* repräsentiert die manuelle Auslösung einer neuen Zuordnung eines menschlichen Akteurs. So ist in weiteren Verfeinerungen des Modells denkbar, dass dieses Ereignis im Rahmen der Verwendung einer Benutzerschnittstelle, wie z.B. auf einem mobilen Endgerät, erzeugt und an die Komponente *WorkflowEngine* weitergereicht wird.

Im Fall der Notwendigkeit einer Anpassung sollen abermals die Funktionen des Hauptprozesses *Assemble Parts* und *Unassemble Parts* angepasst werden. Die beiden genannten Funktionen werden dabei durch einen menschlichen Akteur ausgeführt, welcher hier dargestellt ist durch den Akteur *Human Performer*. Im Vergleich zu *Fall 1* wird die weitere Funktion mit der Bezeichnung *Analyze Workload Profile* in *Fall 2* verwendet. Diese Funktion wird zur Analyse der aktuellen Belastung eines zugewiesenen menschlichen Akteurs verwendet und ist Teil des hier beschriebenen *AC4BPM HPWM*. Dabei wird angenommen, dass diese Funktion selbst durch einen IT-basierten Dienst ausgeführt wird. Ein solcher Dienst kann z.B. im Kontext eines Anwendungsservers existent sein. Auf die Beschreibung eines zugehörigen Akteurs wurde dabei verzichtet, da der Dienst als außerhalb der Sicht des *Adaptivity Engineering* betrachtet werden kann und derartige Umstände nur für Referenzzwecke Teil eines *ACM4BPM* sind.

Low-Level-Gestaltung des HPWM

Durch die *Low-Level-Gestaltung* wird der *AC4BPM* der Funktion *HPWM* durch einen Beobachtungs- und Anpassungsprozess verfeinert. Der hier dargestellte Beobachtungsprozess beschreibt dabei das vorgesehene Verhalten für die Prüfung, ob eine Überlastung des derzeit aktiven menschlichen Akteurs besteht. Im Fall eines zeitgesteuerten Aufrufs wird durch den Task mit der Bezeichnung *AnalyzeWorkloadProfile* die derzeitige Belastung des menschlichen Akteurs überprüft. Ein Zugriff auf Eigenschaften der Komponente mit der Bezeichnung *HumanPerformer* ist dabei durch die beiden Datenobjekte mit der Bezeichnung *WorkloadProfile* und *CurrentHeartrate* dargestellt. Fällt die Prüfung negativ aus, so terminiert der Beobachtungsprozess und es wird keine Anpassung am Hauptprozess ausgeführt. Eine im Rahmen des Tasks durchgeführte Überprüfung könnte im einfachsten Fall dadurch gegeben sein, dass der aktuelle Wert der Herz-

frequenz nicht über dem im *WorkloadProfile* hinterlegten Wert liegen darf. Wie bereits angedeutet stellt dieser Task im Rahmen der Evaluation lediglich ein einfaches Beispiel dar. Weiterführende Analysemethoden lassen sich durch eine Verfeinerung des Tasks und der eingesetzten Datenobjekte umsetzen.

Im alternativen Fall eines ereignisbasierten Aufrufs des *HPWMs* wird keine Analyse von Umgebungsfaktoren durchgeführt. Hier wird angenommen, dass eine manuelle Anzeige einer Überbelastung durch einen menschlichen Akteur hinreichend aussagekräftig ist. Daher wird der bereits durch das *WTM* bekannte Anpassungsprozess mit der Bezeichnung *SwitchPerformer* direkt aufgerufen. Auf eine Beschreibung dieses Anpassungsprozesses wird an dieser Stelle verzichtet und auf Abschnitt 7.1.2 verwiesen.

Ergänzung Das zuvor beschriebene Verhalten des *HPWM* setzt die in Abschnitt 7.1 beschriebene Funktionalität in Form ausgesuchter Lösungsbestandteile dieser Arbeit um. So ist es möglich, im Bedarfsfall einen Wechsel von zugewiesenen menschlichen Akteuren im *Shared-Workspace* auf unterschiedliche Arten durchzuführen. Dabei wird sowohl eine automatisierte als auch eine manuelle Variante unterstützt. Der zuvor beschriebene *AC4BPM* des *HPWM* sowie seine Verfeinerungen, hier gegeben durch den Beobachtungs- und Anpassungsprozess, stellen ein weiteres Beispiel zum Ziel der Plausibilisierung der vorgestellten Lösung dar. Auch hier können weitere Eigenschaften des Systems und seiner Umgebung bei einer Anpassung im Rahmen des *HPWM* berücksichtigt werden, sofern sie zuvor dem *AVM4BPM* hinzugefügt worden sind.

7.1.4 Fall 3: Separation of Business and Adaptivity Logic

Der letzte im Rahmen der Evaluation betrachtete Fall des Szenarios beschäftigt sich mit der Umstrukturierung von bestehenden Prozessen hinsichtlich der Trennung von Anpassungs- und Anwendungslogik. Hierzu soll der in Abbildung 7-2 eingeführte Hauptprozess *Human-Robot-Team* so umstrukturiert werden, dass die beiden genannten Aspekte unter Verwendung der Sprache *ACML4BPM* getrennt voneinander gestaltet werden. Zuvor müssen hierfür jedoch Elemente der beiden Logiken identifiziert werden. Hierzu ist in Abbildung 7-6 das Ergebnis einer durchgeführten Analyse gezeigt. Elemente der Anpassungslogik sind in der Farbe *Grün* und Elemente der Anwendungslogik in der Farbe *Blau* hinterlegt dargestellt.

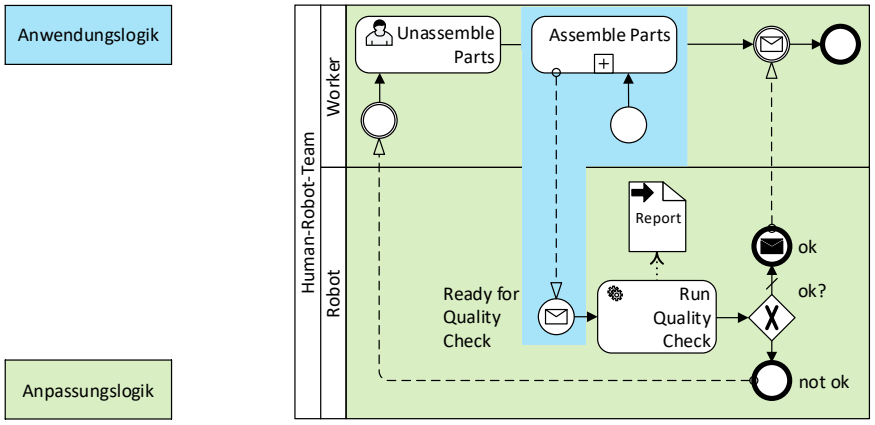


Abbildung 7-6: Analyse des Hauptprozesses

Ergebnis der Analyse

Der Prozess enthält dabei nicht nur Funktionen zur Montage und der Qualitätssicherung. So ist in Abhängigkeit zum dargestellten Qualitätsbericht (*Report*) auch eine Funktion zur Demontage vorgesehen, in deren Anschluss die Montage erneut beginnt. Dies kann als eine spezielle Art von *Flexibility-by Design* in Form des Aspekts *Iteration* verstanden werden (siehe Abschnitt 5.2.3). Im Rahmen des Aspekts *Iteration* können bestimmte Funktionen eines Ablaufs iterativ ausgeführt werden. Das dargestellte Verhalten für die Qualitätssicherung stellt dabei die Bedingung für die Durchführung einer weiteren Iteration dar. Das Verhalten in Form des Tasks *Unassemble Parts* stellt Verhalten zur Rückabwicklung von Verhalten der Anwendungslogik in Form des Subprozesses *Assemble Parts* dar.

In bestehenden Prozessen kann Flexibilität wie zuvor beschrieben in verschiedenen Formen vorkommen. Soll das *Adaptivity Engineering* als methodischer Rahmen zur Umstrukturierung von Prozessen eingesetzt werden, so sollten deren Modelle, Instanzen und Prozesshistorien sorgfältig analysiert werden. Hierdurch kann eine Verbesserung der Prozesse aus Sicht des *Adaptivity Engineering* in zukünftigen Iterationen des *BPM-Lebenszyklus* unterstützt werden. Ein exemplarisches Resultat einer solchen Verbesserung hinsichtlich der zuvor gegebenen Analyse ist in Abbildung 7-7 dargestellt.

High-Level-Gestaltung SoC

Die durch die Umstrukturierung vorgenommene Verbesserung führt dabei auf der Ebene der *High-Level-Gestaltung* sowohl einen neuen *AC4BPM* mit der Bezeichnung *Check Product Quality* ein als auch eine Funktion mit der Bezeichnung *Run Quality Check*. Bereits zuvor vorgestellte Funktionen der Montage sind hier durch die Funktionen *Assemble Parts* und *Unassemble Parts* dargestellt. Dabei wird jedoch die Funktion *Unassemble Parts* nicht mehr als anzupassende Funktion, sondern als Teilfunktion des *AC4BPM*

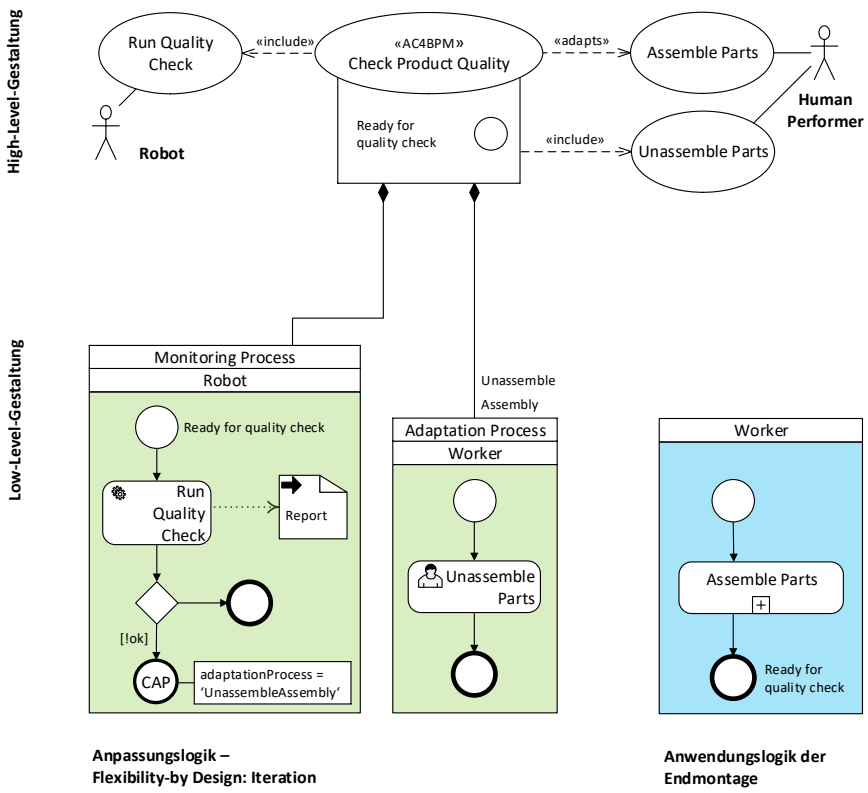


Abbildung 7-7: AC4BPM für das Separation of Business and Adaptivity Logic

verwendet. Der *AC4BPM* beschreibt hierbei, dass im Fall einer unzureichenden Qualität des montierten Produkts eine Anpassung an der Montage durchgeführt werden soll. Ob dieser Fall eingetreten ist, kann durch die Funktion *Run Quality Check* detektiert werden. Sie wird als ein Teil des *AC4BPM* verwendet. Ausführender Akteur bleibt dabei *Robot*.

Als wesentlicher Unterschied ist jedoch zu bemerken, dass diese Funktion nicht mehr als Teil des Hauptprozesses dargestellt wird und somit bereits eine Trennung von unterschiedlichen Aspekten stattgefunden hat. Als weiterer Teil des *Separation-of-Concerns* bleibt die Funktion *Unassemble Parts*, welche zur Demontage eingesetzt wird. Sie wird ebenfalls als Teil des *AC4BPM* verwendet, da hierdurch eine Anpassung am montierten Produkt vorgenommen werden soll. Sie wird daher im Rahmen des dargestellten Anpassungsprozesses eingesetzt. Ausführender Akteur bleibt hier der menschliche Akteur *Human Performer* bzw. die Rolle *Worker*.

Low-Level-Gestaltung SoC

Das auf der Ebene der *Low-Level-Gestaltung* beschriebene Verhalten des *AC4BPM* beschreibt dabei die Qualitätssicherung durch den Roboter im Rahmen des Beobachtungsprozesses. Die Demontage wird in dem Anpassungsprozess durch die Rolle *Worker* übernommen. Dies lässt sich für den Fall der Qualitätssicherung durch den Roboter derartig begründen, dass hierdurch Bedingungen zur Auslösung einer Anpassung ausgewertet werden. Diese Auswertung stellt dabei exakt den für das Konzept des Beobachtungsprozesses angedachten Zweck dar, der durch die beiden Funktionen *Monitor* und *Analyze* aus *MAPE-K* [KC03] angedacht ist. Im Fall des hier dargestellten Anpassungsprozesses kann von einer speziellen Art der Kompensation gesprochen werden. Die dargestellte Funktion *Unassemble Assembly* stellt dabei eine Maßnahme zur Rückabwicklung und somit eine Anpassung selbst dar.

Ergänzung

Neben dem zuvor gezeigten Beispiel für die Verbesserung von bestehenden Prozessen auf Basis von *Flexibility-by Design* und dem Aspekt *Iteration* lassen sich in der Praxis auch bestehende Prozesse finden, auf die weitere in dieser Arbeit aufgeführte Entwurfsmuster für flexible und anpassbare Prozesse anwendbar sind. Für diese bestehenden Prozesse können unter Verwendung der Perspektive des *Adaptivity Engineering* in weiteren Iterationen des *BPM-Lebenszyklus* Verbesserungen erzielt werden, in dem generelle Beschreibungen aus Kapitel 5 als Leitfaden für die Gestaltung verwendet werden.

7.1.5 Zusammenfassung

In dem vorangegangenen Abschnitt wurden Beispiele für den Einsatz von den in dieser Arbeit vorgestellten Lösungsansätzen in Form der Sprache *ACML4BPM*, den Entwurfsmustern sowie der zugehörigen Methode *Adapt Cases 4 BPM* in Anlehnung an ein Szenario gegeben. Dabei wurden insgesamt drei Fälle beschrieben, für die das Ziel der Plausibilisierung von Lösungsteilen angedacht war.

Die beschriebenen Fälle des Szenarios stellen einfache aber aussagekräftige Beispiele für die Verwendung des vorgestellten Ansatzes auf unterschiedlichen Ebenen der Gestaltung von flexiblen und anpassbaren Prozessen dar. So wurde veranschaulicht, wie auf den Ebenen der *High-Level-Gestaltung* und der *Low-Level-Gestaltung* ausgesuchte Beispiele von

Sprachelementen verwendet werden können. In der *High-Level-Gestaltung* werden von dem System und seiner Umgebung zu realisierende Funktionen in Form von *UML Use Cases* (Anwendung) und in Form von *Adapt Case 4 BPMs* (Anpassung) beschrieben. Dabei wurde gezeigt, wie in Bezug zu an der Praxis orientierten Beispielen ein Zusammenhang zwischen Anpassungs- und Anwendungslogik beschrieben werden kann. In einer mit der *High-Level-Gestaltung* integrierten Darstellung wurde zudem die Verfeinerung von *AC4BPM* in Form von entsprechenden Beobachtungs- und Anpassungsprozessen beschrieben. Hierbei wurden neben unterschiedlichen Arten der Auslösung von Beobachtungsprozessen auch Operationen zur Anpassung von Prozessen (siehe Abschnitt 4.3.3) im Rahmen von Anpassungsprozessen verwendet. Neben der Neugestaltung von Teilen der Anpassungslogiken in den Fällen *1* und *2* wurde in *Fall 3* zudem beschrieben, wie auch bestehende Prozesse in Unternehmen so gestaltet werden können, dass Anpassungs- und Anwendungslogik voneinander getrennt gestaltet werden können. Die drei betrachteten Fälle des Szenarios fokussieren dabei jeweils nicht die tiefer gehenden Analysemethoden oder -techniken zur Identifikation einer Situation, in der die Notwendigkeit zur Anpassung besteht, sondern die Trennung der Anpassungs- von der Anwendungslogik. Dennoch wurde sich für Beispiele entschieden, die so auch in der Praxis vorkommen können.

Das vorgestellte Szenario bildet zudem eine Grundlage für die Bewertung von weiteren Kriterien zur Evaluation (siehe Abschnitt 7.2). Für einen konkreten Bezug zwischen dem Szenario und der Bewertung der Evaluationskriterien wird an dieser Stelle auf Abschnitt 7.4 verwiesen.

7.2 Kriterien

In diesem Abschnitt werden insgesamt zwei Kataloge von Kriterien zur Bewertung von vorgestellten Konzepten eingeführt. Der erste Katalog adressiert einzelne Lösungsteile von *Adapt Cases 4 BPM* in Bezug zu dem Aspekt der Anpassbarkeit (engl. *Adaptivity*) auf Basis des durch *Andersson et. al* [And+09] vorgestellten Ansatzes. *Andersson et. al* führen dabei einen Katalog von verschiedenen Eigenschaften zur Bewertung der Modellierbarkeit von selbst-adaptiven Systemen ein. Dabei wird nachfolgend auf die Gruppen von Eigenschaften *Goals*, *Change*, *Mechanisms* und *Effects* eingegangen.

Hier orientiert sich das Vorgehen zur Bewertung maßgeblich an der durch *Luckey* [LE13] bzw. *Biser* [Bis11] vorgestellten Evaluation des Ansatzes

Adapt Cases. Hierdurch wird ein Vergleich zu dem ursprünglichen Ansatz *Adapt Cases* ermöglicht. Dies lässt sich damit begründen, dass es sich bei *Adapt Cases 4 BPM* um eine domänenspezifische Redefinition des zuvor genannten Ansatzes handelt. So soll durch die Evaluation festgestellt werden, wie sich Ausprägungen von einzelnen Spracheigenschaften des generelleren Ansatzes *Adapt Cases* in dem hier vorgestellten Ansatz *Adapt Cases 4 BPM* darstellen.

Der zweite Katalog basiert auf den in Kapitel 1 beschriebenen Anforderungen an den Ansatz *Adapt Cases 4 BPM*. Dabei werden für die Sprache *ACML4BPM* Kriterien verwendet, die sich anteilig auch in einem Vergleich mit dem Ansatz *Adapt Cases* verwenden lassen. Für Anforderungen hinsichtlich der Entwurfsmuster für flexible Prozesse wird zudem ein Vergleich mit existierenden Arbeiten aus der Domäne *BPM* beschrieben.

7.2.1 Kriterien der Anpassbarkeit

In diesem Abschnitt werden Evaluationskriterien für den Aspekt der Anpassbarkeit (engl. *Adaptivity*) vorgestellt. Sie basieren dabei auf den durch *Andersson et al.* [And+09] beschriebenen Gruppen von Dimensionen für die Gestaltung von selbst-adaptiven Systemen. Es wurde sich für den vorliegenden Katalog von Evaluationskriterien entschieden, weil er bereits für die Evaluation von *ACML* in [Bis11] eingesetzt worden ist. Das Ziel, einen Vergleich zwischen den beiden Ansätzen *Adapt Cases* und *Adapt Cases 4 BPM* geben zu können, kann hierdurch unterstützt werden. Nachfolgend werden die Gruppen von Dimensionen *Goals*, *Change*, *Mechanisms* und *Effects* für die Gestaltung von selbst-adaptiven Systemen vorgestellt. Für eine detaillierte Beschreibung einzelner Dimensionen wird auf [And+09] verwiesen.

Auf Basis der zuvor genannten Gruppen von unterschiedlichen Dimensionen der Gestaltung von selbst-adaptiven Systemen werden in einem späteren Teil der Arbeit (siehe Abschnitt 7.4.1) Bewertungen hinsichtlich der Ausdrucksfähigkeit des Ansatzes *Adapt Cases 4 BPM* vorgenommen. Hierfür wurde bereits in der Arbeit von *Biser* [Bis11] eine Reihe von verschiedenen Fragen ausgearbeitet, die im Rahmen der Evaluation von *Adapt Cases* verwendet worden sind. Sie wurden daher als Grundlage für eine Redefinition von Fragen wiederverwendet, mit der eine Bewertung des vorliegenden Ansatzes durchgeführt wird. Wie bereits in [Bis11] verfahren worden ist, werden die zuvor referenzierten Dimensionen als Ausgangspunkt für die Evaluation verwendet.

Gruppe Goals

Die Gruppe *Goals* enthält Dimensionen, mit denen sich Ziele beschreiben lassen, für deren Erfüllung ein selbst-adaptives System einzelne oder eine Reihe von Anpassungen durchführt. Hiervon kann sowohl das System selbst aber auch seine Umgebung betroffen sein. Ein Ziel lässt sich nach *Andersson et al.* auch als eine Komposition verschiedener Teilziele beschreiben, für die sich jeweils spezifische Anpassungen sowohl sequentiell als auch parallel zur Erreichung des übergeordneten Ziels anwenden lassen. Durch [And+09] werden für Ziele die Dimensionen *Evolution*, *Flexibility*, *Duration*, *Multiplicity* und *Dependency* aufgeführt. Die abgeleiteten Evaluationskriterien für diese Dimensionen werden in Tabelle 7-1 dargestellt.

Tabelle 7-1: Evaluationskriterien für die Gruppe Goals

ID	Dimension	Frage
EK1	Evolution	Kann in der Sprache ACML4BPM ausgedrückt werden, dass die Ziele eines Systems statisch oder dynamisch sind?
EK2	Flexibility	Kann in der Sprache ACML4BPM der Grad an Unbestimmtheit eines Ziels formuliert werden?
EK3	Goal Duration	Kann die Validität eines Ziels über die Lebenszeit des Systems beschrieben werden?
EK4	Multiplicity	Können einzelne oder mehrere Ziele beschrieben werden?
EK5	Dependency	Ist es möglich, voneinander abhängige oder unabhängige Ziele sowie, falls vorhanden, deren Relation zu beschreiben?

Gruppe Change

Die Gruppe *Change* beschreibt Dimensionen in Hinsicht auf die Auslösung (engl. *Trigger*) einer Anpassung, die in dem selbst-adaptiven System oder seiner Umgebung stattfinden soll. Derartige Auslöser können durch eine Veränderung am System selbst oder in seiner Umgebung gegeben sein und lassen sich z.B. durch Ereignisse beschreiben. Im Rahmen der Gruppe *Change* werden die Dimensionen *Source*, *Type*, *Frequency* und *Anticipation* aufgeführt. Die abgeleiteten Evaluationskriterien für diese Dimensionen werden in Tabelle 7-2 dargestellt.

Tabelle 7-2: Evaluationskriterien für die Gruppe Change

ID	Dimension	Frage
EK6	Source	Kann in der Sprache ACML4BPM die ausgehende Quelle für eine Auslösung entlang der beiden Grade extern und intern unterschieden werden?
EK7	Change Type	Ist es möglich, die Art einer Auslösung hinsichtlich der Typen funktional, nicht funktional oder technologisch zu beschreiben?
EK8	Frequency	Kann die Häufigkeit einer Auslösung beschrieben werden?
EK9	Anticipation	Kann die Vorhersage einer Auslösung beschrieben werden?

Gruppe Mechanisms

Die dritte Gruppe *Mechanisms* beschreibt Dimensionen hinsichtlich von Anpassungen an einem selbst-adaptiven System oder seiner Umgebung (siehe Tabelle 7-3). Dabei werden derartige Anpassungen auf Basis zuvor aufgekommener Auslösungen ausgeführt. Die durch die Gruppe *Mechanisms* gegebenen Dimensionen *Autonomy*, *Organization* und *Triggering* charakterisieren diese Anpassungen.

Tabelle 7-3: Evaluationskriterien für die Gruppe Mechanisms

ID	Dimension	Frage
EK10	Mechanism Type	Können durch das Konzept Adapt Case 4 BPM Anpassungen in Bezug zu Parametern von Systemkomponenten oder von der Struktur des Systems beschrieben werden?
EK11	Autonomy	Wie autonom sind Anpassungen, die durch das Konzept Adapt Case 4 BPM beschrieben werden können?
EK12	Organization	Kann eine Anpassung durch eine Komponente oder durch mehrere Komponenten durchgeführt werden?
EK13	Scope	Ist es möglich, zu beschreiben, ob eine Anpassung lokale Eigenschaften oder das gesamte System betrifft?
EK14	Mechanism Duration	Ist es möglich, zu beschreiben wie lange der Vorgang einer Anpassung dauern wird?
EK15	Timeliness	Kann durch das Konzept Adapt Case 4 BPM beschrieben, werden, ob eine bestimmte Zeitdauer für eine Anpassung garantiert werden kann?
EK16	Triggering	Ist es möglich, Anpassungen zu beschreiben, die durch ein Ereignis oder durch eine zeitliche Bedingung ausgelöst werden?

Gruppe Effects

In der vierten und letzten Gruppe *Effects* sind Dimensionen enthalten, die den Einfluss einer Anpassung auf das selbst-adaptive System oder seine Umgebung beschreiben. So sind bspw. Charakterisierungen möglich, die eine Anpassung als kritisch, vorhersehbar, aufwändig oder das System bzw. seine Umgebung als widerstandsfähig darstellen. Die entsprechenden Dimensionen sind *Criticality*, *Predictability*, *Overhead* und *Resilience*, für die die abgeleiteten Evaluationskriterien in Tabelle 7-4 dargestellt sind.

Tabelle 7-4: Evaluationskriterien für die Gruppe Effects

ID	Dimension	Frage
EK17	Criticality	Kann die Konsequenz einer gescheiterten Anpassung durch das Konzept Adapt Case 4 BPM beschrieben werden?
EK18	Predictability	Ist es möglich, die Vorhersage einer Konsequenz einer durchgeführten Anpassung durch das Konzept Adapt Case 4 BPM zu beschreiben?
EK19	Overhead	Kann eine negative Konsequenz einer Anpassung beschrieben werden?
EK20	Resilience	Kann die Lebensdauer einer vertrauenswürdigen Diensterbringung beschrieben werden?

7.2.2 Kriterien für die Anforderungen an Adapt Cases 4 BPM

In Tabelle 7-5 werden in Anlehnung an die beschriebenen Anforderungen an den Ansatz *Adapt Cases 4 BPM* (siehe Abschnitt 1.3) zugehörige Evaluationskriterien dargestellt. Durch die Bewertung werden zwei Ziele verfolgt. Zum einen soll ebenso wie zuvor für den Aspekt der Anpassbarkeit (siehe Abschnitt 7.2.1) ein Vergleich zwischen den beiden Sprachen *ACML* und *ACML4BPM* durchgeführt werden. Zum anderen sind domänenspezifische Anforderungen an den Ansatz *Adapt Cases 4 BPM* gesetzt worden, die nicht mit *Adapt Cases* verglichen werden können. Für derartige Anforderungen soll ein Vergleich mit Arbeiten aus der Domäne *BPM* durchgeführt werden.

Tabelle 7-5: Evaluationskriterien für die Anforderungen an Adapt Cases 4 BPM

ID	Dimension	Frage
EK21	Separation-of-Concerns	Ist eine Trennung von Anpassungs- und Anwendungslogik durch den Ansatz Adapt Cases 4 BPM möglich?
EK22	Kontrollschleife	Wie gut können Kontrollschleifen durch die Verwendung von ACML4BPM gestaltet werden?
EK23	Ausdrucks-fähigkeit	Wie hoch ist der Grad der Ausdrucksfähigkeit von der Sprache ACML4BPM im Vergleich zu der Sprache ACML und UML Use Cases?
EK24	UML-Konsistenz	Wie gut ist die Konsistenz gegenüber der Sprache UML?
EK25	BPMN2.0-Konsistenz	Wie hoch ist die Konsistenz zwischen den beiden Sprachen ACML4BPM und BPMN2.0?
EK26	Musterbasierte Unterstützung	Können verschiedene Aspekte von Flexibilität durch die erarbeiteten Entwurfsmuster unterstützt werden?
EK27	Integration	Ist der Ansatz Adapt Cases 4 BPM entlang einer spezifischen Methode der Domäne BPM einsetzbar?

Ein Vergleich mit der Sprache *ACML* kann auf Basis der Evaluationskriterien *EK21, EK22, EK23* sowie *EK24* durchgeführt werden. Dies lässt sich dadurch begründen, dass sie bereits in der Evaluation des Ansatzes *Adapt Cases* verwendet worden sind und einen Teil der Anforderungen umfassen, die auch für den vorliegenden Ansatz gesetzt worden sind. Die zugehörige Bewertung dieser Kriterien kann daher im Rahmen der hier durchgeführten Evaluation zum Zweck des Vergleichs wiederverwendet werden.

Weitere aufgeführte Evaluationskriterien sind für die Bewertung von Erfüllungsgraden der Anforderungen an den in dieser Arbeit vorgestellten Ansatz *Adapt Cases 4 BPM* vorgesehen. Basierend auf den vorgestellten Evaluationskriterien soll im Rahmen einer Selbstevaluation eine Bewertung von Erfüllungsgraden durchgeführt und offene Punkte identifiziert werden.

7.3 Bewertungseinheit

Für die Bewertung von Kriterien zur Evaluation der einzelnen Lösungsteile wird in diesem Abschnitt ein Skalenniveau eingeführt. Zum besseren Vergleich der beiden Sprachen *ACML* und *ACML4BPM* wurde sich für eine Ordinalskala entschieden. Dabei wird maßgeblich die gleiche Ausprägung verwendet, wie sie bereits in [Bis11] für die Evaluation der Sprache *ACML* eingeführt wurde.

