ulm university universität
uulm

Institut für Datenbanken und Informationssysteme
(Leiter: Prof. Dr. P. Dadam)

Management von Prozessvarianten

Dissertation
zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften und Informatik
der Universität Ulm

vorgelegt von
Alena Janice Hallerbach (geb. Büttler)
aus Kirchheimbolanden

2009

Amtierender Dekan:	Prof. Dr.-Ing. M. Weber
Gutachter:	Prof. Dr. M. Reichert Prof. Dr. H. Partsch
Externer Gutachter:	Priv.-Doz. Dr. B. Weber
Tag der Promotion:	3. Februar 2010

Kurzfassung

Die fachliche Modellierung von Geschäftsprozessen und deren Ausführung mittels Workflow-Management-Systemen bilden zentrale Aufgaben bei der Realisierung prozessorientierter Informationssysteme. In der Praxis hat sich gezeigt, dass ein Prozess oftmals in zahlreichen Varianten auftritt, die Anpassungen an bestimmte Rahmenbedingungen (z.B. Domäne, landesspezifische Gesetze) darstellen. Die adäquate Modellierung und Ausführung solcher Prozessvarianten stellt eine große Herausforderung dar, der heutige Geschäftsprozessmodellierungswerkzeuge und Workflow-Management-Systeme nicht gerecht werden. Existierende Werkzeuge ermöglichen lediglich das Ausmodellieren aller Prozessvarianten in separaten Prozessmodellen, was in einem hohen Anpassungs- und Wartungsaufwand resultiert: Werden Prozessanpassungen erforderlich, sind meist mehrere Varianten betroffen und deshalb mehrere Prozessmodelle zu adaptieren. Dies wiederum führt schnell zu Inkonsistenzen und mit der Zeit zu degenerierten Prozessmodellen. Um dies zu vermeiden, existiert in der Praxis ein zweiter verbreiteter Ansatz für das Management von Prozessvarianten: Alle Varianten werden in einem „großen“ Prozessmodell abgebildet. Sie sind somit innerhalb der Prozesslogik „versteckt“. Dieser Ansatz führt zwar zu weniger Redundanz, allerdings entstehen dadurch sehr komplexe und unübersichtliche Modelle, so dass bei großer Zahl von Varianten eine effektive Handhabbarkeit nicht mehr möglich ist.

Typischerweise ist eine bestimmte Prozessvariante nur für gewisse Rahmenbedingungen relevant, d.h. sie genügt einem spezifischen Anwendungsfall. Die Informationen über den spezifischen Anwendungskontext einer Variante können bislang nicht in die entsprechenden Prozessmodelle integriert werden. Darüber hinaus ist eine Auswertung dieser Informationen, z.B. zur automatischen Konfiguration von Prozessvarianten, nicht möglich, geschweige denn eine dynamische Reaktion auf Änderungen des Anwendungskontextes zur Laufzeit einer Prozessvariante.

Die vorliegende Arbeit stellt mit Provop (Prozessvarianten mittels Optionen) einen Lösungsansatz zur Handhabung von Prozessvarianten dar, der es erlaubt, ausgehend von einem sog. Basisprozessmodell, alle Varianten eines Prozesstyps abzuleiten. Dazu transformieren verschieden Änderungsoperationen das Basisprozessmodell sukzessiv zu einem Variantenmodell. Dabei berücksichtigt Provop den spezifischen Anwendungskontext einer Prozessvariante, um zu bestimmen, welche der modellierten Änderungsoperationen auf das Basisprozessmodell anzuwenden sind. Mit Hilfe von Auswahlbeschränkungen kann gewährleistet werden, dass nur solche Änderungsoperationen gemeinsam angewendet werden, die auch strukturell und semantisch kompatibel sind.

Dem Problem nicht-kommutativer Änderungsoperationen bei der Konfiguration einer Prozessvariante begegnet Provop durch die Vorgabe einer eindeutigen Anwendungsreihenfolge. Dazu werden die Modellierungsreihenfolge sowie explizite Reihenfolgebeziehungen zwischen den Änderungsoperationen betrachtet. Darüber hinaus wird gewährleistet, dass die Menge aller konfigurierbaren Prozessvarianten den Korrektheitskriterien des zugrundeliegenden Prozess-Metamodells genügt.

Für eine flexible Ausführung von Prozessinstanzen ermöglicht es Provop, zur Laufzeit auf Änderungen des Anwendungskontextes zu reagieren. Dabei stellen wir sicher, dass nur solche Variantenmodelle zur Ausführung kommen bzw. nur dann dynamische Wechsel zwischen Variantenmodellen zur Laufzeit zulässig sind, wenn dadurch die korrekte und stabile Ausführbarkeit der jeweiligen Prozessinstanz nicht beeinträchtigt wird. Das heißt, es können zur Laufzeit keine Ausführungsfehler aufgrund inkorrekter Prozessmodelle auftreten.

Provop unterstützt das Refactoring von Basisprozessmodell und Änderungsoperationen, ohne die Ausführungssemantik der Prozessvarianten zu beeinflussen. Dabei werden sowohl der spezifische Anwendungskontext als auch definierte Auswahlbeschränkungen zwischen Änderungsoperationen berücksichtigt. Mit Hilfe eines Refactorings können Pflege- und Wartungsaufwände für das Management von Prozessvarianten reduziert werden.

Die Ansätze von Provop werden in einem Prototypen realisiert und anhand mehrerer Fallstudien praktisch validiert.

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als Doktorandin in der Abteilung Daten- und Prozessmanagement der Daimler AG. Die Problemstellungen des Managements von Prozessvarianten wurden zunächst im Rahmen des Forschungsprojektes *Process Management Infrastructure (PMI)* identifiziert und anschließend im Teilprojekt Provop intensiv bearbeitet (Provop steht für Prozessvarianten mittels Optionen).

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben. Allen voran danke ich Prof. Dr. Manfred Reichert, der die universitäre Betreuung dieser Arbeit übernommen hat, für sein hohes Engagement als Doktorvater und seine stete Motivation und Begeisterung für das Thema. Dr. Thomas Bauer, der diese Arbeit seitens der Daimler AG betreut hat, danke ich für unsere zahlreichen fachlichen Diskussionen. Sie waren mir große Hilfe und Antrieb und ich habe sie stets sehr genossen.

Ich danke Prof. Dr. Helmuth Partsch für die Übernahme des Zweitgutachtens und seinem Interesse an dem Thema. Ebenfalls danken möchte ich den Mitarbeitern des Instituts für Datenbanken und Informationssysteme, die mich als „Externe“ herzlich aufgenommen haben und mir immer als Ansprechpartner, Ratgeber und Motivatoren beigestanden haben. Des Weiteren möchte ich mich bei meinen studentischen Hilfskräften Christoph Mauroner, Steve Rechtenbach und Daniel Ott für ihr Engagement bei der prototypischen Realisierung der Konzepte bedanken.

Mein persönlicher Dank gilt Reiner Siebert, für die Schaffung der optimalen organisatorischen Rahmenbedingungen innerhalb der Daimler AG. Mein Dank geht auch an das Team Prozessmanagement und allen Kollegen der Daimler AG, mit denen ich in den vergangenen Jahren zusammenarbeiten durfte, für die professionelle und freundschaftliche Arbeitsatmosphäre. Insbesondere möchte ich mich bei meinem Bürokollegen Dr. Thomas Beuter herzlich dafür bedanken, dass er mich bereits als Werkstudentin in seinem Büro aufgenommen hat. Die vielen gemeinsamen Jahre unserer Zusammenarbeit werden mir immer positiv in Erinnerung bleiben.

Zu guter Letzt danke ich meiner Familie und meinen Freunden für ihre Geduld und ihr Verständnis sowie das sorgfältige Korrekturlesen. Insbesondere danke ich meinem Mann Andreas für seinen Rückhalt, ohne den diese Arbeit sicher nicht möglich gewesen wäre.

Ulm, im Oktober 2009

Inhaltsverzeichnis

Teil I:
Motivation und Anforderungen

1 Einführung

Unternehmen sind aus verschiedenen Gründen daran interessiert, die Effizienz und Qualität ihrer Geschäftsprozesse kontinuierlich zu steigern [MRB08]. So wurden im Laufe der letzten Jahre vermehrt Systeme für das *Business Process Management* (BPM) in Unternehmen eingeführt. Gleichzeitig entstanden verschiedene Defacto-Standards zur Spezifikation von Prozessmodellen sowie deren Ausführung. Beispielhaft seien an dieser Stelle Unified Modeling Languange (UML) [UML09], Business Process Modeling Notation (BPMN) [BPM09] und Web Sercives Business Process Execution Language (WSBPEL) [OAS07]) genannt. Generell dokumentiert ein Prozessmodell, welche Aktivitäten eine oder mehrere Organisationen durchzuführen haben, um ein bestimmtes Geschäftsziel zu erreichen. Ebenfalls wird beschrieben, in welcher Reihenfolge diese Aktivitäten ausgeführt werden müssen. Neben der Dokumentation und Abbildung solcher Arbeitsabläufe werden bei der Prozessmodellierung auch Informationsflüsse, Rollen und Organisationsstrukturen erfasst und (meist) graphisch dargestellt [Wes07]. Abbildung 1.1 zeigt die Konzepte der Prozessmodellierung beispielhaft für den Prozess zur Bearbeitung eines Produktänderungsantrags in der Automobilindustrie [VDA05].

Zur Erstellung von Geschäftsprozessmodellen existiert eine Vielzahl von Systemen, etwa ARIS Business Architect [IDS08], Innovator [MID08] und WebSphere Business Modeler [IBM08]. Diese Systeme bieten verschiedene Modellierungstechniken, wobei UML-Aktivitätsdiagramme, BPMN-Diagramme und ereignisgesteuerte Prozessketten (EPK) besonders häufig verwendet werden. Mit der expliziten Modellierung von Prozessen werden verschiedene Ziele verfolgt: Durch Dokumentation der Arbeitsabläufe werden diese für die Prozessbeteiligten transparent und können z.B. durch Analyse und verbessert werden [Sch98, Gad05, Wes07]. Eine andere Anwendung stellt die Automatisierung und Simulation von Prozessen oder Teilen davon dar. In diesem Fall wird ein ausführbares Prozessmodell erstellt, das zur Ausführung in einem Workflow-Management-System (WfMS) instanziiert wird. Das WfMS steuert die Ausführung der einzelnen Prozessaktivitäten entsprechend der modellierten Reihenfolge und weist sie den Mitarbeitern (bzw. automatischer Agenten) unter Bereitstellung der erforderlichen Daten und Anwendungen zu [DvdAtH05, LR99, GHS95].

Prozessunterstützung wird heute in nahezu allen Domänen benötigt. In der Automobilindustrie etwa sind der Produktentwicklungsprozess, die Verarbeitung eines Produktänderungs-

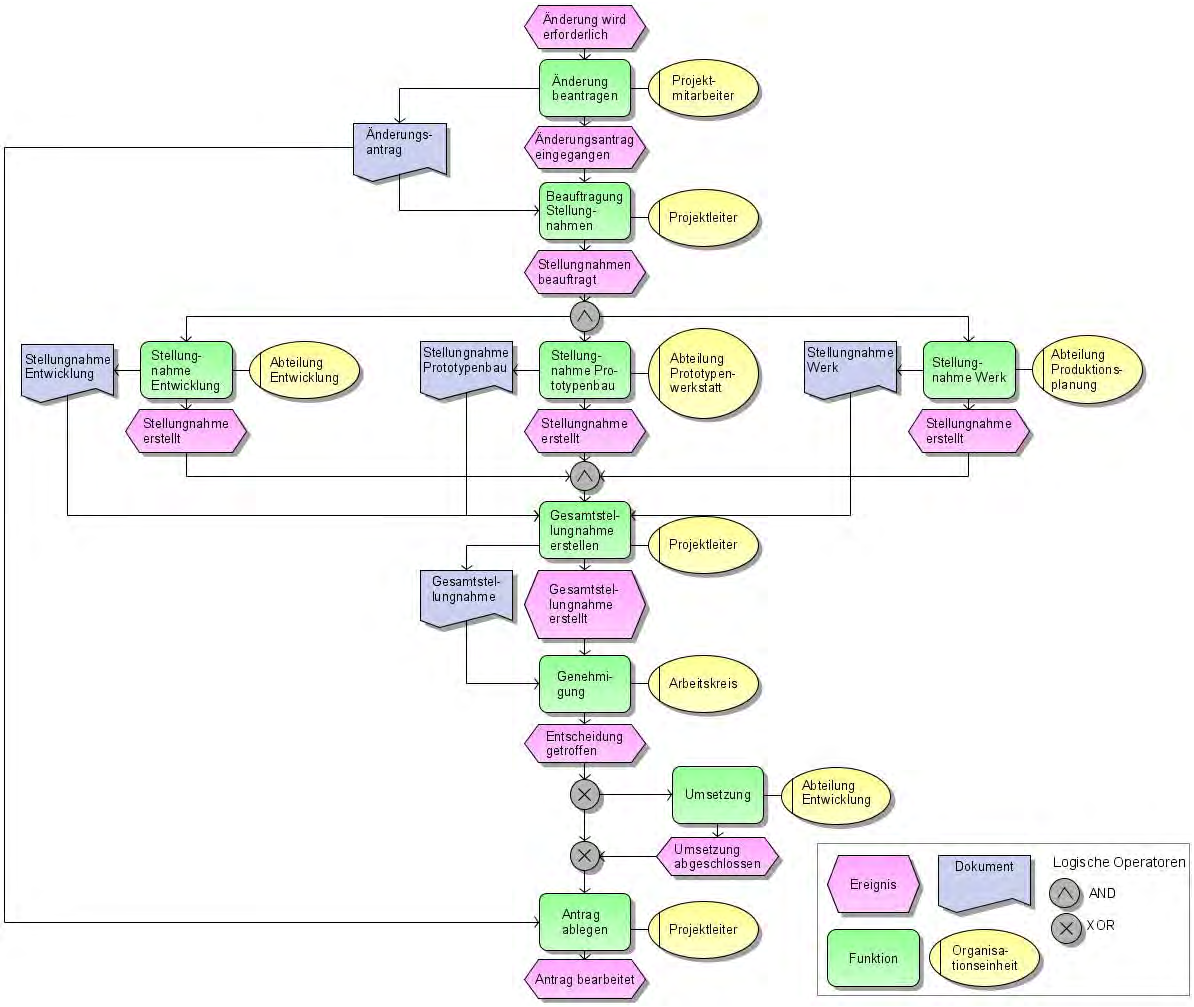


Abbildung 1.1: Prozessmodell aus der Domäne Änderungsmanagement

antrags [VDA05] und die Abnahme von Produkten [MHHR06] charakteristische Prozesse. In der Praxis treten diese Prozesse häufig in unterschiedlichen Varianten auf. Diese besitzen meist eine ähnliche Struktur und der Anteil identischer Prozesselemente (wie z.B. Aktivitäten) ist sehr hoch. Prozessvarianten entstehen meist aufgrund bestimmter Rahmenbedingungen, die eine Abweichung von einem Prozess erforderlich machen. Zum Beispiel treten bei der Produktentwicklung zahlreiche Prozessvarianten, abhängig vom zu entwickelnden Produkt, von organisatorischen Zuständigkeiten, strategischen Ausrichtungen und sonstigen Rahmenbedingungen auf. Diese Rahmenbedingungen beschreiben im Prinzip den Kontext eines Prozesses bzw. einer Prozessvariante. Prozessvarianten stellen somit spezifische Lösungen für bestimmte Kontexte dar.

Beispiele für variantenbehaftete Prozesse finden sich in nahezu allen Domänen. Bei der Patientenbehandlung etwa werden Prozessvarianten, abhängig von Patienteninformationen wie Alter, Allergien oder medizinischer Vorgeschichte, erforderlich [LR07]. Bei der Bestellabwicklung sind verschiedene Lieferbedingungen (z.B. die Beschaffenheit der Ware, Art der Lieferung) für die Bildung von Prozessvarianten zu berücksichtigen. Die Erfassung und Handhabung solcher Prozessvarianten ist für verschiedene Prozessbereiche von Unternehmen und Einrichtungen von Bedeutung. Die Notwendigkeit eines adäquaten Managements dieser Varianten ist somit nicht auf einzelne Branchen oder bestimmte Prozesse begrenzt. Die Ergebnisse dieser Arbeit sind vielmehr für alle Bereiche mit variantenbehafteten Prozessen relevant.

1.1 Kontext der Arbeit

Die vorliegende Arbeit stellt die entwickelten Lösungen unseres Projektes Provop (Prozessvarianten durch Optionen) für ein adäquates Management von Prozessvarianten vor. Provop ist Teil eines größeren Forschungsprojektes der Daimler AG. Dieses Forschungsprojekt befasst sich mit der konkreten Fragestellung, wie eine partner- bzw. organisationsübergreifende, standardisierte Prozessmanagement-Infrastruktur zur Realisierung prozessorientierter Applikationen gestaltet werden soll [BD99, Bau05]. Abbildung 1.2 zeigt eine Übersicht über die wesentlichen Einzelthemen dieses Projektes: Neben der Prozessmodellierung und -steuerung werden die Themen Arbeitslisten-Management [UB08], Applikationsintegration [BP09], Prozessvisualisierung [BRB05, BRB07, BBb07] und Performance-Management [BBa07] untersucht.

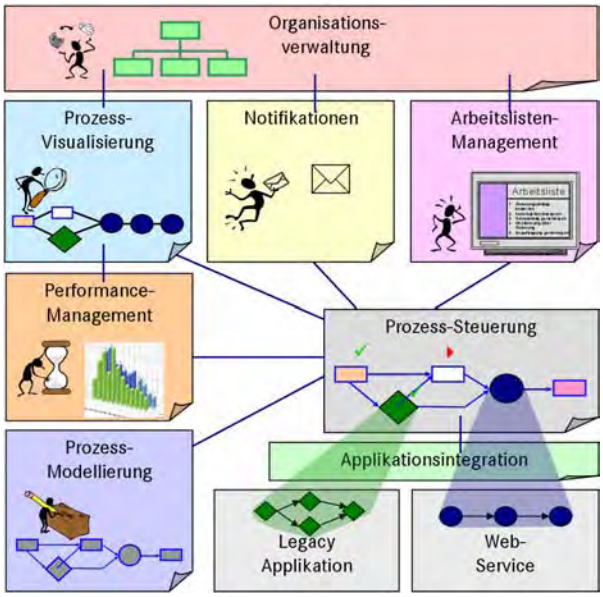


Abbildung 1.2: Elemente einer standardisierten Prozessmanagement-Infrastruktur

Ziel des Forschungsprojektes ist es, die einzelnen Komponenten der Infrastruktur (z.B. Prozesssteuerung, Organisationverwaltung) mit Hilfe kommerzieller Technologien zu realisieren. Für nicht abgedeckte Problemstellungen sowie unreife Technologien werden darüber hinaus innovative Lösungen entwickelt.

Eine bisher nicht ausreichend berücksichtigte Problemstellung einer Prozessmanagement-Infrastruktur betrifft das durchgängige Management von Prozessvarianten. Dies führte zur Entstehung des Projektes Provop. Ziel von Provop ist es gewesen, einen generischen (d.h. nicht Fachprozess-spezifischen) Lösungsansatz zu entwickeln, der ein transparentes und kontextabhängiges Management von Prozessvarianten über alle Phasen des Prozesslebenszyklus erlaubt. Das heißt, wir möchten von der Modellierung und Konfiguration der Prozessvarianten, über ihre Ausführung und Evolution hinweg, eine durchgängige Unterstützung anbieten.

Die Umsetzung der entwickelten Lösungsansätze in verfügbaren Werkzeugen ist Voraussetzung für den breiten Einsatz dieser Technologien in Unternehmen. Ein wesentliches Ziel des Projektes ist es daher, dass neu entwickelte Konzepte von Systemherstellern aufgegriffen und realisiert werden. Der Austausch mit den Systemherstellern entsprechender Werkzeuge, etwa der IDS SCHEER AG und der GBTEC Software + Consulting AG, war daher ein wichtiger Teil unserer Arbeit. Der Erfolg dieser Kooperation ist der teilweise Transfer von Provop-Konzepten in das Werkzeug ARIS der IDS SCHEER AG.

1.2 Problemstellung

Für das adäquate Management von Prozessvarianten ergeben sich primär folgende Forschungsfragestellungen:

- Wie können Varianten transparent und explizit modelliert werden?
- Wie kann der Modellierungs- und Pflegeaufwand für eine Vielzahl von Prozessvarianten reduziert werden?
- Wie können Prozessvarianten in Abhängigkeit von gegebenen Rahmenbedingungen (d.h. dem Prozesskontext) konfiguriert werden?
- Wie kann die Korrektheit von Prozessvarianten gewährleistet werden?
- Wie können Prozessvarianten flexibel ausgeführt werden?
- Wie können Prozessvarianten über den gesamten Prozesslebenszyklus hinweg unterstützt werden?
- Wie kann ein generischer Ansatz (d.h. unabhängig von Prozess-Metamodell oder Domäne) gestaltet werden?

Detaillierte technische Anforderungen im Zusammenhang mit obigen Problemstellungen werden in Kapitel 2 behandelt.

1.3 Beitrag der Arbeit

Das von uns in Provop entwickelte Rahmenwerk für das Management von Prozessvarianten adressiert alle zuvor aufgelisteten Fragestellungen. Dabei liegt der Hauptbeitrag unserer Arbeit in der Unterstützung von Prozessvarianten hinsichtlich des gesamten Prozesslebenszyklus. Das heißt, von der expliziten und transparenten Modellierung und kontextabhängigen Konfiguration der Varianten über deren flexible Ausführung, bis hin zur effizienten Evolution. Darüber hinaus werden in Provop geeignete Mechanismen zur Gewährleistung der Korrektheit der Prozessmodelle angeboten. Solche Mechanismen stellen die korrekte Erstellung und anschließende Ausführung der Prozessvarianten in einem Workflow-Management-System (WfMS) sicher. Diese Durchgängigkeit stellt den entscheidenden Mehrwert unserer Lösung gegenüber existierenden Ansätzen dar.

Es ist unser Ziel, einen generischen und flexibel übertragbaren Ansatz zu entwickeln, daher basiert Provop nicht auf einem konkreten Prozess-Metamodell. Um die Konzepte dieser Arbeit geeignet motivieren und darstellen zu können, ist der Bezug zu einem Prozess-Metamodell unumgänglich. Wir verwenden daher die wesentlichen Konzepte bekannter Modellierungssprachen (z.B. EPK [Sch98], BPMN [BPM09]) und beschränken uns auf die wichtigsten Sprachelemente. Die in dieser Arbeit vorgestellten Konzepte sind mit entsprechenden Anpassungen auf bekannte Modellierungssprachen übertragbar. Wir realisieren unsere Ansätze in einem Prototypen und validieren sie praktisch anhand mehrerer Fallstudien aus verschiedenen Prozessdomänen.

1.4 Forschungsmethodik

Zur Erhebung von Anforderungen an das Management von Prozessvarianten werden verschiedene Fallbeispiele aus der Automobildomäne untersucht. Bei der Analyse der Anforderungen zeigen wir auf, dass eine durchgängige Lösung erforderlich ist, d.h. das zu entwickelnde Konzept muss alle Phasen des Prozesslebenszyklus unterstützen. Wir entwickeln daher das Provop-Rahmenwerk entlang des Prozesslebenszyklus und berücksichtigen dabei alle zuvor identifizierten Anforderungen.

Im Anschluss an die Entwicklung des Rahmenwerks führen wir eine Validation unseres Lösungsansatzes durch. Dazu wird ein Prototyp entwickelt, welcher die Modellierung, Konfiguration und Ausführung von Prozessvarianten unterstützt. Die Konzepte und entwickelten Technologien werden anschließend in mehreren Fallstudien eingesetzt. Diese umfassen ein weites Spektrum an Anwendungsfällen und stammen aus den Domänen Automobilindustrie, Stadtverwaltung und Klinik. Sie zeigen auf, welche Vor- und Nachteile Provop im Vergleich zu konventionellen Methoden des Variantenmanagements aufweist und welche Erweiterungen des Provop-Ansatzes in Zukunft noch denkbar sind.

1.5 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in vier Hauptteile:

Teil I (Einleitung):
Im ersten Teil der Arbeit geben wir eine Motivation für das Management von Prozessvarianten. Dazu diskutiert Kapitel 1 das Auftreten von Varianten im Bereich des Prozessmanagements. Anschließend stellt Kapitel 2 die wichtigsten Anforderungen an ein adäquates Management von Prozessvarianten vor.

Teil II (Konzept):
Im zweiten Teil stellen wir in Kapitel 3 zunächst die Grundlagen zum Verständnis der Arbeit vor. Dazu erklären wir wichtige Begriffe aus dem Bereich der Prozessmodellierung und beschreiben das Prozess-Metamodell für diese Arbeit. Kapitel 4 führt den Lösungsansatz von Provop ein. Anschließend erfolgt eine Betrachtung des Lösungsansatzes im Detail, entlang der Phasen des Prozesslebenszyklus (d.h. Modellierung, Konfiguration, Ausführung und Evolution von Prozessvarianten) in den Kapiteln 5 bis 8. Schließlich diskutiert Kapitel 9 phasenübergreifende Aspekte.

Teil III (Validation):
Im dritten Teil validieren wir unseren Provop-Ansatz. Dazu präsentiert Kapitel 10 den entwickelten Prototyp. Anschließend beschreibt Kapitel 11 verschiedene Fallstudien, die wir im Rahmen dieser Arbeit durchführen, und fasst wesentliche Ergebnisse und Erfahrungen zusammen.

Teil IV (Abschluss):
Zum Abschluss erfolgt im vierten Teil eine Diskussion verwandter wissenschaftlicher Arbeiten in Kapitel 12. Außerdem fassen wir die Arbeit und ihre Beiträge zusammen. Der Ausblick in Kapitel 13 informiert über weitere interessante Themen auf dem Gebiet des Variantenmanagements, die aufbauend auf den entwickelten Konzepten in nachfolgenden Arbeiten vertiefend betrachtet werden können.

Teil V (Anhang):
Der Anhang dieser Arbeit enthält eine Übersicht der Provop-spezifischen Funktionen und weitere Abbildungen.

2

Anforderungsanalyse

In diesem Kapitel erläutern wir die spezifischen Anforderungen an das Management von Prozessvarianten. Zunächst geben die Abschnitte 2.1 und 2.2 je ein Fallbeispiel für variantenreiche Prozesse. Diese Beispiele sind vereinfachte, aber dennoch realitätsnahe Szenarien aus der Automobildomäne. Sie werden im Verlauf der Arbeit referenziert, um ausgewählte Aspekte des Provop-Lösungsansatzes zu erläutern. Anschließend beschreibt Abschnitt 2.3, welche funktionalen und nicht-funktionalen Anforderungen aus diesen Fallbeispielen abgeleitet werden können. Abschnitt 2.4 stellt abschließend den Stand der Technik vor, d.h. die Umsetzung von Variantenmanagementkonzepten in existierenden Werkzeugen.

2.1 Fallbeispiel 1: Änderungsmanagement

Während der Entwicklung eines Produktes tritt oft der Fall ein, dass Entwicklungsstände von Produktkomponenten überarbeitet und angepasst werden müssen. Entsprechende Änderungen können sich auf andere Produktkomponenten auswirken und z.B. Entwicklungszeiten und -kosten erhöhen. Eine Änderung der Geometrie eines Autodaches wirkt sich z.B. auf den Rohbau des Gesamtfahrzeugs aus. Wird die Komponente Autodach erst zu einem späten Zeitpunkt in der Entwicklung geändert (z.B. kurz vor Serienproduktion), so sind die zu erwartenden Änderungskosten entsprechend hoch. Der Zeitpunkt der Beantragung eines Änderungsantrags wirkt sich daher auf die Gestaltung des Prozesses aus.

In der Praxis ist genau vorgegeben, wie Änderungsanträge gestellt und geprüft werden, sowie welche Entscheidungs- und Genehmigungsprozesse befolgt werden müssen, bevor eine Produktänderung tatsächlich umgesetzt werden kann bzw. darf [VDA05]. Abbildung 2.1 zeigt ein stark vereinfachtes Prozessmodell zur Abwicklung von Änderungsanträgen. Der dargestellte Prozess beginnt mit der Beantragung einer Produktänderung durch einen Projektmitarbeiter. Anschließend werden Stellungnahmen der betroffenen Bereiche eingeholt (z.B. Entwicklung, Prototypen-Bau). Nachdem alle Stellungnahmen eingetroffen sind, erstellt der Projektleiter eine Gesamtstellungnahme, welche anschließend einem Gremium als Entscheidungsgrundlage vorgelegt wird. Erfolgt die Genehmigung der Änderung, wird diese entsprechend umgesetzt. Andernfalls wird die Umsetzung ausgelassen.

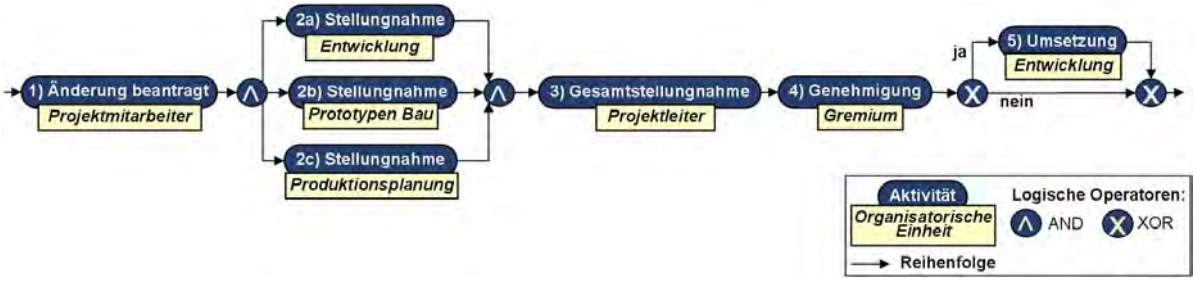


Abbildung 2.1: Vereinfachter Prozess zur Abwicklung eines Änderungsantrags

In der Praxis wird dieser Prozess unter verschiedenen Rahmenbedingungen ausgeführt. Beispiele sind die Umsetzungskosten einer Änderung, die betroffenen Komponenten und die Entwicklungsphase des Produkts. Für diese Bedingungen sind ggf. Anpassungen des vordefinierten Standardprozesses erforderlich. Dies führt zur Entstehung von Prozessvarianten. Abbildung 2.2a bis 2.2c zeigt drei mögliche Varianten des Standardprozesses:

- **Prozessvariante 1** wird unter der Bedingung erforderlich, dass eine qualitätskritische Änderung vorgenommen werden soll. In diesem Fall muss eine zusätzliche Stellungnahme der Abteilung für Qualitätssicherung angefordert werden. Auf Modellebene bedeutet dies, dass die Aktivität `2d` zum Standardprozess hinzugefügt werden muss.

- **Prozessvariante 2** wird während des Anlaufs des Produkts relevant, wenn ein geringes Risiko und kurze Umsetzungszeiten ein frühzeitiges Beginnen mit der Umsetzung der Änderung erlauben. Da nicht auf die Genehmigung eines Gremiums gewartet wird, muss die Umsetzung gegebenenfalls rückgängig gemacht werden, sollte später gegen die Änderung entschieden werden. Für das Prozessmodell bedeutet dies, dass die Aktivität zur Umsetzung der Änderung parallel zum Einholen der Stellungnahmen durchgeführt werden kann. Zusätzlich wird die „Undo“-Aktivität für die Umsetzung nach der Entscheidungsfindung in das Modell eingefügt.

- **Prozessvariante 3** stellt den kombinierten Anwendungsfall von Variante 1 und 2 dar: Bei qualitätskritischen Änderungen, mit geringem Risiko und kurzer Umsetzungszeit, die während der Prozessphase „Anlauf“ angefordert werden, wird eine zusätzliche Stellungnahme der Qualitätsabteilung erforderlich. Gleichzeitig kann die Umsetzung vorgezogen werden und ein gegebenenfalls notwendiges Rücksetzen der Änderung wird eingeplant.

Bei der Erstellung der aufgeführten Varianten sind unterschiedliche Aspekte zu berücksichtigen. So werden unter anderem Aspekte wie die Qualitätsrelevanz einer beantragten Änderung, Risikofaktoren, Umsetzungsaspekte und die Entwicklungsphase des Produktes bei der Variantenbildung betrachtet. Bereits für dieses vereinfachte Beispiel eines variantenbehafteten Prozesses ergeben sich in der Praxis ein knappes Dutzend an Varianten.

2.2 Fallbeispiel 2: Prozesse in einer Werkstatt

Auch für die Phase nach Produktion und Verkauf eines Fahrzeuges gibt es im Automobilbereich variantenbehaftete Prozesse. Zum Beispiel existiert ein Prozess im Kundendienst zur Abwicklung einer Fahrzeugreparatur. Dieser Werkstattprozess beschreibt zunächst ein Standardvorgehen für die verschiedenen Fahrzeugtypen.

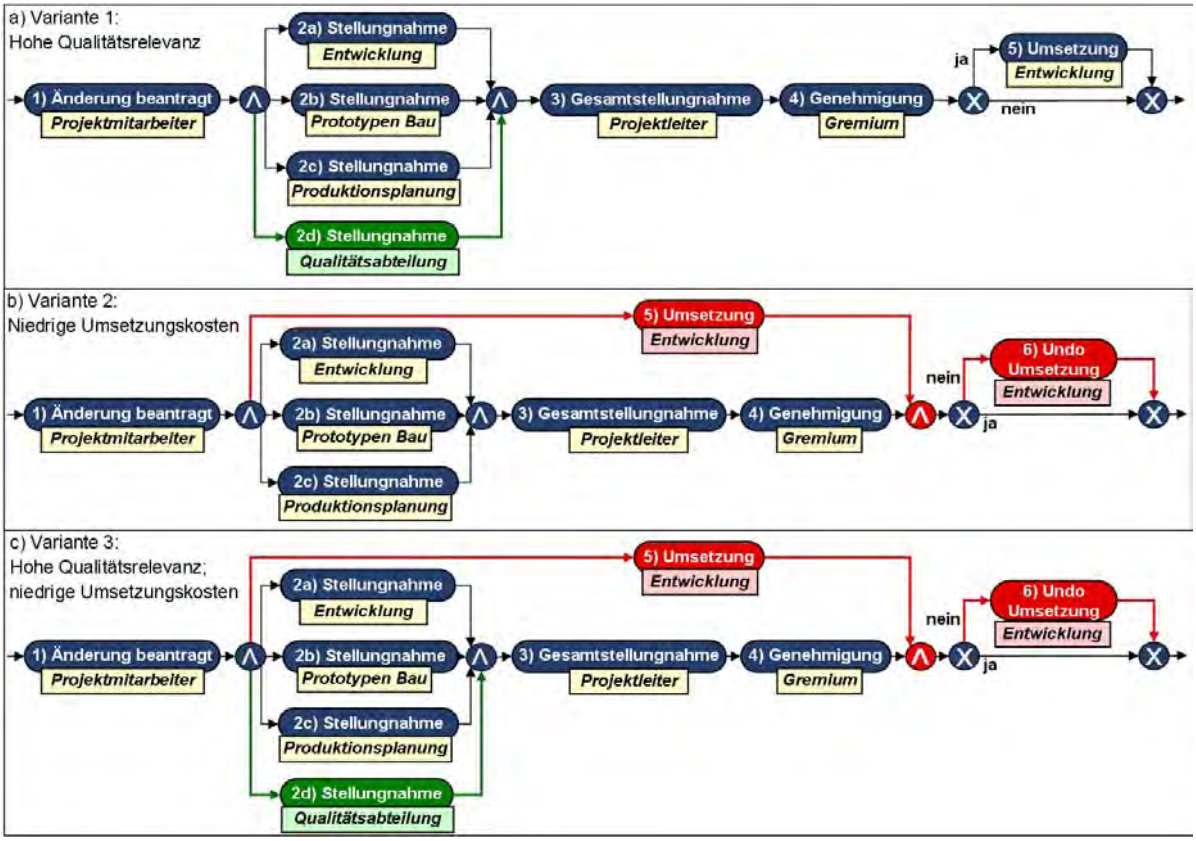


Abbildung 2.2: Varianten zur Abwicklung eines Änderungsantrags

Abbildung 2.3a zeigt den stark vereinfachten, aber dennoch realitätsnahen Standardprozess, der weltweit für alle Werkstätten gültig ist: Nach Annahme des Fahrzeuges vom Kunden wird eine Schadens- bzw. Fehlerdiagnose durchgeführt. Dazu werden fahrzeugspezifische Checklisten verwendet. Wird bei der Diagnose ein Schaden oder Fehler erkannt, wird das Fahrzeug repariert. Andernfalls entfällt dieser Bearbeitungsschritt.

Parallel zur Durchführung von Diagnose und Reparatur findet die Wartung des Fahrzeugs als spezielle Serviceleistung statt; z.B. werden, falls nötig, Öl und Scheibenwischwasser nachgefüllt. Der Prozess endet mit der Übergabe des Fahrzeugs an den Kunden.