Zur Übersicht ist die verwendete Ordinalskala in Tabelle 7-6 gezeigt. Sie besteht aus insgesamt sechs geordneten Kategorien, welche durch natürlichsprachliche und numerische Werte identifizierbar sind. Sie reicht vom niedrigsten Wert 0 (*nicht gesetzt*) bis hin zum höchsten Wert 10 (*exzellent*). Die Semantik der einzelnen Kategorien ist ebenso in Anlehnung an [Bis11] übernommen worden und in Tabelle 7-6 entsprechend dargestellt.

Eine Ergänzung zu der durch *Biser* verwendeten Kategorien ist jedoch durch den Spezialfall des Wertes 0 (*nicht gesetzt*) gegeben. So bestanden bestimmte Anforderungen an den Ansatz *Adapt Cases* zum Zeitpunkt seines Entwurfs nicht, sodass ein Vergleich mit bestehenden Werten im weiteren Verlauf nicht möglich ist. Auf eine nachträgliche Bewertung in Anlehnung an entsprechende Anforderungen wurde in dieser Arbeit verzichtet. Dabei wird an geeigneter Stelle auf entsprechende Anforderungen aufmerksam gemacht.

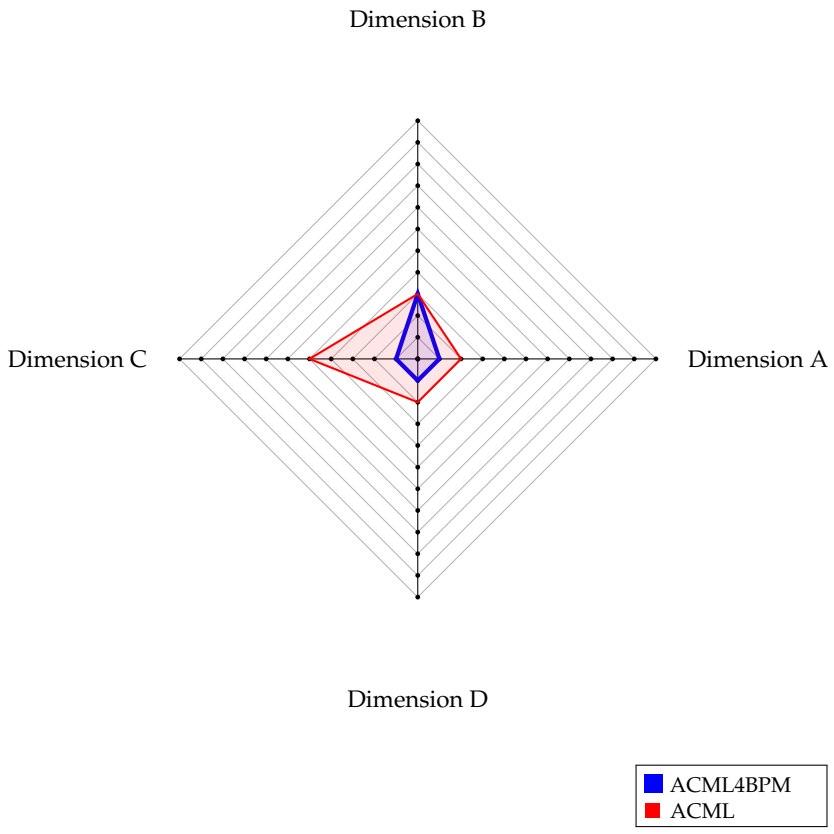
Tabelle 7-6: Bewertungseinheit für Kriterien

Deskriptiv	Numerisch	Semantik
nicht gesetzt	0	Anforderung wurde für den Ansatz nicht gesetzt.
nicht akzeptabel	2	Anforderung kann in keiner Hinsicht erfüllt werden.
moderat	4	Anforderung könnte mit großem zusätzlichen Aufwand realisiert werden.
akzeptabel	6	Anforderung kann nicht erfüllt werden, ist jedoch durch geringe Aufwände zukünftig realisierbar.
gut	8	Anforderung kann mit Abweichung erfüllt werden.
exzellent	10	Anforderung kann vollständig erfüllt werden.

Durch die vorgestellte Bewertungseinheit ist es möglich, den aktuellen Stand der beiden Ansätze *Adapt Cases* und *Adapt Cases 4 BPM* sowie ihrer zugehörigen Sprachen gegenüberzustellen. Für die grafische Darstellung von Ergebnissen der Evaluation werden im Rahmen der Bewertung von Kriterien Netzdiagramme verwendet (siehe Abschnitt 7.4). Dies lässt sich damit begründen, dass sie bereits in der Evaluation von *Adapt Cases* eingesetzt worden sind und so Vergleich ermöglicht wird.

Ein Beispiel für ein solches Netzdiagramm ist in Abbildung 7-8 dargestellt. Insgesamt ist hier die Bewertung von vier verschiedenen Dimensionen gezeigt. Das dargestellte Raster gibt gemäß der in Tabelle 7-6 gezeigten Bewertungseinheit die Bewertung in Zweierschritten für eine Dimension an. Dabei werden Bewertungen für den Ansatz *Adapt Cases 4 BPM* in der Farbe *Blau* und für den Ansatz *Adapt Cases* in der Farbe *Rot* dargestellt.

Abbildung 7-8: Netzdiagramm zur grafischen Darstellung von Ergebnissen



Die hier exemplarisch gezeigte Bewertung beschreibt eine überdeckende Bewertung für die beiden Ansätze hinsichtlich der *Dimension B*. In den weiteren Dimensionen werden exemplarisch Eigenschaften des Ansatzes *ACML* insgesamt als besser bewertet dargestellt.

7.4 Bewertung

In diesem Abschnitt werden Bewertungen vorgestellt, die auf Basis der zuvor in Abschnitt 7.2 vorgestellten Kataloge von Kriterien vorgenommen worden sind. Die Bewertungen von Kriterien des Aspekts der Anpassbarkeit (engl. *Adaptivity*) werden in Abschnitt 7.4.1 vorgestellt. Die Kriterien beruhen dabei auf den durch *Andersson et. al* [And+09] eingeführten Katalog von Eigenschaften zur Bewertung von Fähigkeiten zur Modellierung unterschiedlicher Eigenschaften von selbst-adaptiven Systemen. Dabei wird auf die bereits eingeführten Gruppen dieser Eigenschaften in den zugehörigen Abschnitten 7.4.1.1 bis 7.4.1.4 Bezug genommen.

Anschließend werden Bewertungen für die Anforderungen an den Ansatz *Adapt Cases 4 BPM* in Abschnitt 7.4.2 vorgestellt. Da sich eine Bewertung von unterstützten Aspekten von flexiblen und anpassbaren Prozessen mit *Adapt Cases* nicht sinnvoll in einem Vergleich kombinieren lässt, wird dieser Abschnitt weiter unterteilt. So wird in Abschnitt 7.4.2.1 zunächst allgemein auf die Bewertung für die vorgestellten Anforderungen eingegangen. Dabei wird an sinnvollen Stellen ein Vergleich zum Ansatz *Adapt Cases* vorgenommen. Ergänzend wird in Abschnitt 7.4.2.2 dediziert auf eine Übersicht, eine Bewertung und einen Vergleich von unterstützten Aspekten von flexiblen und anpassbaren Prozessen eingegangen.

7.4.1 Bewertung von Kriterien der Anpassbarkeit

In diesem Abschnitt werden für den Aspekt der Anpassbarkeit Bewertungen vorgestellt. Dabei werden Ergebnisse sowohl tabellarisch als auch grafisch durch Netzdiagramme dargestellt. Zugehörige Daten der Evaluation für die Sprache *ACML* wurden aus der Arbeit von [Bis11] übernommen. Bei den verwendeten Daten der Sprache *ACML4BPM* handelt es sich um eine selbstkritische Selbsteinschätzung, die auf Erfahrungen aus der Erstellung von Beispielen in Bezug zu dem Szenario beruhen (siehe Abschnitt 7.1). Für die durch [And+09] eingeführten Dimensionen zur Gestaltung von selbst-adaptiven Systemen werden zunächst die Ergebnisse der Evaluation der Sprache *ACML4BPM* vorgestellt und diskutiert. Im Anschluss folgt ein Vergleich mit den Ergebnissen für die Sprache *ACML*.

7.4.1.1 Bewertungen für die Gruppe Goals

Die Gestaltung von Zielen wird durch den Einsatz von *ACML4BPM* nicht unterstützt. Ziele können für die zu gestaltenden flexiblen und anpassbaren Prozesse jedoch als eine Art von Anforderungen verstanden werden, für deren Erreichung Funktionen beschrieben werden müssen. So kann ein Prozess, Subprozess oder Task eine solche Funktion zur Erreichung eines Ziels darstellen, die in verschiedene Teilfunktionen bzw. Teilziele zerlegt werden kann. Nachfolgend sind die Bewertungen von Dimensionen der Gruppe *Goals* in Tabelle 7-7 dargestellt. Auf Details und eine zugehörige Diskussion für die Bewertungen von Dimensionen der Gruppe *Goals* wird nachfolgend individuell eingegangen.

Tabelle 7-7: Bewertungen für die Gruppe Goals

ID	Dimension	ACML4BPM	ACML
EK1	Evolution	akzeptabel	akzeptabel
EK2	Flexibility	gut	gut
EK3	Goal Duration	akzeptabel	moderat
EK4	Multiplicity	gut	gut
EK5	Dependency	gut	gut

EK1 In *ACML4BPM* sind keine Sprachelemente zur Gestaltung von Zielen enthalten. Daher können nur implizit statische Ziele durch die Gestaltung von Prozessen bzw. Subprozessen sowie von Teilzielen durch verschiedene Tasks beschrieben werden. Prozesse, Subprozesse und Tasks stellen hier Funktionen dar, die zur Erreichung eines organisatorischen Ziels notwendig sind. Wird ein *AC4BPM* als Beschreibung von Anpassungen von bestehenden Funktionen verwendet, so werden bspw. Funktionen verändert, die der Erreichung des ursprünglichen Ziels unter geänderten Umständen dienen.

Ein Beispiel für die Anpassung eines Prozesses zur weiteren Erreichung seines Ziels ist in Abschnitt 7.1.4 gegeben. Betrachtet man hier als Ziel des Hauptprozesses die Montage eines Produktes unter Einhaltung gewisser Qualitätsanforderungen, so können einzelne Funktionen bzw. Tasks als Maßnahmen zur Erreichung von Teilzielen betrachtet werden. Die in dem Hauptprozess enthaltene *Iteration* stellt dabei sicher, dass ein montiertes Produkt im Fall unzureichender Qualität demontiert und anschließend erneut montiert wird. Das Ziel des Hauptprozesses ist somit statisch. Eine Anpassung von bestehenden Funktionen führt jedoch dazu, dass das jeweilige Teilziel auch als dynamisch betrachtet werden kann.

Die Dimension *Evolution* wird daher als `akzeptabel` realisiert bewertet.

EK2 Bei der Unbestimmtheit handelt es sich um eine Eigenschaft eines Ziels. Da *ACML4BPM* keine Sprachelemente zur Beschreibung von Zielen zur Verfügung stellt, ist die Gestaltung von deren Eigenschaften auch nicht möglich. Jedoch ist es möglich, Funktionen, die zur Realisierung von Teilzielen gestaltet werden, bei Bedarf so anzupassen, dass übergeordnete Ziele und Teilziele auch weiterhin erfüllt werden können. Der hierdurch entstehende Grad an Flexibilität ermöglicht es, auf der Ebene der Gestaltung von Prozessen – insbesondere unter Verwendung der in Kapitel 5 vorgestellten Entwurfsmuster – die Unbestimmtheit eines Ziels umzusetzen.

Ein Beispiel für eine derartige Umsetzung ist in *Fall 2* des Szenarios beschrieben (siehe Abschnitt 7.1.3). Hier ist Flexibilität durch unterschiedliche Auslösungen des *HPWM* gegeben. Dabei wird der Aspekt *Choice* von *Flexibility-by Design* eingesetzt (siehe Abschnitt 5.2.2). Ferner ist Unbestimmtheit auch im Rahmen des Anpassungsprozesses vorgesehen. So ist nicht beschrieben, in welcher Reihenfolge ein nicht zugewiesener menschlicher Akteur zugewiesen wird. Beide Umstände stellen jeweils einen gewissen Grad an Unbestimmtheit dar, auch wenn er selbst nicht durch ein explizites Sprachelement gestaltet worden ist.

Die Dimension *Flexibility* wird daher als `gut` realisiert bewertet.

EK3 Da die Gestaltung von Zielen in *ACML4BPM* nicht explizit möglich ist, kann die Eigenschaft der Validität eines Ziels über die Lebenszeit des Systems nicht beschrieben werden. Es ist jedoch möglich, auf geänderte Eigenschaften der Umgebung zu reagieren. Hierfür wurden eine Reihe von unterschiedlichen Entwurfsmustern vorgestellt, sodass bspw. ungewollte Funktionen, die für die Realisierung von (Teil-)Zielen eines Prozesses verwendet werden, aus dem Prozess entfernt werden können. Dieses Entfernen kann im Rahmen von *Flexibility-by Change* entweder temporär oder permanent durchgeführt werden (siehe Abschnitt 5.3). Ferner können auch bedingte Aufrufe von Funktionen durchgeführt werden, wie es z.B. im Rahmen von *Flexibility-by Design* und dem Aspekt *Choice* beschrieben worden ist (siehe Abschnitt 5.2.2). Die in den Entwurfsmustern vorgesehenen Anpassungen bzw. Aufrufe stellen dabei auch eine Aktion in Hinsicht auf die Eigenschaft der Validität von (Teil-)Zielen dar.

Durch *Fall 3* des Szenarios ist hierzu ein Beispiel gegeben (siehe Abschnitt 7.1.4). So wird eine Funktion zur Demontage eines Produktes nur dann aufgerufen, wenn die Qualität des montierten Produktes nicht ausreichend ist. Dadurch wird eine neue Funktion zur Realisierung eines zuvor nicht vorhandenen Teilziels verwendet. Im Fall eines bedingten Auf-

rufs dieser Funktion verändert sich folglich auch die Validität des zugehörigen Ziels.

Die Dimension *Goal Duration* wird als `akzeptabel` realisiert bewertet.

EK4 Einzelne oder mehrere Ziele können durch *ACML4BPM* nicht explizit gestaltet werden. Stattdessen können aber mehrere *AC4BPM* beschrieben werden, die für die Erreichung gleicher oder unterschiedlicher Ziele verwendet werden. Derartige *AC4BPM* können auch mit weiteren Funktionen des Systems in Relation stehen (siehe *EK5 Dependency*).

Sofern Funktionen als Maßnahmen zur Erreichung von (Teil-)Zielen betrachtet werden, sind entlang der drei Fälle des Szenarios verschiedene Beispiele auf der Ebene der *High-Level-Gestaltung* gegeben. So ist es z.B. möglich, dass eine Funktion durch eine weitere Funktion angepasst wird (siehe Abschnitt 7.1.2). Funktionen können aber auch andere Funktionen enthalten (siehe Abschnitt 7.1.3). Je nach Anforderungen können dabei beliebig viele Funktionen beschrieben werden, die für die Erreichung eines oder mehrerer Ziele notwendig sind.

Die Dimension *Multiplicity* wird daher als `gut` realisiert bewertet.

EK5 Durch *ACML4BPM* ist es möglich, verschiedene Funktionen zur Realisierung von Zielen in eine gegenseitige Abhängigkeit zu stellen. So können durch die Verwendung verschiedener Ereignisse oder durch den Aufruf weiterer Funktionen derartige Abhängigkeiten implizit dargestellt werden. Eine explizite Darstellung ist auf der Ebene der *High-Level-Gestaltung* durch verschiedene Assoziationen wie «*include*», «*adapts*» oder «*extends*» möglich.

Durch *Fall 2* des Szenarios ist hierfür ein Beispiel gegeben (siehe Abschnitt 7.1.3). So wird die Funktion *Analyze Workload Profile* im Rahmen des Beobachtungsprozesses des *AC4BPM HPWM* eingesetzt. Hierdurch ist eine Art der Abhängigkeit beider Funktionen beschrieben, die auf der Ebene der *High-Level-Gestaltung* durch die Assoziation «*include*» beschrieben werden kann.

Die Dimension *Dependency* wird daher als `gut` realisiert bewertet.

Grafische Darstellung und Diskussion

Ergänzend zu der vorherigen textuellen Beschreibung der Bewertungen von Dimensionen der Gruppe *Goals* ist zur besseren Übersicht zudem eine grafische Darstellung in Abbildung 7-9 gegeben. Werden die Bewertungen miteinander verglichen, so fällt eine hohe Überdeckung auf. Dies wird damit begründet, dass die zugrunde liegenden Konzepte in der Sprache

ACML im Rahmen des Entwurfs der Sprache *ACML4BPM* redefiniert worden sind. Dabei wurden wesentliche Konzepte übernommen, sodass in der Bewertung der Dimensionen der Gruppe *Goals* die vorliegende Überdeckung sogar zu erwarten war.

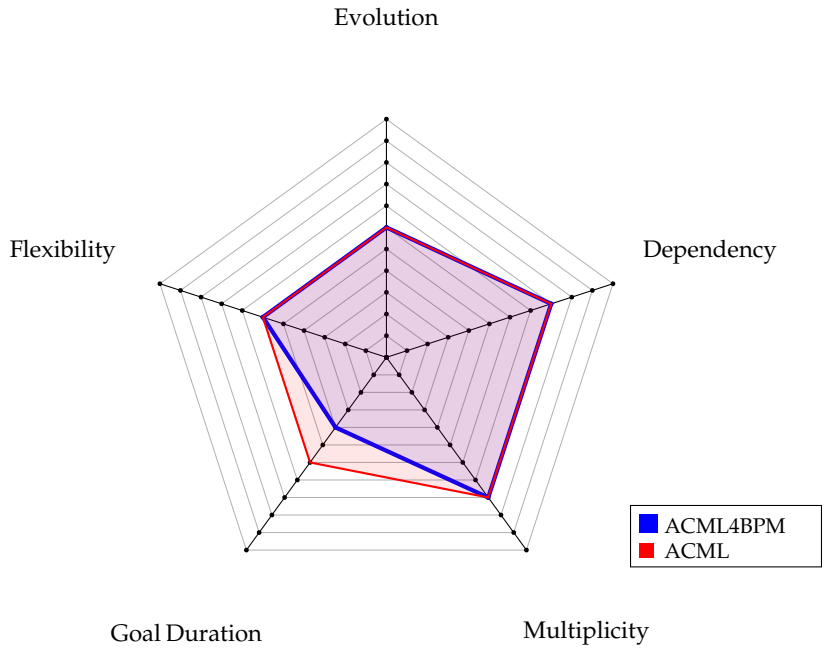


Abbildung 7-9: Netzdiagramm für die Gruppe Goals

Eine Ausnahme bildet jedoch das Kriterium *EK3* in Form der Dimension *Goal Duration*. Dies lässt sich damit begründen, dass in der vorliegenden Arbeit verschiedenste Entwurfsmuster für Aspekte von Flexibilität in der Domäne *BPM* vorgestellt wurden (siehe Kapitel 5). Hierdurch entstehen erweiterte Fähigkeiten des Ansatzes *Adapt Cases 4 BPM*, sodass das Kriterium *EK3* besser bewertet werden kann. Anhand des für die Bewertung von *EK3* gegebenen Beispiels wird eine dieser Fähigkeiten deutlich. So kann durch einen *AC4BPM* implizit die Validität eines Ziels über die Lebenszeit des Systems beschrieben werden, indem temporär oder evolutionär Anpassungen von Prozessen, wie z.B. in Form eines Hinzufügens oder Entfernens einer Funktion, beschrieben werden.

7.4.1.2 Bewertungen für die Gruppe Change

Die Gruppe *Change* enthält Dimensionen in Hinsicht auf mögliche Auslösungen für eine Anpassung am System oder an seiner Umgebung. Dabei können Auslösungen sowohl von dem System selbst als auch in seiner Umgebung vorkommen. In der Sprache *ACML4BPM* können Auslöser durch Ereignisse im Rahmen des *Adaptation View Model 4 BPM*

(AVM4BPM) beschrieben werden (siehe Abschnitt 4.3.4). Derartige Ereignisse sind Teil der Beschreibung von Schnittstellen, die durch System- und Umgebungskomponenten angeboten werden. Nachfolgend sind die Bewertungen von Dimensionen der Gruppe *Change* in Tabelle 7-8 dargestellt. Auf Details und eine zugehörige Diskussion für die dargestellten Bewertungen von Dimensionen der Gruppe *Change* wird nachfolgend individuell eingegangen.

Tabelle 7-8: Bewertungen für die Gruppe Change

ID	Dimension	ACML4BPM	ACML
EK6	Source	exzellent	exzellent
EK7	Change Type	exzellent	exzellent
EK8	Frequency	akzeptabel	akzeptabel
EK9	Anticipation	akzeptabel	moderat

EK6 Ereignisse werden als Teil einer Sensorschnittstelle beschrieben, die durch System- oder Umgebungskomponenten angeboten wird. Die in Sensorschnittstellen von Systemkomponenten beschriebenen Ereignisse lassen sich als *intern* bezeichnen, wohingegen Ereignisse aus Sensorschnittstellen von Umgebungskomponenten als *extern* zu betrachten sind.

In dem in Abschnitt 7.1.1 vorgestellten *AVM4BPM* des Szenarios ist eine derartige Quelle durch die Umgebungskomponente *HumanPerformer* gegeben. Sie bietet die Sensorschnittstelle *HumanMachineInterface* an, in der das Ereignis vom Typ *AdaptationRequestEvent* mit der Bezeichnung *ManualReallocationRequest* beschrieben ist. Da es sich hier um eine Umgebungskomponente handelt, ist dieses Ereignis als extern zu bezeichnen.

Die Dimension *Source* wird daher als `exzellent` realisiert bewertet.

EK7 In der Sprache *ACML4BPM* kann ein breites Spektrum von potentiellen Auslösungen berücksichtigt werden.

In dem in Abschnitt 7.1.3 gegebenen *Fall 2* des Szenarios sind unterschiedliche Typen von Auslösern gegeben. So ist sowohl eine zeitgesteuerte als auch eine manuelle Auslösung einer Anpassung des Hauptprozesses möglich. Beide Auslösungen können sowohl als funktional als auch als nicht funktional betrachtet werden. So sind sie zum einen auslösend für die Durchführung einer Anpassung und zum anderen dienen sie dem Zweck der Menschenzentrierung innerhalb des *Shared-Workspace*.

Die Dimension *Change Type* wird als `exzellent` realisiert bewertet.

EK8 Durch das Konzept *AC4BPM* ist es nicht möglich, die Häufigkeit des Auftretens einer Auslösung zu beschreiben. Durch zeitbasierte Ereignisse kann aber ein Intervall oder ein bestimmter Zeitpunkt der Ausführung des Verhaltens eines Beobachtungs- und Anpassungsprozesses beschrieben werden.

Ein Beispiel für ein derartiges Ereignis wird in Fall 1 des Szenarios gegeben (siehe Abschnitt 7.1.2). Hier wird ein Ereignis vom Typ *TimerEvent* dazu verwendet, dass der zugehörige Beobachtungsprozess alle *20 Minuten* ausgeführt werden soll.

Durch diese nur geringe Einschränkung wird die Dimension *Frequency* als `akzeptabel` realisiert eingeschätzt.

EK9 Eine Vorhersage des Auftretens von Auslösern einer Anpassung kann durch ein eigenes Element in der Sprache *ACML4BPM* nicht dargestellt werden. Durch *Andersson et al.* [And+09] werden die genannten Typen von Vorhersagen *foreseen*, *foreseeable* und *unforeseen* betrachtet. Für diese Typen ist jedoch in Beobachtungs- und Anpassungsprozessen die Gestaltung von spezialisierten Funktionen in Hinsicht auf die Analyse der bisherigen Historie der Ausführung möglich. Auch wenn kein explizites Sprachelement vorhanden ist, so können derartige Funktionen dennoch in der Gestaltung Berücksichtigung finden. Neben den zuvor genannten Typen zählen *Andersson et al.* aber auch die Behandlung von Fehlern zu den Eigenschaften der Dimension *Anticipation*. Eine solche Behandlung sollte stets in einer Gestaltung von flexiblen Prozessen mitberücksichtigt werden. Sie stellt neben vielen anderen Gründen für die Durchführung einer Anpassung einen wichtigen Beitrag zur Gewährleistung der ordnungsgemäßen Funktion des Gesamtsystems dar.

Die Behandlung von auftretenden Fehlern kann durch Beobachtungs- und Anpassungsprozesse gestaltet werden. Hierzu können entsprechende Maßnahmen bereits in der Neugestaltung oder in nachfolgenden Iterationen des *BPM-Lebenszyklus* hinzugefügt werden. Dabei bietet sich die Verwendung des Flexibilitätsaspekts *Flexibility-by Underspecification* und seiner Untertypen *Late Selection* bzw. *Late Modeling* (siehe Abschnitt 5.5.1) besonders an. So lassen sich im Bedarfsfall Funktionen dynamisch auch zur Laufzeit gestalten und binden, sodass adäquat auf einen aufgekommenen Fehler reagiert werden kann. Die Verwendung dieser Entwurfsmuster stellt selbst eine implizite Vorhersage für auftretende Ereignisse dar, die jeweils als *foreseen* oder *foreseeable* betrachtet werden können. Dies lässt sich dadurch begründen, dass unter Einsatz dieser Entwurfsmuster bereits

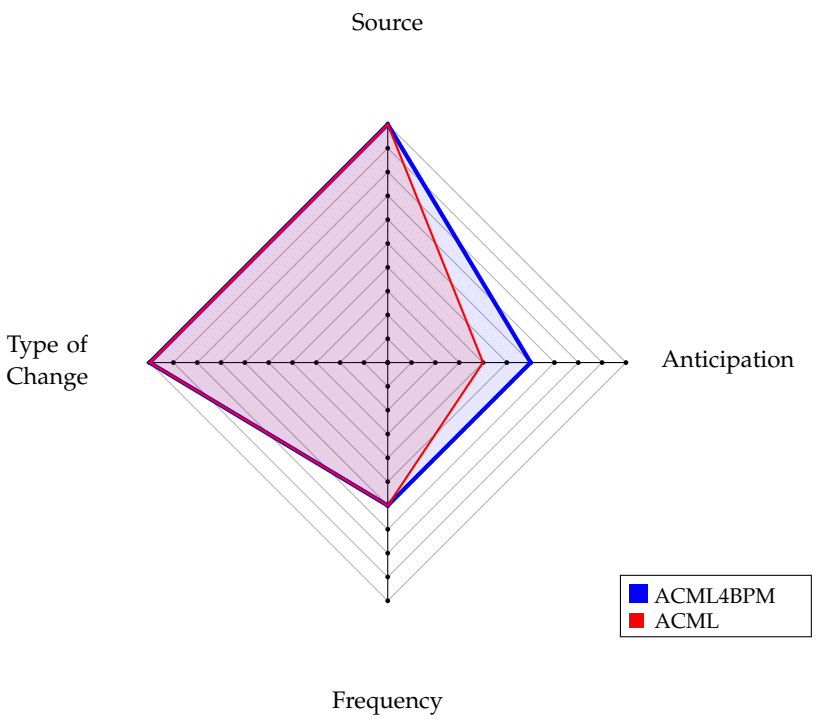
zum Zeitpunkt der Gestaltung Situationen berücksichtigt werden können, die eine dynamische Bindung von Prozessfragmenten *(Late Selection)* ermöglichen oder aber das Aufkommen von undefinierten Situationen *(Late Modeling)* berücksichtigen.

Die Dimension *Anticipation* wird daher als `akzeptabel` realisiert bewertet.

Grafische Darstellung und Diskussion

Eine grafische Darstellung der zuvor beschriebenen Ergebnisse für die Gruppe *Change* ist in Abbildung 7-10 gegeben. Wie zuvor bereits für die Gruppe *Goals* beschrieben, besteht auch hier eine hohe Überdeckung in den Bewertungen. Es existiert in beiden Sprachen kein dediziertes Sprachelement zur Beschreibung der Vorhersage des Auftretens einer Auslösung. Dabei können sowohl im Rahmen der *Monitoring Activity* (*ACML*) als auch im Rahmen des Beobachtungsprozesses (*ACML4BPM*) erweiterte Funktionalitäten beschrieben werden, die eine derartige Vorhersage, wie z.B. auf Basis einer Historie der Ausführung, ermöglichen.

Abbildung 7-10: Netzdiagramm für die Gruppe Change



Eine Unterscheidung lässt sich hier lediglich für die Dimension *Anticipation* erkennen. Dies lässt sich dadurch erklären, dass im Kontext der Methode *Adapt Cases 4 BPM* auch Entwurfsmuster für unterschiedlichste Aspek-

te von Flexibilität vorgestellt worden sind. So kann, wie zuvor beschrieben, z.B. durch die Verwendung der Entwurfsmuster *Late Selection* oder *Late Modeling* eine implizite Vorhersage des Auftretens einer Auslösung beschrieben werden. Die Entwurfsmuster stellen daher in Bezug zur Dimension *Anticipation* einen Mehrwert dar, sodass die Bewertung entsprechend besser ausfallen kann.

7.4.1.3 Bewertungen für die Gruppe Mechanisms

Die Dimensionen der Gruppe *Mechanisms* betreffen – basierend auf aufkommenden Auslösern der Dimension *Change* – durchzuführende Anpassungen an Eigenschaften eines Systems bzw. seiner Umgebung. Unter der Verwendung der Sprache *ACML4BPM* werden in diesem Bezug typischerweise zunächst Entscheidungen im Rahmen von Beobachtungsprozessen getroffen. Als Folge einer Entscheidung können Anpassungsprozesse aufgerufen werden, in denen Operationen zur Anpassung von Prozessen und deren Umgebung ausgeführt werden können. Nachfolgend sind die Bewertungen von Dimensionen der Gruppe *Mechanisms* in Tabelle 7-9 dargestellt. Auf Details und eine zugehörige Diskussion für die dargestellten Bewertungen von Dimensionen der Gruppe *Mechanisms* wird nachfolgend individuell eingegangen.

ID	Dimension	ACML4BPM	ACML
EK10	Mechanism Type	gut	gut
EK11	Autonomy	gut	moderat
EK12	Organization	gut	gut
EK13	Scope	nicht akzeptabel	nicht akzeptabel
EK14	Mechanism Duration	nicht akzeptabel	nicht akzeptabel
EK15	Timeliness	moderat	nicht akzeptabel
EK16	Triggering	exzellent	exzellent

Tabelle 7-9: Bewertungen für die Gruppe Mechanisms

EK10 In *ACML4BPM* ist keine Anpassung der Struktur des Systems oder seiner Umgebung möglich, sofern hiermit eine Anpassung der System- und Umgebungsarchitektur gemeint ist. Unter derartigen Anpassungen kann das Hinzufügen, Entfernen oder Modifizieren von System- oder Umgebungskomponenten verstanden werden, die im Rahmen des *Adaptation View Model 4 BPM (AVM4BPM)* beschrieben worden sind.

Stattdessen können jedoch verschiedenste Eigenschaften (hier: *Parameter*), die durch System- und Umgebungskomponenten gekapselte Inhalte darstellen, wie z.B. Prozesse, durch entsprechende Operationen angepasst

werden. In *ACML4BPM* werden dabei insgesamt drei Typen von Komponenten unterschieden, für die eine Reihe von Operationen zur Anpassung in Abschnitt 4.3.3 vorgestellt worden sind. Dabei wurden wesentliche anpassbare Elemente in Bezug zu Perspektiven von Prozessen nach *Curtis et. al* [CKO92] identifiziert und eine Menge von zugehörigen Operationen vorgestellt. Die Menge enthält Operationen, mit denen Anpassungen an der organisatorischen Struktur, dem Kontroll- und dem Datenfluss ermöglicht werden können. Im Vergleich zu bereits existierenden Arbeiten, wie z.B. [Sch+08; WRR08; Ger13], sind so erweiterte Fähigkeiten zur Anpassung von Elementen unterschiedlicher Perspektiven vorhanden.

Daher kann hier von einer Realisierung der Dimension *Mechanisms Type* gesprochen werden, die als `gut` angesehen wird.

EK11 *ACML4BPM* unterstützt verschiedene Grade von Autonomie in Hinsicht auf anzuwendende Anpassungen. So kann durch einen *AC4BPM* spezifisches Verhalten beschrieben werden, in dessen Rahmen im Beobachtungsprozess zunächst eine Analyse erfolgt und anschließend eine Entscheidung für die Auswahl einer benötigten Anpassung getroffen wird. Nachfolgend kann ein Anpassungsprozess konkrete Anpassungen durch eine Reihe von Operationen durchführen. Die hierdurch beschriebene Anpassung benötigt dabei nicht zwangsläufig eine Einbindung von weiteren Ressourcen, wie z.B. einen menschlichen Akteur.

Dies kann – wie für *Fall 2* des Szenarios erörtert wurde – jedoch sinnvoll sein und z.B. durch die Verwendung des Entwurfsmusters *Late Modeling* umgesetzt werden. Durch die Verwendung von *ACML4BPM* und der vorgestellten Entwurfsmuster wird somit ein breites Spektrum unterschiedlicher Grade von Autonomie unterstützt. Je nach Anforderung an das System kann die Verwendung eines Entwurfsmusters einen kleineren oder höheren Grad an Autonomie bedeuten.

Da aber sowohl Beobachtungs- als auch Anpassungsprozesse zuvor manuell gestaltet werden müssen, wird die Realisierung der Dimension *Autonomy* nicht als `exzellent`, sondern als `gut` erfüllt betrachtet.

EK12 In *ACML4BPM* können Anpassungen sowohl an gekapselten Inhalten einer einzelnen Komponente (zentral) als auch an mehreren Komponenten (dezentral) durchgeführt werden. So können in einem Anpassungsprozess Operationen zur Anpassung verschiedenster Komponenten verwendet werden. Der Anpassungsprozess orchestriert dabei die jeweilige Fähigkeit zur Anpassung durch die verwendeten Operationen der beteiligten Komponenten.

Verschiedene Beispiele für eine einfache Orchestrierung von Anpassungen sind in *Fall 1* und *Fall 2* des Szenarios gegeben. Hier enthält der verwendete Anpassungsprozess eine Operation zur Anpassung der Eigenschaft *Performer* eines Tasks. Die Fähigkeit der Anpassung ist demnach durch die Modifikation dieser Eigenschaft gegeben.

Dabei ist die Herkunft einer Operation zur Anpassung nicht alleine auf Komponenten zur Kapselung von Prozessen und deren Instanzen beschränkt. Wie in Abschnitt 4.3.2 bereits beschrieben, können z.B. auch die Effektorschnittstellen von Umgebungskomponenten dedizierte Operationen zur Anpassung ihrer gekapselten Inhalte anbieten. Der in einem Anpassungsprozess enthaltene Kontrollfluss kann dabei aus Operationen verschiedener Komponenten bestehen, die im Rahmen des *AVM4BPM* beschrieben worden sind. Somit können durch einen *AC4BPM* komplexe und verteilte Anpassungen am System und seiner Umgebung beschrieben werden.

Die Erfüllung der Realisierung der Dimension *Organization* kann damit als `gut` angegeben werden.

EK13 Die Sprache *ACML4BPM* enthält keine Sprachelemente zur Gestaltung einer zentralisierten oder globalen Rekonfiguration in Hinsicht auf das Gesamtsystem.

Somit ist die Bewertung der Dimension *Scope* als `nicht akzeptabel` vorgenommen worden.

EK14 Durch die Sprache *ACML4BPM* kann die Zeitdauer einer Anpassung nicht beschrieben werden.

Somit ist die Bewertung der Realisierung für die Dimension *Mechanisms Duration* `nicht akzeptabel`.

EK15 Die Reihenfolge von Anpassungen durch verschiedene Instanzen eines Anpassungsprozesses kann in *ACML4BPM* nicht bestimmt werden. Lediglich die Reihenfolge von Instanzen eines Beobachtungsprozesses und der durch ihn aufgerufenen Anpassungsprozesse ist vorgegeben.

Dennoch können im Rahmen des Verhaltens eines Anpassungsprozesses verschiedene Ereignisse eingesetzt werden, die weitere Anpassungsfälle aufrufen. Hierdurch könnte z.B. der Umstand gegeben sein, dass bei der weiteren Ausführung des ausgehenden Anpassungsprozesses auf eine Synchronisation mit den durch ihn aufgerufenen weiteren Anpassungsprozessen gewartet wird. Für derartige Fälle kann garantiert werden, dass aufgerufene Anpassungen im fehlerfreien Betrieb zuerst beendet werden.

Die Bewertung der Realisierung der Dimension *Timeliness* wird aufgrund der beschriebenen Möglichkeit zur Beschreibung von Reihenfolgen als `moderat` bewertet.

EK16 In *ACML4BPM* ist es möglich, auf verschiedenste Typen von Ereignissen zu reagieren. So werden neben expliziten Ereignissen auch implizite Ereignisse unterstützt (siehe Abschnitt 4.3.4.1 bzw. Abschnitt 4.3.4.2).

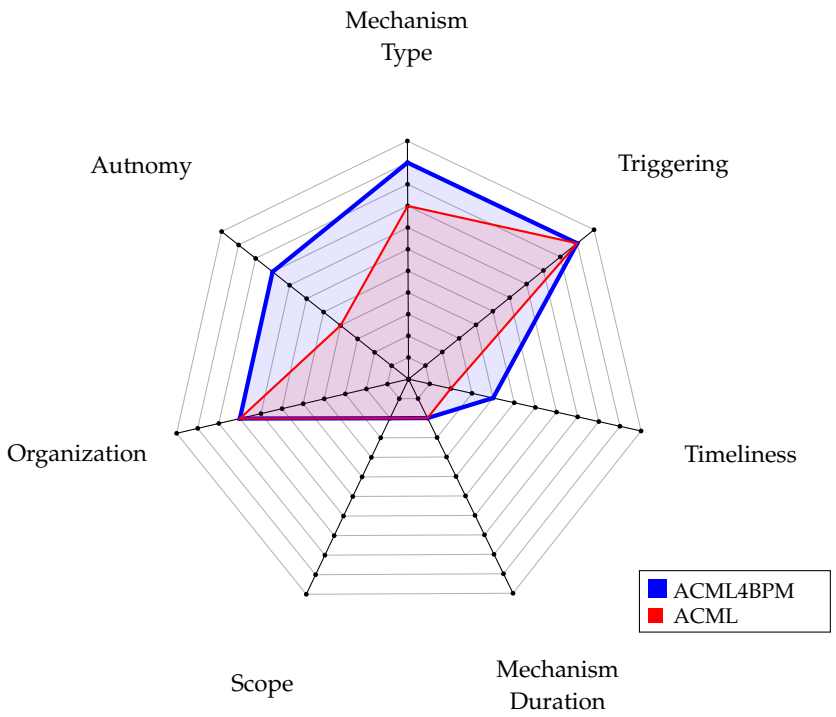
In dem Szenario werden dabei verschiedene Untertypen von expliziten Ereignissen verwendet. Neben zeitgesteuerten Ereignissen (*Fall 1* und *Fall 2*) wird in *Fall 3* auch ein Endereignis zur Auslösung verwendet.

Die Bewertung der Dimension *Triggering* kann daher als `exzellent` vorgenommen werden.

Grafische Darstellung und Diskussion

Eine grafische Darstellung der zuvor beschriebenen Bewertungen für die Gruppe *Mechanisms* ist in Abbildung 7-11 gegeben. Im Gegensatz zu den zuvor beschriebenen Gruppen des Aspekts der Anpassbarkeit unterscheiden sich die Bewertungen in gleich mehreren Dimensionen. Auf eine Erläuterung dieser Unterschiede mit dem Zweck des Vergleichs wird nachfolgend eingegangen.

Abbildung 7-11: Netzdiagramm für die Gruppe Mechanisms



Hinsichtlich der Dimension *Mechanism Type* lässt ich eine Abweichung der Ergebnisse damit begründen, dass der Fokus der Methode *Adapt Cases 4 BPM* auf der Gestaltung von flexiblen und anpassbaren Prozessen liegt. Der Ansatz *Adapt Cases* hingegen stellt einen generelleren Ansatz des *Adaptivity Engineering* dar, der insbesondere domänenspezifische Aspekte der Domäne *BPM* außer Acht lässt. So wurden in diesem Zusammenhang relevante System- und Umgebungskomponenten im Rahmen einer umfassenden Analyse identifiziert. Darauf basierend stehen entsprechende Sprachelemente in Form spezifischer Systemkomponenten in der Gestaltung von flexiblen und anpassbaren Prozessen zur Verfügung. Ferner konnten für diese Systemkomponenten Schnittstellen und Operationen zur Anpassung vorgestellt werden, die auf Basis von Perspektiven in Prozessen ein breites Spektrum an Anpassbarkeit unterstützen.

Im Gegensatz zu *ACML* unterstützt *ACML4BPM* verschiedene Grade an Autonomie des resultierenden Systems. Hier konnte durch die Arbeit an verschiedenen Aspekten von Flexibilität von Prozessen sowie der Einführung entsprechender Entwurfsmuster ein Mehrwert gegenüber *ACML* dargestellt werden. Dies lässt sich dadurch begründen, dass verschiedene Operationen zur Anpassung oder auch die Einbindung von Mitarbeitern oder anderen Nutzern des Systems entweder in der Sprache integriert oder durch die Entwurfsmuster bereits vorgesehen sind. Daher unterscheiden sich die Bewertungen in Hinsicht auf die Dimension *Autonomy*.

Die letzte Unterscheidung in der Bewertung ist für die Dimension *Timeliness* gegeben. Dabei konnte zuvor argumentiert werden, dass eine gewisse Art an Erfüllung des Evaluationskriteriums in *ACML4BPM* sehr wohl vorhanden ist, auch wenn es sich hierbei um einen Spezialfall handelt. In den Ergebnissen auf Basis von [Bis11] wird dieser Spezialfall nicht beschrieben. Dabei ist jedoch vorstellbar, dass dieser auch im Rahmen von *ACML* umsetzbar wäre. Die Realisierung der Dimension mit der Bewertung `nicht akzeptabel` wird aufgrund des Vorhandenseins des genannten Spezialfalls als nicht übertragbar auf *ACML4BPM* angesehen.

Ein Vergleich weiterer Ergebnisse mit gleicher Bewertung für die Gruppe *Mechanisms* lässt sich dadurch herleiten, dass zugehörige Konzepte der beiden Sprachen vergleichbar beschrieben worden sind. Zwei Beispiele sind hier durch die Dimensionen *Triggering* und *Organization* gegeben. Dabei können in beiden Sprachen unterschiedliche Arten von Auslösern, wie z.B. zeitbezogene Ereignisse, zum Aufruf eines *Adapt Case* bzw. eines *Adapt Case 4 BPM* genutzt werden. Ferner lassen sich Anpassungen an dem System oder seiner Umgebung auch derartig gestalten, dass sie an verteil-

ten Stellen vorkommen. Die Evaluationskriterien *Scope* und *Mechanisms Duration* können dabei durch keine der beiden Sprachen `moderat` erfüllt werden.

7.4.1.4 Bewertungen für die Gruppe Effects

Durch die Gruppe *Effects* werden verschiedene Dimensionen aufgeführt, die sich maßgeblich auf Eigenschaften einer Anpassung in Hinsicht auf die Auswirkung für das Gesamtsystem beziehen. Dabei ist zu bemerken, dass die Gestaltung oder Fähigkeit zur Analyse von Auswirkungen außerhalb des Fokus von *Adapt Cases* oder *Adapt Cases 4 BPM* liegt. Nachfolgend sind die Bewertungen der Dimensionen der Gruppe *Effects* in Tabelle 7-10 dargestellt. Auf Details und eine zugehörige Diskussion für die dargestellten Bewertungen von Dimensionen der Gruppe *Effects* wird nachfolgend individuell eingegangen.

Tabelle 7-10: Bewertungen für die Gruppe Effects

ID	Dimension	ACML4BPM	ACML
EK17	Criticality	gut	nicht akzeptabel
EK18	Predictability	akzeptabel	moderat
EK19	Overhead	moderat	nicht akzeptabel
EK20	Resilience	akzeptabel	nicht akzeptabel

EK17 Ein eigenes Sprachelement zur Kennzeichnung einer gescheiterten Anpassung ist in der Sprache *ACML4BPM* nicht notwendig, da die in der Sprache *BPMN2.0* enthaltenen Ereignisse vom Typ *BoundaryEvent* verwendet werden können. Ein solches Ereignis kann im Fall einer gescheiterten Anpassung zur Auslösung von weiterem Verhalten genutzt werden, durch das geeignete Maßnahmen umgesetzt werden. Eine Veranschaulichung der Verwendung von Ereignissen des Typs *BoundaryEvent* wurde bereits für *Flexibility-by Design* und den Aspekt *Cancellation* in Abschnitt 5.2.1.6 gegeben.

Ein Beispiel für derartige Maßnahmen ist durch Verhalten gegeben, das zum Zweck einer Kompensation bisher durchgeführter Anpassungen ausgeführt wird. Alternativ kann durch ein solches Ereignis und durch die Verwendung des Entwurfsmusters *Late Modeling* bspw. auch ein menschlicher Akteur eingebunden werden. Als Folge dieser Einbindung lassen sich durch diesen Akteur adäquate Maßnahmen in Form von Prozessfragmenten gestalten und anschließend binden. Hierdurch wäre es möglich, durch benutzerspezifische Prozessfragmente, auf das Scheitern einer Anpassung reagieren zu können. Die beiden zuvor genannten Beispiele zeigen zwei

mögliche Arten, die Konsequenz einer gescheiterten Anpassung entweder explizit (*Kompensation*) oder implizit (*Late Modeling*) zu beschreiben.

Die Dimension *Criticality* wird daher als `gut` realisiert bewertet.

EK18 Für die Vorhersage einer Konsequenz einer gescheiterten Anpassung ist geeignetes Wissen notwendig, welches im Rahmen des *AVM4BPM* gestaltet werden kann. Wird dieses Wissen nicht spezifiziert, lassen sich zugehörige Funktionen zur Vorhersage nicht umsetzen. Auch wenn somit kein dediziertes Sprachelement zur Vorhersage einer Konsequenz in der Sprache *ACML4BPM* vorhanden ist, so ist durch die weiterführende Gestaltung von Funktionen zur Analyse des genannten Wissens eine Vorhersage dennoch möglich.

Die Dimension *Predictability* wird daher als `akzeptabel` erfüllt bewertet.

EK19 Der Aufwand einer Anpassung und die daraus resultierenden Konsequenzen können durch die Sprache *ACML4BPM* nicht gestaltet werden. Negative Konsequenzen können z.B. dadurch gegeben sein, dass zeitlich später angestoßene Anpassungen durchführt worden sind, bevor früher begonnene Anpassungen beendet werden. Stehen diese Anpassungen in einem Abhängigkeitsverhältnis, sind sowohl Inkonsistenzen zwischen Prozessmodellen und deren Instanzen als auch unbestimmte System- und Umgebungszustände möglich.

Eine mögliche Lösung könnte sich mit der Vergabe von Prioritäten und einem geeigneten Schedulingverfahren beschäftigen. Dabei lassen sich einfache Prioritäten im Rahmen eines einzelnen *AC4BPM* durch eine entsprechende Gestaltung von Verzweigungen (siehe Abschnitt 5.2.2), also in Form von *Flexibility-by Design* und dem Aspekt *Choice*, implizit umsetzen.

Damit ist die Dimension *Overhead* mit `moderat` realisiert zu bewerten.

EK20 Durch *ACML4BPM* ist nicht gestaltbar, ob nach der Durchführung einer Anpassung das System bzw. seine Umgebung weiter in der Lage ist, die benötige Funktionalität zu erbringen.

Damit ist die Dimension *Resilience* mit `nicht akzeptabel` realisiert zu bewerten.

Grafische Darstellung und Diskussion

Eine grafische Darstellung der zuvor beschriebenen Bewertungen für die Gruppe *Effects* ist in Abbildung 7-12 gegeben. Generell lässt sich hier ablesen, dass sich der Erfüllungsgrad von Evaluationskriterien dieser Dimension im Durchschnitt als `moderat` bis `nicht akzeptabel` darstellt. Das

lässt sich dadurch begründen, dass der Fokus in beiden Ansätzen nicht auf der Abschätzung oder Gestaltung von möglichen Konsequenzen einer durchzuführenden Anpassung liegt.

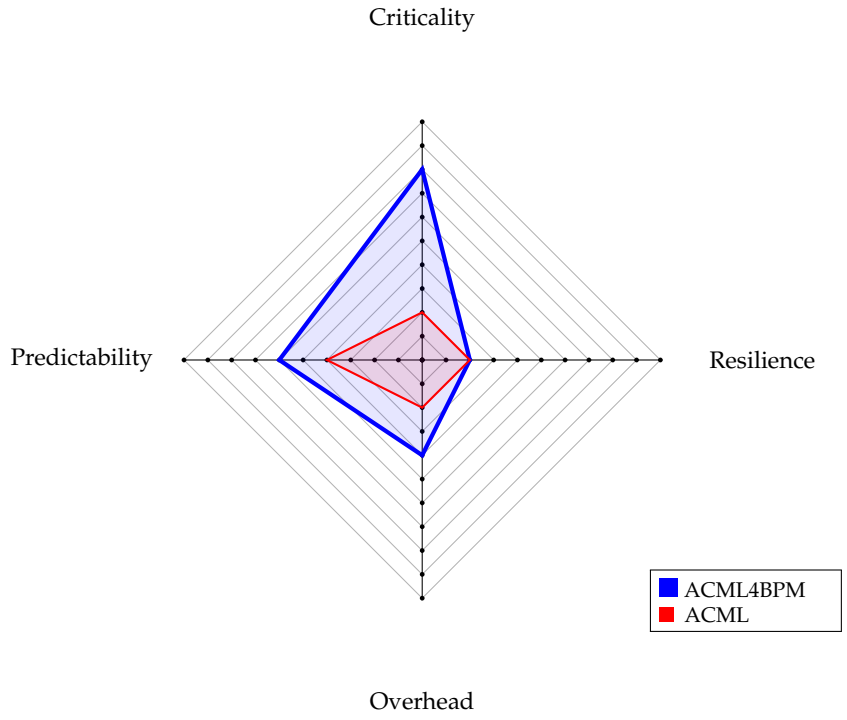


Abbildung 7-12: Netzdiagramm für die Gruppe Effects

Die Bewertung unterscheidet sich unmaßgeblich in drei der vier aufgeführten Dimensionen. Dies lässt sich aus zwei unterschiedlichen Perspektiven begründen. Zum einen ist es möglich, dass die Bewertung von *ACML4BPM* an diesen Stellen optimistischer ausfällt, als sie es für *ACML* war. Wobei hier jeweils in den einzelnen Beschreibungen entsprechend argumentiert worden ist. Zum anderen lässt sich für die Dimension *Criticality* die Verwendung von den in der Sprache *BPMN2.0* bestehenden Elementen und den erarbeiteten Entwurfsmustern beschreiben, welche so für den Ansatz *Adapt Cases* nicht vorhanden sind. Der dadurch entstehende Mehrwert stellt damit die Basis für eine höhere Bewertung dar.

7.4.2 Bewertung von Kriterien an Adapt Cases 4 BPM

In diesem Abschnitt werden Bewertungen für die in Abschnitt 7.2.2 beschriebenen Kriterien vorgestellt. Diese Bewertungen werden in zwei Gruppen vorgenommen, auf die nachfolgend kurz eingegangen wird. So werden zunächst vergleichbare Bewertungen der Evaluation von *ACML*

und *ACML4BPM* in Abschnitt 7.4.2.1 vorgestellt. Hiervon betroffen sind die Evaluationskriterien *EK21*, *EK22*, *EK23* sowie *EK24*. Zugehörige Daten der Evaluation für die Sprache *ACML* wurden aus der Arbeit von [Bis11] übernommen.

Die Evaluationskriterien *EK25*, *EK26* und *EK27* sind für *ACML* nicht sinnvoll in einem Vergleich anwendbar. Für *EK26* wird ein Vergleich mit alternativen existierenden Arbeiten der Domäne *BPM* in Abschnitt 7.4.2.2 vorgenommen. Zugehörige Daten der Evaluation für das Evaluationskriterium *EK26* wurden aus der Arbeit von [Sch+08] übernommen. Auf einen Vergleich zwischen *EK25* und *EK27* wird verzichtet. Die Ergebnisse aus beiden Gruppen werden jeweils sowohl tabellarisch als auch zum Teil grafisch durch Netzdiagramme dargestellt.

7.4.2.1 Bewertungen für die Anforderungen an Adapt Case 4 BPM

Nachfolgend sind Bewertungen für die Anforderungen an den Ansatz *Adapt Cases 4 BPM* in Tabelle 7-11 dargestellt. Auf Details und eine zugehörige Diskussion für die dargestellten Bewertungen wird nachfolgend individuell eingegangen.

ID	Dimension	ACML4BPM	ACML
EK21	Separation-of-Concerns	gut	exzellent
EK22	Kontrollschleife	exzellent	exzellent
EK23	Ausdrucksfähigkeit	exzellent	exzellent
EK24	UML-Konsistenz	exzellent	exzellent
EK25	BPMN2.0-Konsistenz	exzellent	nicht gesetzt
EK26	Musterbasierte Unterstützung	exzellent	nicht gesetzt
EK27	Integration	exzellent	nicht gesetzt

Tabelle 7-11: Bewertungen für die Anforderungen an Adapt Cases 4 BPM

EK21 Eine wesentliche Anforderung war es, die Trennung von Anpassungs- und Anwendungslogik in der Gestaltung von flexiblen und anpassbaren Prozessen unterstützen zu können. Hierzu wurden in Kapitel 4 die Sprache *ACML4BPM* und in Kapitel 5 eine Reihe von Entwurfsmustern vorgestellt. Dabei wurden grundlegende Funktionsweisen von Konzepten und schematische Beispiele für die Trennung der genannten Logiken gegeben. Im Rahmen der in diesem Kapitel betrachteten Fälle des Szenarios wurden zusätzlich praxisnahe Beispiele beschrieben, in denen eine getrennte Gestaltung dieser Logiken vorgenommen wurde.

Eine Trennung der beiden Logiken kann somit unterstützt werden. Für konkrete Beispiele für die Trennung von Anpassungs- und Anwendungslogiken wird an dieser Stelle auf die Fälle des Szenarios in Abschnitt 7.1

verwiesen. Dabei ist anzumerken, dass sowohl in dem Szenario als auch in den Entwurfsmustern verschiedene Fälle existieren, bei denen spezielle Sprachelemente verwendet werden müssen, um die Anpassungslogik mit der Anwendungslogik zu verbinden. Hierfür werden bspw. im Rahmen des Szenarios verschiedene Ereignisse verwendet. Eine vollkommene Trennung der beiden Logiken ist daher nicht erreicht worden, weil es auch weiterhin verbindende Elemente gibt.

Als Konsequenz fällt die Bewertung für die Dimension *Separation-of-Concerns* lediglich als `gut` aus.

EK22 Durch die enge Anlehnung an grundlegende Konzepte des Ansatzes *Adapt Cases* konnte auch in *Adapt Cases 4 BPM* eine Unterstützung des Musters der Kontrollschleife *MAPE-K* [KC03] in der Gestaltung von flexiblen und anpassbaren Prozessen umgesetzt werden. So werden grundlegende Konzepte des Musters in Form der Funktionen *Analysis* und *Execution* (siehe Kapitel 1) durch den Beobachtungs- bzw. Anpassungsprozess umgesetzt.

Hierzu enthält *Fall 2* des Szenarios ein besonders geeignetes Beispiel. So kann im Rahmen des Beobachtungsprozesses auf Basis verschiedener Auslöser aus dem Kontext des Systems (hier: *Prozess*) so reagiert werden, dass adäquate Anpassungen (hier: *Zuweisung*) durchgeführt werden. Dabei werden auch erweiterte Analysefunktionen unterstützt, die in regelmäßigen Abständen die Notwendigkeit einer Anpassung prüfen.

Die Dimension *Kontrollschleife* kann damit als `exzellent` erfüllt bewertet werden.

EK23 Der in dieser Arbeit vorgestellte Ansatz *Adapt Cases 4 BPM* stellt eine domänenspezifische Redefinition des Ansatzes *Adapt Cases* dar. So wird in beiden Ansätzen das Verhalten fallbasiert beschrieben. Durch dieses Verhalten werden Anpassungen von Eigenschaften des betrachteten Systems und seiner Umgebung vorgenommen. Beide Ansätze sind dabei für die Anwendung in einer frühen Phase in der Gestaltung (hier: *Design & Analyse*) vorgesehen. Werden benötigte Funktionen der Anwendungslogik durch *UML Use Cases* beschrieben, so können Funktionen der Anpassungslogik durch das Konzept des *Adapt Case 4 BPM* beschrieben werden. Dabei kann – wie für *EK5 Dependency* beschrieben – ein *AC4BPM* mit anderen Funktionen der Anwendungs- und Anpassungslogik in Relation stehen.

Ein Beispiel für die gemeinsame Verwendung von *UML Use Cases* und dem Konzept des *Adapt Case 4 BPM* wurde in den drei vorgestellten Fällen des

Szenarios im Rahmen der *High-Level-Gestaltung* von Funktionen des betrachteten Systems und seiner Umgebung gegeben (siehe Abschnitt 7.1). Dabei wurde gezeigt, dass neben der für das *Adaptivity Engineering* klassischen Assoziation «*adapts*» auch weitere Assoziationen sinnvoll sein können (siehe Abschnitt 7.1.3 bzw. Abschnitt 7.1.4).

Die Dimension *Ausdrucksfähigkeit* kann für den Ansatz *Adapt Cases 4 BPM* daher als `exzellent` bewertet werden.

EK24 In der Sprache *ACML4BPM* wurden verschiedene Elemente auf Basis bestehender Konzepte der beiden Sprachen *ACML* und *UML* redefiniert. Hierdurch wurde eine hohe Konsistenz hinsichtlich der Verwendung von *ACML4BPM* bei der Gestaltung von flexiblen und anpassbaren Prozessen gegenüber der Sprache *ACML* bzw. *UML* erreicht.

Da es sich bei *Adapt Case 4 BPM* um eine domänenspezifische Redefinition handelt, werden an notwendigen Stellen Elemente der Sprache *BPMN2.0* verwendet. Hierfür existieren verschiedene Beispiele, wie die beiden Verfeinerungen eines *AC4BPM* in Form von Beobachtungs- und Anpassungsprozessen oder die zur Auslösung verwendeten Ereignisse.

Bei den zuvor genannten Abweichungen handelt es sich um notwendige Einschränkungen in der Konsistenz, sodass als Konsequenz die Dimension *UML-Konsistenz* dennoch als `exzellent` erfüllt bewertet wird.