Der skizzierte Standardprozess wird für verschiedene Fahrzeugtypen zentral definiert und anschließend weltweit kommuniziert (vgl. Abbildung 2.3b). In den einzelnen Ländern findet dann eine Anpassung der Prozessmodelle statt. Dabei werden im Wesentlichen sprachliche bzw. begriffliche Anpassungen vorgenommen und die Prozessmodelle der jeweiligen Gesetzeslage angepasst. Bereits in diesem Schritt entsteht weltweit eine Vielzahl neuer Varianten des Werkstattprozesses. Im nächsten Schritt finden weitere Anpassungen auf der Ebene der Werkstätten statt (vgl. Abbildung 2.3c). Diese werden notwendig, um den Werkstätten Handlungsspielraum für bestimmte Serviceleistungen zu bieten oder gewachsene Strukturen zu erhalten. Auch dynamische Aspekte, etwa das Reagieren auf das Erreichen von Auslastungsgrenzen können Werkstätten individuell in entsprechenden Prozessvarianten dokumentieren.

Die Vielzahl von variantenbildenden Faktoren im Bereich der Werkstattprozesse führt zu hunderten Prozessvarianten weltweit.

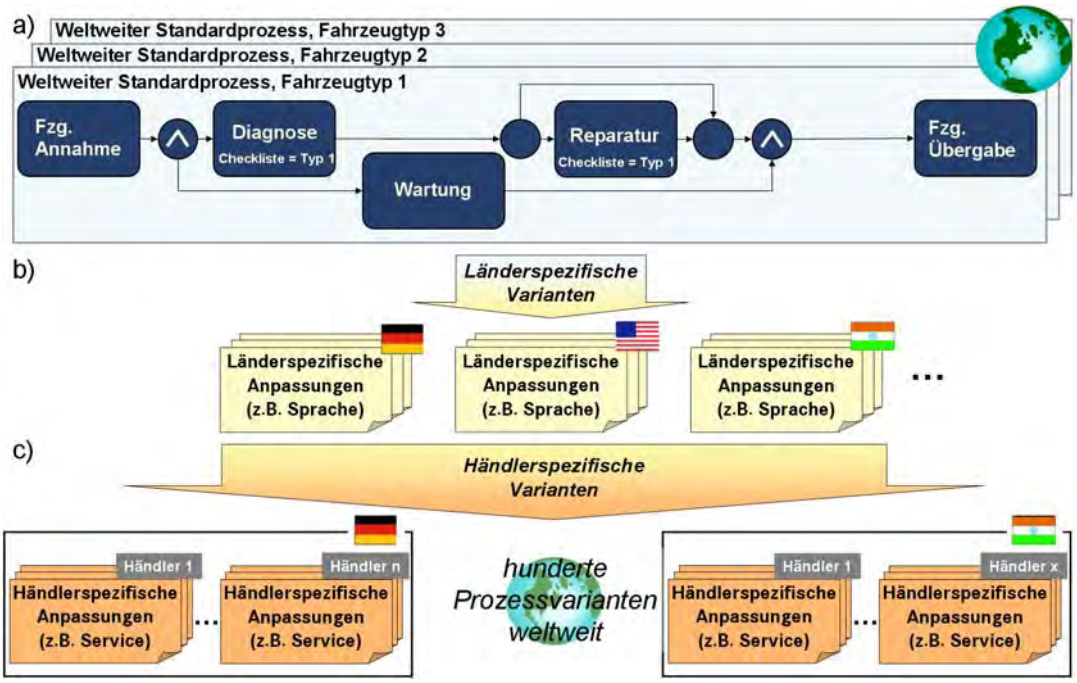


Abbildung 2.3: Entstehung von Prozessvarianten des weltweiten Standard-Werkstattprozesses

2.3 Anforderungen

Zur Identifikation der Anforderungen an das Management von Prozessvarianten sind von uns verschiedene Anwendungsfälle in der Automobilindustrie und anderen Bereichen (z.B. Registrierungsprozesse in Stadtverwaltungen und klinische Untersuchungsprozesse) untersucht worden. Dieser starke Praxisbezug ist notwendig, um einen vollständigen und praktikablen Ansatz für das Management von Prozessvarianten zu realisieren.

Die identifizierten Anforderungen beziehen sich auf verschiedene Aspekte, etwa die Modellierung einer großen Zahl von Prozessvarianten und ihre flexible Ausführung in einem WfMS. Es handelt sich somit um Anforderungen, die über den gesamten Lebenszyklus eines Prozesses hinweg relevant sind [HBR08a, HBR07]. Abbildung 2.4 zeigt den vereinfachten Prozesslebenszyklus. Er besteht aus den Phasen der Modellierung, Konfiguration, Ausführung und Evolution von Prozessvarianten.

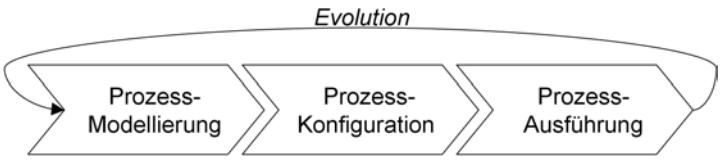


Abbildung 2.4: Prozesslebenszyklus

Im Folgenden beschreiben wir die wichtigsten Anforderungen an das adäquate Management von Prozessvarianten entlang der Phasen des Prozesslebenszyklus. Abschließend behandeln wir Anforderungen, die nicht einer konkreten Phase zugeordnet werden können, sondern phasenübergreifend relevant sind (vgl. Tabelle 2.1). Zur Motivation der Anforderungen beziehen wir uns auf die in den Abschnitten 2.1 und 2.2 vorgestellten Fallstudien des Änderungsmanagements und der Werkstattprozesse und greifen wichtige Problemstellungen dieser Bereiche auf.

Tabelle 2.1: Überblick der Anforderungen an das Management von Prozessvarianten

Modellierung
Anforderung 2.1 *Prozessvarianten müssen explizit und transparent modellierbar sein.*
Anforderung 2.2 *Die Modellierungs- und Pflegeaufwände müssen durch Wiederverwendung minimiert werden können.*
Konfiguration
Anforderung 2.3 *Bei der Konfiguration einer Prozessvariante müssen strukturelle und semantische Abhängigkeiten zwischen verschiedenen Konfigurationseinstellungen berücksichtigt werden.*
Anforderung 2.4 *Die manuelle sowie die automatisierte Konfiguration von Prozessvarianten muss in Abhängigkeit vom Prozesskontext möglich sein.*
Ausführung
Anforderung 2.5 *Die Ausführung von Prozessvarianten in einem WfMS muss möglich sein.*
Anforderung 2.6 *Es muss zur Laufzeit möglich sein, die Prozessausführung auf eine andere Variante „umzustellen“.*
Anforderung 2.7 *Die Migration zwischen Prozessvarianten während der Ausführung muss unterstützt werden.*
Evolution
Anforderung 2.8 *Die Aufwände zur Evolution von Prozessen und ihren Prozessvarianten muss reduziert werden.*
Anforderung 2.9 *Die einfache Pfleg- und Wartbarkeit der Prozessvarianten muss durch geeignete Maßnahmen (d.h. Refactorings) bewahrt werden können.*
Phasenübergreifende Anforderungen
Anforderung 2.10 *Die Prüfung der (hinsichtlich eines gegebenen Prozess-Metamodells) korrekten Modellierung, Konfiguration und Ausführung der Prozessvarianten muss möglich sein.*
Anforderung 2.11 *Die Analyse und Vergleichbarkeit von Varianten muss unterstützt werden.*
Anforderung 2.12 *Die Skalierbarkeit der Konzepte bei geringer bzw. sehr großer Anzahl von Prozessvarianten muss gegeben sein.*

Modellierung

Bei der Modellierung einer Vielzahl von Prozessvarianten ist der Modellierer durch eine intuitive Modellierungsmethodik zu unterstützen. Die Prozessvarianten sind explizit als solche zu kennzeichnen und müssen dem Modellierer transparent dargestellt werden (vgl. Anforderung 2.1).

Wie das Fallbeispiel der Werkstattprozesse in Abschnitt 2.2 zeigt, können für einen einzelnen Prozess hunderte von Prozessvarianten vorliegen. Es ist nicht wünschenswert, diese sehr ähnlichen Prozessmodelle, redundant zu modellieren. Vielmehr müssen Redundanzen durch Wiederverwendung geeignet reduziert werden (vgl. Anforderung 2.2). Gleichbleibende Pro-

zesselemente (z.B. Aktivitäten) sollen z.B. in die Modelle aller Prozessvarianten übernommen werden können.

Konfiguration

In dieser Phase muss aus der Vielzahl an modellierten Prozessvarianten diejenige ausgewählt bzw. konfiguriert werden können, die aktuell für Analysen oder zur Ausführung in einem WfMS benötigt wird. Eine Variante muss dabei sowohl manuell als auch automatisch generiert werden können (vgl. Anforderung 2.4).

Die Fallbeispiele zeigen, dass Prozessvarianten meist in einem spezifischen Kontext relevant sind (z.B. im Kontext eines bestimmten Landes oder Händlers beim Werkstattprozess). Für die automatische Konfiguration von Prozessvarianten soll dieser Kontext berücksichtigt werden. Dazu muss der Modellierer zunächst definieren können, in welchem Kontext eine bestimmte Prozessvariante gültig ist. Dies erfordert ein Kontextmodell, welches den Prozesskontext einer bestimmten Domäne erfassen kann. Des Weiteren muss der Kontextmodellierer den Zusammenhang zwischen einer spezifischen Prozessvariante und dem gegebenen Kontext in einer expliziten Form festlegen können und zwar derart, dass diese Information vom System zur Bestimmung der aktuell benötigten Prozessvariante ausgewertet werden kann. Allerdings muss es auch bei fehlenden Kontextinformationen möglich sein, eine ausführbare Prozessvariante zu erstellen.

Nicht alle theoretisch konfigurierbaren Prozessvarianten sind tatsächlich in der Praxis relevant oder hinsichtlich der Kriterien des entsprechenden Prozess-Metamodells korrekt. Die strukturelle und semantische Korrektheit der Prozessvarianten ist daher bei der Konfiguration zu berücksichtigen (vgl. Anforderung 2.3).

Ausführung

In der Praxis sollen Prozesse oft EDV-gestützt ausgeführt bzw. durch ein EDV-System implementiert werden. Dazu muss eine Prozessvariante in einem WfMS in ein ausführbares Workflow-Modell transformiert werden (vgl. Anforderung 2.5). Diese Transformation soll automatisch durchgeführt werden. Alle relevanten Daten der vorangegangenen Prozesslebenszyklusphasen sind entsprechend zu übertragen.

Oftmals fehlen beim Start einer Prozessvariante bestimmte Kontextinformationen, die sich erst während der Ausführung des Prozesses ergeben (z.B. der Schweregrad des diagnostizierten Fahrzeugschadens oder die Höhe von Änderungskosten). In solchen Fällen muss es möglich sein, erst zur Laufzeit festzulegen, welche Prozessvariante tatsächlich ausgeführt werden soll (vgl. Anforderung 2.6).

Ein weiterer Aspekt ist die Dynamik des Anwendungskontextes eines Prozessmodells. So können Änderungen in Kontextinformationen (z.B. Rekalkulationen von Kosten nach zusätzlichen Stellungnahmen) dazu führen, dass eine andere Prozessvariante, als die gerade ausgeführte, im aktuellen Kontext erforderlich ist. Tritt dieser Fall ein, muss das „Umschalten“ zwischen definierten Prozessvarianten unterstützt werden (vgl. Anforderung 2.7).

Evolution

Generell ist nicht zu erwarten, dass variantenbehaftete Prozesse einmalig modelliert und anschließend keine Änderungen mehr vorgenommen werden. Die Änderbarkeit von Prozessmodellen ist daher grundsätzlich zu unterstützen. Bei einer Vielzahl von Prozessvarianten kommt die Problemstellung hinzu, dass bei grundlegenden Änderungen sehr viele Prozessmodelle anzupassen sind (z.B. eine Gesetzesänderung für alle Werkstätten Mitteleuropas).

Dies kann nicht nur zu fehlerhaften Modellen führen, sondern ist vor allem auch sehr aufwendig. Die durchzuführenden Anpassungen müssen daher für alle betroffenen Prozessvarianten mit möglichst geringem Arbeitsaufwand vorgenommen werden, d.h. im besten Fall in einem einzigen Arbeitsschritt (vgl. Anforderung 2.8).

Ein anderer Aspekt betrifft die einfache Handhabung von Prozessvarianten (vgl. Anforderung 2.9). Zu diesem Zweck können sog. *Refactorings* verwendet werden. Ein Refactoring ist eine strukturelle Anpassung von Prozessmodellen, wobei das Ausführungsverhalten der Prozessemodelle nicht verändert wird. Refactorings werden eingesetzt, um Redundanzen zu minimieren und somit die Les- und Pflegbarkeit der Prozessvarianten zu erhöhen.

Korrektheit

Ein wichtiger Aspekt für das adäquate und durchgängige Management von Prozessvarianten ist die Betrachtung der Korrektheit von Variantenmodellen- und ausführungen (vgl. Anforderung 2.10). Relevante Anforderungen betreffen dabei den gesamten Prozesslebenszyklus. So muss das Modell einer konfigurierten Prozessvariante hinsichtlich der Kriterien des zugrunde liegenden Prozess-Metamodells korrekt sein (z.B. dürfen in bestimmten Metamodellen keine zyklischen Ausführungspfade modelliert werden), um eine fehlerhafte Ausführung der Modelle gewährleisten zu können.

Die Unterstützung des dynamischen Umschaltens zwischen Prozessvarianten und das sukzessive Festlegen des eigentlichen Prozessmodells zur Laufzeit, machen es erforderlich, auch während der Ausführung der Prozessmodelle die Korrektheit der Prozessvarianten sicherzustellen, damit nicht zu einer inkorrekten Prozessvariante gewechselt werden kann bzw. sich nach einem solchen Wechsel ein inkorrekter Zustand ergibt. Der Endanwender ist über auftretende Probleme zu informieren und muss bei der Auflösung dieser Probleme vom System unterstützt werden.

Darstellung

Zur Harmonisierung und Standardisierung parallel entwickelter Prozessvarianten müssen ihre Prozessmodelle miteinander verglichen werden können (vgl. Anforderung 2.11). Dazu sind anwenderorientierte Darstellungskonzepte erforderlich. Die Modelle der Varianten müssen für den Modellierer je nach Anwendungsfall durch unterschiedliche Sichten und Darstellungsformen visualisiert werden können. Die einzelnen Visualisierungen sind automatisch vom System zu generieren und ein Wechsel zwischen den verschiedenen Visualisierungsformen muss unterstützt werden. Darüber hinaus müssen Vergleiche von verschiedenen Prozessvarianten mit Hilfe einer geeigneten Visualisierung unterstützt werden. Dabei sind Aspekte, wie die Platzierung der Objekte auf der Modellierungsoberfläche, im Sinne der sog. *Mental Map* [MELS95], zu berücksichtigen.

Skalierbarkeit

Die Fallbeispiele zeigen, dass die Anzahl der erforderlichen Prozessvarianten stark variieren kann. Während für den Änderungsmanagementprozess nur ein knappes Dutzend von Prozessvarianten existieren, sind es für Werkstattprozesse bereits mehrere hundert. Daher resultiert die Anforderung nach Skalierbarkeit des Ansatzes (vgl. Anforderung 2.12).

2.4 Stand der Technik

Im Folgenden analysieren und bewerten wir existierende Werkzeuge hinsichtlich der vorgestellten Anforderungen an das Variantenmanagement. Tabelle 2.2 zeigt unsere Bewertungen im Überblick.

ARIS Business Architect

Das Modellierungswerkzeug ARIS Business Architect (kurz: ARIS) von IDS Scheer (vgl. Abbildung 2.5a) wird zur Dokumentation und Visualisierung von Geschäftsprozessmodellen eingesetzt [Sch98, IDS08]. Neben verschiedenen Prozess-Metamodellen (z.B. EPK und BPMN) wird z.B. auch die Erstellung von Datendiagrammen und Organigrammen unterstützt. In ARIS können von Prozessmodellen oder einzelnen Symbolen (z.B. Funktionen) Varianten angelegt werden. Dazu werden diese Objekte kopiert und ein Verweis auf das Original erstellt. Mit Hilfe dieser Verweisstruktur können wir für ein Objekt ermitteln, ob Varianten existieren und in welchen anderen Modellen diese verwendet werden. Wir können außerdem feststellen, ob das untersuchte Objekt selbst eine Variante darstellt. Über die Verweisstruktur kann allerdings keine weitere Information transportiert werden. Zum Beispiel werden Änderungen am Original nicht automatisch auf alle Prozessvarianten übertragen. ARIS ist ein reines Prozessmodellierungswerkzeug. Ausführungsaspekte, wie der Wechsel zwischen Prozessvarianten zur Laufzeit, können daher nicht bewertet werden. Durch den Einsatz weiterer Werkzeuge der ARIS-Produktfamilie kann ein Monitoring der Prozessmodelle vorgenommen werden. Diese berücksichtigen jedoch nicht explizit den Aspekt der kontinuierlichen Verbesserung und Erhebung von Prozessvarianten. Die vergleichende Darstellung von Varianten bzw. Prozessmodellen ist in ARIS eingeschränkt möglich. So können zwei Prozessvarianten in einer Ansicht direkt verglichen werden. Das System hebt dabei das Delta der Prozessmodelle hervor. Diese Funktion wurde nicht spezifisch für die Modelle von Prozessvarianten entwickelt, sondern steht allgemein für Prozessmodelle zur Verfügung. Die Prozessmodelle können in ARIS nur bedingt auf Korrektheit überprüft werden. So ist eine Validation der strukturellen Anordnung der Symbole möglich, eine Prüfung der korrekten Ausführung wird allerdings nicht unterstützt.

Innovator

Das Werkzeug Innovator von MID (vgl. Abbildung 2.5b) wird für die rechnergestützte Softwareentwicklung (engl. *computer-aided Software Engineering (CASE)*) eingesetzt und unterstützt den Standard UML 2.0 [MID08]. Innovator wird vorallem verwendet, um systemunabhängige technische Prozesse zu modellieren. Dieses Werkzeug bietet kein explizites Variantenmanagementkonzept an. So können Varianten weder graphisch miteinander verglichen noch aus bestehenden Prozessmodellen konfiguriert werden. Strukturelle und semantische Abhängigkeiten sind nur über Namenskonventionen abbildbar. Laufzeitaspekte (z.B. Auswahl von Prozessvarianten zur Laufzeit) werden nicht berücksichtigt. Die gezielte Evolution von Prozessmodellen wird in Innovator ebenfalls nicht unterstützt. Die Korrektheit von Prozessmodellen wird kontinuierlich geprüft, d.h. bereits während der Modellierung werden nur korrekte Prozesse zugelassen.