EK25 Die Sprache *BPMN2.0* stellt den De-facto-Standard zur Beschreibung von Prozessen in der Domäne *BPM* dar. Aus diesem Grund wurden bestehende Konzepte des Ansatzes *Adapt Cases* so redefiniert, dass spezifische Konzepte der Sprache *BPMN2.0* verwendet worden sind. So lässt sich im Rahmen der *Low-Level-Gestaltung* die durch einen *AC4BPM* beschriebene Funktion mit Hilfe von Beobachtungs- und Anpassungsprozessen verfeinern. Die Beschreibung von dem in diesen Prozessen enthaltenen Verhalten kann somit durch die Verwendung üblicher *BPMN2.0* Elemente durchgeführt werden. Ferner können auch neu eingeführte Elemente der Sprache *ACML4BPM*, wie z.B. verschiedene Operationen oder Ereignisse, in diesen Prozessen verwendet werden. Sie wurden dabei auf Basis bestehender Elemente der Sprache *BPMN2.0* definiert. Hierdurch entsteht die Möglichkeit, mit Konzepten des *Adaptivity Engineering* und in enger Anlehnung an die Sprache *BPMN2.0* entsprechend Prozesse zu gestalten.

Die beiden zuvor ausgeführten Eigenschaften lassen daher den Schluss zu, dass eine hohe Konsistenz zu der Sprache *BPMN2.0* erwartet werden kann. Das Szenario (siehe Abschnitt 7.1) sowie die Beispiele entlang Kapitel 4 und Kapitel 5 dokumentieren den bestehenden Grad an Konsistenz.

Als Konsequenz wird die Dimension *BPMN2.0-Konsistenz* als `exzellent` erfüllt bewertet.

EK26 Im Rahmen dieser Arbeit wurde eine Reihe von verschiedenen Aspekten von flexiblen und anpassbaren Prozessen nach [Sch+08] vorgestellt. Basierend auf dieser Arbeit wurden Entwurfsmuster beschrieben (siehe Kapitel 5). Diese Entwurfsmuster umfassen neben verschiedenen notwendigen Spracherweiterungen der Sprache *BPMN2.0* auch spezifische Operationen für Anpassungen. Ferner wird die Verwendungsweise der vorgestellten Entwurfsmuster beschrieben, sodass ein Mehrwert für die Gestaltung von flexiblen und anpassbaren Prozessen entsteht.

Die Bewertung des Kriteriums *EK26* wird an dieser Stelle vorweggenommen und mit `exzellent` angeben. Eine Übersicht über unterstützte Entwurfsmuster, die detaillierte Bewertung sowie ein Vergleich mit existierenden Arbeiten aus der Domäne *BPM* werden im Anschluss an die Diskussion gegeben.

EK27 Sowohl die vorgestellte Sprache *ACML4BPM* als auch die Entwurfsmuster benötigen für ihre Verwendung einen methodischen Rahmen. Hierzu wurde die Methode *Adapt Cases 4 BPM* vorgestellt, die wesentliche Aktivitäten und Artefakte eines domänenspezifischen *Adaptivity Engineering* beschreibt. Dabei wurde sich an einem in der Domäne *BPM* verbreiteten methodischen Rahmen in Form des *BPM-Lebenszyklus* nach *Weske* [Wes12] orientiert.

Eine beispielhafte Anwendung der Methode *Adapt Cases 4 BPM* wurde in Teilen im Rahmen des Szenarios gezeigt (siehe Abschnitt 7.1). Dabei wurden die drei Aktivitäten *Anforderungsanalyse*, *High-Level-Gestaltung* und *Low-Level-Gestaltung* sowie ihre Artefakte und Abhängigkeiten an Beispielen beschrieben. Eine weiterführende Verwendung der Methode, wie z.B. in einem realen Projekt, wurde nicht durchgeführt. Daher muss eine Bewertung der Methode auf Basis praxisnaher Kriterien an die zukünftige Forschung am vorgestellten domänenspezifischen *Adaptivity Engineering* delegiert werden.

Es wird davon ausgegangen, dass durch die in Kapitel 6 gegebene Beschreibung der Methode *Adapt Cases 4 BPM* sowie durch das Szenario und seine drei Fälle die Integration in eine Methode der Domäne *BPM* plausibilisiert werden konnte. Die Dimension *Integration* wird daher mit `exzellent` bewertet.

Grafische Darstellung und Diskussion

Die Bewertungen in Hinsicht auf die Anforderungen an den Ansatz *Adapt Cases 4 BPM* lassen sich wie in Abbildung 7-13 gezeigt grafisch darstellen. Generell lässt sich erkennen, dass die Bewertungen von hier betrachteten Dimensionen als `exzellent` eingestuft werden können. Das lässt sich dadurch begründen, dass sich bei der Erstellung des Konzepts für den Ansatz *Adapt Cases 4 BPM* eng an dem bestehenden Ansatz *Adapt Cases* orientiert wurde. Die Ergebnisse der untersuchten Dimensionen sind daher bis auf die Ausnahme *Separation-of-Concerns* als Deckungsgleich zu bezeichnen. Eine Erklärung für die Abweichung bei der Bewertung der Dimension *Separation-of-Concerns* wird nachfolgend in Form einer Abschätzung durchgeführt.

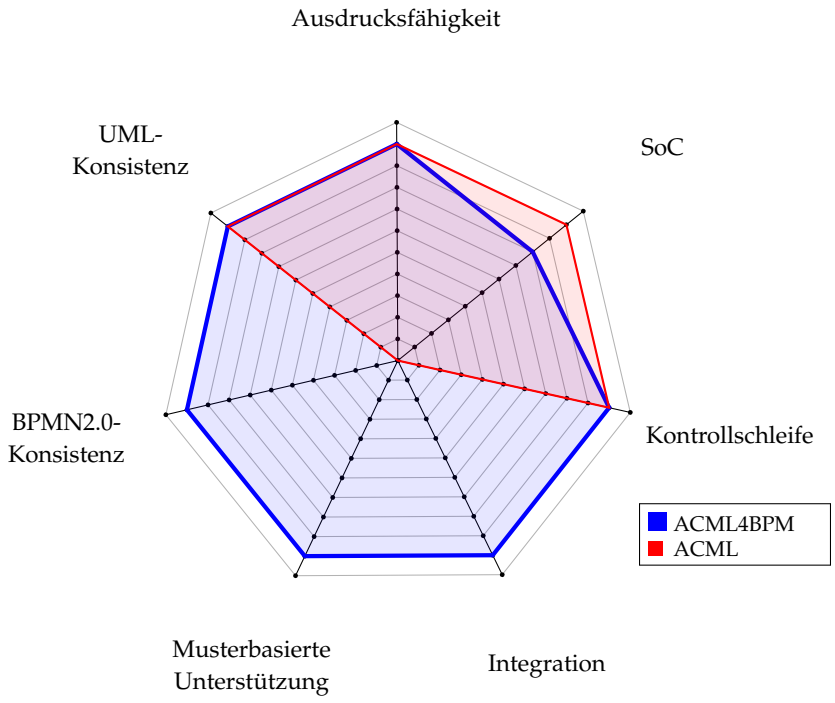


Abbildung 7-13: Netzdiagramm für die Anforderungen an Adapt Cases 4 BPM

In der Bewertung der Dimension *Separation-of-Concerns* wurde sich gegen die Vergabe der Bewertung *exzellent* entschlossen, da in einigen Fällen sehr wohl Elemente der verschiedenen Logiken gemeinsam vorkommen. So wurden bereits in Abschnitt 5.2 unterschiedliche Arten der Integration des durch einen *AC4BPM* beschriebenen Verhaltens vorgestellt. Diese Arten weisen nicht immer eine klare Trennung beteiligter Logiken auf, sodass es sich hier um eine geringe Einschränkung des vorliegenden Ansatzes

Adapt Cases 4 BPM in Bezug zur Dimension *Separation-of-Concerns* handelt. Ferner kann davon ausgegangen werden, dass diese gemeinsame Verwendung von Elementen der beiden Logiken auch im Ansatz *Adapt Cases* bereits präsent war. Dies wurde jedoch im Rahmen der Bewertung der betreffenden Kriterien offensichtlich als weniger relevant für das Endergebnis eingeschätzt. Eine Abweichung in den Evaluationen der beiden Ansätze *Adapt Cases* und *Adapt Cases 4 BPM* in Hinsicht auf die Dimension *Separation-of-Concerns* ist daher vermutlich der jeweiligen subjektiven Bewertung zuzuordnen.

Die Bewertungen der beiden Dimensionen *BPMN2.0-Konsistenz* und *Musterbasierte Unterstützung* sind als `exzellent` eingestuft. Dabei wird an dieser Stelle auf einen Vergleich mit dem Ansatz *Adapt Cases* verzichtet, da es sich um spezifische Anforderungen für den vorliegenden Ansatz *Adapt Cases 4 BPM* handelt. Die Bewertungen für den Ansatz *Adapt Cases* sind daher als `nicht gesetzt` dargestellt und können hier vernachlässigt werden. Nachfolgend werden in Abschnitt 7.4.2.2 eine Übersicht, eine Bewertung sowie ein Vergleich zwischen alternativen Ansätzen in Bezug zur Dimension *Musterbasierte Unterstützung* gegeben.

7.4.2.2 Vergleich mit existierenden Ansätzen hinsichtlich der Dimension *Musterbasierte Unterstützung*

Bei der Beschreibung von verschiedenen Entwurfsmustern wurde die Sprache *ACML4BPM* verwendet (siehe Kapitel 5). In diesem Abschnitt sollen eine Übersicht, eine Bewertung sowie ein Vergleich alternativer Ansätze zur Gestaltung von flexiblen und anpassbaren Prozessen gegeben werden. Hierdurch soll die in dem vorherigen Abschnitt beschriebene Argumentation zur Bewertung der Dimension *EK26* mit dem Wert `exzellent` begründet werden.

Die nachfolgende Bewertung basiert dabei zum einen auf Daten, die bereits durch die von [Sch+08] durchgeführte Evaluation erstellt worden sind. Zum anderen werden Daten hinsichtlich der Bewertung vorgestellter Entwurfsmuster unter Verwendung der Sprache *ACML4BPM* hinzugefügt. Diese Daten sind durch eine selbstkritische Bewertung des Ansatzes entstanden. Dadurch kann der Vergleich zwischen verschiedenen Frameworks zur Gestaltung von flexiblen und anpassbaren Prozessen unterstützt werden.

Bei diesen Frameworks handelt es sich vornehmlich um *ADEPT1* [RRD03], *YAWL* [AT05; Ada+06; Ada+07], *FLOWer* [AWG05] und *Declare* [PA06;

Pes+07]. Neben diesen vorrangig wissenschaftlich geprägten Frameworks existiert selbstverständlich eine Reihe von freien oder kommerziellen Frameworks, die entweder kostenpflichtig oder oft mit reduziertem Funktionsumfang verfügbar sind. Beispiele hierfür sind durch *Camunda Tool Suite*[1], *Signavio Business Transformation Suite*[2], *jBPM*[3] oder *Bonita Toolsuite*[4] gegeben. Die genannten freien und kommerziellen Frameworks unterstützen die Verwendung von in der Sprache *BPMN2.0* spezifizierten Prozessmodellen. Für eine umfangreichere Übersicht über verfügbare freie und kostenpflichtige Frameworks sowie über ihre Kompatibilität zur Sprache *BPMN2.0* kann auf verschiedene Arbeiten verwiesen werden [ES11; GW13; Kur16]. Es wurde sich gegen eine Aufnahme derartiger Frameworks im Rahmen des hier vorliegenden Vergleichs entschieden, da sie ausgesuchte Aspekte der Spezifikation der Sprache *BPMN2.0* teilweise unterschiedlich umsetzen und ein solcher Vergleich für diese Arbeit nicht im Fokus liegt.

Flexibility-by Design Die Bewertung von *Adapt Cases 4 BPM* in Bezug zum Aspekt *Flexibility-by Design* ist in Tabelle 7-12 dargestellt. Nachfolgend wird auf die Ausprägung sowie auf einen Vergleich zu den genannten alternativen Ansätzen eingegangen.

Tabelle 7-12: Bewertungen in Hinsicht auf Flexibility-by Design

Flexibility-by Design	**ADEPT1**	**YAWL**	**FLOWer**	**Declare**	**Adapt Cases 4 BPM**
Parallelism	✓	✓	✓	✓	✓
Choice	✓	✓	✓	✓	✓
Iteration	✓	✓	✓	✓	✓
Interleaving	✗	✓	✓	✓	✓
Multiple Instances	✗	✓	✓	✓	✓
Cancellation	✗	✓	✗	✓	✓

Der Flexibilitätsaspekt *Flexibility-by Design* und seine weiteren Aspekte zur Gestaltung lassen sich bereits durch die Sprache *BPMN2.0* umsetzen. Hierfür wurden bereits je Gestaltungsaspekt diverse Beispiele beschrieben (siehe Abschnitt 5.2.1). Die Sprache *ACML4BPM* wurde auf Basis von Elementen der Sprache *BPMN2.0* definiert, sodass in Beobachtungs- und Anpassungsprozessen verschiedene Aspekte, wie z.B. *Choice* oder *Iteration*, enthalten sein können. Ferner lässt sich durch die Verwendung der Sprache *ACML4BPM* bei der Gestaltung von Aspekten von *Flexibility-by Design* auch eine Trennung der Anpassungs- und Anwendungslogik durchführen (siehe *EK21*). Davon sind die Gestaltungsaspekte *Choice*, *Iteration* und

[1] `https://camunda.org/` Letzter Zugriff: 01.09.2018
[2] `https://www.signavio.com/` Letzter Zugriff: 01.09.2018
[3] `https://www.jbpm.org/` Letzter Zugriff: 01.09.2018
[4] `https://www.bonitasoft.com/` Letzter Zugriff: 01.09.2018

Cancellation unmittelbar betroffen. Im Vergleich zu alternativen Ansätzen, wie z.B. *ADEPT1* oder *FLOWer*, kann der Ansatz *Adapt Cases 4 BPM* somit einen Vorteil bieten, da die aufgeführten Gestaltungsaspekte zum einen durch *BPMN2.0* bereits abgedeckt sind und zum anderen eine Trennung der beiden Logiken ermöglicht wird.

Flexibility-by Change Die Bewertung von *Adapt Cases 4 BPM* in Bezug zum Aspekt *Flexibility-by Change* ist in Tabelle 7-13 dargestellt. Nachfolgend wird auf die Ausprägung sowie einen Vergleich zu den genannten alternativen Ansätzen eingegangen.

Tabelle 7-13: Bewertungen in Hinsicht auf Flexibility-by Change

Flexibility-by Change	**ADEPT1**	**YAWL**	**FLOWer**	**Declare**	**Adapt Cases 4 BPM**
Momentary Change	✓	✗	✗	✓	✓
Evolutionary Change	✗	✓	✗	✓	✓
Strategien zur Migration					
Forward Recovery	✗	✓	✗	✗	✓
Backward Recovery	✗	✓	✗	✗	✓
Proceed	✗	✗	✗	✓	✓
Transfer	✗	✓	✗	✓	✓
Zeitpunkt der Anwendung					
Entry Time	✓	✗	✗	✓	✓
On-the-Fly	✓	✓	✗	✓	✓

Im Rahmen des vorgestellten Entwurfsmusters für den Flexibilitätsaspekt *Flexibility-by Change* wurde neben einer Spracherweiterung der Sprache *BPMN2.0* zur Unterscheidung von Elementen aus Prozessmodellen und deren Instanzen auch eine Reihe von Operationen vorgestellt, mit denen Migrationen von Anpassungen von Prozessen durchgeführt werden können (siehe Abschnitt 5.3). Die vorgestellten Inhalte decken dabei sowohl die aufgeführten Typen von Anpassungen *Momentary Change* und *Evolutionary Change* sowie die Zeitpunkte der Anwendung einer Anpassung *Entry Time* und *On-the-Fly* als auch die aufgeführten Strategien zur Migration ab. Dabei wurde für jede Migrationsstrategie eine Operation vorgestellt, die unter entsprechender Parametrisierung das jeweilige Funktionsprinzip einer Strategie umsetzt. Hierdurch können die in Tabelle 7-13 dargestellten Aspekte von *Flexibility-by Change* jeweils als erfüllt betrachtet werden.

Im Vergleich zu anderen aufgeführten Ansätzen unterstützt *Adapt Cases 4 BPM* ebenso wie *Declare* beide Typen von Anpassungen. Ferner werden im Rahmen von Anpassungen des Typs *Evolutionary Change* jede der vier aufgeführten Migrationsstrategien von *Adapt Cases 4 BPM* unterstützt. Die

beiden Ansätze *YAWL* und *Declare* unterstützen hierbei lediglich eine Teilmenge dieser Strategien.

Flexibility-by Deviation Die Bewertung von *Adapt Cases 4 BPM* in Bezug zum Aspekt *Flexibility-by Deviation* ist in Tabelle 7-14 dargestellt. Nachfolgend wird auf die Ausprägung sowie auf einen Vergleich zu den genannten alternativen Ansätzen eingegangen.

Flexibility-by Deviation	**ADEPT1**	**YAWL**	**FLOWer**	**Declare**	**Adapt Cases 4 BPM**
Operationen in imperativen Sprachen					
Redo	✗	✗	✓	✗	✓
Undo	✗	✗	✓	✗	✓
Skip	✗	✗	✓	✗	✓
Create additional Instances	✗	✗	✓/✗	✗	✓
Invoke Task	✗	✗	✓	✗	✓
Operationen in deklarativen Sprachen					
Violation of Constraints	✗	✗	✗	✓	✗

Tabelle 7-14: Bewertungen in Hinsicht auf Flexibility-by Deviation

Hier bietet der Ansatz *Adapt Cases 4 BPM* die Möglichkeit der Unterstützung bei der Gestaltung von flexiblen und anpassbaren Prozessen in der Form, dass eine Reihe von spezifischen Operationen zur Verfügung steht. Diese Operationen lassen sich so einsetzen, dass verschiedene Abweichungen vom im Prozessmodell vorgesehenen Kontrollfluss ermöglicht werden, ohne diesen explizit anzupassen. Dabei sind diese Operationen jedoch in dieser Weise nur in imperativen Sprachen möglich. Im Rahmen der in dieser Arbeit vorgenommen Definition der Sprache *ACML4BPM* werden keine deklarativen Sprachen und entsprechenden Operationen zur Abweichung unterstützt. Im Vergleich zu den Ansätzen *FLOWer* und *Declare* fällt auf, dass jeweils entweder Operationen für imperative oder aber deklarative Sprachen unterstützt werden. Dies lässt sich damit begründen, dass das Konzept der zugrundeliegenden Basissprache entsprechend imperativ oder deklarativ ist. Eine Unterstützung von Operationen für die beiden Typen von Sprachen scheint daher nur dann sinnvoll, wenn die Basissprache sowohl imperative als auch deklarative Beschreibungen von Prozessen explizit vorsieht. Dies ist bei *BPMN2.0* jedoch nicht der Fall, sodass dieser Missstand als vernachlässigbar angesehen wird.

Flexibility-by Underspecification Die Bewertung von *Adapt Cases 4 BPM* in Bezug zum vierten und letzten Aspekt *Flexibility-by Underspecification* ist in Tabelle 7-15 dargestellt. Nachfolgend wird auf die Ausprägung sowie auf einen Vergleich zu den genannten alternativen Ansätzen eingegangen.

Tabelle 7-15: Bewertungen in Hinsicht auf Flexibility-by Underspecification

Flexibility-by Underspecification	**ADEPT1**	**YAWL**	**FLOWer**	**Declare**	**Adapt Cases 4 BPM**
Late Binding	✗	✓	✗	✗	✓
Late Modeling	✗	✓	✗	✗	✓
Zeitpunkt der Anwendung					
Preliminary	✗	✗	✗	✗	✓
On Activation	✗	✓	✗	✗	✓
Mehrfachanwendung					
Static	✗	✗	✗	✗	✓
Dynamic	✗	✓	✗	✗	✓

Der Ansatz *Adapt Cases 4 BPM* unterstützt die Gestaltung der beiden Typen *Late Selection* und *Late Modeling* durch die Verwendung von verschiedenen Operationen. Dabei können diese Operationen so parametrisiert werden, dass sowohl beide Typen von Zeitpunkten als auch die Mehrfachanwendung ermöglicht werden kann.

YAWL unterstützt ebenso die beiden genannten Typen des Flexibilitätsaspekts *Flexibility-by Underspecification*. Ferner können dort mehrfach Bindungen bei Aktivierung eines Platzhalters durchgeführt werden. Der Ansatz *Adapt Cases 4 BPM* bietet hier jedoch vielfältigere Einsatzmöglichkeiten, da insgesamt mehr Eigenschaften unterstützt werden können.

7.5 Gültigkeit

Die Evaluation des in dieser Arbeit vorgestellten Ansatzes *Adapt Cases 4 BPM* basiert auf insgesamt zwei vorgestellten Kriterienkatalogen sowie auf einem an der Praxis orientierten Szenario. Dabei wurden neben den bereits durch [Bis11] vorgestellten Kriterien auch weitere Kriterien zur Evaluation in Bezug zur Gestaltung von flexiblen und anpassbaren Prozessen hinzugefügt. Selbstverständlich können durch die zuvor genannten Kriterien und durch das Szenario nur bestimmte Teile des Ansatzes unter einer sinnvollen Verwendung existierender Ressourcen evaluiert werden. Derartige Ressourcen sind bspw. gegeben durch die finanzielle Unterstützung sowie durch die Verfügbarkeit von industriellen Partnern, mit denen der Ansatz z.B. auch in der Praxis hätte evaluiert werden können.

Rückkopplung mit Experten aus Wissenschaft und Industrie

Nichtsdestotrotz profitierte die Qualität dieser Arbeit durch die Einbettung in das *NRW Fortschrittskolleg „Gestaltung von flexiblen Arbeitswelten“*. Hier konnten in verschiedenen Expertengesprächen und wissenschaftlichen sowie industriellen Workshops einzelne Teile der vorliegenden Arbeit regelmäßig vorgelegt und diskutiert werden. Durch das breite Spektrum an unterschiedlichen Experten aus Wissenschaft und Praxis konnte die Plausibilisierung unterschiedlichster Aspekte bereits vor und während der Ausarbeitung vorgenommen werden. Als ein Beispiel kann hier insbesondere das Szenario sowie verschiedenste Aspekte von Flexibilität und deren Entwurfsmustern genannt werden, die auf Basis diverser Gespräche verfeinert werden konnten. Der vorliegende Arbeitsstand unter der gegebenen Qualität wäre ohne derartige Gespräche, welche ebenso eine Art der Evaluation darstellen, nicht möglich gewesen.

Art der Bewertung

Die Bewertung von vorgestellten Kriterien stellen eine selbstkritische und damit subjektive Bewertung dar. Unter Verwendung von Experten der Domäne *BPM* wäre es ebenso möglich gewesen, eine empirische Untersuchung hinsichtlich der Anwendbarkeit der Sprache oder der Benutzerfreundlichkeit (engl. *Usability*) durchzuführen. Der Ansatz hätte hierdurch in Hinsicht auf die Evaluation von grundlegenden Konzepten, die bereits Teil des Ansatzes *Adapt Cases* gewesen sind, einen Mehrwert bieten können.

Referenzpunkte zur Bewertung

Ein weiterer Punkt, der aus Sicht einer kritischen Betrachtung der vorgestellten Bewertungen Berücksichtigung finden sollte, ist dadurch gegeben, dass vergleichbare Kriterien eingesetzt wurden, wie sie auch schon für den Ansatz *Adapt Cases* verwendet worden sind. Dies lässt sich damit begründen, dass beide Ansätze verglichen werden sollten. Wären alternative Vorgehensweisen für die Bewertungen verwendet worden, hätte auch der Ansatz *Adapt Cases* neu bewertet werden müssen.

Bei dem gegebenen Vergleich von vorgestellten Entwurfsmustern mit alternativen Arbeiten lag der Fokus lediglich auf der Gestaltung von flexiblen und anpassbaren Prozessen. Dabei werden durch die aufgeführten alternativen Ansätze oftmals auch wissenschaftliche Prototypen zur Ausführung auf Basis gestalteter Prozesse angeboten. Ein derartiger Prototyp ist für *Adapt Cases 4 BPM* so nicht vorhanden und lag auch nicht im Fokus der Arbeit. Durch die Definition der Sprachelemente in enger Anlehnung an existierenden Elementen der Sprache *BPMN2.0* wird an dieser Stelle davon ausgegangen, dass Ergänzungen, die eine Ausführung der gestalteten Prozesse ermöglichen, durch geringe Anpassungen von entsprechenden Softwarelösungen ermöglicht werden können.

Kapitel 8

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden verschiedene Konzepte präsentiert, die für die Gestaltung von flexiblen und anpassbaren Prozessen eingesetzt werden können. In diesem Kapitel werden diese Konzepte zunächst in Abschnitt 8.1 kurz zusammengefasst. Ferner wird auf den geleisteten wissenschaftlichen Beitrag in Anlehnung an existierenden Ansätzen des *Adaptivity Engineering* eingegangen und Bezug zu anfangs vorgestellten Forschungsfragen genommen (siehe Kapitel 1). Ausgesuchte weiterführende Fragestellungen für zukünftige Forschungsthemen für das *Adaptivity Engineering* mit einem speziellen Bezug zur Gestaltung von flexiblen und anpassbaren Prozessen werden in Form eines Ausblicks in Abschnitt 8.2 vorgestellt.

8.1 Zusammenfassung

In dieser Arbeit wurde sich mit der Gestaltung von flexiblen und anpassbaren Prozessen in der Domäne *BPM* auseinandergesetzt. Die vorgestellten Lösungsteile umfassen eine für diese Domäne neuartige Sprache (*Adapt Case Modeling Language 4 BPM*), Entwurfsmuster zur Gestaltung von flexiblen und anpassbaren Prozessen sowie eine zugehörige Methodik (*Adapt Cases 4 BPM*). Diese drei Lösungsteile stellen den durch diese Arbeit geleisteten wissenschaftlichen Beitrag dar. In Abbildung 8-1 ist eine Übersicht dieses Beitrags gezeigt, auf dessen Details nachfolgend eingegangen wird.

Adapt Case Modeling Language 4 BPM

Beantwortung von Forschungsfrage 1

In dieser Arbeit wurde eine neuartige domänenspezifische Sprache zur Gestaltung von flexiblen und anpassbaren Prozessen vorgestellt. Die Sprache

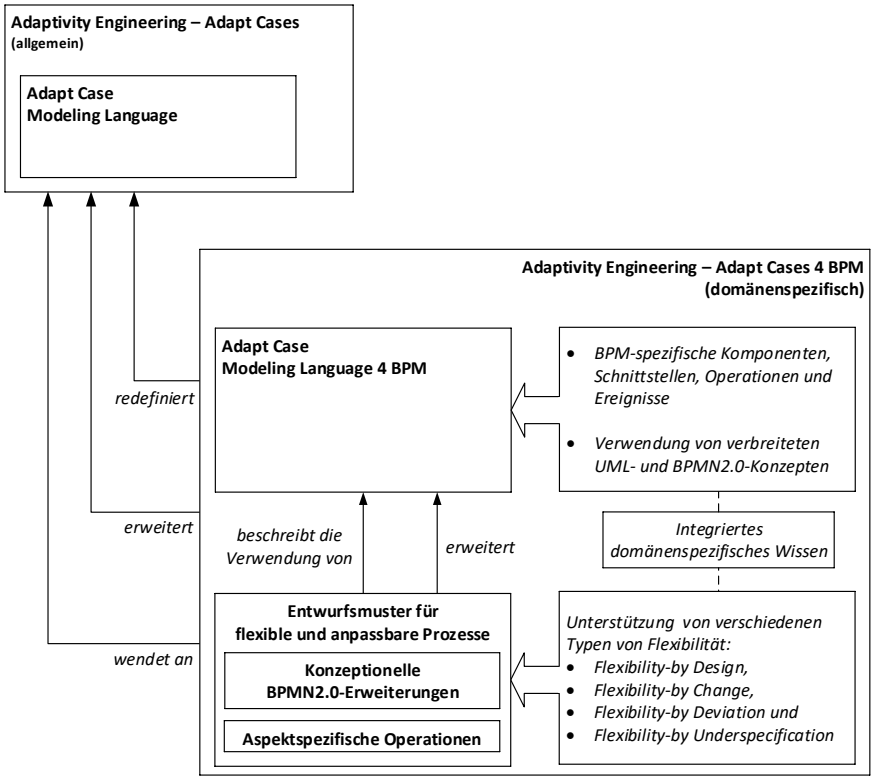


Abbildung 8-1: Übersicht über den wissenschaftlichen Beitrag

Adapt Case Modeling Language 4 BPM (*ACML4BPM*) stellt dabei eine Redefinition der Sprache *Adapt Case Modeling Language* (*ACML*) dar, welche durch [Luc+11] erstmals vorgestellt worden ist. Hierdurch können grundlegende Konzepte des *Autonomic Computing* in Form der Referenzarchitektur *MAPE-K* [KC03] wiederverwendet werden.

Die Sprache *ACML4BPM* kann in einer frühen Phase der Gestaltung zur Beschreibung einzelner Funktionen eines Systems verwendet werden. Hierbei liegt die Trennung der Anpassungs- von der Anwendungslogik im Fokus. Eine Trennung beider Logiken kann die Qualität der Gestaltung steigern, da beteiligte Akteure detaillierte Aspekte von einer der beiden Logiken fokussieren können.