TIBCO Business Studio

Das Business Studio von TIBCO (vgl. Abbildung 2.5c) – vormals Staffware – wird zur Geschäftsprozessmodellierung eingesetzt und unterstützt die BPMN Modellierungsnotation. Business Studio wird in erster Linie zur Simulation von Prozessmodellen verwendet. Zur

Durchführung der Simulation kann das Prozessmodell zunächst validiert werden. Auf diese Weise werden z.B. Endlosschleifen oder nicht eindeutige Prozesselemente identifiziert. Ein Variantenmanagementkonzept wird nicht angeboten. Auch vergleichende Darstellungen von Prozessmodellen sind nicht möglich, allerdings werden die Simulationsergebnisse in einer Übersicht angegeben. Ein im TIBCO Business Studio erstellter Geschäftsprozess kann als Workflow-Modell exportiert werden.[1] Das heißt die Durchgängigkeit von einem Geschäftsprozess zu einem ausführbaren Workflow-Modell ist zumindest ansatzweise gegeben.

WebSphere Business Modeler und WebSphere Integration Developer

Der WebSphere Business Modeler (vgl. Abbildung 2.5d) [IBM08] von IBM wird zur Modellierung von Geschäftsprozessen mit der BPMN verwendet. Dabei ist es das Ziel dieser Modellierung, den so erstellten Prozess später in einem WfMS, dem WebSphere Process Server [IBM09], ausführen zu können. Der WebSphere Integration Developer (vgl. Abbildung 2.5e) [IBM09] wird zur Modellierung ausführbarer Workflow-Modelle eingesetzt. Dabei wird die Darstellung der Modelle an die BPMN angelehnt und intern BPEL-Code generiert [OAS07]. Dieser BPEL-Code kann auf Korrektheit geprüft werden, bevor die Modelle direkt auf dem integrierten WebSphere Process Server ausgeführt werden können. Mit den Systemen der WebSphere Produktfamilie ist die Durchgängigkeit von der Modellierung fachlicher Modelle zu ausführbaren Workflow-Modellen vom Prinzip her gegeben. Die Modellierungswerkzeuge WebSphere Business Modeler und WebSphere Integration Developer bieten kein Variantenmanagementkonzept für Prozesse an. Das heißt es gibt keine Ansätze zur effektiven Modellierung und automatischen Konfiguration von Prozessvarianten sowie deren vergleichenden Darstellung in der Modellierungsoberfläche. Zur Unterstützung einer flexiblen Ausführung existieren außerdem nur rudimentäre Ansätze, wie das Auslassen einzelner Aktivitäten, das Springen im Prozessmodell und das Ändern von Daten. Eine Änderung der Prozessinstanz zur Laufzeit ist nicht möglich. Konzepte zur Evolution von Prozessvarianten sind nicht realisiert.

AristaFlow BPM Suite

Die AristaFlow BPM Suite von AristaFlow (vgl. Abbildung 2.5f) ist ein WfMS, das im Rahmen des *ADEPT* Projektes [RRKD05, Rei00] entwickelt wird. Ziel ist die flexible Ausführung von Prozessmodellen in der Praxis [DR09, DRRM+09]. AristaFlow unterstützt das Management von Prozessvarianten nicht explizit. Das heißt, auch hier können Variantenbeziehungen nur über Namenskonventionen der Modellbezeichner oder eine Verzeichnisstruktur abgebildet werden. Prozessmodelle können nicht direkt miteinander verglichen werden. Eine Form der Konfiguration von Prozessvarianten ist mit einer Weiterentwicklung möglich: Der *ProCycle* Ansatz erlaubt das fallbasierte Speichern von dynamischen Änderungen [WWRD07]. Die gespeicherten „Fälle“ können bei Erreichen einer ähnlichen Problem-Situation zur Laufzeit wiederverwendet werden, d.h. die dynamischen Änderungen werden für das Prozessmodell vorgenommen. AristaFlow unterstützt die flexible Ausführung von Prozessmodellen und ist in der Lage, zur Laufzeit fallspezifische (d.h. instanzspezifische) Änderungen am zugrunde liegenden Prozessmodell in die Ausführung zu propagieren. Damit wird die Evolution von Prozessmodellen explizit unterstützt. Funktionen zur Erhebung oder Optimierung von Prozessvarianten werden nicht angeboten. In AristaFlow wird die Korrektheit bereits bei der Prozessmodellierung geprüft und kann auch bei dynamischen Änderungen zur Laufzeit sichergestellt werden.

[1] Bei einem Export wird aus dem BPMN Modell ein XPDL Dokument generiert. XPDL steht für XML Process Definition Language (dt. Prozessbeschreibungssprache) [Coa08, W3C09].

Tabelle 2.2: Erfüllung der Anforderungen an das Variantenmanagement: ausreichende Konzepte (+), unzureichende Konzepte (0), keine Konzepte (-)

Werkzeug / Anforderung	ARIS	Innovator	TIBCO	WBM[a]	WID[b]	AristaFlow
Modellierung	0	-	-	-	-	-
Konfiguration	-	-	-	-	-	-
Ausführung	-	-	0	0	0	+
Evolution	+	-	-	-	-	+
Korrektheit	0	+	+	0	0	+
Darstellung	0	-	-	-	-	0
Skalierbarkeit	0	0	0	0	0	0

[a] WBM = WebSphere Business Modeler

[b] WID = WebSphere Integration Developer

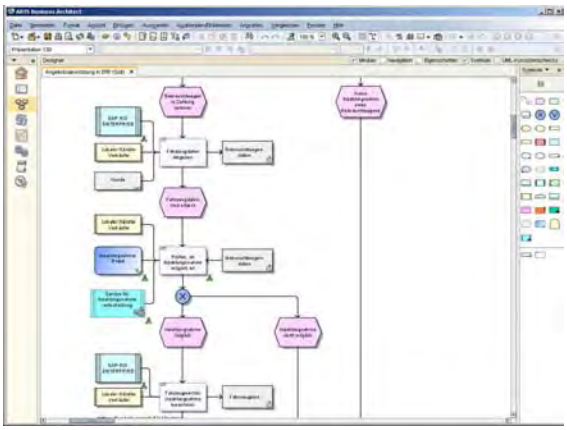
Fazit: Keines der untersuchten Werkzeuge erfüllt alle gestellten Anforderungen an das Management von Prozessvarianten. Es fällt auf, dass mit Ausnahme des ARIS Business Architects kein Werkzeug ein explizites Variantenmanagementkonzept anbietet. Daraus resultieren fehlende Ansätze zur Abbildung des Kontextes zur automatischen Konfiguration einer Prozessvariante oder spezifische Darstellungsmöglichkeiten. Zudem wird auf die Korrektheitsbetrachtung einzelner Prozessmodelle fokussiert.

2.5 Zusammenfassung

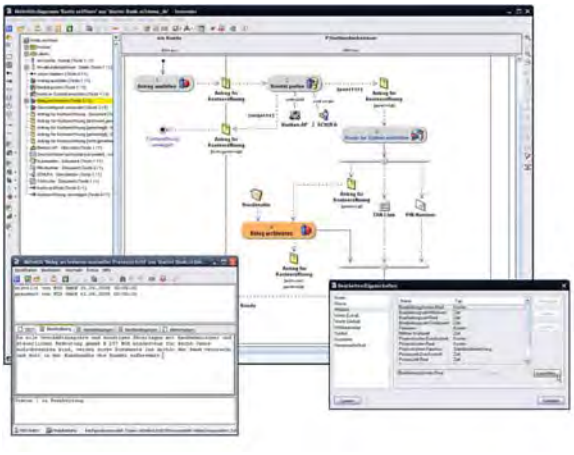
In diesem Kapitel haben wir zwei Fallbeispiele aus der Automobildomäne vorgestellt. Diese Prozessbeispiele der Bereiche Änderungsmanagement und Werkstatt zeigen, welche Prozessvarianten auftreten können und unter welchen Rahmenbedingungen dies geschieht. Mit Hilfe dieser Beispiele haben wir charakteristische Anforderungen identifiziert. Diese Anforderungen betreffen den kompletten Prozess-Lebenszyklus von der expliziten und transparenten Modellierung der Varianten, über deren kontextabhängige Konfiguration, die flexible Ausführung in einem WfMS, bis hin zur aufwandsreduzierten Evolution der Prozessmodelle. Darüber hinaus haben wir phasenübergreifende Anforderungen identifiziert, etwa die Darstellung und Korrektheit von Prozessmodellen und die Skalierbarkeit der Ansätze für eine beliebige Anzahl an Prozessvarianten.

Die Analyse kommerzieller Werkzeuge des Prozessmanagements hinsichtlich unserer Anforderungen hat gezeigt, dass kein System alle Anforderungen erfüllt. Ein explizites, durchgängiges und kontextabhängiges Management von Prozessvarianten wird bisher in keinem Werkzeug unterstützt. Vielmehr sind Variantenmanagementkonzepte in den Systemen oftmals überhaupt nicht realisiert.

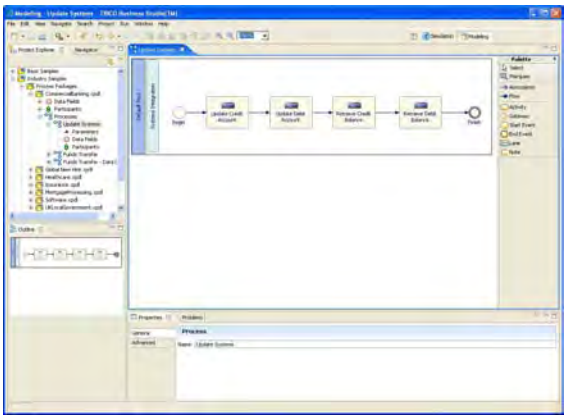
Im Folgenden stellen wir die Grundlagen zum weiteren Verständnis dieser Arbeit vor. Außerdem präsentieren wir den Provop-Lösungsansatz für ein adäquates Variantenmanagement über alle Prozess-Lebenszyklusphasen hinweg und zeigen, wie wir die hier gestellten Anforderungen erfüllen. Wir beziehen uns dabei auf die beschriebenen Fallbeispiele des Änderungsmanagements und der Werkstattprozesse.



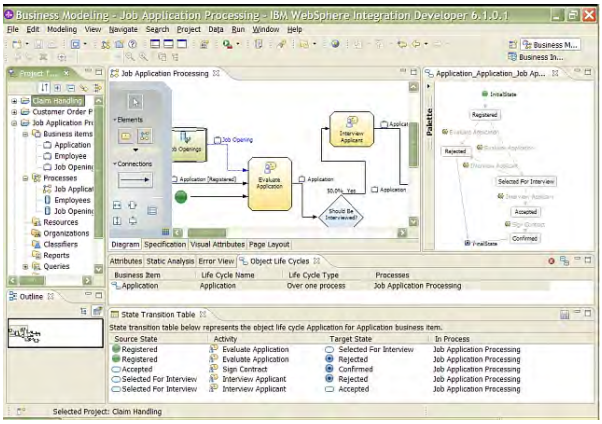
(a) ARIS Business Architekt



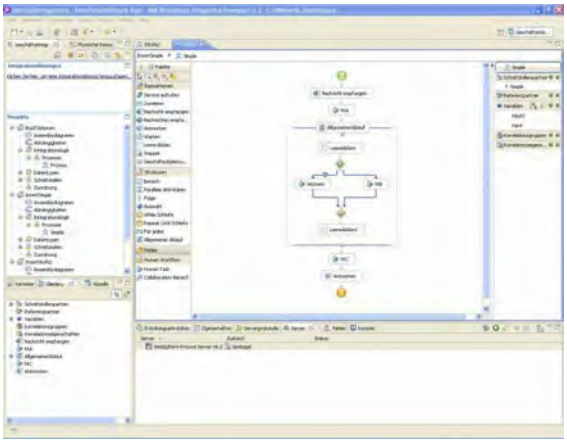
(b) Innovator



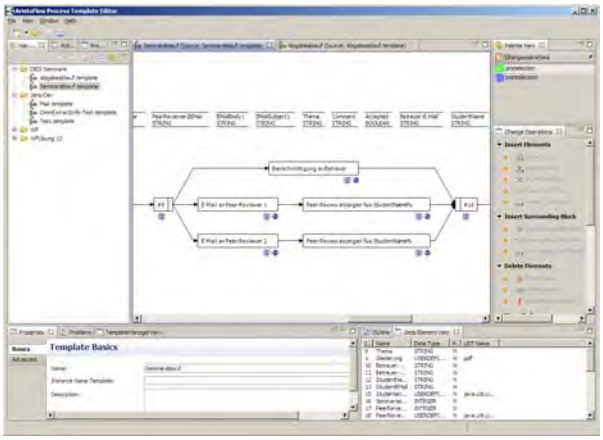
(c) TIBCO Business Studio



(d) WebSphere Business Modeler



(e) WebSphere Integration Developer



(f) AristaFlow BPM Suite

Abbildung 2.5: Prozessmodellierungswerkzeuge

Teil II:
Lösungskonzepte des Provop-Ansatzes

3

Grundlagen

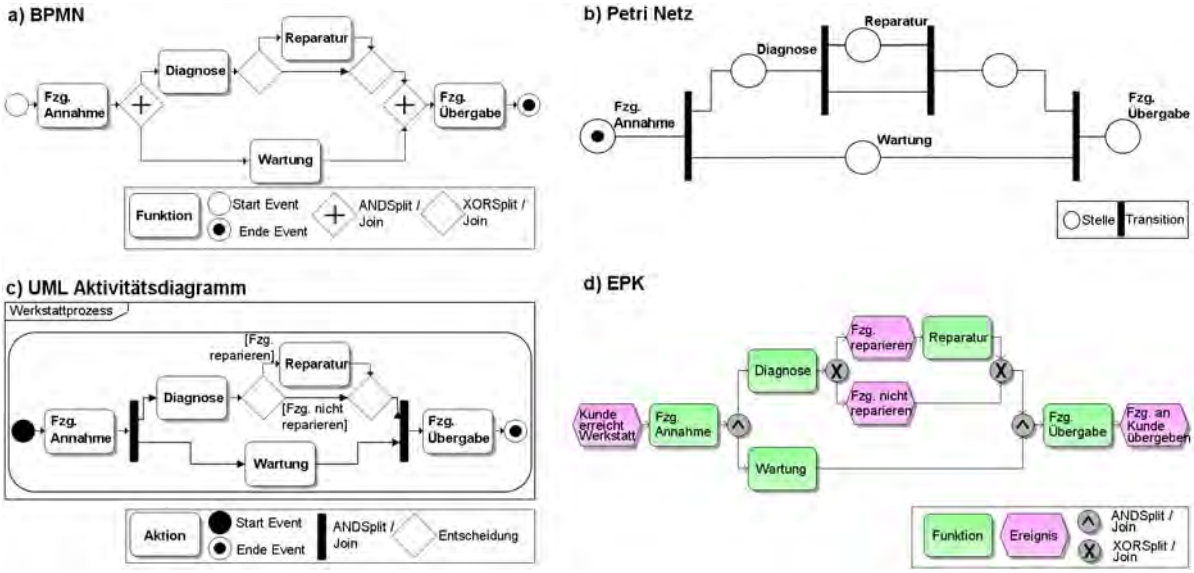
In Kapitel 2 haben wir zwei Fallbeispiele aus der Automobildomäne vorgestellt und entlang dieser Beispiele charakteristische Anforderungen an ein adäquates Variantenmanagement beschrieben. In diesem Kapitel fassen wir wichtige Grundlagen zur Prozessmodellierung in Provop zusammen.

Kapitel 3 gliedert sich wie folgt: Nach einer kurzen Motivation in Abschnitt 3.1 stellt Abschnitt 3.2 das dieser Arbeit zugrunde liegende Prozess-Metamodell vor. Dieses orientiert sich an existierenden Prozessbeschreibungssprachen, nimmt aber einige Reduzierungen vor, um den Fokus auf das Management von Prozessvarianten setzen zu können. Abschnitt 3.3 erläutert, welche Eigenschaften wir von korrekten Prozessmodellen fordern. Abschnitt 3.4 beschreibt verschiedene Möglichkeiten, wie die Struktur von Prozessmodellen ggf. vereinfacht werden kann, ohne dabei ihr Ausführungsverhalten zu verändern. Solche Vereinfachungsregeln sind sinnvoll, um aufwendige Sonderfallbehandlungen bei der Konfiguration bzw. Änderung von Prozessmodellen zu vermeiden. Kapitel 3 schließt mit einer Diskussion in Abschnitt 3.5 und einer Zusammenfassung in Abschnitt 3.6.

3.1 Motivation

Ein Prozess beschreibt, welche Aktivitäten zur Erreichung eines bestimmten Ziels durchgeführt werden müssen. Typischerweise kann die Reihenfolge dieser Aktivitäten durch Modellierung von Sequenzen, parallelen oder bedingten Verzweigungen sowie Schleifen explizit vorgegeben werden. Die Erfassung und Abbildung solcher Abfolgen von Aktivitäten erfolgt durch die *Prozessmodellierung*. Prozessmodelle werden üblicherweise als gerichtete Graphen dargestellt. Zu diesem Zweck existieren verschiedene graphische Beschreibungssprachen für Prozessmodelle. An dieser Stelle seien beispielhaft ereignisgesteuerte Prozessketten (EPK) [Sch98], Business Process Modelling Notation (BPMN) [BPM09], UML-Aktivitätsdiagramme [Par98, Esh02, UML09] und Petri-Netze [Pet81, vdA98] genannt. Abbildung 3.1 zeigt für ein einfaches Prozessmodell beispielhaft, wie dieses mit verschiedenen Prozess-Metamodellen definiert werden kann.

Abbildung 3.1: Beispiele für Prozessmodelle als BPMN Modell a), Petri-Netz b), UML Aktivitätsdiagramm c) und EPK d)



Existierende Prozess-Metamodelle sind für unterschiedliche Anwendungszwecke entwickelt worden. Zum Beispiel liegt der Schwerpunkt der EPK-Modellierung auf der Dokumentation und Visualisierung der Prozessmodelle. UML-Modelle werden hingegen für die systemunabhängige Beschreibung von Software verwendet. Petri-Netze werden in erster Linie zur Beschreibung dynamischer und nebenläufiger Prozesse eingesetzt.

Unabhängig vom Anwendungszweck sind mit all diesen Prozess-Metamodellen auch Prozessvarianten abzubilden. Ein umfassendes Variantenmanagement-Konzept sollte somit prinzipiell auf all diese Prozess-Metamodelle rückführbar sein. Auf Grund der Forderung nach einer generischen Lösung verzichten wir auf die Wahl eines speziellen Prozess-Metamodells in Provop. Damit wir aber die in Provop entwickelten Konzepte in dieser Arbeit beschreiben und dokumentieren können, ist eine präzise Notation und Semantik von Prozesselementen, also ein Prozess-Metamodell, unabdingbar. Um nicht ein eigenes, neues Prozess-Metamodell oder eine zu Metamodell-spezifische Lösung zu entwickeln, setzen wir auf (mehreren) bekannten Prozess-Metamodellen auf. Diese besitzen allerdings sehr umfassende Spezifikationen, die eine Vielzahl von einzelnen abbildbaren Prozesselementen und -strukturen beschreiben. Eine solche Komplexität ist jedoch meist nicht erforderlich [MR08]. Zur Beschreibung des Provop-Ansatzes ist deshalb nur eine geringe Menge an Prozesselementen und -strukturen relevant. Diese werden im Rahmen dieser Arbeit identifiziert und spezifiziert, und in einer allgemein anerkannten Form verwendet. Die Notation und Anordnung der Prozesselemente entnehmen wir bekannten Metamodellen (z.B. EPK und BPMN). Die Semantik der Prozesselemente basiert auf bekannten Mustern (engl. Patterns) des Workflow-Managements [RAvdAM06, RAE04].

Um auf andere Prozess-Metamodelle möglichst einfach übertragbar zu sein, nehmen wir in Provop nur minimale Einschränkungen an den möglichen Prozessstrukturen vor. Zum Beispiel fordern wir keine Blockstrukturierung [Rei00] und erlauben somit das „Verschränken“ von Verzweigungs- und Zusammenführungskonstrukten.

3.2 Prozess-Metamodell

In graphbasierten Modellierungssprachen werden verschiedene Knoten- und Kantentypen verwendet, um komplexe Graphen zu bilden. Bei der Prozessmodellierung werden diese Knoten- und Kantentypen als *Prozesselemente* bezeichnet. Dementsprechend stellen die Graphen dann *Prozessmodelle* dar. Im Folgenden definieren wir zunächst den Begriff des Prozessmodells. Anschließend stellen wir die einzelnen Prozesselemente im Detail vor und geben eine formale Definition für Prozessmodelle an.

3.2.1 Prozessmodell

Abbildung 3.2 zeigt ein Beispiel für ein Prozessmodell aus der Domäne „Änderungsmanagement“ (vgl. Abschnitt 2.1). Das dargestellte Prozessmodell besteht aus einer Menge von *Aktivitäten*, die ausgeführt werden müssen, um einen Änderungsantrag (z.B. Antrag auf Änderung einer Fahrzeugkomponente) zu bearbeiten. *Kontrollfluss-Kanten* und *Strukturknoten* (z.B. `ANDSplit`- und `XORJoin`-Knoten) zeigen an, in welcher Reihenfolge und unter welchen Bedingungen (z.B. parallel, alternativ oder sequentiell) die Aktivitäten ausgeführt werden sollen. Dies entspricht dem *Kontrollfluss* des Prozesses. Neben dem Kontrollfluss zeigt Abbildung 3.2 auch den *Datenfluss* des Prozesses. Die Aktivität `Änderung beantragen` etwa schreibt das Datenobjekt `Änderungsantrag`, das anschließend von den Aktivitäten `Beauftragung Stellungnahmen` und `Antrag ablegen` gelesen wird.

Ein solches Prozessmodell, das für eine spezifische Domäne (z.B. Änderungsmanagement oder Werkstatt) modelliert wird und einen generischen Charakter hat, bezeichnen wir als *Prozesstyp*.

3.2.2 Prozesselemente

Neben der Notation von Prozesselementen, d.h. ihrer Darstellung im Prozessmodell, beschreiben wir im Folgenden auch, welche Regeln für ihre Anordnung gelten. Wir verzichten in dieser Arbeit auf eine formale Beschreibung der Semantik der verschiedenen Prozesselemente und verweisen stattdessen auf entsprechende Spezifikationen [BPM09, DM08, RAvdAM06, RAE04, Sch98].

3.2.2.1 Prozesselemente für azyklische Graphen

Im Folgenden gehen wir zunächst von azyklischen Graphen aus. Rücksprünge und Schleifen werden an dieser Stelle noch nicht behandelt, später aber eingeführt.

Start- und Endknoten

Beschreibung. In Anlehnung an [BPM09] markieren wir die Prozesssein- und Prozessausgänge jeweils durch einen speziellen Knotentyp: Die *Startknoten* stellen die Ausgangsknoten eines Prozessmodells dar. Sie besitzen keine Vorgängerknoten bzw. eingehende Kontrollfluss-Kanten. Um das Ende eines Ausführungspfades zu markieren, verwenden wir *Endknoten*. Sie besitzen weder Nachfolgerknoten noch ausgehende Kontrollfluss-Kanten. Ein Prozessmodell hat mindestens einen Start- und mindestens einen Endknoten.[1]

[1] Abhängig vom zugrunde liegenden Prozess-Metamodell kann die Anzahl der Start- und Endknoten in einem Prozessmodell jeweils auch auf exakt einen Start- bzw. Endknoten begrenzt sein. In Provop nehmen wir hier jedoch keine entsprechende Einschränkung vor.

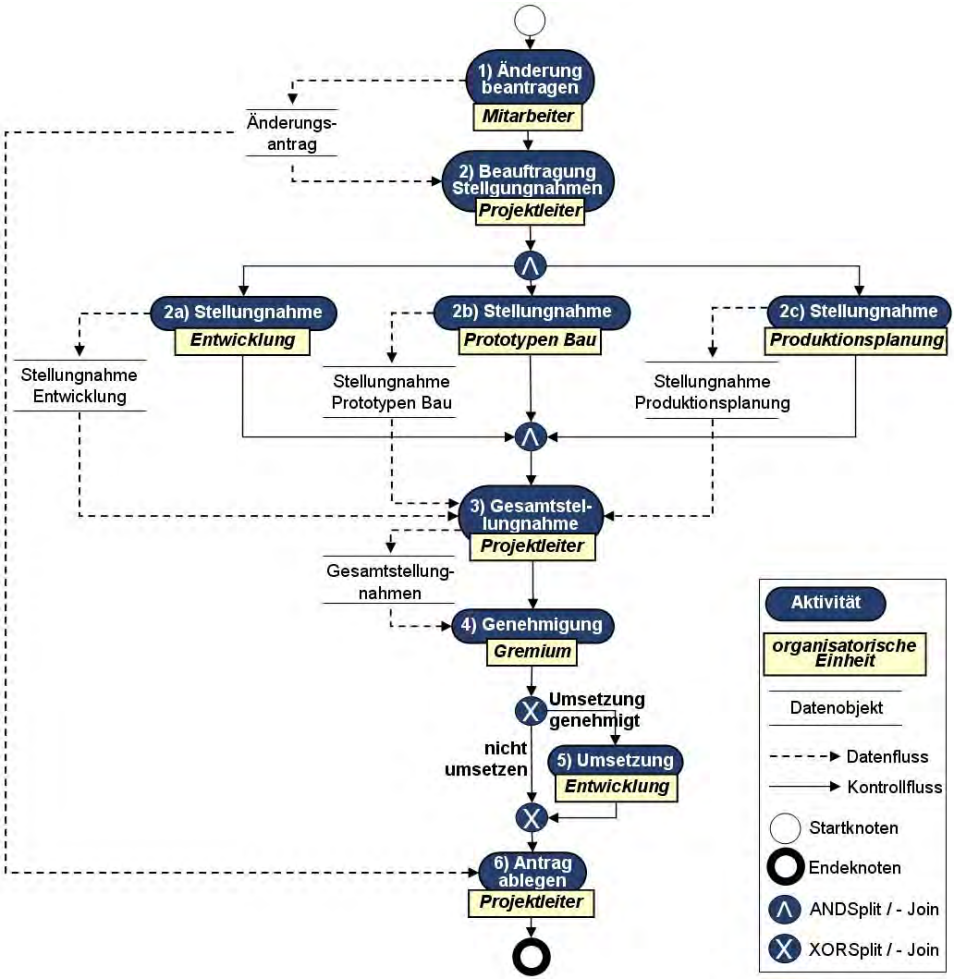


Abbildung 3.2: Prozessmodell aus der Domäne Änderungsmanagement

Notation. Tabelle 3.1 zeigt die in dieser Arbeit verwendeten Symbole für Start- und Endknoten. Im folgenden wird aber auf die Darstellung dieser Knoten aus Gründen der kompakteren Darstellbarkeit unserer Beispiele meist verzichtet. Ihre Betrachtung ist allerdings für bestimmte Algorithmen auf Graphstrukturen relevant. Sie sind daher Teil des Provop-Metamodells.

Tabelle 3.1: Notation für Start- a) und Endknoten b)

	Symbol	Beschreibung
a)	○	Startknoten
b)	◎	Endknoten

Semantik. Die Ausführung eines Prozessmodells beginnt zeitgleich an allen definierten Startknoten. An ihnen wird dann direkt zu den nachfolgenden Prozesselementen durchgeschaltet (vgl. Pattern *All Start Places* in [DM08]). Wird während der Ausführung ein Endknoten erreicht, terminiert der zugehörige Ausführungspfad. Entsprechend terminiert das gesamte Prozessmodell sobald alle Ausführungspfade terminiert sind (vgl. Pattern *Implicit Termination* in [RAvdAM06]). Das heißt, es werden anschließend keine weiteren Aktivitäten ausgeführt.

Aktivitäten

Beschreibung. Eine Aktivität stellt einen Arbeitsschritt eines Prozessmodells dar. Üblicherweise besitzen Aktivitäten immer genau einen Vorgänger- und einen Nachfolgerknoten (vgl. [Sch98, BPM09]). Komplexe Aktivitäten, welche eine Zusammensetzung aus mehreren einzelnen Aktivitäten oder Aktionen darstellen (sog. Subprozesse), werden in dieser Arbeit nicht betrachtet.[2] Neben „normalen" Aktivitäten können auch „leere" Knoten modelliert werden. Zur Anordnung dieser Knoten gelten die gleichen Regeln wie für Aktivitäten.

Mit Hilfe ihrer Attribute können wir eine Aktivität detailliert beschrieben. Beispielhaft seien an dieser Stelle der Bezeichner der Aktivität, die maximale Dauer ihrer Ausführung und ihre Bearbeiterrollen (bei interaktiven Aktivitäten) genannt. In Provop sind zudem benutzerdefinierte Attribute möglich, d.h. einer Aktivität können beliebige neue Attribute zwecks ihrer Beschreibung hinzugefügt werden.

Notation. In dieser Arbeit werden Aktivitäten in Anlehnung an [Sch98, BPM09] als Rechteck mit abgerundeten Ecken dargestellt (vgl. Tabelle 3.2a). Die optionale Bezeichnung einer Aktivität entspricht, falls vorhanden, der Beschriftung innerhalb des Rechtecks. Leere Knoten dienen der Strukturierung von Modellen und werden, wie Tabelle 3.2b zeigt, ausgegraut und mit gestricheltem Rahmen dargestellt.

Tabelle 3.2: Aktivität mit optionaler Bezeichnung

	Symbol	Beschreibung
a)	[Name] /Attribut	Aktivität mit optionaler Bezeichnung und optionalem Attribut
b)		Leerer Knoten

Semantik. Eine Aktivität befindet sich während der Ausführung eines Prozessmodells in genau einem von fünf Zuständen. Abbildungen 3.3a bis 3.3e zeigen die verwendeten Symbole zur Darstellung dieser Zustände. Die Zustandsübergänge zeigt Abbildung 3.3f. Eine Aktivität befindet sich initial im Zustand `not activated`. Dieser Zustand gibt an, dass die Aktivität im Augenblick (noch) nicht ausgeführt werden kann. Dies ist erst möglich, wenn die Aktivität in den Zustand `activated` versetzt wird.[3] Anschließend kann die Aktivität ausgeführt werden. Sie befindet sich dann im Zustand `running`. Ist die Ausführung abgeschlossen, befindet sich die Aktivität im Endzustand `finished`. Soll eine Aktivität nicht ausgeführt werden, kann sie direkt in den Endzustand `skipped` versetzt werden. Wir werden später noch zeigen, dass zur erneuten Ausführung einer Aktivität, z.B. innerhalb einer Schleife, das Zurücksetzen des Zustands von einem Endzustand (d.h. `finished` oder `skipped`) zum Startzustand `not activated` erforderlich ist.

Leere Knoten werden bei der Ausführung ignoriert. Der Zustand `true_signaled` oder `false_signaled` der eingehenden Kante wird von der ausgehenden Kante übernommen.

[2] Dies stellt jedoch keine Einschränkung dar, da die Provop-Konzepte vom Prinzip her auf Sub-Prozesse übertragen werden können.

[3] Zustandsübergang werden von den Zuständen der eingehenden Kanten einer Aktivität bestimmt. Wir werden später noch zeigen, dass Kontrollfluss-Kanten die Zustände `true_signaled`, `false_signaled` und `not_signaled` annehmen können.

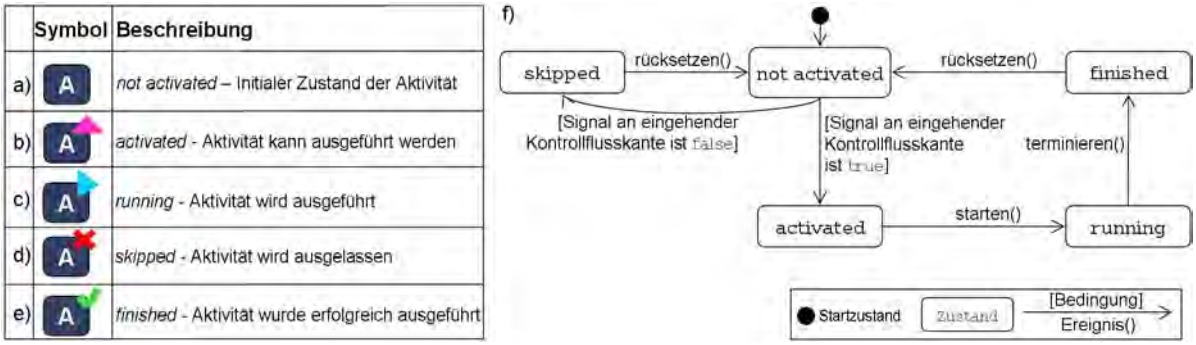


Abbildung 3.3: Zustände von Aktivitäten zur Laufzeit und mögliche Zustandsübergänge

Kontrollfluss-Kanten

Beschreibung und Notation. Der Kontrollfluss beschreibt die Ausführungsreihenfolge der Aktivitäten eines Prozesses. Er wird über gerichtete Kanten, sog. *Kontrollfluss-Kanten*, dargestellt (vgl. [Sch98, BPM09, UML09, vdA98]). Kontrollfluss-Kanten werden zwischen je zwei Knoten (z.B. Aktivitäten) eines Prozesses gezogen. Dabei wird die Quelle einer Kontrollfluss-Kante als *Quell-* und ihr Ziel als *Zielknoten* bezeichnet. Für diese Knoten stellen die Kontrollfluss-Kanten entsprechend aus- bzw. eingehende Kontrollfluss-Kanten dar. Beispiel 3.1 erläutert einige allgemein als zulässig und unzulässig erachtete Anordnungen von Kontrollfluss-Kanten.

Beispiel 3.1 (Zulässige und unzulässige Anordnung von Kontrollfluss-Kanten)
Abbildung 3.4a ist zulässig und bedeutet, dass Aktivitäten `A` *und* `B` *sequentiell ausgeführt werden. Die Beispiele aus Abbildung 3.4b bis 3.4e sind in Provop unzulässig. Die beabsichtigte Struktur des Prozessmodells kann mit anderen zulässigen Prozesselementen, wie Schleifen und Strukturknoten, jeweils korrekt abgebildet werden. Solche Einschränkungen sind nicht in allen Formalismen gegeben. So erlauben z.B. Aktivitätennetze mehrere ausgehende Kanten für Aktivitäten. Die Konzepte sind allerdings aufeinander abbildabr, daher schränken wir die Übertragbarkeit des Provop-Ansatzes auf Aktivitätennetze an dieser Stelle nicht ein.*

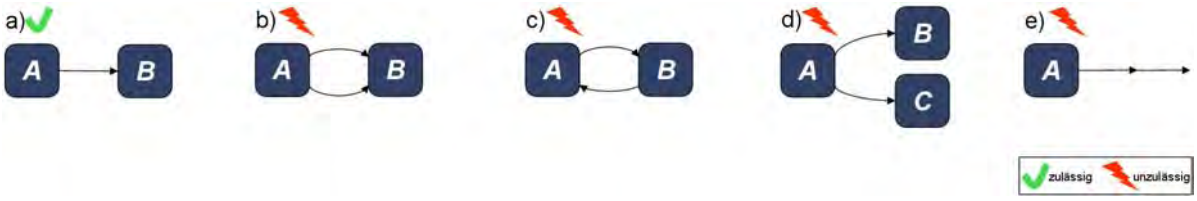


Abbildung 3.4: Beispiele für zulässige und unzulässige Kontrollfluss-Konstrukte

Eine Kontrollfluss-Kante kann mit einer Transitionsbedingung annotiert werden. Wir verwenden diese *Kantenbedingung*, um zur Laufzeit alternative Ausführungspfade auswählen zu können. Wie wir später vertiefen werden, verwenden wir in Provop Kantenbedingungen ausschließlich in Kombination mit Struktur- und Schleifenknoten.

Semantik. Eine Kontrollfluss-Kante kann die Zustände `true_signaled`, `false_signaled` und `not_signaled` annehmen. Sie ist initial im Zustand `not_signaled`. Abbildungen 3.5a bis 3.5e zeigen die verwendeten Symbole zur Darstellung der Zustände. Darauf aufbauend zeigt Abbildung 3.5d die möglichen Zustandsübergänge von Kontrollfluss-Kanten: Die Entscheidung, ob eine Kontrollfluss-Kante in den Zustand `true_signaled` oder `false_signaled` übergeht,

wird nach Auswertung der Kantenbedingung bzw. abhängig vom Zustand ihres Quellknotens getroffen: Wird eine Vorgängeraktivität beendet (d.h. sie befindet sich im Zustand `finished`), geht die Kontrollfluss-Kante in den Zustand `true_signaled` über. In diesem Fall wird der Zustand der direkt folgenden Aktivität auf `activated` gesetzt (vgl. Abbildung 3.6a). Wird die vorangegangene Aktivität ausgelassen (d.h. sie befindet sich im Zustand `skipped`), geht die ausgehende Kontrollfluss-Kante in den Zustand `false_signaled` über und wir nehmen eine sog. „Dead Path Elimination" vor [LR99]: Aufgrund des Zustands `false_signaled` wird der Zielknoten der Kante in den Zustand `skipped` versetzt, wodurch wiederum die ausgehende Kante dieses Knotens in den Zustand `false_signaled` übergeht. Eine „Dead Path Elimination" führt also dazu, dass alle Aktivitäten in den Zustand `skipped` versetzt werden, die im weiteren Verlauf nicht mehr ausgeführt werden können (vgl. Abbildung 3.6b).