Grundsätzlich wird bei der Verwendung der Sprache in die *High-Level-Gestaltung* und die *Low-Level-Gestaltung* unterschieden. Im Rahmen der *High-Level-Gestaltung* werden in Anlehnung an *UML Use Case-Diagramme* einzelne Funktionen eines Systems beschrieben. Dabei können Funktionen der Anpassungslogik von Funktionen der Anwendungslogik separiert werden. Eine Unterscheidung von Funktionen beider Logiken wird damit bereits frühzeitig unterstützt. Die *Low-Level-Gestaltung* hingegen verfeinert

das im Rahmen der *High-Level-Gestaltung* spezifizierte Verhalten der Anpassungslogik. Dabei werden in *ACML4BPM* Sprachelemente der Sprache *BPMN2.0* verwendet, welche den De-facto-Standard zur Gestaltung von Prozessen darstellt. Hierdurch kann eine dedizierte domänenspezifische Gestaltung von Prozessen unterstützt werden, die so durch die ursprüngliche Sprache *ACML* nicht möglich gewesen ist.

Als wissenschaftlichen Beitrag ist neben der Möglichkeit einer dedizierten Gestaltung unter Verwendung von Konzepten der Domäne *BPM* auch weiteres domänenspezifisches Wissen in die Sprache *ACML4BPM* integriert worden. Dabei kennzeichnen sich domänenspezifische Sprachen insbesondere durch die Kapselung spezifischer Konzepte einer ausgewählten Domäne aus, wodurch u.a. die Qualität in der Gestaltung erhöht werden kann. Beispiele für derartiges Wissen sind durch spezifische Komponenten, Operationen zur Anpassung sowie durch unterschiedliche Ereignisse zur Auslösung von Anpassungen gegeben. Dabei konnte insbesondere dadurch ein Mehrwert generiert werden, dass im Gegensatz zu existierenden Ansätzen verschiedene Elemente aus allen vier durch *Curtis* [CKO92] eingeführten Perspektiven von Prozessen angepasst werden können. Die Sprache *ACML4BPM* unterstützt daher die *BPM*-spezifische und qualitativ hochwertige Gestaltung von flexiblen und anpassbaren Prozessen.

Entwurfsmuster für flexible und anpassbare Prozesse

Beantwortung von Forschungsfrage 2

Damit die Gestaltung von flexiblen und anpassbaren Prozessen weiter unterstützt werden kann, wurden unterschiedliche Facetten von Flexibilität in Prozessen analysiert und zugehörige Entwurfsmuster vorgestellt. Die Entwurfsmuster basieren dabei auf einem Katalog etablierter Flexibilitätsaspekte von Prozessen in der Domäne *BPM* (siehe [Sch+08]).

So wurden im Rahmen der Entwurfsmuster notwendige konzeptionelle Erweiterungen der Sprache *BPMN2.0* und aspektspezifische Operationen zur Anpassung vorgestellt. Die Funktionsprinzipien der Operationen wurden an schematischen Beispielen verdeutlicht. Die Gesamtheit aller Entwurfsmuster stellt Akteuren ein breites Spektrum an Möglichkeiten zur Verfügung, um die Gestaltung von flexiblen und anpassbaren Prozessen unter Verwendung der Sprache *ACML4BPM* vornehmen zu können.

Wie bereits an verschiedenen Stellen der Evaluation beschrieben (siehe Kapitel 7), stellen die Entwurfsmuster einen Mehrwert gegenüber der Sprache *ACML* dar, da eine erweiterte Verwendung der Sprache ermöglicht wird. Auch wenn die Erweiterungen der Sprache *BPMN2.0* sowie aspektspezifischer Operationen nicht Teil der Basissprache (hier: *ACML4BPM*)

sind, lassen sie sich als Erweiterung von *ACML4BPM* verstehen und einsetzen.

Als weiterer wissenschaftlicher Beitrag ist aber auch die Gesamtheit der Entwurfsmuster zu betrachten. So wurden für die durch [Sch+08] beschriebenen Flexibilitätsaspekte Entwurfsmuster unter Verwendung der Sprache *ACML4BPM* beschrieben. Dies stellt einen Mehrwert dar, da sich bisher existierende Arbeiten lediglich auf die Umsetzung einzelner Flexibilitätsaspekte ohne Trennung der Anwendungs- von der Anpassungslogik unter Verwendung der Sprache *BPMN2.0* beziehen. Beispiele für derartige Arbeiten sind durch [CMT10; Mur+13] gegeben, in denen die Flexibilitätsaspekte *Late Binding* bzw. *Late Selection* behandelt werden. Die in dieser Arbeit einheitlich beschriebenen Entwurfsmuster können somit auch als Leitfaden für die Gestaltung von flexiblen und anpassbaren Prozessen verstanden werden. Hierdurch können verschiedene an der Gestaltung beteiligte Akteure die Gesamtheit der Entwurfsmuster als Ausgangslage zur Auswahl eines geeigneten Typs von Flexibilität in Prozessen nutzen.

Adaptivity Engineering – Adapt Cases 4 BPM

Beantwortung von Forschungsfrage 3

Der durch *Luckey* [Luc13] vorgestellte Ansatz *Adapt Cases* ermöglicht die Gestaltung von selbst-adaptiven Systemen mit Fokus auf der Trennung der Anpassungs- und Anwendungslogik. Die Methode *Adapt Cases 4 BPM* greift diese grundlegende Idee für die methodische Gestaltung von flexiblen und anpassbaren Prozessen wieder auf. Dabei werden unter Verwendung von Konzepten der Domäne *BPM* verschiedene Redefinitionen, Erweiterungen und Anwendungen ursprünglicher Konzepte des Ansatzes *Adapt Cases* vorgenommen. Dabei kann die Methode *Adapt Cases 4 BPM* ebenso wie *Adapt Cases* in einer frühen Phase in der Gestaltung eingesetzt werden.

Der daraus resultierende wissenschaftliche Beitrag kann aus verschiedenen Perspektiven betrachtet werden. So stellt der Ansatz *Adapt Cases 4 BPM* ein Beispiel für die Anwendung bereits existierender Konzepte des *Adaptivity Engineering* auf eine spezifische Anwendungsdomäne (hier: *BPM*) dar. Dabei war es notwendig, dass für die Anwendungsdomäne *BPM* neue Elemente hinzugefügt oder existierende Elemente in Anlehnung an neue spezifische Konzepte aus der Domäne redefiniert werden mussten. Hierdurch entstanden neue Eigenschaften in der Sprache *ACML4BPM*, durch die die Ausdrucksfähigkeit gegenüber der Sprache *ACML* gesteigert werden konnte. Ein weiterer Mehrwert, der durch den Ansatz *Adapt Cases 4 BPM* gegenüber dem Ansatz *Adapt Cases* entsteht,

ist eine erweiterte Fähigkeit zur Gestaltung von flexiblen und anpassbaren Prozessen. Die Verwendung von Elementen der Sprache *BPMN2.0* anstelle von *UML Aktivitätsdiagrammen* stellt in diesem Bezug einen für die Domäne *BPM* spezifischeren Weg der Gestaltung dar. Hierdurch können verschiedene Akteure in die Lage versetzt werden, auf Basis des De-facto-Standards *BPMN2.0* Prozesse geeigneter gestalten zu können.

Zuletzt bilden die Entwurfsmuster mit ihren erarbeiteten konzeptionellen Erweiterungen der Sprache *BPMN2.0* und ihren Operationen zur Anpassung einen Mehrwert. So waren bspw. derartige Entwurfsmuster für den Ansatz *Adapt Cases* nicht existent. Die vorgenommene Integration von domänenspezifischem Wissen durch unterschiedliche Flexibilitätsaspekte ermöglicht somit eine erweiterte methodische Fähigkeit des erarbeiteten Ansatzes *Adapt Cases 4 BPM*.

8.2 Ausblick

In dieser Arbeit konnten drei wichtige Lösungsteile für die Gestaltung von flexiblen und anpassbaren Prozessen erarbeitet werden. Im Rahmen der Bearbeitung sind dabei weitere Fragestellungen entstanden. Durch eine Beantwortung dieser Fragestellungen kann das *Adaptivity Engineering* sowohl im generellen als auch im spezifischen Kontext einer Domäne weiterentwickelt werden. Nachfolgend wird auf wichtige Eckpunkte dieser Fragestellungen eingegangen.

Evaluation im industriellen Kontext

Ausgesuchte Inhalte der vorliegenden Arbeit wurden auf Basis eines praxisnahen Szenarios plausibilisiert. Zusätzlich wurden verschiedene Kataloge von Kriterien verwendet, mit denen Eigenschaften der Sprache *ACML4BPM* im Rahmen einer selbstständig durchgeführten Bewertung untersucht worden sind. Eine sinnvolle Ergänzung stellt eine zusätzliche Evaluation im industriellen Kontext dar. So könnten weitere Kriterien für Eigenschaften der Sprache untersucht, bewertet und gegebenenfalls Bedarf für Erweiterungen identifiziert werden.

Eine mögliche Kategorie dieser Eigenschaften ist durch die Benutzerfreundlichkeit (engl. *Usability*) gegeben. So stellt sich die Frage, ob vorgestellte Elemente der Sprache *ACML4BPM* durch verschiedene in der Gestaltung beteiligte Akteure adäquat verstanden werden können. Diese Art von Verständnis der vorgestellten Sprache ist insbesondere dann wichtig,

wenn verschiedene Akteure parallel oder sequentiell die Gestaltung von Prozessen auf Basis von mit *ACML4BPM* erstellten Modellen vornehmen. Daher ist zu prüfen, ob ein gewisser Grad an Benutzerfreundlichkeit bei der Verwendung des vorgestellten Ansatzes besteht. Beispiele für bestehende Ansätze zur Evaluation von Eigenschaften der Benutzerfreundlichkeiten sind durch *Moody* und *Hillegersberg* [MH08; Moo09] gegeben.

Ein weiterer Aspekt, der durch eine Evaluation im industriellen Kontext untersucht werden könnte, ist durch spezifische Anforderungen aus der Praxis gegeben. So können aus einem konkreten industriellen Kontext weitere Anforderungen an eine Sprache und an die zugehörige Entwurfsmethodik gesetzt werden. Für den Gesamtansatz *Adapt Cases 4 BPM* stellt sich in diesem Bezug die Fragestellung, ob beschriebene Konzepte derartige Anforderungen bereits erfüllen oder ob weitere Konzepte in den Ansatz integriert werden müssen.

Werkzeugunterstützung

Weitere Arbeiten in Bezug zu dem Ansatz *Adapt Cases 4 BPM* können hinsichtlich der Bereitstellung einer Werkzeugunterstützung durchgeführt werden. Dabei können die Aspekte der Phase *Gestaltung* sowie der Phase *Ausführung* und der Phase *Evaluation* besonders hervorgehoben werden.

In dieser Arbeit wurden verschiedene Konzepte vorgestellt, mit denen flexible und anpassbare Prozesse gestaltet werden können. Dabei wurde auf die Entwicklung eines Softwarewerkzeugs zur Gestaltung bewusst verzichtet. Ein solches Softwarewerkzeug ist z.B. durch einen grafischen Editor gegeben, welcher die Akzeptanz und Anwendbarkeit des vorgestellten Ansatzes, insbesondere auch in Bezug zu der zuvor genannten Evaluation im industriellen Kontext, erhöhen kann.

Der zweite Aspekt hinsichtlich der Phase *Ausführung* und der Phase *Evaluation* geht über das für diese Arbeit gesetzte Ziel der Gestaltung von flexiblen und anpassbaren Prozessen hinaus und beschäftigt sich vornehmlich mit weiterführenden Phasen des *BPM-Lebenszyklus*. So stellen sich bspw. für die Phase *Ausführung* und die Phase *Evaluation* die Frage, wie beschriebene Konzepte hier übertragen werden können.

Für die Phase *Ausführung* gilt dabei zu untersuchen, ob und welche Vorteile eine Trennung der Anpassungs- und Anwendungslogik ermöglichen oder aber weiterhin sinnvoll erscheinen lässt. So ist es vorstellbar, dass für spezifische Anwendungen unterschiedliche Anforderungen für die Ausführung der beteiligten Logiken bestehen können. Als Beispiel lässt sich

eine Anforderung beschreiben, in deren Realisierung beide Logiken auch in der Phase *Ausführung* getrennt ausgeführt werden sollen. Im Gegensatz zu dieser Anforderung steht die Integration beider Logiken, wie z.B. die Ausführung im Rahmen einer gemeinsamen Workflow-Engine. Dabei lassen sich auch weitere Anforderungen beschreiben, für die weitere Überlegungen notwendig sind.

In der Phase *Evaluation* stehen häufig auch statistische Daten über aktive und bereits beendete Prozesse zur Verfügung. Dieser Umstand kann auf Basis neuartiger Methoden aus dem Bereich des *Machine Learning* [WFH11] so eingesetzt werden, dass Verbesserungen von Prozessen auch (teil-)automatisiert werden können. Dabei können Verbesserungen von bestehenden Prozessen bereits durch die Verwendung des Entwurfsmusters *Flexibility-by Change* unterstützt werden. Eine sinnvolle Erweiterung um fortgeschrittene Analysetechniken, die z.B. im Rahmen des *Machine Learning* existent sind, kann insbesondere auch im Umfeld der Automatisierung von Prozessen einen großen Mehrwert bieten.

Gestaltung von *-zentrierten Prozessen

Diese Forschungsarbeit wurde durch verschiedene Fragestellungen aus dem Kontext des *NRW Fortschrittskollegs „Gestaltung von flexiblen Arbeitswelten"* eingeleitet. Sie betreffen zum einen die Flexibilisierung von Prozessen, auf die in dieser Arbeit explizit eingegangen worden ist. Zum anderen stehen sie aber auch in Bezug zum Konzept der Menschenzentrierung. Eine derartige Zentrierung kann als eine Art von Perspektive verstanden werden, in der Eigenschaften von den im Fokus stehenden Entitäten berücksichtigt werden sollen. Hierbei können unterschiedliche Aktivitäten entlang der Phasen des *BPM-Lebenszyklus* betroffen sein. So kann z.B. allgemein von einer **-zentrierten* Gestaltung oder Ausführung von Prozessen gesprochen werden, wenn der Fokus auf Entitäten wie Menschen, Maschinen oder IT-Dienste gesetzt wird.

Für das in dieser Arbeit vorgestellte domänenspezifische *Adaptivity Engineering* ergeben sich hieraus potentiell neue Möglichkeiten durch eine **-zentrierte* Gestaltung von Prozessen. So könnte aus der Perspektive eines Prozesses für jede dieser Entitäten eine Komponente in seiner Umgebung zur Verfügung stehen, die wesentliche Konzepte zur **-zentrierung* enthält. Durch zugehörige Sensor- und Effektorschnittstellen lassen sich kontrollierte lesende Zugriffe auf Eigenschaften bzw. auch Anpassungen ermöglichen. Dabei ergeben sich jedoch offene Fragestellungen hinsichtlich eines eingesetzten Ansatzes zur Gestaltung dieser Eigenschaften der Entitäten

Mensch, Maschine und IT-Dienst. Eine Möglichkeit wäre eine sprachbasierte Lösung anzustreben, die insbesondere verschiedenes domänenspezifisches Wissen in einer gemeinsamen Sprache zur Gestaltung von Umgebungskomponenten integriert. Hierdurch wäre ein erweitertes *Adaptivity Engineering* in der Lage, Konzepte dediziert im Kontext von Prozessen zu berücksichtigen, um eine **-zentrierung* zu erreichen.

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[Aal16] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016 (siehe S. 34, 201).

[Ada+06] Michael Adams, Arthur H. M. ter Hofstede, David Edmond und Wil M. P. van der Aalst. „Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows“. In: *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences, CoopIS, DOA, GADA, and ODBASE 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part I*. Hrsg. von Robert Meersman und Zahir Tari. Bd. 4275. Lecture Notes in Computer Science. Springer, 2006, S. 291–308 (siehe S. 60, 169, 256).

[Ada+07] Michael Adams, Arthur H. M. ter Hofstede, Wil M. P. van der Aalst und David Edmond. „Dynamic, Extensible and Context-Aware Exception Handling for Workflows“. In: *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I*. Hrsg. von Robert Meersman und Zahir Tari. Bd. 4803. Lecture Notes in Computer Science. Springer, 2007, S. 95–112 (siehe S. 60, 256).

[AHW03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede und Mathias Weske. „Business Process Management: A Survey“. In: *Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings*. Hrsg. von Wil M. P. van der Aalst, Arthur H. M. ter Hofstede und Mathias Weske. Bd. 2678. Lecture Notes in Computer Science. Springer, 2003, S. 1–12 (siehe S. 29, 31, 32, 43).

[AJ00] Wil M. P. van der Aalst und Stefan Jablonski. „Dealing with workflow change: identification of issues and solutions“. In: *Computer systems science and engineering* 15.5 (2000), S. 267–276 (siehe S. 43).

[And+09] Jesper Andersson, Rogério de Lemos, Sam Malek und Danny Weyns. „Modeling Dimensions of Self-Adaptive Software Systems“. In: *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*. Hrsg. von Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi und Jeff Magee. Bd. 5525. Lecture Notes in Computer Science. Springer, 2009, S. 27–47 (siehe S. 228–230, 235, 241).

[Ard+11] Danilo Ardagna, Luciano Baresi, Sara Comai, Marco Comuzzi und Barbara Pernici. „A Service-Based Framework for Flexible Business Processes“. In: *IEEE Software* 28.2 (2011), S. 61–67 (siehe S. 169).

[ARD07] Wil M. P. van der Aalst, Michael Rosemann und Marlon Dumas. „Deadline-based escalation in process-aware information systems“. In: *Decision Support Systems* 43.2 (2007), S. 492–511 (siehe S. 43).

[AT05] Wil M. P. van der Aalst und Arthur H. M. Ter Hofstede. „YAWL: yet another workflow language“. In: *Information systems* 30.4 (2005), S. 245–275 (siehe S. 41, 60, 256).

[AWG05] Wil M. P. van der Aalst, Mathias Weske und Dolf Grünbauer. „Case handling: a new paradigm for business process support“. In: *Data & Knowledge Engineering* 53.2 (2005), S. 129–162 (siehe S. 36, 60, 256).

[AWM04] Wil M. P. van der Aalst, Ton Weijters und Laura Maruster. „Workflow Mining: Discovering Process Models from Event Logs“. In: *IEEE Transactions on Knowledge & Data Engineering* 16.9 (2004), S. 1128–1142 (siehe S. 34).

[Ayo+16] Clara Ayora, Victoria Torres, Jose Luis de la Vara und Vicente Pelechano. „Variability management in process families through change patterns“. In: *Information & Software Technology* 74 (2016), S. 86–104 (siehe S. 60).

[Bar+11] Angineh Barkhordarian, Frederik Demuth, Kristof Hamann, Minh Hoang, Sonja Weichler und Sonja Zaplata. „Migratability of BPMN 2.0 Process Instances“. In: *Service-Oriented Computing - ICSOC 2011 Workshops - ICSOC 2011, International Workshops WESOA, NFPSLAM-SOC, and Satellite Events, Paphos, Cyprus, December 5-8, 2011. Revised Selected Papers*. Hrsg. von George Pallis, Mohamed Jmaiel, Anis Charfi, Sven Graupner, Yücel Karabulut, Sam Guinea, Florian Rosenberg, Quan Z. Sheng, Cesare Pautasso und Sonia Ben Mokhtar. Bd. 7221. Lecture Notes in Computer Science. Springer, 2011, S. 66–75 (siehe S. 133).

[BCW17] Marco Brambilla, Jordi Cabot und Manuel Wimmer. *Model-Driven Software Engineering in Practice, Second Edition*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017 (siehe S. 15–17, 19, 23, 26).

[BDP14] Paolo Bocciarelli, Andrea D'Ambrogio und Emiliano Paglia. „A Language for Enabling Model-Driven Analysis of Business Processes“. In: *MODELSWARD 2014 - Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, Lisbon, Portugal, 7 - 9 January, 2014*. Hrsg. von Luis Ferreira Pires, Slimane Hammoudi, Joaquim Filipe und Rui César das Neves. SciTePress, 2014, S. 325–332 (siehe S. 61).

[Bis11] Adnan Biser. „Evaluation of Adapt Cases“. Masterarbeit. Universität Paderborn, 2011 (siehe S. 228, 229, 233, 235, 247, 251, 260).

[Boc+14a] Paolo Bocciarelli, Andrea D'Ambrogio, Andrea Giglio, Emiliano Paglia und Daniele Gianni. „A Transformation Approach to Enact the Design-Time Simulation of BPMN Models". In: *2014 IEEE 23rd International WETICE Conference, WETICE 2014, Parma, Italy, 23-25 June, 2014*. Hrsg. von Sumitra Reddy. IEEE Computer Society, 2014, S. 199–204 (siehe S. 61).

[Boc+14b] Paolo Bocciarelli, Andrea D'Ambrogio, Andrea Giglio, Emiliano Paglia und Daniele Gianni. „Empowering business process simulation through automated model transformations". In: *2014 Spring Simulation Multiconference, SpringSim '14, Tampa, FL, USA, April 13-16, 2014, Proceedings of the Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*. ACM, 2014, S. 39 (siehe S. 61).

[Boc+16] Paolo Bocciarelli, Andrea D'Ambrogio, Andrea Giglio und Emiliano Paglia. „A BPMN Extension to Enable the Explicit Modeling of Task Resources." In: *CIISE*. 2016, S. 40–47 (siehe S. 61).

[Boc+17] Paolo Bocciarelli, Andrea D'Ambrogio, Andrea Giglio und Emiliano Paglia. „A BPMN extension for modeling Cyber-Physical-Production-Systems in the context of Industry 4.0". In: *14th IEEE International Conference on Networking, Sensing and Control, ICNSC 2017, Calabria, Italy, May 16-18, 2017*. Hrsg. von Giancarlo Fortino, MengChu Zhou, Zofia Lukszo, Athanasios V. Vasilakos, Francesco Basile, Carlos E. Palau, Antonio Liotta, Maria Pia Fanti, Antonio Guerrieri und Andrea Vinci. IEEE, 2017, S. 599–604 (siehe S. 61).

[Bru+09] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger M. Kienle, Marin Litoiu, Hausi A. Müller, Mauro Pezzè und Mary Shaw. „Engineering Self-Adaptive Systems through Feedback Loops". In: *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*. Hrsg. von Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi und Jeff Magee. Bd. 5525. Lecture Notes in Computer Science. Springer, 2009, S. 48–70 (siehe S. 6, 9).

[Can+08] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito und Maria Luisa Villani. „A framework for QoS-aware binding and re-binding of composite web services". In: *Journal of Systems and Software* 81.10 (2008), S. 1754–1769 (siehe S. 169).

[Cas+99] Fabio Casati, Stefano Ceri, Stefano Paraboschi und Giuseppe Pozzi. „Specification and Implementation of Exceptions in Workflow Management Systems". In: *ACM Transactions on Database Systems* 24.3 (1999), S. 405–451 (siehe S. 36).

[CDM09] Anis Charfi, Tom Dinkelaker und Mira Mezini. „A Plug-in Architecture for Self-Adaptive Web Service Compositions". In: *IEEE International Conference on Web Services, ICWS 2009, Los Angeles, CA, USA, 6-10 July 2009*. IEEE Computer Society, 2009, S. 35–42 (siehe S. 169).

[CKO92] Bill Curtis, Marc I. Kellner und Jim Over. „Process Modeling“. In: *Communications of the ACM* 35.9 (Sep. 1992), S. 75–90 (siehe S. 42–44, 89, 92, 93, 105, 244, 265).

[Cla+09] Peter Clark, William R. Murray, Philip Harrison und John A. Thompson. „Naturalness vs. Predictability: A Key Debate in Controlled Languages“. In: *Controlled Natural Language, Workshop on Controlled Natural Language, CNL 2009, Marettimo Island, Italy, June 8-10, 2009. Revised Papers*. Hrsg. von Norbert E. Fuchs. Bd. 5972. Lecture Notes in Computer Science. Springer, 2009, S. 65–81 (siehe S. 202).

[CMT10] Pierre Châtel, Jacques Malenfant und Isis Truck. „QoS-based Late-Binding of Service Invocations in Adaptive Business Processes“. In: *IEEE International Conference on Web Services, ICWS 2010, Miami, Florida, USA, July 5-10, 2010*. IEEE Computer Society, 2010, S. 227–234 (siehe S. 37, 266).

[Coa96] Workflow Management Coalition. *Workflow Management Coalition terminology and glossary*. Techn. Ber. WFMC-TC-1011. Workflow Management Coalition, 1996 (siehe S. 30).

[Deu+15] Jochen Deuse, Kirsten Weisner, André Hengstebeck und Felix Busch. „Gestaltung von Produktionssystemen im Kontext von Industrie 4.0“. In: *Zukunft der Arbeit in Industrie 4.0*. Springer, 2015, S. 99–109 (siehe S. 3).

[Dij76] Edsger Wybe Dijkstra. *A discipline of programming*. Bd. 1. prentice-hall Englewood Cliffs, 1976 (siehe S. 7).

[DP10] Ken Decreus und Geert Poels. „A Goal-Oriented Requirements Engineering Method for Business Processes“. In: *Information Systems Evolution - CAiSE Forum 2010, Hammamet, Tunisia, June 7-9, 2010, Selected Extended Papers*. Hrsg. von Pnina Soffer und Erik Proper. Bd. 72. Lecture Notes in Business Information Processing. Springer, 2010, S. 29–43 (siehe S. 202).

[Dum+18] Marlon Dumas, Marcello La Rosa, Jan Mendling und Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018 (siehe S. 32, 199).

[DZK11] Markus Döhring, Birgit Zimmermann und Lars Karg. „Flexible Workflows at Design- and Runtime Using BPMN2 Adaptation Patterns“. In: *Business Information Systems - 14th International Conference, BIS 2011, Poznan, Poland, June 15-17, 2011. Proceedings*. Hrsg. von Witold Abramowicz. Bd. 87. Lecture Notes in Business Information Processing. Springer, 2011, S. 25–36 (siehe S. 39, 59, 169).

[Ecl] Eclipse Foundation. *Eclipse Modeling Framework* (siehe S. 18).

[Eco] Ecore. *Ecore Metamodell* (siehe S. 18).

[Eng+00] Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel und Stefan Sauer. „Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML“. In: *«UML» 2000 - The Unified Modeling Language, Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, Proceedings*. Hrsg. von Andy Evans, Stuart Kent und Bran Selic. Bd. 1939. Lecture Notes in Computer Science. Springer, 2000, S. 323–337 (siehe S. 16, 21, 50).

[Eng+18] Gregor Engels, Günter W. Maier, Sonja K. Ötting, Eckhard Steffen und Alexander Teetz. „Gerechtigkeit in flexiblen Arbeits- und Managementprozessen“. In: *Zukunft der Arbeit – Eine praxisnahe Betrachtung*. Springer, 1. Jan. 2018 (siehe S. 214).

[ES11] Florian Evequoz und Christoph Sterren. *Waiting for the miracle: Comparative analysis of twelve business process management systems regarding the support of BPMN 2.00 palette and export*. Techn. Ber. Tech. rep., University of Applied Sciences Western Switzerland Google Scholar, 2011 (siehe S. 257).

[EST18] Gregor Engels, Thim Strothmann und Alexander Teetz. „Adapt Cases 4 BPM - A Modeling Framework for Process Flexibility in IIoT“. In: *22nd IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2018, Stockholm, Sweden, October 16-19, 2018*. IEEE Computer Society, 2018, S. 59–68 (siehe S. 214).

[ESW07] Gregor Engels, Christian Soltenborn und Heike Wehrheim. „Analysis of UML Activities Using Dynamic Meta Modeling“. In: *Formal Methods for Open Object-Based Distributed Systems, 9th IFIP WG 6.1 International Conference, FMOODS 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings*. Hrsg. von Marcello M. Bonsangue und Einar Broch Johnsen. Bd. 4468. Lecture Notes in Computer Science. Springer, 2007, S. 76–90 (siehe S. 50).

[ET18] Gregor Engels und Alexander Teetz. „Flexible Arbeitsprozesse“. In: *Handbuch Gestaltung digitaler und vernetzter Arbeitswelten*. Springer, 1. Jan. 2018 (siehe S. 214).

[Eva03] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Pearson Education, 2003 (siehe S. 23).

[Faz16] Masud Fazal-Baqaie. „Project-specific software engineering methods: composition, enactment, and quality assurance“. Dissertation. Universität Paderborn, 2016 (siehe S. 170).

[FKK08] Norbert E. Fuchs, Kaarel Kaljurand und Tobias Kuhn. „Attempto Controlled English for Knowledge Representation“. In: *Reasoning Web, 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures*. Hrsg. von Cristina Baroglio, Piero A. Bonatti, Jan Maluszynski, Massimo Marchiori, Axel Polleres und Sebastian Schaffert. Bd. 5224. Lecture Notes in Computer Science. Springer, 2008, S. 104–124 (siehe S. 202).

[Fow10] Martin Fowler. *Domain Specific Languages*. 1st. Addison-Wesley Professional, 2010 (siehe S. 21).

[FR14] Jakob Freund und Bernd Rücker. *Praxishandbuch BPMN 2.0*. Carl Hanser Verlag GmbH Co KG, 2014 (siehe S. 29).

[Gad08] Andreas Gadatsch. *Grundkurs Geschäftsprozess-Management: Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker*. Springer-Verlag, 2008 (siehe S. 27, 30, 41).

[Ger13] Christian Gerth. *Business Process Models. Change Management*. Bd. 7849. Lecture Notes in Computer Science. Springer, 2013 (siehe S. 29, 88, 105, 244).