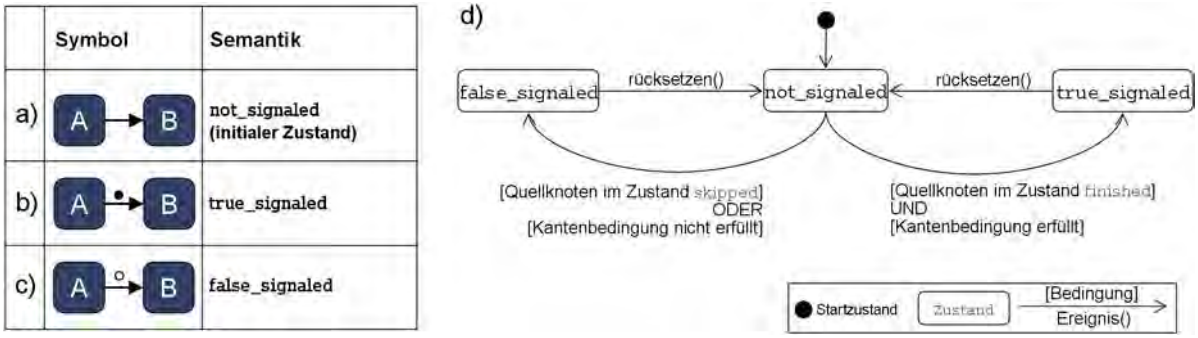


Abbildung 3.5: Zustände an Kontrollfluss-Kanten

Die Beschreibung möglicher Zustandsübergänge von Kanten und die Durchführung einer *Dead Path Elimination* sind auch im Kontext von Verzweigungen im Prozessmodell relevant, z.B. dann, wenn aufgrund der Verzweigungsbedingung ein alternativer Ausführungspfad ausgelassen werden soll. Wir werden die Beschreibung dieser Vorgehensweise an späterer Stelle nochmals aufgreifen.

Abbildung 3.6: Zusammenhang der Zustände von Aktivitäten und Kanten

Strukturknoten

Beschreibung. In Anlehnung an [Sch98] verwenden wir zur Beschreibung von parallelen und bedingten Verzweigungen logische Konnektoren – sog. *Strukturknoten*. Diese können eine Verzweigung (engl. *Split*) oder eine Zusammenführung (engl. *Join*) von Ausführungspfaden darstellen. Ein Split-Knoten besitzt exakt eine eingehende und mindestens eine ausgehende Kontrollfluss-Kante.[4] Umgekehrt besitzt ein Join-Knoten mindestens eine eingehende und exakt eine ausgehende Kante. Für jeden Split- bzw. Join-Knoten wird ein Strukturknotentyp angegeben. Dieser liegt in der Menge der logischen Operatoren `AND`, `OR` und `XOR`. Der logische Operator beschreibt entsprechend die Verzweigungs- bzw. Zusammenführungslogik, d.h. parallel (AND), alternativ (OR) und echt alternativ (XOR).

[4] Es ist somit auch zulässig nur jeweils eine aus- bzw. eingehende Kante für Split bzw. Join Knoten zu modellieren. Allerdings können solche Konstrukte meist einfacher modelliert werden. Entsprechende Vereinfachungsregeln werden in Abschnitt 3.4 vorgestellt.

Notation. Die Split- und Join-Knoten werden als Kreis oder Oval dargestellt. Der jeweilige Typ bzw. logische Operator (`AND`, `OR` und `XOR`) eines Split- oder Join-Knotens wird durch ein entsprechendes Symbol innerhalb des Strukturknotens angezeigt (vgl. Tabelle 3.3). Nicht alle Konstrukte, die durch Verwendung von Split- und Join-Knoten möglich sind, sind in Provop auch zulässig. Beispiel 3.2 zeigt exemplarisch, welche Konstrukte wir als unzulässig bewerten.

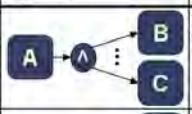
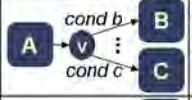
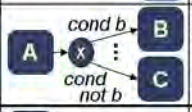
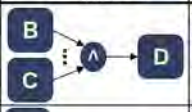
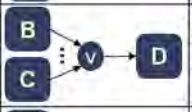
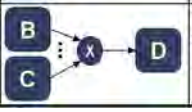
	Symbol	Beschreibung	Semantik
a)		Parallele Verzweigung „AND - Split"	Alle nachfolgenden Pfade werden ausgeführt.
b)		Bedingte Verzweigung „OR - Split"	Mindestens einer der nachfolgenden Pfade wird ausgeführt: Aktivität B wird aktiviert, wenn Kantenbedingung „cond b" wahr ist. Aktivität C wird aktiviert, wenn Kantenbedingung „cond c" wahr ist.
c)		Exklusive Verzweigung „XOR - Split"	Genau einer der nachfolgenden Pfade wird ausgeführt: Aktivität B wird aktiviert, wenn Kantenbedingung „cond b" wahr ist. Aktivität C wird aktiviert, wenn Kantenbedingung „cond not b" wahr ist.
d)		Parallele Synchronisation „AND - Join"	Es wird auf einen Zustand ungleich **not_signaled** aller eingehenden Kanten gewartet. Damit die ausgehende Kante des ANDJoin-Knotens in den Zustand **true_signaled** übergeht, müssen alle eingehenden Kanten im Zustand **true_signaled** sein.
e)		Bedingte Synchronisation „OR - Join"	Es wird auf einen Zustand ungleich **not_signaled** aller eingehenden Kanten gewartet. Damit die ausgehende Kante des ORJoin-Knotens in den Zustand **true_signaled** übergeht, muss mindestens eine der eingehenden Kanten im Zustand **true_signaled** sein.
f)		Exklusive Synchronisation „XOR - Join"	Es wird auf einen Zustand ungleich **not_signaled** aller eingehenden Kanten gewartet. Damit die ausgehende Kante des XORJoin-Knotens in den Zustand **true_signaled** übergeht, muss genau eine der eingehenden Kanten im Zustand **true_signaled** sein.

Tabelle 3.3: Notation und Semantik von Split- und Join-Knoten

Beispiel 3.2 (Unzulässige Konstrukte mit Strukturknoten) *Das in Abbildung 3.7a dargestellte Konstrukt ist nicht zulässig, da ein Split-Knoten in Provop nicht gleichzeitig einen Join-Knoten darstellen kann. Da wir zunächst von azyklischen Graphen ausgehen, ist der Zyklus in Abbildung 3.7b nicht zulässig. Wir werden in Abschnitt 3.2.2.2 diskutieren, wie wir zyklische Graphen mit Hilfe expliziter Schleifenkonstrukte in einem Prozessmodell korrekt abbilden können.*

Abbildung 3.7: Beispiele für die ungültige Verwendung der Split- und Join-Knoten

Eine Frage bei der Verwendung von Split- und Join-Knoten ist, ob eine regelmäßige Blockstrukturierung gefordert ist. In einem blockstrukturierten Prozessmodell werden Kontrollflussstrukturen (z.B. Sequenzen, Verzweigungen) in Blöcke eingeteilt, die eindeutige Ein- bzw. Ausgang besitzen [Rei00]. Überlappungen von Blöcken sind nicht erlaubt, allerdings können sie beliebig verschachtelt werden.

Beispiel 3.3 (Blockstrukturierung) *Abbildung 3.8a zeigt ein Beispiel mit drei verschachtelten Blöcken. Block 1 umfasst das gesamte Prozessmodell. Block 2 beschreibt einen* `AND`*-Block innerhalb des Prozessmodells, in welchem Block 3 (ein* `OR`*-Block) enthalten ist. Das Prozessmodell in Abbildung 3.8b besitzt die gleiche Semantik wie das Modell aus Abbildung 3.8a, allerdings ist hier durch gemeinsame Zusammenführung aller aufgesplitteten Ausführungspfade keine „reine" Blockstrukturierung gegeben (diese lässt sich im vorliegenden Fall allerdings einfach herstellen).*

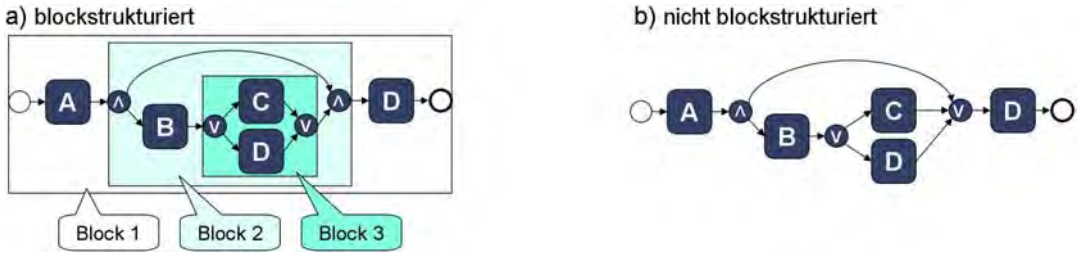


Abbildung 3.8: Blockstrukturiertes a) und nicht blockstrukturiertes Prozessmodell b)

Der Vorteil einer Blockstrukturierung ist die eindeutige Zuordnung von Split- und Join-Knoten. Dies vereinfacht die Analyse des Prozessmodells, z.B. zur Erkennung von Zyklen oder anderen inkorrekten Strukturen [Rei00]. Trotz dieser Vorteile fordern wir in Provop jedoch keine Blockstrukturierung. Der Grund dieser Entwurfsentscheidung ist, dass wir weniger restriktiv sein wollen, um somit die Übertragbarkeit unseres Ansatzes auf verschiedene Prozess-Metamodelle (d.h. auch auf nicht blockstrukturierte) zu erleichtern.

Semantik von Verzweigungen. Generell ermöglichen Split-Knoten die parallele bzw. (echte) alternative Ausführung von Aktivitäten. Dazu kann ein Ausführungspfad an einem Split-Knoten in mehrere Ausführungspfade aufgespalten werden. Um festzulegen welche der ausgehenden Kanten des Split-Knotens ausgeführt werden sollen, verwenden wir die oben genannten logischen Operatoren `AND`, `OR` und `XOR`. Mit jedem dieser Operatoren ist ein anderes Ausführungsverhalten verknüpft. Für alle Split-Knoten gilt jedoch, dass wenn die eingehende Kante im Zustand `false_signaled` ist, alle ausgehenden Kanten des Split-Knotens in den Zustand `false_signaled` übergehen.

Ein `ANDSplit` führt zu einer Parallelität im Prozessmodell und ermöglicht somit die parallele Ausführung von Aktivitäten. Ist der Zustand der eingehenden Kante des `ANDSplit`-Knotens `true_signaled`, werden alle ausgehenden Kontrollfluss-Kanten in den Zustand `true_signaled` versetzt und die Pfade werden parallel ausgeführt. Diese Semantik entspricht dem in [RAvdAM06] beschriebenen Ausführungspattern *Parallel Split*. Abbildung 3.9 zeigt die Semantik des `ANDSplit`-Knotens beispielhaft. Das Konstrukt aus Abbildung 3.9c stellt keine gültige Semantik des `ANDSplit`-Knotens dar.

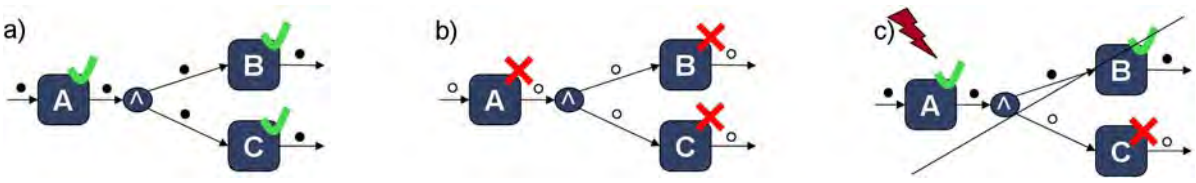


Abbildung 3.9: Semantik von `ANDSplit`-Knoten

Bei einem `ORSplit` entstehen alternative Pfade, deren Ausführung jeweils von einer Bedingung abhängig ist (vgl. Pattern *Multi Choice* [RAvdAM06]). Ist der Zustand der eingehenden

Kontrollfluss-Kante `true_signaled`, können nach Auswertung der Transitions- bzw. Kantenbedingungen die Zustände der ausgehenden Kontrollfluss-Kanten entsprechend festgelegt werden. Abbildung 3.10 zeigt die Semantik des `ORSplit`-Knotens beispielhaft.

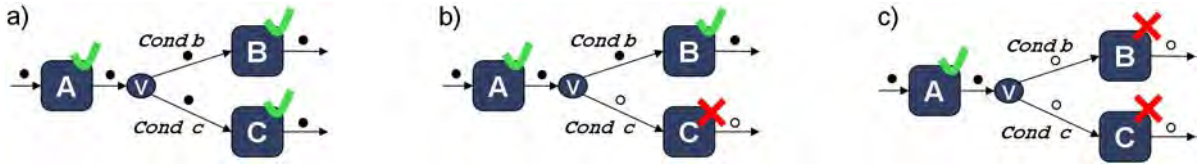


Abbildung 3.10: Semantik von `ORSplit`-Knoten

Ein `XOR-Split` ermöglicht die Definition von echt alternativen Ausführungspfaden. Das heißt, exakt eine ausgehende Kante des `XORSplit`-Knotens wird in den Zustand `true_signaled` versetzt (vgl. Pattern *Exclusive Choice* in [RAvdAM06]). Welche Kante dies ist, wird bei der Auswertung der Kantenbedingungen entschieden. Abbildung 3.11 zeigt die Semantik des `XORSplit`-Knotens beispielhaft. Die Konstrukte aus Abbildung 3.11a und 3.11c stellen keine gültige Semantik des `XORJoins` dar.

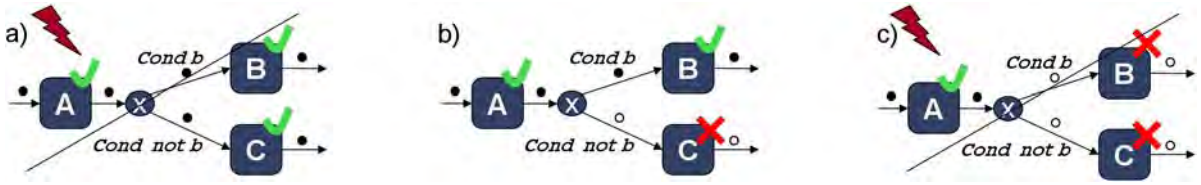


Abbildung 3.11: Semantik von `XORSplit`-Knoten

Semantik von Synchronisationen. An einem Join-Knoten wird auf den Zustand `true_signaled` oder `false_signaled` aller eingehenden Kontrollfluss-Kanten gewartet. Anschließend werden entsprechend der Semantik des gewählten Join-Operators (`AND`, `OR` oder `XOR`) die Zustände der ausgehenden Kanten festgelegt. Eine Zusammenführung an einem Join-Knoten schlägt generell fehl, wenn alle eingehenden Kanten im Zustand `false_signaled` sind. In diesem Fall wird auch die ausgehende Kante eines Join-Knotens in den Zustand `false_signaled` versetzt. Dies führt dann zu einer *Dead Path Elimination*.

Damit die ausgehende Kante eines `ANDJoin`-Knotens in den Zustand `true_signaled` übergeht, müssen alle eingehenden Kanten im Zustand `true_signaled` sein (vgl. Abbildung 3.12). Ist dies nicht der Fall, d.h. ist mindestens eine eingehende Kante im Zustand `false_signaled`, wird die ausgehende Kante ebenfalls in den Zustand `false_signaled` überführt (vgl. Pattern *Synchronization* [RAvdAM06]).

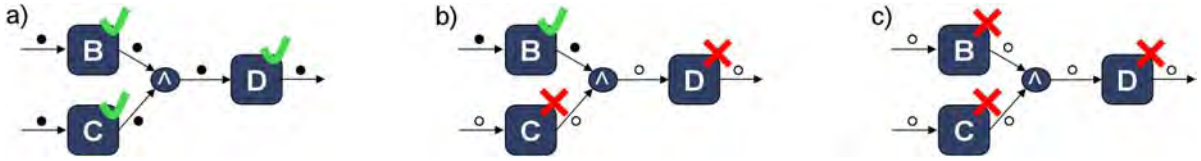


Abbildung 3.12: Semantik von `ANDJoin`-Knoten

Die Semantik von `ORJoin`-Knoten wird in der Literatur stark diskutiert [WEvdAtH05, MNN05, DGHW07]. Neben dem beschriebenen Ansatz des Wartens auf einen Zustand aller eingehenden Kanten ungleich dem Startzustand (d.h. `not_signaled`) gibt es auch Ansätze, in denen jeder Zustandsübergang einer eingehenden Kante des `ORJoins` in den Zustand `true_signaled`

zu einer Aktivierung nachfolgender Aktivitäten führt (vgl. Beispiel 3.4). Dies wird als *Mehrfachinstanziierung* bezeichnet und wird in tokenbasierten Metamodellen, z.B. Petri-Netze [vdA98] und YAWL [vdAtH05], eingesetzt.[5] Um eine Mehrfachinstanziierung in tokenbasierten Prozess-Metamodellen zu vermeiden, ist die Blockierung oder der Abbruch aller noch nicht beendeter Ausführungspfade und zwar nachdem die Synchronisationsbedingung des `ORJoins` erfüllt wurde (vgl. Pattern *Blocking Partial Join* und *Cancelling Partial Join* [RAvdAM06]).

Beispiel 3.4 (Mehrfachinstanziierung) *In Abbildung 3.13a wird nach Ausführung der Aktivität* `B` *direkt Aktivität* `D` *aktiviert, ohne dass Aktivität* `C` *zuvor beendet worden ist. Ohne eine „Blockierung" des zweiten Pfades ist es daher möglich, dass nach Ausführung von Aktivität* `C` *die Aktivität* `D` *ein zweites Mal aktiviert und ausgeführt wird. Im Gegensatz dazu wird in Abbildung 3.13b die Aktivität* `D` *erst dann ausgeführt, wenn an allen eingehenden Kanten des* `ORJoins` *in einem Zustand ungleich* `not_signaled` *sind und mindestens eine dieser Kanten im Zustand* `true_signaled` *ist.*

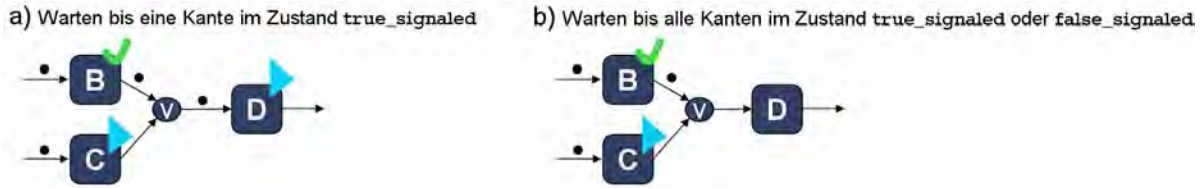


Abbildung 3.13: Warten auf Zustandsübergang der eingehenden Kanten eines `ORJoins`

Die beschriebene Semantik ist für den Prozessmodellierer schwer nachvollziehbar und widerspricht dem Prinzip einer (zeitlichen) Synchronisation von Ausführungspfaden an einem Join-Knoten. In Provop gehen wir deshalb davon aus, dass an einem `ORJoin` auf alle eingehenden Kanten gewartet wird. Diese Forderung ist jedoch nicht unbedingt trivial umzusetzen. Die Herausforderung besteht darin, dass wir am `ORJoin`-Knoten erkennen müssen, welcher Pfad noch ausgeführt wird, welcher bereits beendet worden ist und welcher nicht ausgeführt werden soll.[6] In Provop gehen wir zur Lösung dieses Problems von den bereits vorgestellten Kantenzuständen aus. Diese helfen uns zwischen den drei Fällen (d.h. wird ausgeführt, beendet und ausgelassen) zu unterscheiden. Entsprechend warten wir an einem `ORJoin` auf das Erreichen des Zustands `true_signaled` oder `false_signaled` aller eingehenden Kanten. Ist der Zustand von mindestens einer eingehenden Kante `true_signaled`, wird die ausgehende Kante in den Zustand `true_signaled` überführt (vgl. Pattern *Acyclic Synchronizing Merge* [RAvdAM06]). Beispiele hierzu zeigt Abbildung 3.14.