[Got+08] Florian Gottschalk, Wil M. P. van der Aalst, Monique H. Jansen-Vullers und Marcello La Rosa. „Configurable Workflow Models“. In: *International Journal of Cooperative Information Systems* 17.2 (2008), S. 177–221 (siehe S. 39).

[Gra+16] Imen Graja, Slim Kallel, Nawal Guermouche und Ahmed Hadj Kacem. „BPMN4CPS: A BPMN Extension for Modeling Cyber-Physical Systems“. In: *25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2016, Paris, France, June 13-15, 2016*. Hrsg. von Sumitra Reddy und Walid Gaaloul. IEEE Computer Society, 2016, S. 152–157 (siehe S. 62).

[GTG15] Jennifer E. Gerow, Jason Bennett Thatcher und Varun Grover. „Six types of IT-business strategic alignment: an investigation of the constructs and their measurement“. In: *EJIS* 24.5 (2015), S. 465–491 (siehe S. 202).

[GW13] Matthias Geiger und Guido Wirtz. „BPMN 2.0 Serialization - Standard Compliance Issues and Evaluation of Modeling Tools“. In: *Enterprise Modelling and Information Systems Architectures: Proceedings of the 5th International Workshop on Enterprise Modelling and Information Systems Architectures, EMISA 2013, St. Gallen, Switzerland, September 5-6, 2013*. Hrsg. von Reinhard Jung und Manfred Reichert. Bd. 222. LNI. GI, 2013, S. 177–190 (siehe S. 257).

[Hau05] Jan Hendrik Hausmann. „Dynamic META modeling: a semantics description technique for visual modeling languages“. Dissertation. Universität Paderborn, 2005 (siehe S. 16, 21).

[HBR10] Alena Hallerbach, Thomas Bauer und Manfred Reichert. „Configuration and management of process variants“. In: *Handbook on Business Process Management*. Springer, 2010, S. 237–255 (siehe S. 39).

[JB96] Stefan Jablonski und Christoph Bussler. *Workflow management - modeling concepts, architecture and implementation*. International Thomson, 1996 (siehe S. 43).

[Jes+14] Sabina Jeschke, René Vossen, Ingo Leisten, Florian Welter, Stella Fleischer und Thomas Thiele. „Industrie 4.0 als Treiber der demografischen Chancen“. In: *Automation, Communication and Cybernetics in Science and Engineering 2013/2014*. Springer, 2014, S. 75–85 (siehe S. 4).

[Kar+14] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler und Steven Völkel. „Design Guidelines for Domain Specific Languages“. In: *CoRR* abs/1409.2378 (2014). arXiv: 1409.2378 (siehe S. 26).

[Kau15] Timothy Kaufmann. *Geschäftsmodelle in Industrie 4.0 und dem Internet der Dinge: Der Weg vom Anspruch in die Wirklichkeit*. Springer-Verlag, 2015 (siehe S. 3).

[KC03] Jeffrey O. Kephart und David M. Chess. „The Vision of Autonomic Computing“. In: *IEEE Computer* 36.1 (2003), S. 41–50 (siehe S. 51, 62, 74, 77, 79, 216, 227, 252, 264).

[KJP15] Martin Krzywdzinski, Ulrich Jürgens und Sabine Pfeiffer. „Die vierte Revolution Wandel der Produktionsarbeit im Digitalisierungszeitalter“. In: *WZB Mitteilungen* 149 (2015), S. 6–9 (siehe S. 3).

[Kur16] Matthias Kurz. „BPMN Model Interchange: The Quest for Interoperability“. In: *Proceedings of the 8th International Conference on Subject-oriented Business Process Management, S-BPM ONE 2016, Erlangen, Germany, April 7-8, 2016*. Hrsg. von Jorge L. Sanz. ACM, 2016, 6:1–6:10 (siehe S. 257).

[Lag+07] François Lagarde, Huáscar Espinoza, François Terrier und Sébastien Gérard. „Improving uml profile design practices by leveraging conceptual domain models“. In: *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*. Hrsg. von R. E. Kurt Stirewalt, Alexander Egyed und Bernd Fischer. ACM, 2007, S. 445–448 (siehe S. 25).

[Las+14] Heiner Lasi, Privatdozent Dr Peter Fettke, Hans-Georg Kemper, Dipl-Inf Thomas Feld und Dipl-Hdl Michael Hoffmann. „Industrie 4.0“. In: *Wirtschaftsinformatik* 56.4 (2014), S. 261–264 (siehe S. 3).

[LE13] Markus Luckey und Gregor Engels. „High-quality specification of self-adaptive software systems“. In: *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2013, San Francisco, CA, USA, May 20-21, 2013*. Hrsg. von Marin Litoiu und John Mylopoulos. IEEE Computer Society, 2013, S. 143–152 (siehe S. 6, 51, 53–57, 68, 104, 216, 228).

[Luc+11] Markus Luckey, Benjamin Nagel, Christian Gerth und Gregor Engels. „Adapt cases: extending use cases for adaptive systems“. In: *2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2011, Waikiki, Honolulu , HI, USA, May 23-24, 2011*. Hrsg. von

Holger Giese und Betty H. C. Cheng. ACM, 2011, S. 30–39 (siehe S. 6, 9, 50, 51, 56, 67, 70, 103, 104, 264).

[Luc13] Markus Luckey. „Adaptivity engineering: modeling and quality assurance for self-adptive software systems“. Dissertation. Universität Paderborn, 2013 (siehe S. 6, 7, 15, 18, 50, 72, 199, 202, 207, 266).

[Lud+16] Thomas Ludwig, Christoph Kotthaus, Martin Stein, Hartwig Durt, Constanze Kurz, Julian Wenz, Thorsten Doublet, Maximilian Becker, Volkmar Pipek und Volker Wulf. „Arbeiten im Mittelstand 4.0–KMU im Spannungsfeld des digitalen Wandels“. In: *HMD Praxis der Wirtschaftsinformatik* 53.1 (Jan. 2016), S. 71–86 (siehe S. 2).

[MG09] Milan Milanovic und Dragan Gasevic. „Towards a Language for Rule-Enhanced Business Process Modeling“. In: *Proceedings of the 13th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2009, 1-4 September 2009, Auckland, New Zealand*. IEEE Computer Society, 2009, S. 64–73 (siehe S. 60).

[MH08] Daniel Moody und Jos van Hillegersberg. „Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams“. In: *International Conference on Software Language Engineering*. Springer. 2008, S. 16–34 (siehe S. 268).

[MHW17] Sankalita Mandal, Marcin Hewelt und Mathias Weske. „A Framework for Integrating Real-World Events and Business Processes in an IoT Environment“. In: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I*. Hrsg. von Hervé Panetto, Christophe Debruyne, Walid Gaaloul, Mike P. Papazoglou, Adrian Paschke, Claudio Agostino Ardagna und Robert Meersman. Bd. 10573. Lecture Notes in Computer Science. Springer, 2017, S. 194–212 (siehe S. 61).

[MM17] Andrea Marrella und Massimo Mecella. „Cognitive Business Process Management for Adaptive Cyber-Physical Processes“. In: *Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers*. Hrsg. von Ernest Teniente und Matthias Weidlich. Bd. 308. Lecture Notes in Business Information Processing. Springer, 2017, S. 429–439 (siehe S. 63).

[Moo09] Daniel L. Moody. „The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering“. In: *IEEE Transactions on Software Engineering* 35.6 (2009), S. 756–779 (siehe S. 268).

[MRH15] Sonja Meyer, Andreas Ruppen und Lorenz M. Hilty. „The Things of the Internet of Things in BPMN“. In: *Advanced Information Systems Engineering Workshops - CAiSE 2015 International Workshops, Stockholm, Swe-*

den, June 8-9, 2015, Proceedings. Hrsg. von Anne Persson und Janis Stirna. Bd. 215. Lecture Notes in Business Information Processing. Springer, 2015, S. 285–297 (siehe S. 60).

[MRM13] Sonja Meyer, Andreas Ruppen und Carsten Magerkurth. „Internet of Things-Aware Process Modeling: Integrating IoT Devices as Business Process Resources“. In: *Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings*. Hrsg. von Camille Salinesi, Moira C. Norrie und Oscar Pastor. Bd. 7908. Lecture Notes in Computer Science. Springer, 2013, S. 84–98 (siehe S. 60).

[Mur+13] Aitor Murguzur, Goiuria Sagardui, Karmele Intxausti und Salvador Trujillo. „Process Variability through Automated Late Selection of Fragments“. In: *Advanced Information Systems Engineering Workshops - CAiSE 2013 International Workshops, Valencia, Spain, June 17-21, 2013. Proceedings*. Hrsg. von Xavier Franch und Pnina Soffer. Bd. 148. Lecture Notes in Business Information Processing. Springer, 2013, S. 371–385 (siehe S. 37, 168, 266).

[Mur89] Tadao Murata. „Petri nets: Properties, analysis and applications“. In: *Proceedings of the IEEE* 77.4 (1989), S. 541–580 (siehe S. 41).

[Nag15] Benjamin Nagel. „Goal-oriented business process engineering“. Dissertation. Universität Paderborn, 2015 (siehe S. 202).

[OAS07] OASIS. *Web Services Business Process Execution Language (WS-BPEL)*. Techn. Ber. Version 2.0. Web Services Business Process Execution Language (WS-BPEL). Organization for the Advancement of Structured Information Standards (OASIS), 2007 (siehe S. 40, 104).

[OMG10] OMG. *Unified Modeling Language TM (OMG UML): Superstructure*. Techn. Ber. February. Unified Modeling Language TM (OMG UML): Superstructure. Object Management Group, 2010 (siehe S. 7, 10, 18, 19, 22, 24, 44, 81, 82, 98, 151, 179).

[OMG11] OMG. *Business Process Model and Notation (BPMN)*. Techn. Ber. Version 2.0. Business Process Model and Notation (BPMN). Object Management Group, 2011 (siehe S. 10, 18, 19, 24, 25, 36, 40, 44, 46, 83, 97, 116).

[OMG14a] OMG. *Model Driven Architecture (MDA): MDA Guide*. Techn. Ber. Version 2.0. Model Driven Architecture (MDA): MDA Guide. Object Management Group, 2014 (siehe S. 18).

[OMG14b] OMG. *Object Contraint Language*. Techn. Ber. Version 2.0. Object Contraint Language. Object Management Group, 2014 (siehe S. 18, 24, 26).

[OMG15a] OMG. *Meta Object Facility (MOF)*. Techn. Ber. Version 2.5. Meta Object Facility (MOF). Object Management Group, 2015 (siehe S. 16).

[OMG15b] OMG. *Unified Modeling Language (UML): Activity Diagrams*. Techn. Ber. Object Management Group, 2015 (siehe S. 36, 44, 104).

[OMG16a] OMG. *Case Model Management and Notation*. Techn. Ber. Object Management Group, 2016 (siehe S. 104).

[OMG16b] OMG. *Decision Model and Notation*. Techn. Ber. Object Management Group, 2016 (siehe S. 104).

[PA06] Maja Pesic und Wil M. P. van der Aalst. „A Declarative Approach for Flexible Business Processes Management“. In: *Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006, Proceedings*. Hrsg. von Johann Eder und Schahram Dustdar. Bd. 4103. Lecture Notes in Computer Science. Springer, 2006, S. 169–180 (siehe S. 60, 256).

[Par02] David Lorge Parnas. „On the Criteria To Be Used in Decomposing Systems into Modules (Reprint)“. In: *Software Pioneers.* Hrsg. von Manfred Broy und Ernst Denert. Springer Berlin Heidelberg, 2002, S. 411–427 (siehe S. 7).

[Pes+07] Maja Pesic, M. H. Schonenberg, Natalia Sidorova und Wil M. P. van der Aalst. „Constraint-Based Workflow Models: Change Made Easy“. In: *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I*. Hrsg. von Robert Meersman und Zahir Tari. Bd. 4803. Lecture Notes in Computer Science. Springer, 2007, S. 77–94 (siehe S. 60, 257).

[Poe+13] Geert Poels, Ken Decreus, Ben Roelens und Monique Snoeck. „Investigating Goal-Oriented Requirements Engineering for Business Processes“. In: *Journal of Database Management* 24.2 (2013), S. 35–71 (siehe S. 202).

[RA07] Michael Rosemann und Wil M. P. van der Aalst. „A configurable reference modelling language“. In: *Information Systems* 32.1 (2007), S. 1–23 (siehe S. 39).

[RD09] Manfred Reichert und Peter Dadam. „Enabling Adaptive Process-aware Information Systems with ADEPT2.“ In: *Handbook of Research on Business Process Modeling* (Jan. 2009) (siehe S. 41).

[Ren03] Arend Rensink. „The GROOVE Simulator: A Tool for State Space Generation“. In: *Applications of Graph Transformations with Industrial Relevance, Second International Workshop, AGTIVE 2003, Charlottesville, VA, USA, September 27 - October 1, 2003, Revised Selected and Invited Papers*. Hrsg. von John L. Pfaltz, Manfred Nagl und Boris Böhlen. Bd. 3062. Lecture Notes in Computer Science. Springer, 2003, S. 479–485 (siehe S. 50).

[RG02] Mark Richters und Martin Gogolla. „OCL: Syntax, Semantics, and Tools“. In: *Object Modeling with the OCL, The Rationale behind the Object Constraint Language*. Hrsg. von Tony Clark und Jos Warmer. Bd. 2263. Lecture Notes in Computer Science. Springer, 2002, S. 42–68 (siehe S. 18).

[RR10] Stefanie Rinderle-Ma und Manfred Reichert. „Advanced Migration Strategies for Adaptive Process Management Systems“. In: *12th IEEE Conference on Commerce and Enterprise Computing, CEC 2010, Shanghai, China, November 10-12, 2010*. Hrsg. von Kuo-Ming Chao, Christian Huemer, Birgit Hofreiter, Yinsheng Li und Nazaraf Shah. IEEE Computer Society, 2010, S. 56–63 (siehe S. 133).

[RRD03] Manfred Reichert, Stefanie Rinderle und Peter Dadam. „Adept workflow management system“. In: *International Conference on Business Process Management*. Springer. 2003, S. 370–379 (siehe S. 60, 256).

[RSS06] Gil Regev, Pnina Soffer und Rainer Schmidt. „Taxonomy of Flexibility in Business Processes“. In: *Proceedings of the CAISE*06 Workshop on Business Process Modelling, Development, and Support BPMDS '06, Luxemburg, June 5-9, 2006*. Hrsg. von Gil Regev, Pnina Soffer und Rainer Schmidt. Bd. 236. CEUR Workshop Proceedings. CEUR-WS.org, 2006 (siehe S. 35, 194).

[Rus13] Siegfried Russwurm. „Software: Die Zukunft der Industrie“. In: *Industrie 4.0*. Springer, 2013, S. 21–36 (siehe S. 3).

[RW12] Manfred Reichert und Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012 (siehe S. 8, 35–37, 40, 108, 194).

[Sai+15] Imen Ben Said, Mohamed Amine Chaâbane, Rafik Bouaziz und Eric Andonoff. „Flexibility of collaborative processes using versions and adaptation patterns“. In: *9th IEEE International Conference on Research Challenges in Information Science, RCIS 2015, Athens, Greece, May 13-15, 2015*. IEEE, 2015, S. 400–411 (siehe S. 60).

[Sch+08] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar und Wil M. P. van der Aalst. „Process Flexibility: A Survey of Contemporary Approaches“. In: *Advances in Enterprise Engineering I, 4th International Workshop CIAO! and 4th International Workshop EOMAS, held at CAiSE 2008, Montpellier, France, June 16-17, 2008. Proceedings*. Hrsg. von Jan L. G. Dietz, Antonia Albani und Joseph Barjis. Bd. 10. Lecture Notes in Business Information Processing. Springer, 2008, S. 16–30 (siehe S. 8, 35–40, 108, 109, 117, 127, 129–131, 133, 148, 149, 155, 157, 158, 165, 166, 170, 194, 244, 251, 254, 256, 265, 266).

[Sch+12] Ina Schaefer, Rick Rabiser, Dave Clarke, Lorenzo Bettini, David Benavides, Goetz Botterweck, Animesh Pathak, Salvador Trujillo und Karina Villela. „Software diversity: state of the art and perspectives“. In: *STTT* 14.5 (2012), S. 477–495 (siehe S. 133).

[SCV11] Luis Jesús Ramón Stroppi, Omar Chiotti und Pablo David Villarreal. „Extending BPMN 2.0: Method and Tool Support“. In: *Business Process Model and Notation*. Hrsg. von Remco Dijkman, Jörg Hofstetter und Jana Koehler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 59–73 (siehe S. 25).

[Sei+15] Ronny Seiger, Christine Keller, Florian Niebling und Thomas Schlegel. „Modelling complex and flexible processes for smart cyber-physical environments“. In: *J. Comput. Science* 10 (2015), S. 137–148 (siehe S. 62).

[Sei+16] Ronny Seiger, Steffen Huber, Peter Heisig und Uwe Assmann. „Enabling Self-adaptive Workflows for Cyber-physical Systems“. In: *Enterprise, Business-Process and Information Systems Modeling - 17th International Conference, BPMDS 2016, 21st International Conference, EMMSAD 2016, Held at CAiSE 2016, Ljubljana, Slovenia, June 13-14, 2016, Proceedings*. Hrsg. von Rainer Schmidt, Wided Guédria, Ilia Bider und Sérgio Guerreiro. Bd. 248. Lecture Notes in Business Information Processing. Springer, 2016, S. 3–17 (siehe S. 62).

[Sel07] Bran Selic. „A Systematic Approach to Domain-Specific Language Design Using UML“. In: *Tenth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2007), 7-9 May 2007, Santorini Island, Greece*. IEEE Computer Society, 2007, S. 2–9 (siehe S. 25).

[SHS18] Ronny Seiger, Steffen Huber und Thomas Schlegel. „Toward an execution system for self-healing workflows in cyber-physical systems“. In: *Software and System Modeling* 17.2 (2018), S. 551–572 (siehe S. 62).

[SMM11] Klaus Sperner, Sonja Meyer und Carsten Magerkurth. „Introducing Entity-Based Concepts to Business Process Modeling“. In: *Business Process Model and Notation - Third International Workshop, BPMN 2011, Lucerne, Switzerland, November 21-22, 2011. Proceedings*. Hrsg. von Remco M. Dijkman, Jörg Hofstetter und Jana Koehler. Bd. 95. Lecture Notes in Business Information Processing. Springer, 2011, S. 166–171 (siehe S. 61).

[Sof05] Pnina Soffer. „On the notion of flexibility in business processes“. In: *Proceedings of the CAiSE*. Bd. 5. 2005, S. 35–42 (siehe S. 35).

[Sol13] Christian Soltenborn. „Quality assurance with dynamic meta modeling“. Dissertation. Universität Paderborn, 2013 (siehe S. 16, 21).

[Spa+13] Dieter Spath, Oliver Ganschar, Stefan Gerlach, Moritz Hämmerle, Tobias Krause und Sebastian Schlund. *Produktionsarbeit der Zukunft – Industrie 4.0*. Fraunhofer Verlag Stuttgart, 2013 (siehe S. 1, 3).

[Sta+06] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase und Simon Helsen. *Model-driven software development - technology, engineering, management*. Pitman, 2006 (siehe S. 26).

[Str+11] Luis Jesús Ramón a Stroppi, Luis, Omar Chiotti und Pablo Villarreal. „A BPMN 2.0 Extension to Define the Resource Perspective of Business Process Models“. In: *IV Congreso Iberoamericano en Software Engineering* (Nov. 2011) (siehe S. 25).

[Tor+12] Victoria Torres, Stefan Zugal, Barbara Weber, Manfred Reichert, Clara Ayora und Vicente Pelechano. „A qualitative comparison of approaches supporting business process variability“. In: *International Conference on Business Process Management*. Springer. 2012, S. 560–572 (siehe S. 39, 40).

[Voe+13] Markus Voelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart CL Kats, Eelco Visser und Guido Wachsmuth. *DSL engineering: Designing, implementing and using domain-specific languages*. dslbook. org, 2013 (siehe S. 7, 16, 22).

[Wes12] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition*. Springer, 2012 (siehe S. 10, 31, 32, 34, 80, 198, 207, 254).

[WFH11] Ian H. Witten, Eibe Frank und Mark A. Hall. *Data mining: practical machine learning tools and techniques*. 3rd Edition. Morgan Kaufmann, Elsevier, 2011 (siehe S. 269).

[Whi+09] Jon Whittle, Peter Sawyer, Nelly Bencomo, Betty H. C. Cheng und Jean-Michel Bruel. „RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems“. In: *RE 2009, 17th IEEE International Requirements Engineering Conference, Atlanta, Georgia, USA, August 31 - September 4, 2009*. IEEE Computer Society, 2009, S. 79–88 (siehe S. 202).

[Wie13] Wieland, Matthias. „Methoden zur Modellierung und Ausführung kontextbezogener Workflows in Produktionsumgebungen“. Dissertation. Universität Stuttgart, 2013 (siehe S. 2).

[Wim09] Manuel Wimmer. „A semi-automatic approach for bridging DSMLs with UML“. In: *International Journal of Web Information Systems* 5 @InProceedingsStroppi.etal2011, author = Stroppi, Luis Jesús Ramón and Chiotti, Omar and Villarreal, Pablo David, title = Extending BPMN 2.0: method and tool support, booktitle = International Workshop on Business Process Modeling Notation, year = 2011, pages = 59–73, organization = Springer.3 (2009), S. 372–404 (siehe S. 25).

[WRN14] Hans-Peter Wiendahl, Jürgen Reichardt und Peter Nyhuis. *Handbuch Fabrikplanung: Konzept, Gestaltung und Umsetzung wandlungsfähiger Produktionsstätten*. Carl Hanser Verlag GmbH Co KG, 2014 (siehe S. 3).

[WRR07] Barbara Weber, Stefanie Rinderle und Manfred Reichert. „Change Patterns and Change Support Features in Process-Aware Information Systems“. In: *Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007, Procee-*

dings. Hrsg. von John Krogstie, Andreas L. Opdahl und Guttorm Sindre. Bd. 4495. Lecture Notes in Computer Science. Springer, 2007, S. 574–588 (siehe S. 38, 88).

[WRR08] Barbara Weber, Manfred Reichert und Stefanie Rinderle-Ma. „Change patterns and change support features - Enhancing flexibility in process-aware information systems“. In: *Data & Knowledge Engineering* 66.3 (2008), S. 438–466 (siehe S. 38, 88, 108, 244).

[ZLS11] Sema Zor, F Leymann und D Schumm. „A proposal of BPMN extensions for the manufacturing domain“. In: *Proceedings of 44th CIRP international conference on manufacturing systems*. Citeseer. 2011 (siehe S. 62).

Anhang

A

Operationen des AVM4BPM

In diesem Abschnitt werden Operationen vorgestellt, die als Teil der in Abschnitt 4.3 beschriebenen Teilsprache für die Gestaltung eines *Adaptation View Model 4 BPM* (*AVM4BPM*) verstanden werden können. Sie stellen Beispiele für mögliche Operationen dar, die im Rahmen der Gestaltung von Anpassungsprozessen eingesetzt werden können. Hierzu werden zunächst gemeinsame Eigenschaften dieser Operationen vorgestellt. Anschließend wird jeweils die Signatur, die konkrete Syntax als auch die Anwendung einer Operation im Rahmen eines Beispiels gezeigt. In Abbildung A-1 wird die Darstellungsweise von Operationen sowie von Ein- und Ausgabeparametern anhand eines konzeptionellen Beispiels gezeigt.

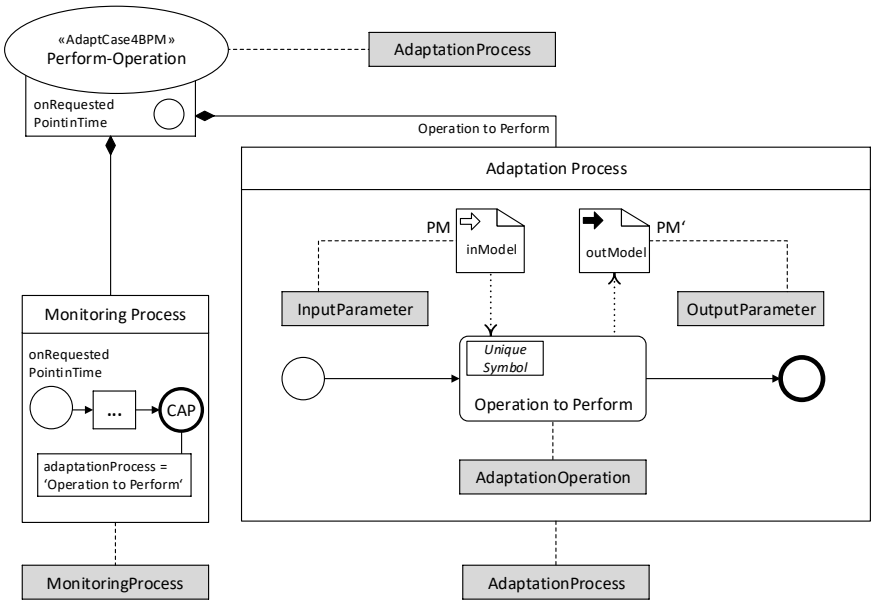


Abbildung A-1: *Darstellung von Operationen sowie von Ein- und Ausgabeparametern*

Konkrete Syntax von Operationen

Die grafische Darstellung von Operationen wird durch eine an die Sprache *BPMN2.0* angelehnte Darstellung von Tasks gewählt. In dieser Arbeit vorgestellte Operation verfügen über ein eindeutiges Symbol. Das Symbol einer Operation wird im linken oberen Bereich der Operation dargestellt.

Konkrete Syntax von Ein- und Ausgabeparametern

Die Parameter einer Operation werden in Anlehnung an Datenelemente der Sprache *BPMN2.0* grafisch dargestellt. Dabei wird jeweils die Darstellung von Input-Datenelementen für Eingabeparameter und die Darstellung von Output-Datenelementen für Ausgabeparameter verwendet. Soll die Angabe von Werten durchgeführt werden, so können diese in textueller Form eingefügt werden. Die in diesem Ansatz enthaltenen Operationen können Parameter verlangen, die in Form von Ein- und Ausgabeparameter vorliegen. Zudem können auch optionale Eingabeparameter vorkommen, die je Operation beschrieben werden.

Textuelle Schreibweise

Für die Operationen ist auch eine textuelle Schreibweise vorgesehen. Ein Beispiel für die textuelle Schreibweise der Operation *ModifyPropertyOfNode* ist nachfolgend gegeben. Parameter werden in der Reihenfolge aufgeführt, in der sie in der Signatur vorkommen (siehe 4-18). Eine Unterscheidung zwischen Operationen für Prozessmodelle und deren Instanzen ist in der textuellen Schreibweise nicht vorgesehen.

ModifyPropertyOfNode (m, Task, 'Name', 'Task A') : m'

In dieser Arbeit stellen die Operationen einen konzeptionellen Gegenstand dar. So sind sie dafür angedacht, prinzipielle Vorgehensweisen für Anpassungen von Prozessen in der frühen Gestaltung von Anpassungsprozessen einsetzen zu können. Soll auf Basis von Beobachtungs- und Anpassungsprozessen die weitere Gestaltung von Prozessen durchgeführt werden, so sind je nach einzusetzender IT-Unterstützung spezifische Verfeinerungen und Implementierungen der vorgestellten Operationen notwendig.

Ein Beispiel bilden hier insbesondere Operationen, die für die Anpassung von Prozessinstanzen eingesetzt werden. Für eine in der Praxis nutzbare Menge von Operationen müssen hierbei plattformspezifische Eigenschaften einer IT-Unterstützung in Form einer Workflow-Engine berücksichtigt werden. Existierende Workflow-Engines haben dabei oftmals verschiedene Repräsentationen von Prozessinstanzen, die die reale Umsetzung von Operationen erschweren können.

Die Auswahl einer derartigen Plattform findet dabei im Rahmen der Phase *Konfiguration* statt. Ebenso wird in dieser Phase die Aufgabe der Implementierung übernommen. Der in dieser Arbeit vorgestellte Ansatz bezieht sich dabei aber auf die frühe Gestaltung im Rahmen der Phase *Design & Analyse*. Die Auswahl einer Plattform sowie die zugehörige Implementierung von Operationen steht für den in dieser Arbeit gesetzten Schwerpunkt nicht im Fokus und wird daher auch nicht angeboten.

Für die Beschreibung von Operationen wurde sich dafür entschieden, lediglich ihre generelle Funktionsweise zu beschreiben. Auf eine Berücksichtigung von spezifischen Eigenschaften von Prozessmodellen und deren Instanzen wird im Rahmen der integrierten Operationen des *AVM4BPM* daher verzichtet. Die in Kapitel 5 vorgestellten Entwurfsmuster für die Gestaltung von flexiblen und anpassbaren Prozessen greifen den Aspekt derartiger Eigenschaften aber in ausgesuchten Teilen wieder auf und zeigen wie sie auch zu einem frühen Zeitpunkt in der Phase *Design & Analyse* berücksichtigt werden können.

Auf die Darstellung aller 24 Operationen des *AVM4BPM* für die Anpassung von Prozessmodellen und deren Instanzen wird nachfolgend verzichtet. Stattdessen wird auf das generelle Funktionsprinzip von Operationen Bezug genommen, das für beide betroffenen Artefakte angewendet werden kann. Daraus ergeben sich insgesamt 12 Operationen für Knotenelemente (siehe Anhang A.1), Kantenelemente (siehe Anhang A.2) und Containerelemente (siehe Anhang A.3). Zusätzlich wird in Anhang A.3 das Funktionsprinzip der Operation *ModifyPositionOfNodesInContainer* vorgestellt, welche für den Flexibilitätsaspekt *Flexibility-by Underspecification* benötigt wird.

Die nachfolgende Beschreibung der Signatur, der konkreten Syntax und Beispiele für die Anwendung von Operationen bezieht sich dabei auf Prozessmodelle. Konzeptionell ändern sich für Operationen an den Prozessinstanzen die Ein- und Ausgabeparameter sowie das spezifische Symbol.

A.1 Operationen zur Anpassung von Knotenelementen

Die Perspektiven *Funktion, Verhalten* und *Informationen* enthalten Knotenelemente. Für die Anpassung derartiger Elemente werden in diesem Abschnitt Operationen vorgestellt. Hierzu gibt Abbildung A-2 einige Beispiele für Elemente aus der Sprache *BPMN2.0* in Bezug zur zugehörigen Perspektive.

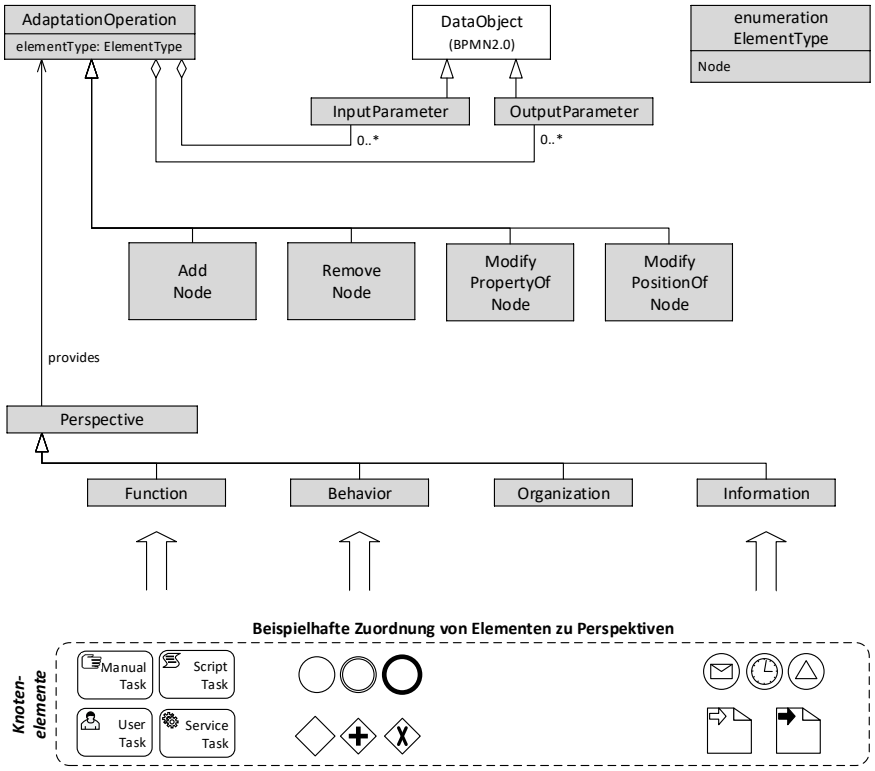


Abbildung A-2: *Übersicht über die Operationen für Knotenelemente*

Mit den dargestellten Operationen ist die Gestaltung von Anpassungen von Prozessmodellen möglich. Im Rahmen der Definition der Operationen werden zunächst die Signatur und anschließend die konkrete Syntax in grafischer Notation exemplarisch dargestellt. Ferner wird für jede Operation ein Beispiel einer möglichen Anwendung der eingeführten Operationen auf Basis von Prozessmodellen gezeigt.

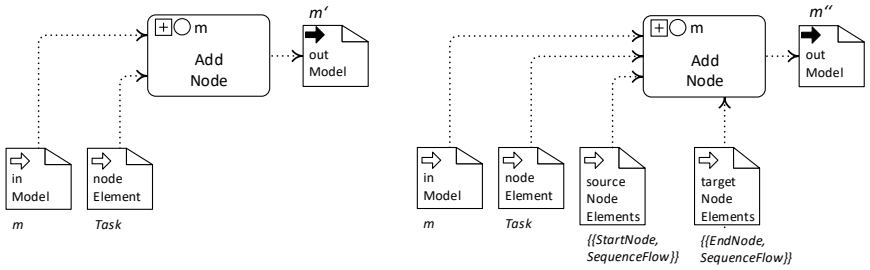
A.1.1 AddNode

Eine Operation vom Typ *AddNode* fügt ein neues Knotenelement in einen Prozess ein. Dabei werden zwei verschiedene Mechanismen unterschieden. Zum einen kann das hinzugefügte Knotenelement lediglich der Menge der vorhandenen Elemente im Prozess hinzufügt werden. Zum anderen

kann es aber auch sinnvoll sein, das Knotenelement mit anderen Elementen in Beziehung zu setzen. So könnte ein hinzugefügtes Knotenelement mit dem bereits vorhandenen Kontrollfluss verbunden werden. Ein solches Knotenelement kann z.B. durch einen der vorhandenen Untertypen von Aktivitäten oder Gateways gegeben sein. Die Signatur der Operation *AddNode* und die konkrete Syntax sind in Abbildung A-3 angegeben.

Abbildung A-3: *Signatur und konkrete Syntax der Operation AddNode*

	Parametername	**Parametertyp**
IN :	*inModel* *nodeElement*	*ProcessModel* *NodeElement*
IN-Optional :	*sourceNodeElements* *targetNodeElements*	$Set\langle NodeElement, AssociationType\rangle$ $Set\langle NodeElement, AssociationType\rangle$
OUT :	*outModel*	*ProcessModel*



Parameter Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist ein in Anlehnung an die ausgeführte Operation geändertes Prozessmodell (*outModel*). Ein weiterer Parameter der Operation ist durch das hinzuzufügende Knotenelement (*nodeElement*) gegeben.

Optionale Parameter Soll das Knotenelement mit bestehenden Elementen verbunden werden, so ist die Angabe weiterer Parameter notwendig. So können durch die Angabe der Mengen *sourceNodeElements* und *targetNodeElements* Quell- bzw. Zielknotenelemente *(NodeElement)* sowie der Typ der einzusetzenden Assoziation *(AssociationType)* angegeben werden. Insgesamt existieren die drei Typen für Assoziationen *SequenceFlow*, *DataAssociation* und *MessageFlow*. Die Menge *sourceNodeElements* enthält alle Knotenelemente, von denen ausgehend eine Assoziation mit dem einzufügenden Knotenelement verbunden werden soll. Ferner werden ausgehend vom dem einzufügenden Knotenelement Assoziationen zu allen Knotenelementen der Menge *targetNodeElements* hinzugefügt. Hierdurch ist es bspw. möglich, einen Task oder ein Gateway in einen bestehenden Kontrollfluss zu integrieren.

Beispiel einer Anwendung der Operation AddNode

Eine Anwendung der in Abbildung A-3 spezifizierten Operationen ist in Abbildung A-4 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. Das BPD zeigt ein Start- und ein Endereignis, die durch eine Assoziation vom Typ *SequenceFlow* verbunden sind.

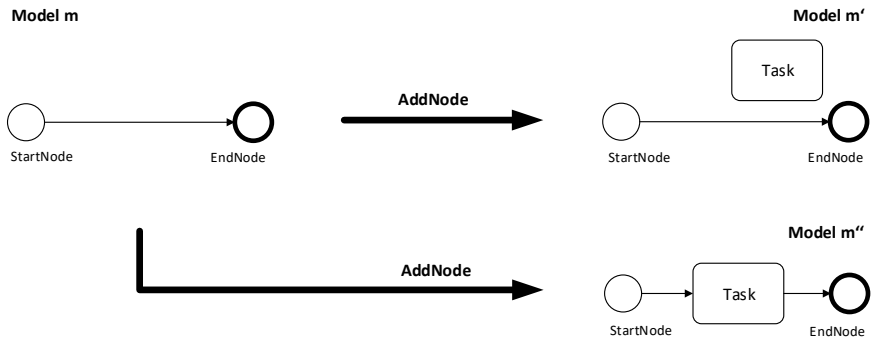


Abbildung A-4: *Beispielhafte Anwendungen der Operation AddNode*

Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m'* ist im oberen Beispiel dargestellt. In dem BPD wurde ein Task hinzugefügt und nicht mit dem existierenden Kontrollfluss verbunden.

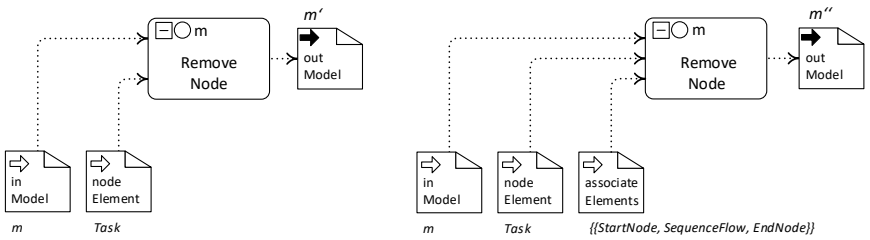
Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m''* ist im unteren Beispiel dargestellt. In dem BPD ist ein Kontrollfluss entstanden, in dem ausgehend vom Startereignis der eingefügte Task folgt und schließlich mit dem Endereignis endet.

A.1.2 RemoveNode

Eine Operation vom Typ *RemoveNode* entfernt ein vorhandenes Knotenelement sowie alle mit ihm in Verbindung stehenden ein- und ausgehenden Kantenelemente aus einem Prozess. Dabei wird ebenso wie bei der Operation *AddNode* zwischen zwei unterschiedlichen Mechanismen unterschieden. Zum einen kann das zu entfernende Knotenelement aus der Menge der vorhandenen Elemente des Prozesses entfernt werden, ohne dass vorhandene Elemente wieder miteinander verbunden werden. Zum anderen kann es aber auch sinnvoll sein, verbleibende Knotenelemente miteinander zu verbinden. So könnten diese z.B. zu einem durchgängigen Kontrollfluss verbunden werden. Die Signatur und konkrete Syntax der Operation *RemoveNode* sind in Abbildung A-5 angegeben.

	Parametername	Parametertyp
IN :	*inModel* *nodeElement*	*ProcessModel* *NodeElement*
IN-Optional :	*associateElements*	*Set*⟨*NodeElement*, *AssociationType*, *NodeElement*⟩
OUT :	*outModel*	*ProcessModel*

Abbildung A-5: Signatur und konkrete Syntax der Operation RemoveNode



Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist ein in Anlehnung an die ausgeführte Operation geändertes Prozessmodell (*outModel*). Ein weiterer Parameter der Operation ist durch das zu entfernende Knotenelement (*nodeElement*) gegeben.

Optionale Parameter

Sollen verbleibende Elemente in einem Prozessmodell durch die Anwendung der Operation verbunden werden, so ist die Angabe weiterer Parameter notwendig. So können durch die Angabe der Menge *associateElements* Tripel angegeben werden. Ein Tripel beschreibt dabei, von welchem Knotenelement *(NodeElement)* ausgehend mit welchem Kantenelement *(AssociationType)* ein weiteres Knotenelement *(NodeElement)* verbunden werden soll. Insgesamt existieren die drei Typen für Assoziationen *SequenceFlow*, *DataAssociation* und *MessageFlow*. Hierdurch ist es z.B. möglich, einen durch die Anwendung der Operation *RemoveNode* unterbrochenen Kontrollfluss wieder zu vervollständigen.

Beispiel einer Anwendung der Operation RemoveNode

Eine Anwendung der beiden in Abbildung A-5 spezifizierten Operationen ist in Abbildung A-6 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. Das BPD enthält ein Startereignis, gefolgt von einem Task und abschließend mit einem Endereignis. Die genannten Knotenelemente sind durch Assoziationen vom Typ *SequenceFlow* verbunden.

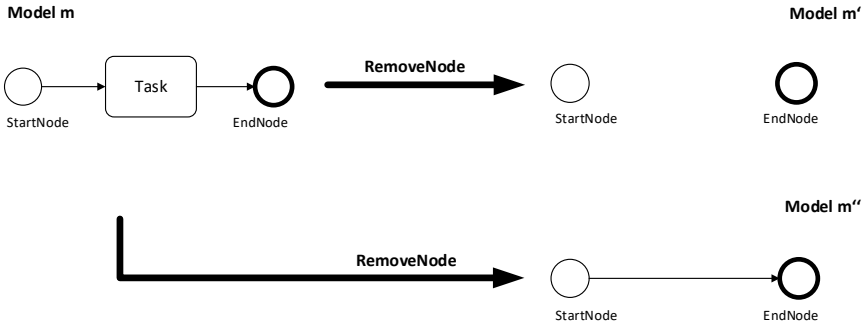


Abbildung A-6: *Beispielhafte Anwendungen der Operation RemoveNode*

Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m'* ist im oberen Beispiel dargestellt. In dem zugehörigen BPD wurden der Task und seine ein- und ausgehenden Assoziationen vom Typ *SequenceFlow* entfernt.

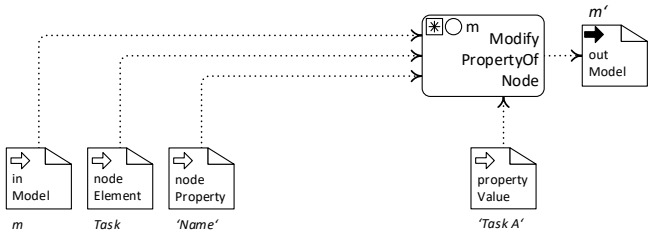
Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m''* ist im unteren Beispiel dargestellt. In dem BPD ist aus dem bestehenden Kontrollfluss der Task entfernt worden. Die verbleibenden Start- und Endereignisse wurden durch eine neu eingefügte Assoziation vom Typ *SequenceFlow* miteinander verbunden.

A.1.3 ModifyPropertyOfNode

Eine Operation vom Typ *ModifyPropertyOfNode* modifiziert den Wert einer Eigenschaft eines Knotenelements in einem Prozess. Die Signatur und konkrete Syntax der Operation *ModifyPropertyOfNode* sind in Abbildung A-7 angegeben.

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	nodeElement	*NodeElement*
	nodeProperty	*Property*
	propertyValue	*Value*
OUT :	*outModel*	*ProcessModel*

Abbildung A-7: *Signatur und konkrete Syntax der Operation ModifyPropertyOfNode*



Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Weitere Parameter der Operation sind durch das betreffende Knotenelement (*nodeElement*), seine zu modifizierende Eigenschaft (*nodeProperty*) und den zugehörigen Wert (*propertyValue*) gegeben. Ein Bezeichner der zu modifizierenden Eigenschaft wird durch ein String-Literal angegeben. Der Typ der zu ändernden Werte ist in der dargestellten Signatur generisch als *Value* angegeben, da es verschiedene Typen wie z.B. String, Integer oder auch komplexe Datentypen geben könnte.

Beispiel einer Anwendung der Operation ModifyPropertyOfNode

Eine Anwendung der in Abbildung A-7 spezifizierten Operation ist in Abbildung A-8 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. In dem BPD sind ein Startereignis, gefolgt von einem Task und abschließend mit einem Endereignis in einer Sequenz durch Assoziationen vom Typ *SequenceFlow* verbunden.

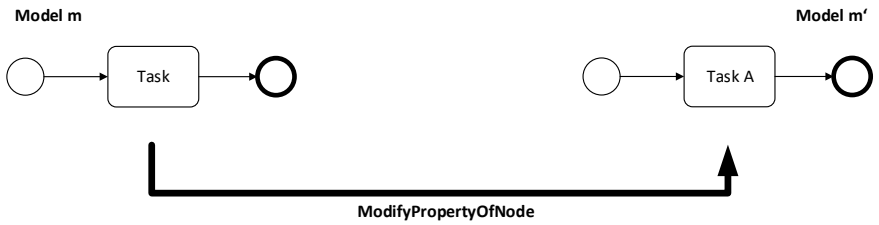


Abbildung A-8: Beispielhafte Anwendung der Operation ModifyPropertyOfNode

Die Anpassung von Prozessmodell *m* hin zu Prozessmodell *m'* ändert die Eigenschaft *Name* des Tasks, sodass der neue Wert dieser Eigenschaft *Task A* ist.

A.1.4 ModifyPositionOfNode

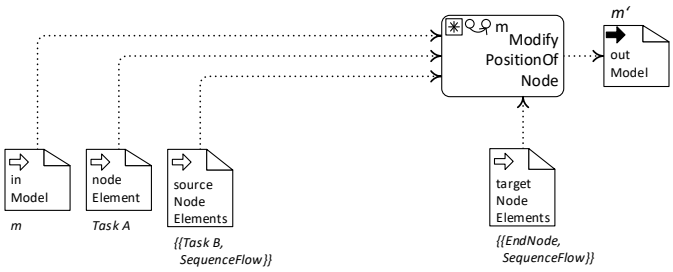
Eine Operation vom Typ *ModifyPositionOfNode* modifiziert die Position eines Knotenelements in einem Prozess innerhalb eines bestehenden Kontroll- oder Datenflusses. Die Signatur und konkrete Syntax der Operation *ModifyPositionOfNode* sind in Abbildung A-9 angegeben.

Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell

	Parametername	Parametertyp
IN :	*inModel*	*ProcessModel*
	nodeElement	*NodeElement*
	sourceNodeElements	$Set\langle NodeElement, AssociationType\rangle$
	targetNodeElements	$Set\langle NodeElement, AssociationType\rangle$
OUT :	*outModel*	*ProcessModel*

Abbildung A-9: *Signatur und konkrete Syntax der Operation ModifyPositionOfNode*



(*outModel*). Ein weiterer Parameter der Operation ist durch das zu verschiebende Knotenelement (*nodeElement*) gegeben. Die neue Position des Knotenelements kann durch die Angabe der Parameter *sourceNodeElements* und *targetNodeElements* angegeben werden. Dabei wird das Knotenelement zwischen den Knotenelementen *(NodeElement)* der zuvor genannten Mengen mit den angegebenen Typen von Assoziationen *(AssociationType)* eingefügt. Zuvor bestehende ein- und ausgehende Kantenelemente werden dabei entfernt.

Beispiel einer Anwendung der Operation ModifyPositionOfNode

Eine Anwendung der in Abbildung A-3 spezifizierten Operation ist in Abbildung A-10 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. In dem BPD ist ein Startereignis, gefolgt von den Tasks *Task A* und *Task B* sowie abschließend mit einem Endereignis durch Assoziationen vom Typ *SequenceFlow* zu einer Sequenz verbunden.

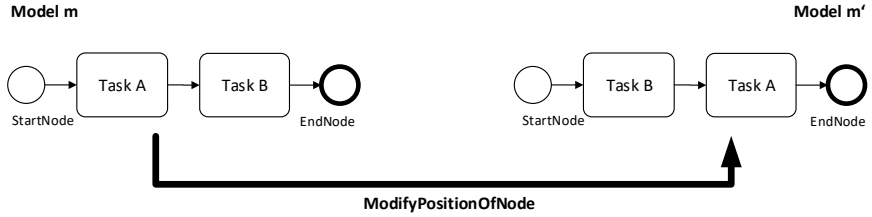


Abbildung A-10: *Beispielhafte Anwendung der Operation ModifyPositionOfNode*

Die Anpassung von Prozessmodell *m* hin zu Prozessmodell *m'* ändert die Position von *Task A*, sodass er nach *Task B* in der Sequenz vorkommt. Hier-

für wurden zusätzlich die in der Ausgabe *m′* dargestellten ein- und ausgehenden Assoziationen von *Task A* vom Typ *SequenceFlow* hinzugefügt.

Die Ausgabe *m′* kann alternativ auch aus einer Kombination der Operationen *RemoveNode* und *AddNode* erreicht werden. Hierbei wird *Task A* zunächst entfernt und anschließend dem Prozessmodell *m′* neu hinzugefügt.

A.2 Operationen zur Anpassung von Kantenelementen

In den Perspektiven *Verhalten* und *Informationen* können Kantenelemente vorkommen. Hierzu zeigt Abbildung A-11 eine Übersicht über die durch die Operationen betrachteten Elemente aus der Domäne *BPM*.

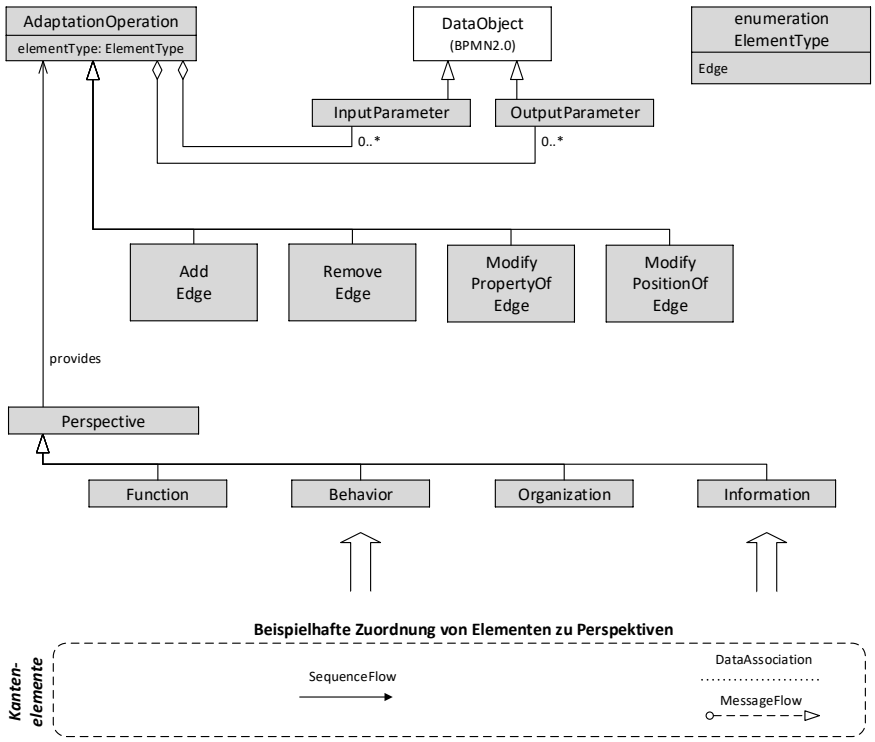


Abbildung A-11: *Übersicht über die Operationen für Kantenelemente*

Mit den dargestellten Operationen ist die Gestaltung von Anpassungen von Prozessmodellen möglich. Im Rahmen der Definition der Operationen werden zunächst die Signatur und anschließend die konkrete Syntax in grafischer Notation exemplarisch dargestellt. Ferner wird für jede Operation ein Beispiel einer möglichen Anwendung der eingeführten Operationen auf Basis von Prozessmodellen gezeigt.

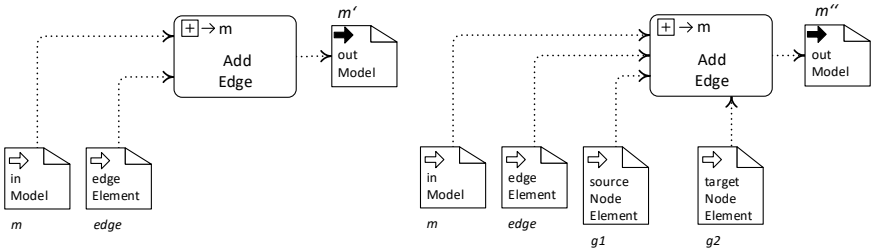
A.2.1 AddEdge

Eine Operation vom Typ *AddEdge* fügt ein neues Kantenelement in einen Prozess ein. Dabei werden zwei verschiedene Mechanismen unterschieden. Zum einen kann das hinzugefügte Kantenelement lediglich der Menge der vorhandenen Elemente im Prozess hinzufügt werden. Zum anderen kann es aber auch sinnvoll sein, das Kantenelement mit anderen Elementen in Beziehung zu setzen. So könnte ein hinzugefügtes Kantenele-

ment des Typs *SequenceFlow* den bereits vorhandenen Kontrollfluss ergänzen, indem es z.B. Tasks oder Gateways miteinander verbindet. Die Signatur und konkrete Syntax der Operation *AddEdge* sind in Abbildung A-12 angegeben.

Abbildung A-12: *Signatur und konkrete Syntax der Operation AddEdge*

	Parametername	**Parametertyp**
IN :	*inModel* *edgeElement*	*ProcessModel* *EdgeElement*
IN-Optional :	*sourceNodeElement* *targetNodeElement*	*NodeElement* *NodeElement*
OUT :	*outModel*	*ProcessModel*



Parameter Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Ein weiterer Parameter der Operation ist durch das hinzuzufügende Kantenelement (*edgeElement*) gegeben.

Optionale Parameter Soll das Kantenelement mit bestehenden Elementen verbunden werden, so ist die Angabe weiterer Parameter notwendig. Durch die Angabe der Parameter *sourceNodeElement* und *targetNodeElement* vom Typ *NodeElement* können jeweils ein Quell- und ein Zielknotenelement benannt werden. Dabei beschreibt der Parameter *sourceNodeElement* das Knotenelement, von dem ausgehend das eingefügte Kantenelement mit verbunden wird. Der Parameter *targetNodeElement* gibt dabei das Knotenelement an, welches als Ziel des Kantenelements gesetzt werden soll. Die Operation *AddEdge* kann eingesetzt werden, um Kantenelemente der Typen *SequenceFlow*, *DataAssociation* und *MessageFlow* einzufügen.

Beispiel einer Anwendung der Operation AddEdge Eine Anwendung der in Abbildung A-12 spezifizierten Operation ist in Abbildung A-13 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang ein Auszug aus einem Prozessmodell *m* dargestellt. In dem BPD sind die beiden Gateways *g1* und *g2* sowie ein Task in einer Sequenz dargestellt.

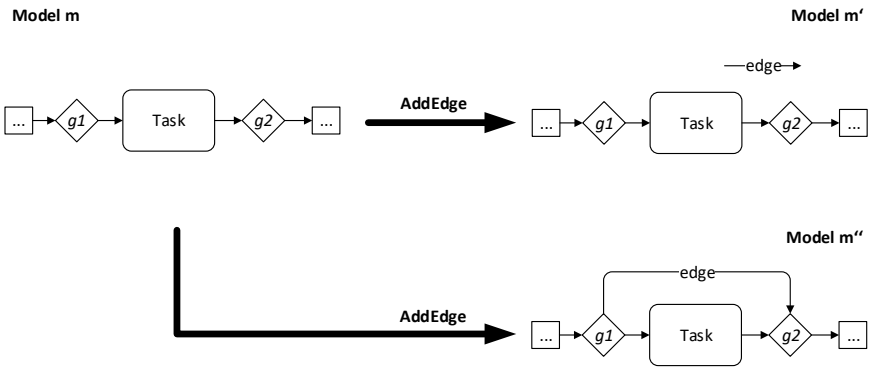


Abbildung A-13: *Beispielhafte Anwendungen der Operation AddEdge*

Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m'* ist im oberen Beispiel dargestellt. In dem BPD wurde eine Assoziation vom Typ *SequenceFlow* mit dem Namen *'edge'* hinzugefügt und nicht in den existierenden Kontrollfluss integriert.

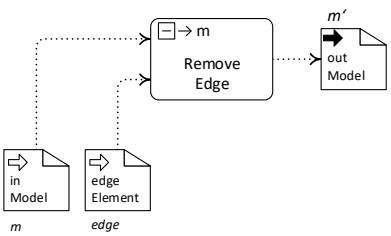
Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m''* ist im unteren Beispiel dargestellt. In dem BPD ist ein Kontrollfluss entstanden, in dem ausgehend vom Gateway *g1* ein alternativer Kontrollfluss entstanden ist, mit dem es möglich ist, die Ausführung des Tasks zu überspringen, indem direkt zum Gateway *g2* gewechselt werden kann.

A.2.2 RemoveEdge

Eine Operation vom Typ *RemoveEdge* entfernt ein vorhandenes Kantenelement aus einem Prozess. So können Assoziationen der Typen *SequenceFlow*, *DataAssociation* und *MessageFlow* entfernt werden. Die Signatur und konkrete Syntax der Operation *RemoveNode* sind in Abbildung A-14 angegeben.

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	edgeElement	*EdgeElement*
OUT :	*outModel*	*ProcessModel*

Abbildung A-14: *Signatur und konkrete Syntax der Operation RemoveEdge*



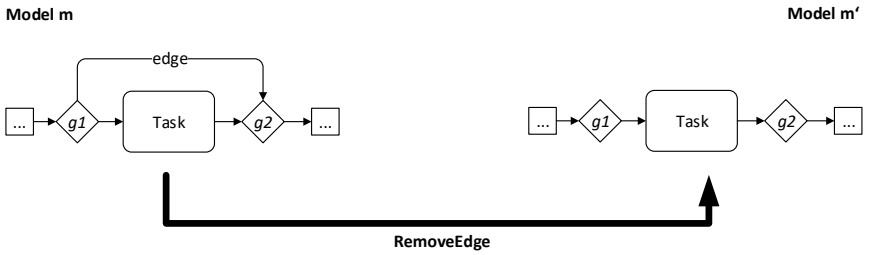
Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Ein weiterer Parameter der Operation ist durch das zu entfernende Kantenelement (*edgeElement*) gegeben.

Beispiel einer Anwendung der Operation RemoveEdge

Eine Anwendung der in Abbildung A-14 spezifizierten Operation ist in Abbildung A-15 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang der Auszug eines Prozessmodells *m* dargestellt. In dem BPD sind zwei Gateways sowie ein Task als eine Sequenz dargestellt. Ausgehend vom Gateway *g1* ist es möglich, die Ausführung des Tasks mit einem alternativen Pfad zu überspringen.