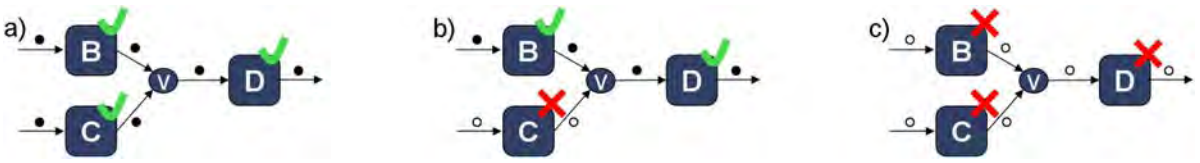


Abbildung 3.14: Semantik von `ORJoin`-Knoten

Eine ähnliche Diskussion, wie bzgl. der Semantik des `ORJoin`, kann für `XORJoin`-Knoten geführt werden. Analog zum `ORJoin` fordern wir auch hier, dass keine Mehrfachinstanziierung

[5] In tokenbasierten Prozess-Metamodellen zeigen Token bzw. Marken an, an welcher Position sich die Prozessausführung aktuell befindet, d.h. welche Aktivität gerade ausgeführt wird. Es gibt keine Zustände für Aktivitäten oder Kanten, wie bei der *Dead Path Elimination*.

[6] In diesem Zusammenhang wird der `ORJoin` auch als *nichtlokaler Konnektor* bezeichnet.

erfolgt. Wir gehen außerdem davon aus, dass zur Identifizierung der ausgeführten oder ausgelassenen Pfade entsprechende Zustandsinformationen von Aktivitäten und Kanten verwendet werden können. Basierend auf diesen Vorgaben erfordert ein `XORJoin`-Knoten, wie in Tabelle 3.3f dargestellt, dass exakt eine eingehende Kante im Zustand `true_signaled` ist, damit die ausgehende Kante ebenfalls in den Zustand `true_signaled` überführt werden kann (vgl. Pattern *Simple Merge* [RAvdAM06]). Abbildungen 3.15a bis 3.15c zeigen die möglichen Fälle einer Zusammenführung von zwei Ausführungspfaden mit entsprechenden Kantenzuständen an einem `XORJoin`.

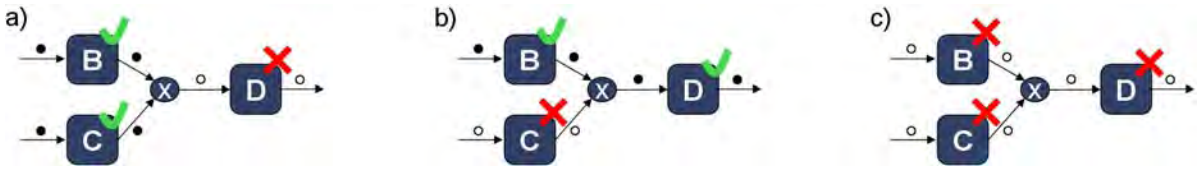


Abbildung 3.15: Semantik von `XORJoin`-Knoten

Datenfluss

Beschreibung und Semantik. Der Datenfluss beschreibt die Verwendung und Weitergabe von Daten zwischen den Aktivitäten eines Prozessmodells. In Anlehnung an das Pattern *Case Data* in [RAE04] modellieren wir den Datenfluss mit Hilfe von Datenobjekten, die in einem globalen Datencontainer liegen. Global bedeutet, dass sie von jeder Aktivität eines Prozessmodells gelesen bzw. geschrieben werden können. Weitergehende Festlegungen, wie die Einschränkung des Sichtbarkeitsbereichs von globalen Variablen [RAE04], werden an dieser Stelle ausgeklammert, da sie nicht im Fokus dieser Arbeit liegen.

Notation. Datenfluss-Kanten werden in Provop mittels gestrichelter Pfeile dargestellt. Ein Pfeil von einer Aktivität zu einem Datenobjekt visualisiert einen Schreibzugriff auf das des Datenobjekt. Wir bezeichnen diese Kante als *schreibende Datenfluss-Kante*. Ein Pfeil von einem Datenobjekt zu einer Aktivität beschreibt das Lesen des Objektes. Entsprechend bezeichnen wir diese Kante als *lesende Datenfluss-Kante*. Ein Datenobjekt wird als *Satellitenobjekt* der (lesend oder schreibend) zugreifenden Aktivität im Prozessmodell dargestellt. Der Bezeichner des Datenobjekts wird von zwei horizontalen Balken eingerahmt (vgl. Abbildung 3.16).

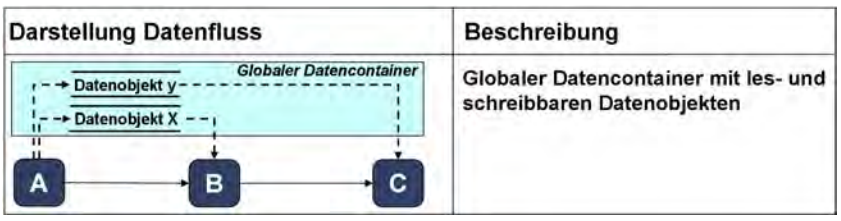


Abbildung 3.16: Darstellung der Datenweitergabe zwischen Aktivitäten

Identifikation von Elementen im Prozessmodell

Prozesselemente werden in einer Datenbank gespeichert und besitzen eine eindeutige Identifikationsnummer (im Folgenden als Repository-ID bezeichnet). Bei der Verwendung eines Elementes in einem Prozessmodell wird zusätzlich eine eindeutige Prozesselement-ID vergeben (kurz PE-ID oder ID), welche sich von der Repository-ID (logisch) unterscheidet. Auf diese Weise kann ein Element mehrfach in einem Prozessmodell verwendet werden und ist dennoch eindeutig identifizierbar.

3.2.2.2 Prozesselemente für zyklische Graphen

Zur wiederholten Ausführung von Aktivitäten können wir in zyklischen Graphen Rücksprünge und Schleifen modellieren. Abbildung 3.17 zeigt, wie die wiederholte Ausführung von Aktivitäten mit Hilfe von Strukturknoten darstellbar ist. Das Problem dieses Ansatzes besteht darin, dass die Semantik des Rücksetzens der Zustände von wiederholt auszuführenden Prozesselementen in diesem Fall nicht klar ist. Weiter ist unklar, ob Aktivität `D` nach jedem Schleifendurchlauf erneut aktiviert wird. Aufgrund dieser Unklarheit verwenden wir in Provop explizite *Schleifen* mit eindeutiger Semantik.

Abbildung 3.17: Beispiele für Schleifen in einem Prozessmodell mit Hilfe von `ORSplit-/ ORJoin`-Knoten a) und `XORSplit-/ XORJoin`-Knoten b)

Eine Schleife besteht aus folgenden Elementen: Dem *Schleifenrumpf*, d.h. den wiederholt auszuführenden Aktivitäten, und einer *Schleifenbedingung*, welche über die nächste Iteration des Schleifenrumpfes bzw. den Abbruch der Schleife entscheidet.

Es gibt unterschiedliche Schleifentypen. Diese unterscheiden sich darin, wie die Schleifenbedingung angegeben wird und wann diese Bedingung geprüft wird. Bekannte Schleifentypen sind `FOR`, `WHILE` und `REPEAT-UNTIL`. In Provop beschränken wir und auf `REPEAT-UNTIL`-Schleifen. Auf Grundlage dieses Schleifentyps lassen sich, in Kombination mit anderen Kontrollfluss-Konstrukten (z.B. Strukturknoten), alle relevanten Verhaltensweisen von Schleifen realisieren.

Notation. Abbildung 3.18 zeigt die Darstellung einer Schleife in einem Prozessmodell: Zur Kennzeichnung des Anfangs eines Schleifenrumpfs verwenden wir einen `LoopEntry`-Knoten. Analog beschreibt der `LoopExit`-Knoten das Ende des Schleifenrumpfs. Von einem `LoopExit`-Knoten zu einem `LoopEntry`-Knoten wird eine *Schleifen-Kante* gezogen. Sie stellt einen besonderen Typ einer Kontrollfluss-Kante dar und trägt die Schleifenbedingung, welche über die Weiterführung der Schleife entscheidet. Bei Verwendung von Schleifen fordern wir, dass der `LoopEntry`-Knoten im Ausführungspfad vor dem entsprechenden `LoopExit`-Knoten liegt. Außerdem muss genau eine Schleifen-Kante zwischen diesen Knoten angegeben werden.

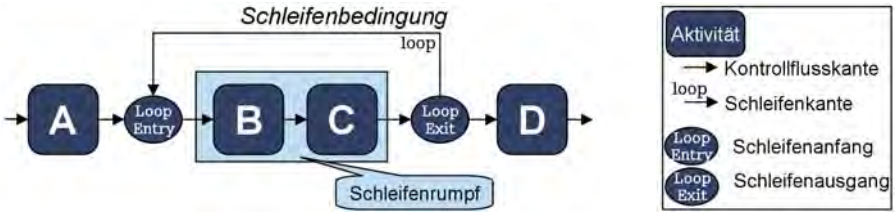


Abbildung 3.18: `REPEAT-UNTIL`-Schleife

Bei Ausführung von Schleifen können Probleme auftreten, wenn ein Schleifenkonstrukt nicht korrekt gebildet wurde. Dies ist z.B. der Fall, wenn die Schleifenbedingung keine Terminierung des Prozessmodells zulässt, d.h. der Schleifenrumpf unendlich oft wiederholt wird. Für die Definition eines Prozess-Metamodells ist es daher unabdingbar, zwischen zulässigen und unzulässigen Schleifenkonstrukten zu unterscheiden. Zu betrachten sind nicht nur Endlosschleifen, sondern u.A. auch Schachtelungen und Überlappungen von Schleifen. Wir

verzichten an dieser Stelle auf detaillierte Untersuchungen, da die Beschreibung eines Prozess-Metamodells nicht im Fokus der Arbeit liegt und die bisher beschriebene Verwendung von Schleifen zur Darstellung des Provop-Lösungsansatzes ausreicht.

Semantik. In Anlehnung an das Pattern *Structured Loop* [RAvdAM06] definieren wir die Semantik von Schleifen wie folgt: An einem `LoopEntry`-Knoten wird die ausgehende Kontrollfluss-Kante in den gleichen Zustand überführt, in welchem die eingehenden Kontrollfluss-Kante ist. Nach der einmaligen Ausführung des Schleifenrumpfes wird am `LoopExit`-Knoten die Schleifenbedingung geprüft. Ist diese `true`, wird die Schleifen-Kante in den Zustand `true_signaled` überführt. Damit wird ein Rücksprung zum `LoopEntry` Knoten ermöglicht. Der Schleifenrumpf wird dann zurückgesetzt, d.h. alle Aktivitäten werden in den Zustand `not activated` und die Kontrollfluss-Kanten in den Zustand `not_signaled` gesetzt. Eine wiederholte Ausführung des Schleifenrumpfes ist damit möglich. Evaluiert die Schleifenbedingung jedoch zu `false`, wird die ausgehende Kontrollfluss-Kante des `LoopExit`-Knotens in den Zustand `true_signaled` überführt und es wird mit der Ausführung des Prozesses fortgefahren. Der Schleifenrumpf wird dann nicht zurückgesetzt bzw. erneut iteriert.

3.2.3 Formale Definition eines Prozessmodells

Ein Prozessmodell ist ein zusammenhängender Prozessgraph, bei dem jeder Knoten von (mindestens) einem Startknoten (`StartNode`) aus erreichbar ist. Außerdem ist von jedem Knoten mindestens ein Endknoten erreichbar (`EndNode`). Kontrollfluss-Kanten mit gleichen Quell- und Zielknoten sind nicht möglich (d.h. ein Prozessmodell in Provop ist kein Multigraph).Wir definieren ein Prozessmodell, wie in Definition 3.1 angegeben.

Definition 3.1 (Prozessmodell und Prozessfragment)

- *Ein Prozessmodell ist ein Tupel* $P = (N, E, NT, ET, EC)$ *mit*
  - N *ist eine Menge von Knoten und* E *eine Menge von gerichteten Kanten; (N, E) bildet einen zusammenhängenden, gerichteten Graphen.*
  - $NT : N \rightarrow \{Start, End, Activity, ANDSplit, ORSplit, XORSplit, ANDJoin, ORJoin, XORJoin, LoopEntry, LoopExit, Silent, DataObj\}$ *ordnet jedem Knoten* $n \in N$ *einen Knotentyp* $NT(n)$ *zu.*
  - $ET : E \rightarrow \{ControlFlow, DataFlow, Loop\}$ *ordnet jeder Kante* $e \in E$ *einen Kantentyp* $ET(e)$ *zu.*
  - $EC(e)$ *ordnet jeder Kante* e *mit* $ET(e) \in \{ControlFlow, Loop\}$ *eine Transitionsbedingung cond bzw. den Wert* `true` *zu (*`true` *bedeutet, dass die Transitionsbedingungen stets wahr ist). Für jede Kante* e *mit* $ET(e) = DataFlow$ *sei* $EC(e)$ *undefiniert.*
- *Sei* $P = (N, E, NT, ET, EC)$ *ein Prozessmodell. Dann heißt* $PF = (N', E', NT', ET', EC')$ *Prozessfragment von P, falls gilt:* $N' \subseteq N$ *und* $E' \subseteq E$ *und* (N', E') *ist ein zusammenhängender Subgraph von* (N, E) *mit* $NT'(n') = NT(n') \; \forall \; n' \in N'$, $ET'(e') = ET(e') \; \forall \; e' \in E'$ *und* $EC'(e') = EC(e') \; \forall e' \in E'$.

Ein **Prozessfragment** beschreibt eine Menge von Elementen eines Prozessmodells, die zusammenhängend sind, d.h. die Aktivitäten eines Fragments sind durch einen Kontrollfluss miteinander verbunden (vgl. Abbildung 3.19a). Es ist nicht vorgegeben, mit welchem Prozesselementtyp (z.B. Aktivität oder Kante) ein Prozessfragment beginnt oder endet. Einen besonderen Fragment-Typ stellen *Single Entry Single Exit (SESE)*-Fragmente dar. Die Mengen

der eingehenden und ausgehenden Kanten sind hier auf jeweils genau eine Kante beschränkt (vgl. Abbildung 3.19b).

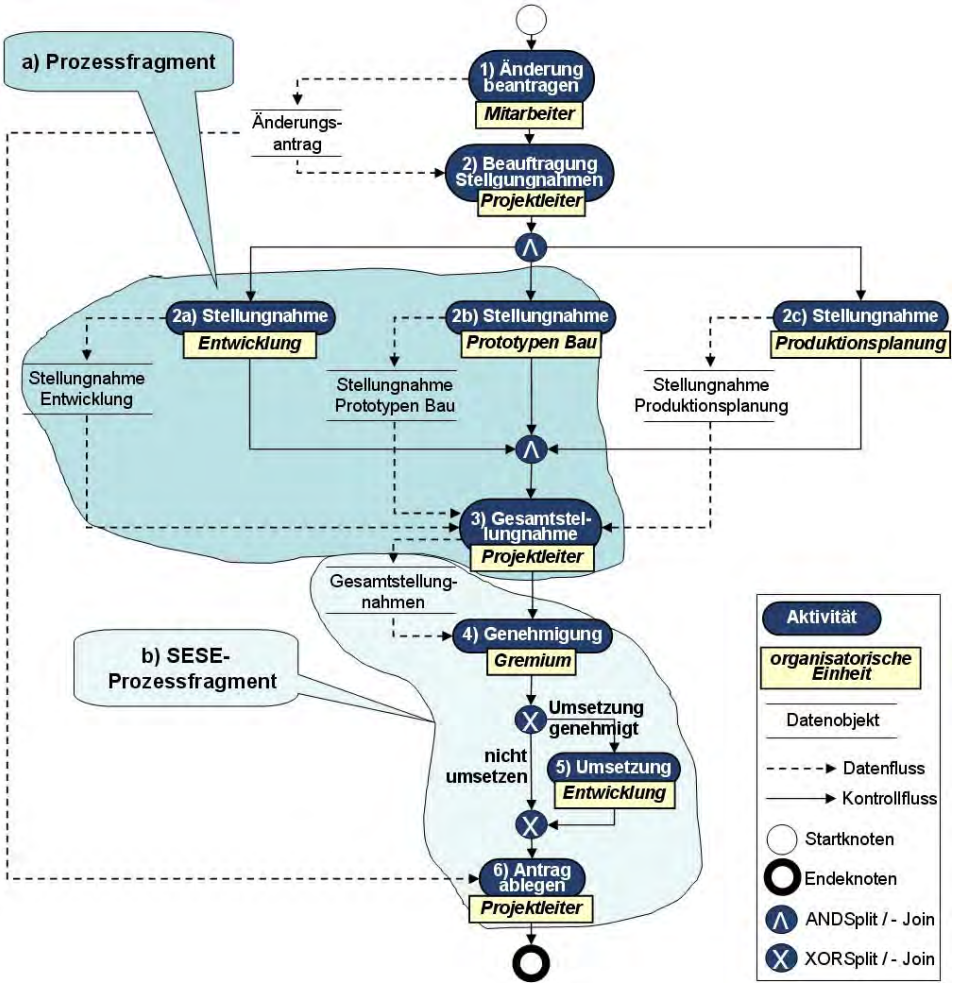


Abbildung 3.19: Prozessmodell mit a) beliebigem Prozessfragment und b) speziellem SESE-Prozessfragment

Zur Ausführung eines Prozessmodells können **Prozessinstanzen** erstellt und ausgeführt werden. Diese umfassen zusätzlich Informationen über die Ausführungszustände der Aktivitäten und Kanten (vgl. Definition 3.2).