Abbildung A-15: *Beispielhafte Anwendung der Operation RemoveEdge*



Die Anpassung im Auszug des Prozessmodells *m* hin zu Prozessmodell *m'* entfernt den alternativen Pfad. In dem BPD wurde somit die Assoziation vom Typ *SequenceFlow* mit der Bezeichnung *edge* entfernt.

A.2.3 ModifyPropertyOfEdge

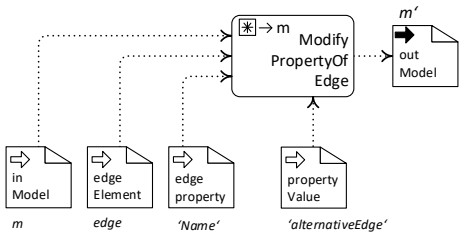
Eine Operation vom Typ *ModifyPropertyOfEdge* modifiziert den Wert einer Eigenschaft eines Kantenelements aus einem Prozess. Die Signatur und konkrete Syntax der Operation *ModifyPropertyOfEdge* sind in Abbildung A-16 angegeben.

Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Weitere Parameter der Operation sind durch das betreffende Kantenelement (*edgeElement*), seine zu modifizierende Eigenschaft (*edgeProperty*) und den zugehörigen Wert (*propertyValue*) gegeben. Ein Bezeichner der zu modifizierenden Eigenschaft wird durch ein String-Literal angegeben. Der Typ der zu ändernden Werte ist in der dargestellten Signatur generisch als *Value* angegeben, da es verschiedene Typen wie z.B. String, Integer oder auch komplexe Datentypen geben könnte.

	Parametername	Parametertyp
IN :	*inModel*	*ProcessModel*
	edgeElement	*EdgeElement*
	edgeProperty	*Property*
	propertyValue	*Value*
OUT :	*outModel*	*ProcessModel*

Abbildung A-16: *Signatur und konkrete Syntax der Operation ModifyPropertyOfEdge*



Eine Anwendung der in Abbildung A-16 spezifizierten Operation ist in Abbildung A-17 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. In dem BPD sind zwei Gateways und ein Task als Sequenz dargestellt. Ferner existiert ausgehend vom Gateway *g1* und endend mit Gateway *g2* ein alternativer Pfad, dessen Assoziation vom Typ *SequenceFlow* ist und den Namen *'edge'* trägt.

Beispiel einer Anwendung der Operation ModifyPropertyOfEdge

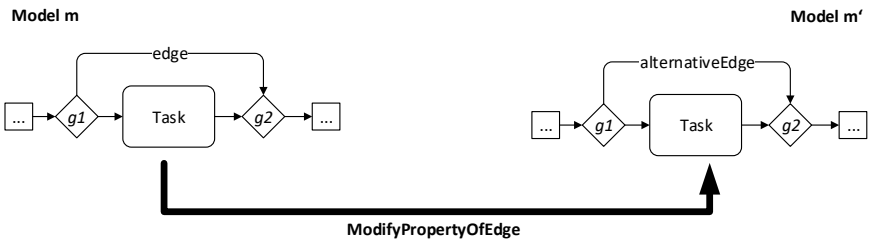


Abbildung A-17: *Beispielhafte Anwendung der Operation Modify-PropertyOfEdge*

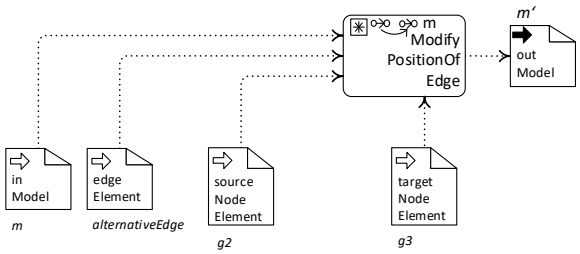
Die Anwendung der Operation ändert die Eigenschaft *Name* des Kantenelements mit der Bezeichnung *edge*, sodass der neue Wert dieser Eigenschaft *'alternativeEdge'* ist.

A.2.4 ModifyPositionOfEdge

Eine Operation vom Typ *ModifyPositionOfEdge* modifiziert die Position eines Kantenelements aus einem Prozess. Eine Position eines Kantenelements wird dabei aus den aus- und dem eingehenden Knotenelement gebildet. Die Signatur und konkrete Syntax der Operation *ModifyPositionOf-Edge* sind in Abbildung A-18 angegeben.

Abbildung A-18: *Signatur und konkrete Syntax der Operation ModifyPositionOfEdge*

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	edgeElement	*EdgeElement*
	sourceNodeElement	*NodeElement*
	targetNodeElement	*NodeElement*
OUT :	*outModel*	*ProcessModel*



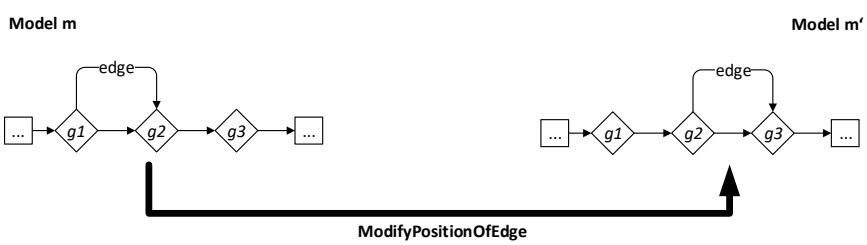
Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Ein weiterer Parameter der Operation ist durch das zu verschiebende Kantenelement (*edgeElement*) gegeben. Die neue Position des Kantenelements kann durch die Angabe der Parameter *sourceNodeElement* und *targetNodeElement* angegeben werden. Dabei wird das Kantenelement ausgehend vom Knotenelement *sourceNodeElement* und endend mit dem Knotenelement *targetNodeElement* verbunden.

Beispiel einer Anwendung der Operation ModifyPositionOfEdge

Eine Anwendung der in Abbildung A-18 spezifizierten Operation ist in Abbildung A-19 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. In dem BPD ist eine Sequenz von drei Gateways (*g1, g2, g3*) dargestellt. Die Gateways sind mit Assoziationen vom Typ *SequenceFlow* verbunden. Ausgehend vom Gateway *g1* existiert ein alternativer Pfad mit der Bezeichnung *edge*, der zum zweiten Gateway *g2* führt.

Abbildung A-19: *Beispielhafte Anwendung der Operation ModifyPositionOfEdge*



Die Anwendung der Operation ändert die Position der Assoziation mit der Bezeichnung *'edge'*, sodass der alternative Pfad ausgehend vom zweiten Gateway *g2* zum dritten Gateway *g3* verläuft. Die Ausgabe *m'* kann alternativ auch aus einer Kombination der Operationen *RemoveEdge* und *AddEdge* erreicht werden. Hierbei wird die Assoziation zunächst entfernt und anschließend dem Prozessmodell *m''* entsprechend neu hinzugefügt.

A.3 Operationen zur Anpassung von Containerelementen

Die Perspektiven *Funktion* und *Organisation* enthalten verschiedene Beispiele für Containerelemente. Bei Containerelementen handelt es ich um Elemente, die weitere Elemente enthalten können. Beispiele hierfür sind durch die drei Typen *SubProcess* und *Pool* bzw. *Lane* gegeben. In Abbildung A-20 ist eine Übersicht über diese Elemente in Anordnung zu ihrer jeweiligen Perspektive gezeigt.

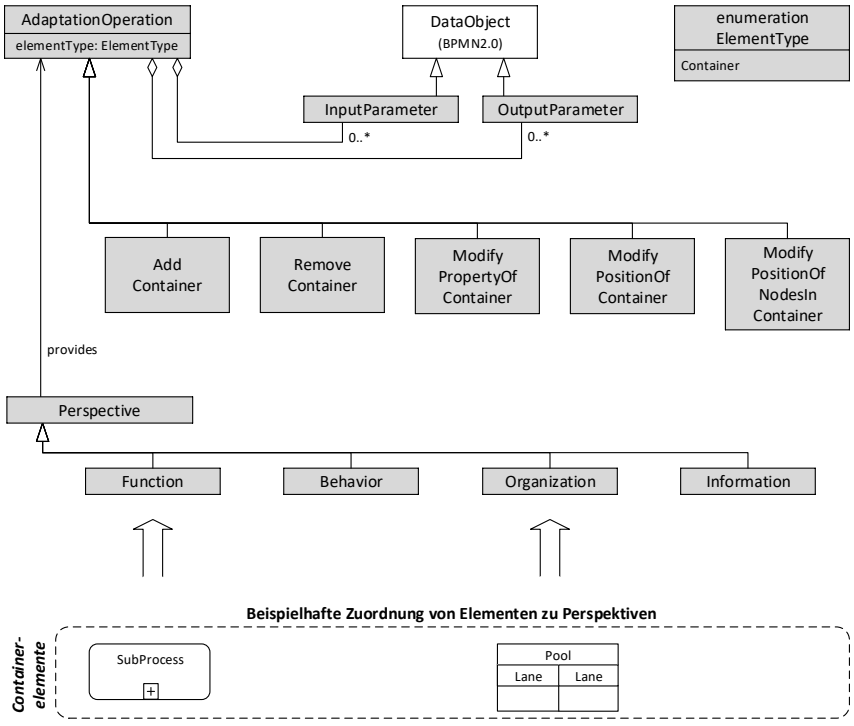


Abbildung A-20: *Übersicht über die Operationen für Containerelemente*

Für die gezeigten Elemente sind im Gegensatz zu den zuvor beschriebenen Operationen unterschiedliche Verfahrensweisen notwendig. So lässt sich bspw. ein Kontrollfluss über mehrere *Lanes* innerhalb eines *Pools* gestalten. Ein Kontrollfluss darf jedoch nicht über einen Subprozess verlaufen. Im Rahmen von Operationen für die betroffenen Perspektiven *Organisation* und *Funktion* sollten Elemente unterschiedlich behandelt werden. Die beschriebenen Beispiele fokussieren die Funktionsprinzipien für Elemente der Perspektive *Organisation*. Eine Anwendung auf Subprozesse kann zu Fehlern führen, die durch Operationen für Knoten- und Kantenelemente korrigiert werden können. Aufgrund des beispielhaften Charakters der

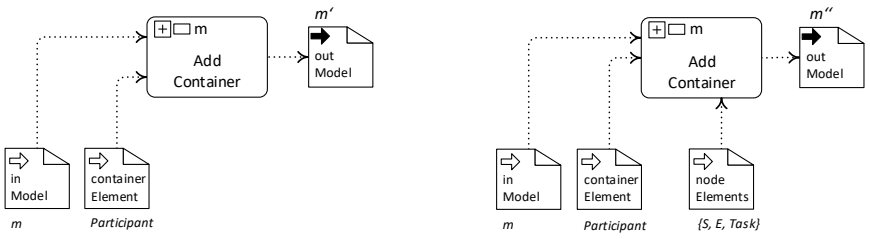
Operationen wurde sich im Rahmen der Beschreibung dafür entschieden, spezifische Korrekturen für die Operationen nicht mitanzugeben.

A.3.1 AddContainer

Eine Operation vom Typ *AddContainer* fügt ein neues Containerelement in einen Prozess ein. Dabei werden zwei verschiedene Mechanismen unterschieden. Zum einen kann das hinzuzufügende Containerelement lediglich der Menge der vorhandenen Elemente in dem Prozess hinzugefügt werden. Zum anderen kann es aber auch sinnvoll sein, existierende Knotenelemente in dem Prozess dem Containerelement zuzuordnen. So könnte ein hinzugefügtes Containerelement bereits existierende Elemente nach Anwendung der Operation enthalten. Die Signatur und konkrete Syntax der Operation *AddContainer* sind in Abbildung A-21 angegeben.

Abbildung A-21: *Signatur und konkrete Syntax der Operation AddContainer*

	Parametername	**Parametertyp**
IN :	*inModel* *containerElement*	*ProcessModel* *ContainerElement*
IN-Optional :	*nodeElements*	*Set⟨NodeElement⟩*
OUT :	*outModel*	*ProcessModel*



Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Ein weiterer Parameter der Operation ist durch das hinzuzufügende Containerelement (*containerElement*) gegeben.

Optionale Parameter

Soll das Containerelement bereits bestehende Elemente enthalten, so ist die Angabe weiterer Parameter notwendig. Es können durch die Angabe der Menge *nodeElements* vom Typ *NodeElement* die Knotenelemente referenziert werden, die im Containerelement enthalten sein sollen.

Beispiel einer Anwendung der Operation AddContainer

Eine Anwendung der beiden in Abbildung A-21 spezifizierten Operationen ist in Abbildung A-22 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. In dem

BPD sind ein Start- und ein Endereignis mit einem dazwischenliegenden Task dargestellt, welche durch Assoziationen vom Typ *SequenceFlow* verbunden sind.

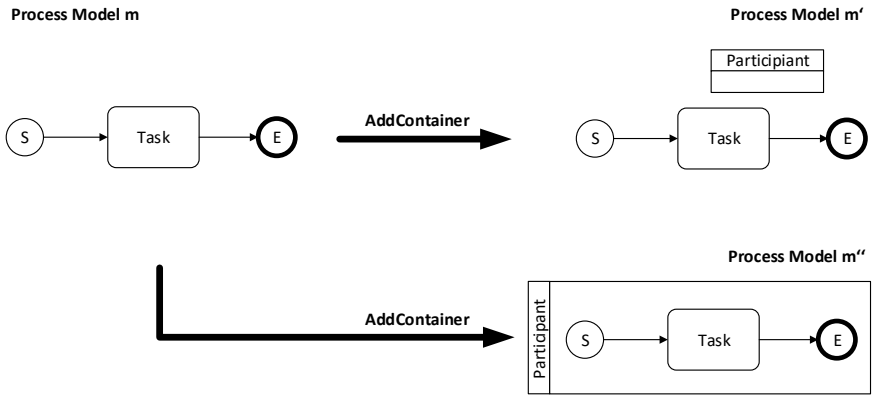


Abbildung A-22: *Beispielhafte Anwendungen der Operation AddContainer*

Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m'* ist im oberen Beispiel dargestellt. In dem BPD wurde die *Lane* mit der Bezeichnung *Participant* hinzugefügt. Dabei enthält diese *Lane* keine der zuvor existierenden Elemente.

Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m''* ist im unteren Beispiel dargestellt. Die in der Anpassungsoperation spezifizierten Elemente mit den Bezeichnungen *S*, *Task* und *E* wurden in der hinzugefügten *Lane* eingebettet.

A.3.2 RemoveContainer

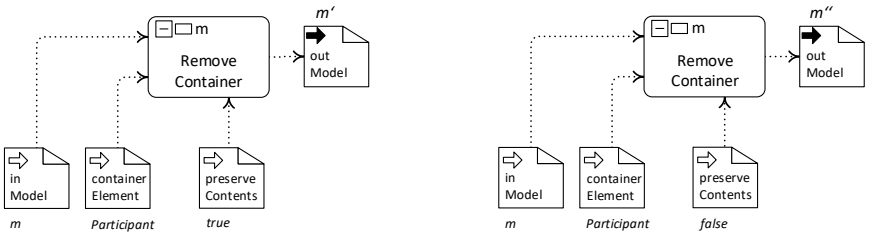
Eine Operation vom Typ *RemoveContainer* entfernt ein vorhandenes Containerelement aus einem Prozess. Es werden zwei Mechanismen zur Entfernung angeboten. So können die in dem Containerelement enthaltenen Elemente entweder ebenfalls entfernt oder erhalten bleiben. Das Erhalten von Elementen kann sinnvoll sein, wenn das in einem Containerelement enthaltene Verhalten bestehen bleiben soll. Die Signatur und konkrete Syntax der Operation *RemoveContainer* sind in Abbildung A-23 angegeben.

Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Ein weiterer Parameter der Operation ist durch das zu entfernende Containerelement (*containerElement*) gegeben. Sind in einem Containerelement enthaltene Elemente vorhanden, so kann durch den Parameter *preserveElements* vom Typ *Boolean* angegeben werden, ob diese Elemente

Abbildung A-23: *Signatur und konkrete Syntax der Operation RemoveContainer*

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	containerElement	*ContainerElement*
	preserveElements	*Boolean*
OUT :	*outModel*	*ProcessModel*

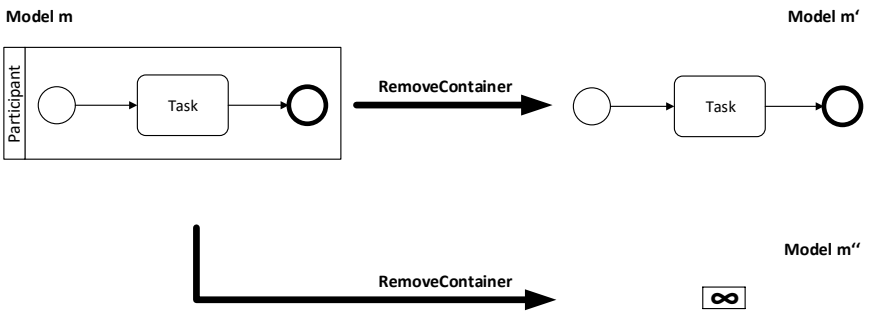


erhalten werden sollen. Dabei steht der Wert *true* für das Erhalten und der Wert *false* für ein Verwerfen bestehender Elemente im Containerelement.

Beispiel einer Anwendung der Operation RemoveContainer

Eine Anwendung der in Abbildung A-23 spezifizierten Operation ist in Abbildung A-24 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. In dem BPD sind ein Startereignis, gefolgt von einem Task und abschließend mit einem Endereignis durch Assoziationen vom Typ *SequenceFlow* verbunden. Die genannten Elemente sind eingebettet in einer *Lane* mit der Bezeichnung *Participant*.

Abbildung A-24: *Beispielhafte Anwendungen der Operation RemoveContainer*



Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m'* ist im oberen Beispiel dargestellt. In dem BPD wurde die *Lane* mit der Bezeichnung *Participant* entfernt. Die zuvor in der *Lane* enthaltene Sequenz bestehend aus einem Startereignis, einem Task sowie einem Endereignis blieb erhalten und ist in diesem Prozessmodell keiner organisatorischen Rolle mehr zugeordnet.

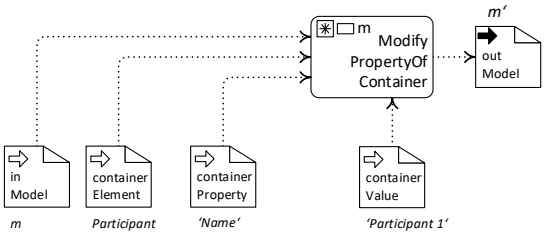
Die Anpassung des Prozessmodells *m* hin zu Prozessmodell *m"* ist im unteren Beispiel dargestellt. Das Resultat ist ein leeres Prozessmodell – hier dargestellt als schwarz-umrandetes Unendlichkeitszeichen.

A.3.3 ModifyPropertyOfContainer

Eine Operation vom Typ *ModifyPropertyOfContainer* modifiziert den Wert einer Eigenschaft eines Containerelements aus einem Prozess. Die Signatur und konkrete Syntax der Operation *ModifyPropertyOfContainer* sind in Abbildung A-25 angegeben.

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	containerElement	*ContainerElement*
	containerProperty	*Property*
	propertyValue	*Value*
OUT :	*outModel*	*ProcessModel*

Abbildung A-25: *Signatur und konkrete Syntax der Operation ModifyPropertyOf-Container*



Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Weitere Parameter der Operation sind durch das betreffende Containerelement (*containerElement*), seine zu modifizierende Eigenschaft (*containerProperty*) und den zugehörigen Wert (*propertyValue*) gegeben. Ein Bezeichner der zu modifizierenden Eigenschaft wird durch ein String-Literal angegeben. Der Typ der zu ändernden Werte ist in der dargestellten Signatur generisch als *Value* angegeben, da es verschiedene Typen wie z.B. String, Integer oder auch komplexe Datentypen geben kann.

Beispiel einer Anwendung der Operation Modify-PropertyOfContainer

Eine Anwendung der in Abbildung A-25 spezifizierten Operation ist in Abbildung A-26 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. In dem BPD ist eine Sequenz bestehend aus einem Startereignis, gefolgt

von einem Task und abschließend mit einem Endereignis gezeigt. Die genannten Elemente sind durch Assoziationen vom Typ *SequenceFlow* verbunden.

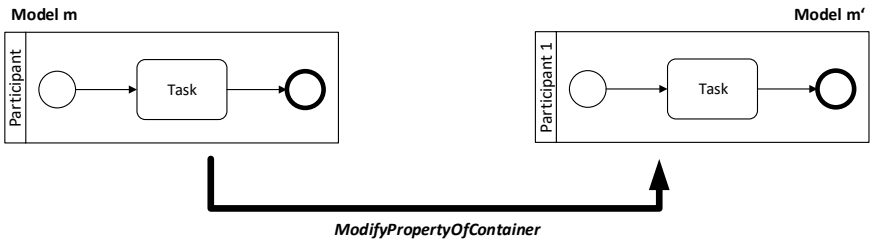


Abbildung A-26: *Beispielhafte Anwendung der Operation ModifyPropertyOfContainer*

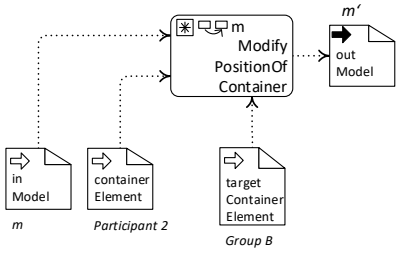
Die Anwendung der Operation ändert die Eigenschaft *Name* der *Lane*, sodass der neue Wert *Participant 1* ist.

A.3.4 ModifyPositionOfContainer

Eine Operation vom Typ *ModifyPositionOfContainer* modifiziert die Position eines Containerelements in einem Prozess. Im Fall von Containerelementen der Perspektive *Organisation* handelt es sich bei der Position des betroffenen Elements um die jeweilige Einbettung in andere Containerelemente der gleichen Perspektive oder auf oberste Ebene im Prozess. Die Signatur und konkrete Syntax der Operation *ModifyPositionOfContainer* sind in Abbildung A-27 angegeben.

Abbildung A-27: *Signatur und konkrete Syntax der Operation ModifyPositionOfContainer*

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	containerElement	*ContainerElement*
	targetContainerElement	*ContainerElement*
OUT :	*outModel*	*ProcessModel*



Parameter

Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Ein weiterer Parameter der Operation ist durch das zu verschiebende Containerelement (*containerElement*) gegeben. Die neue Position des Containerelements kann durch die Angabe der Parameter *targetContainerElement* angegeben werden. Dabei wird das Containerelement in das durch diesen Parameter angegebene Containerelement verschoben. Wird mit dem Parameter kein existierendes Containerelement des Prozessmodells angegeben, so wird das zu verschiebende Containerelement auf oberster Hierarchieebene verschoben.

Beispiel einer Anwendung der Operation ModifyPositionOfContainer

Eine Anwendung der in Abbildung A-27 spezifizierten Operation ist in Abbildung A-28 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. In dem BPD sind die beiden Pools *Group A* und *Group B* gezeigt. In dem Pool mit der Bezeichnung *Group A* sind die beiden Lanes *Participant 1* und *Participant 2* enthalten.

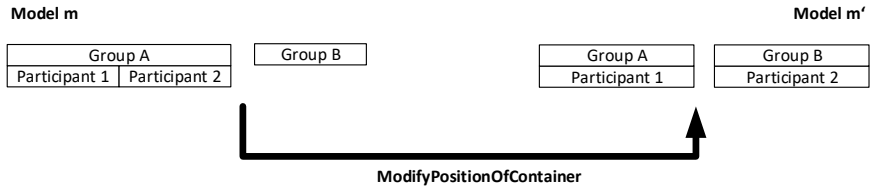


Abbildung A-28: *Beispielhafte Anwendung der Operation ModifyPositionOfContainer*

In dem BPD ist die Lane mit der Bezeichnung *Participant 2* nicht mehr Teil vom Pool *Group A*, sondern Teil von Pool *Group B*. Die Ausgabe m' kann alternativ auch aus einer Kombination der Operationen *RemoveContainer* und *AddContainer* erreicht werden. Hierbei wird die Lane mit der Bezeichnung *Participant 1* zunächst entfernt und anschließend dem Pool *Group B* neu hinzugefügt.

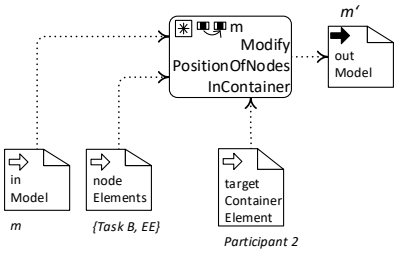
A.3.5 ModifyPositionOfNodesInContainer

Eine Operation vom Typ *ModifyPositionOfNodesInContainer* ändert die Position von Knotenelementen, sodass diese aus einem Containerelement in ein anderes Containerelement in einem Prozess verschoben werden können. Dabei handelt es sich bei dieser Operation um eine Abweichung ge-

genüber den zuvor angewandten Mustern von Operationen zur Anpassung von Prozessen, da es sich um eine Operation in Bezug zu Knotenelementen handelt, sie aber im Abschnitt für Containerelemente beschrieben wird. Dies lässt sich dadurch begründen, dass für ein besseres Verständnis für die Operation *ModifyPositionOfNodesInContainer* zunächst die zuvor beschriebenen Operationen für Containerelemente eingeführt wurden. Die Signatur und konkrete Syntax der Operation *ModifyPositionOfNodesInContainer* sind in Abbildung A-29 angegeben.

Abbildung A-29: *Signatur und konkrete Syntax der Operation ModifyPositionOfNodesInContainer*

	Parametername	**Parametertyp**
IN :	*inModel*	*ProcessModel*
	nodeElements	*Set⟨NodeElement⟩*
	targetContainerElement	*ContainerElement*
IN-Optional :	*copyElements*	*Boolean*
OUT :	*outModel*	*ProcessModel*



Parameter Die Operation erwartet als Eingabe ein Prozessmodell (*inModel*), auf das die Anpassung ausgeführt werden soll. Die Ausgabe der Operation ist in Anlehnung an die ausgeführte Operation ein geändertes Prozessmodell (*outModel*). Durch den Parameter *nodeElements* kann eine Menge vom Typ *NodeElement* angegeben werden, die die zu verschiebenden Knotenelemente enthält. Wird eine leere Menge angegeben, so werden alle Elemente verschoben. Die neue Position der Knotenelemente kann durch die Angabe des Parameters *targetContainerElement* vom Typ *ContainerElement* angegeben werden. Dabei werden die durch den Parameter *nodeElements* angegebenen Knotenelemente in das durch den Parameter *targetContainerElement* angegebene Containerelement verschoben.

Optionale Parameter Sollen Elemente in einem Containerelement erhalten bleiben und stattdessen eine Kopie eingefügt werden, so ist die Angabe eines weiteren Parameters notwendig. Dieser Parameter ist durch *copyElements* vom Typ *Boolean* gegeben. Das Kopieren von Element kann durch Angabe dieses Parameters mit dem Wert *true* durchgeführt werden.

Beispiel einer Anwendung der Operation ModifyPositionOfNodesInContainer

Eine Anwendung der in Abbildung A-29 spezifizierten Operation ist in Abbildung A-30 dargestellt. Dabei wird eine Darstellung in der Sprache *BPMN2.0* durch einen Auszug aus einem BPD gezeigt. Im linken Bereich der Abbildung wird hierzu als Ausgang das Prozessmodell *m* dargestellt. In dem BPD ist eine Sequenz beginnend mit einem Startereignis, gefolgt von *Task A* sowie *Task B* und endend mit einem Endereignis dargestellt. Die genannten Elemente sind durch Assoziationen vom Typ *SequenceFlow* miteinander verbunden. Die beschriebene Sequenz ist eingebettet in einer Lane mit der Bezeichnung *Participant 1*. Sowohl die *Lane* mit der Bezeichnung *Participant 1* als auch eine zweite Lane mit der Bezeichnung *Participant 2* sind eingebettet in einem *Pool* mit der Bezeichnung *Group A*.

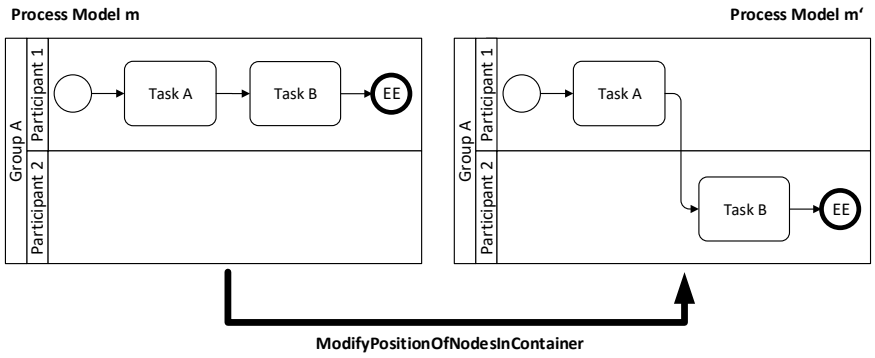


Abbildung A-30: *Beispielhafte Anwendung der Operation ModifyPositionOfNodesInContainer*

Die Anwendung der Operation ändert die Position von *Task B* und des Endereignisses *EE*. Sie werden in der Lane mit der Bezeichnung *Participant 2* eingebettet. Der bestehende Kontrollfluss verläuft anschließend über beiden dargestellten Lanes. Die Ausgabe *m'* kann alternativ auch aus einer Kombination der Operationen *RemoveNode* und *AddNode* erreicht werden. Hierbei wird *Task B* zunächst entfernt und anschließend dem Prozessmodell *m''* neu hinzugefügt.