Definition 3.2 (Prozessinstanz)
Eine Prozessinstanz ist ein Tupel $I = \{P, NS, ES, \mathcal{H}\}$ *mit*

- P *beschreibt das Prozessmodel auf dem* I *basiert.*
- $NS : N \rightarrow \{NotActivated, Activated, Running, Skipped, Completed\}$ *ordnet jedem Knoten* $n \in N$ *einen Ausführungszustand* $NS(n)$ *zu.*
- $ES : E \rightarrow \{true\_signaled, false\_signaled, not\_signaled\}$ *ordnet jeder Kante* $e \in E$ *einen Zustand* $ES(e)$ *zu.*
- $\mathcal{H} : \langle e_1, \ldots, e_m \rangle$ *bezeichnet die Ausführungshistorie von* I*, wobei jeder Eintrag* $e_k \in \mathcal{H}$ *dem Start oder Ende einer Prozessaktivität entspricht.*

3.3 Korrektheit von Prozessmodellen

Die Ausführung eines Prozessmodells mit Hilfe eines WfMS stellt ein wichtiges Ziel der Prozessmodellierung dar. Ein wesentlicher Aspekt ist die stabile und fehlerfreie Ausführung des Prozessmodells. Um diese zu gewährleisten, muss ein Prozessmodell bestimmten Anforderungen bzw. Korrektheitskriterien genügen. Diese Formalismen sind in gewissem Umfang metamodellspezifisch, d.h. die Korrektheitskriterien für EPKs, BPMN-Modelle und Petri-Netze sind nicht identisch.

Die Korrektheit von Prozessvarianten ist ein wichtiger Aspekt dieser Arbeit. Um im Folgenden aufzeigen zu können, wie inkorrekte Variantenmodelle in Provop identifiziert werden und wie wir mit solch inkorrekten Modellen umgehen, haben wir diverse Eigenschaften definiert (vgl. Tabelle 3.4). Diese basieren im Wesentlichen auf den bereits oben erläuterten strukturellen Anordnungsregeln der Prozesselemente. Darüber hinaus formulieren wir Eigenschaften für ein korrektes Ausführungsverhalten und eine korrekte Datenversorgung der Variantenmodelle.[7] Die aufgeführten Eigenschaften entsprechen nicht den spezifischen Eigenschaften eines konkreten Prozess-Metamodells.

Basierend auf den Eigenschaften, definieren wir Korrektheit von Prozessmodellen wie in Definition 3.3 angegeben. Inkorrekte Prozessmodelle erläutern wir in Beispiel 3.5.

Definition 3.3 (Korrektheit von Prozessmodellen)
Sei P ein Prozessmodell.

- *P heißt strukturell korrekt, wenn die Eigenschaften 1 bis 9 aus Tabelle 3.4 zutreffen.*
- *P heißt korrekt, falls P strukturell korrekt ist und zusätzlich die Eigenschaften 10 bis 13 aus Tabelle 3.5 zutreffen.*

Wir verzichten in dieser Arbeit weitestgehend auf eine ausführliche formale Betrachtung zur Korrektheit der Ausführung von Prozessmodellen (sog. Soundness) und verweisen auf entsprechende Literatur [vdA97, SZ06, Wes07].

Beispiel 3.5 (Inkorrekte Prozessmodelle) *Abbildung 3.20 zeigt Prozessmodelle, die nach Definition 3.3 nicht korrekt sind: In Abbildung 3.20a ist ein unzulässiger Zyklus definiert (vgl. Eigenschaft 6). Abbildung 3.20b zeigt ein Beispiel für eine Verklemmung (vgl. Eigenschaft 11): Die Ausführungspfade warten hier jeweils auf das Ende der Ausführung des parallel laufenden Pfades. Auf diese Weise kann die Prozessinstanz nicht terminieren. Abbildung 3.20c zeigt ein Beispiel für eine inkorrekte Datenversorgung (vgl. Eigenschaft 8): Da das Datenobjekt nicht geschrieben wird, wird Aktivität `C` ggf. mit falschen Daten versorgt. Abbildung 3.20e zeigt ein „Lost Update"-Problem (vgl. Eigenschaft 9): Die parallel ausgeführten Aktivitäten `B` und `C` schreiben jeweils das gleiche Datenobjekt. Da nicht eindeutig ist, welche Aktivität als letztes schreibt, ist nicht klar mit welchen Daten Aktivität `D` versorgt wird.*

3.4 Vereinfachung von Prozessgraphen

Die Darstellung von Prozessmodellen mit sehr vielen Prozesselementen wird schnell komplex und unübersichtlich. Dies wird durch Ändern bzw. Bearbeiten eines Prozessmodells verschiedener Modellierer zusätzlich verstärkt. Das Ergebnis ist oft ein Prozessmodell, das

[7] Die korrekte Anordnung bzw. das korrekte Ausführungsverhalten werden auch als *Structural* bzw. *Behavioural Soundness* bezeichnet [vdA97].

Tabelle 3.4: Korrekte Struktur eines Prozessmodells $P \equiv (N, E, NT, ET, EC)$

Eigenschaft 1 *Es existiert mindestens ein Start- und ein Endknoten. Jeder Startknoten besitzt genau eine ausgehende und keine eingehende Kante und jeder Endknoten besitzt genau eine eingehende und keine ausgehende Kante. Das heißt es gilt:*

- $\exists n_1, n_2 \in N$: $NT(n_1) = Start \wedge NT(n_2) = End$
- $\forall\, n \in N$ *mit* $NT(n) = Start$: $\nexists\, (n_1, n_2) \in E$ *mit* $n_2 = n \wedge \exists!(n_1, n_2) \in E$ *mit* $n_1 = n$
- $\forall\, n \in N$ *mit* $NT(n) = End$: $\nexists\, (n_1, n_2) \in E$ *mit* $n_1 = n \wedge \exists!(n_1, n_2) \in E$ *mit* $n_2 = n$

Eigenschaft 2 *Jeder Nichtstartknoten ist über einen Pfad von mindestens einem Startknoten aus erreichbar.* $\forall\, n \in N$ *mit* $NT(n) \neq Start$: $\exists\, st \in N$ *mit* $NT(st) = Start$ *und* $\exists p$ *mit* $p \equiv st = n_1, n_2, \ldots, n_k = n$ *ist ein Pfad in* (N, E) *vom Startknoten st zum Knoten n.*

Eigenschaft 3 *Von jedem Nichtendknoten gibt es mindestens einen Pfad zu einem der Endknoten.* $\forall\, n \in N$ *mit* $NT(n) \neq End$: $\exists\, end \in N$ *mit* $NT(end) = End$ *und* $\exists p$ *mit* $p \equiv n = n_1, n_2, \ldots, n_k = end$ *ist ein Pfad in* (N, E) *vom Knoten n zum Endknoten end.*

Eigenschaft 4 *Jede Aktivität besitzt genau eine ein- und eine ausgehende Kontrollfluss-Kante.* $\forall\, n \in N$ *mit* $NT(n) = Activity$:

- $\exists!(n_1, n_2) \in E$ *mit* $ET(e) = ControlFlow \wedge n_2 = n$
- $\exists!(n_3, n_4) \in E$ *mit* $ET(e) = ControlFlow \wedge n_3 = n$

Eigenschaft 5 `Split`*-Knoten haben genau eine eingehende Kontrollfluss-Kante und mindestens eine ausgehende Kontrollfluss-Kante.* `Join`*-Knoten haben mindestens eine eingehende Kontrollfluss-Kante und genau eine ausgehende Kontrollfluss-Kante.*
$\forall n \in N$ *mit* $NT(n) \in \{XORSplit, ORSplit, ANDSplit\}$

- $\exists!(n_1, n_2) \in E$ *mit* $n_2 = n$
- $\exists(n_3, n_4) \in E$ *mit* $n_3 = n$ *(d.h.* $|\{e \in E | e = (n, n')$ *mit* $n' \in N\}| \geq 1$*)*

$\forall n \in N$ *mit* $NT(n) \in \{XORJoin, ORJoin, ANDJoin\}$

- $\exists!(n_1, n_2) \in E$ *mit* $n_1 = n$
- $\exists(n_3, n_4) \in E$ *mit* $n_4 = n$ *(d.h.* $|\{e \in E | e = (n', n)$ *mit* $n' \in N\}| \geq 1$*)*

Eigenschaft 6 *Zyklen entstehen ausschließlich über die Verwendung von Loopkanten in Verbindung mit Schleifenknoten. Sei* $E' = E - \{e \in E | ET(e) = Loop\}$*. Dann ist* (N, E') *ein azyklischer Prozessgraph.*

Eigenschaft 7 *Eine Schleife verbindet einen Schleifenendknoten le mit einem vorausgehenden Schleifeneingangsknoten ls über eine Schleifenrücksprungkante. Ein konkreter Schleifenknoten kann jeweils nur zur Bildung einer einzigen Schleife verwendet werden.*
$\forall\, e = (ls, le) \in E$ *mit* $ET(e) = Loop$:

- $NT(ls) = LoopExit \wedge NT(le) = LoopEntry$
- $\exists\, p \equiv le = n_1, n_2, \ldots, n_k = ls$ *in* (N, E)

$\forall\, n \in N$ *mit* $NT(n) = LoopExit$: $\exists! e = (n_1, n_2) \in E$ *mit* $ET(e) = Loop \wedge n_1 = n$
$\forall\, n \in N$ *mit* $NT(n) = LoopEntry$: $\exists! e = (n_1, n_2) \in E$ *mit* $ET(e) = Loop \wedge n_2 = n$

Eigenschaft 8 *Auf einem Datenobjekt findet kein Lesen ohne eine vorheriges Schreiben statt.*[a]

Eigenschaft 9 *Ein Datenobjekt kann nicht parallel von mehreren Aktivitäten geschrieben werden.*

[a] Für eine formale Definition für die Eigenschaften 8 und 9 siehe [Rei00].

Tabelle 3.5: Korrektes Ausführungsverhalten eines Prozessmodells $P \equiv (N, E, NT, ET, EC)$

Eigenschaft 10 *Es gibt für jede Aktivität mindestens einen erreichbaren Zustand der Prozessinstanz, in dem die Ausführung der Aktivität möglich ist.*

Eigenschaft 11 *Es treten keine Verklemmungen (engl. Deadlocks) auf. Das heißt in jedem Zustand einer Prozessinstanz ist immer mindestens eine Aktivität aktiviert bzw. laufend oder aber die Prozessinstanz befindet sich in einem gewünschten Endzustand.*

Eigenschaft 12 *Wird ein Endknoten während der Ausführung erreicht, gibt es keine aktivierte oder laufende Aktivität, von der ausgehend es zur erneuten Aktivierung des selben Endknoten kommt.*

Eigenschaft 13 *Eine Schleife terminiert stets nach endlich vielen Durchläufen. Jede gestartete Aktivität wird in endlicher Zeit beendet.*

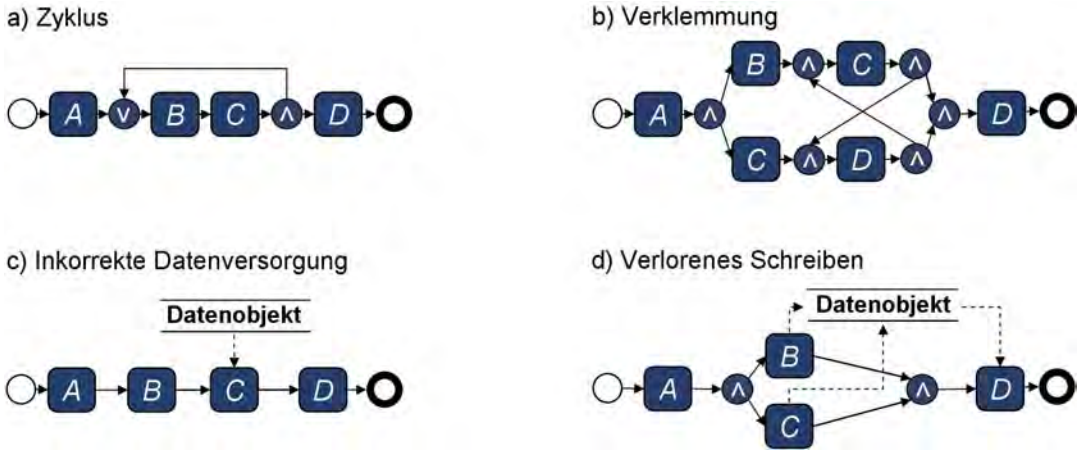


Abbildung 3.20: Beispiele für nicht korrekte Prozessmodelle

strukturell zwar korrekt ist, jedoch überflüssige Prozesselemente enthält. Solche überflüssigen Elemente (z.B. Strukturknoten und Kontrollfluss-Kanten) resultieren z.B. aus dem Löschen einzelner Aktivitäten. Eine übersichtliche Darstellung von großen Prozessmodellen ist in der Praxis unabdingbar. Zu diesem Zweck muss die Darstellung optimiert werden können, so dass keine überflüssigen Elemente im Prozessmodell enthalten sind. Damit wir diese optimierte Darstellung erreichen können, verwenden wir in Provop eine Reihe wohldefinierter Graph-Vereinfachungsoperationen, wobei das Ausführungsverhalten der Prozessmodelle unverändert bleibt.

Die von uns verwendeten Vereinfachungsoperationen sind allgemein anerkannt und werden daher an dieser Stelle nicht formal beschrieben. Abbildung 3.21 zeigt beispielhaft, welche Vereinfachungsoperationen wir an Prozessmodellen durch Entfernen überflüssiger Prozesselemente vornehmen können.

3.5 Diskussion

In der Praxis existieren zur Gestaltung von Prozessmodellen verschiedene Prozess-Metamodelle, wie EPK [Sch98] und BPMN [BPM09]. Diese Prozess-Metamodelle umfassen eine Vielzahl von Prozesselementen und -strukturen. Meist ist eine solche Komplexität jedoch nicht wirklich erforderlich [MR08]. Dies machen wir uns auch für unseren Provop-

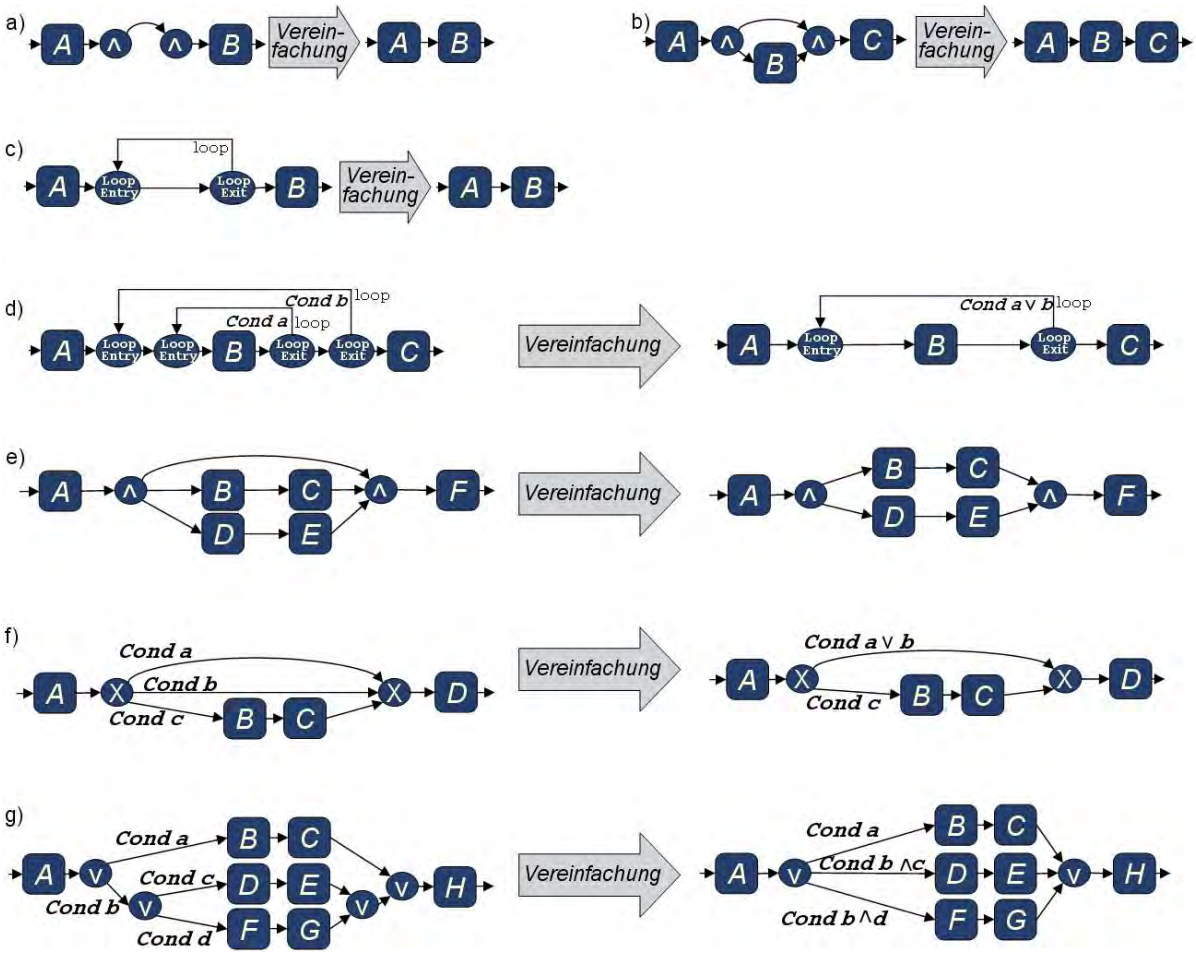


Abbildung 3.21: Vereinfachungen eines Prozessmodells

Ansatz zunutze, zu dessen Erläuterung die für die Praxis häufigsten Prozesselemente und -strukturen verwendet werden. Diese werden im Rahmen dieser Arbeit vorgestellt. Sie beruhen auf bekannten Modellierungssprachen und werden von uns in einer allgemein verständlichen Form verwendet. Das heißt wir übernehmen die Darstellung und Semantik bekannter Konzepte der Prozessmodellierung (z.B. [Sch98, BPM09]) sowie des Workflow-Managements [RAvdAM06, RAE04]. Die von Provop vorgenommenen Einschränkungen sind minimal. Wir fordern z.B. keine Blockstrukturierung und erlauben das beliebige Verschränken von Verzweigungs-, Synchronisations- und Schleifenkonstrukten. Zweck dieser Fokussierung ist die resultierende Generizität des Provop-Metamodells, wodurch der mit Provop entwickelte Lösungsansatz auf verschiedene Prozess-Metamodelle übertragbar ist.

3.6 Zusammenfassung

In diesem Kapitel haben wir das Prozess-Metamodell von Provop informell vorgestellt. Die unterstützten Prozesselemente und -strukturen umfassen Aktivitäten, Kontrollfluss-Kanten, den Datenfluss eines Prozessmodells und komplexe Strukturen wie z.B. Schleifen. Darüber hinaus haben wir die Ausführungssemantik der Prozesselemente informell diskutiert.

Des Weiteren beschreiben wir Kriterien, die angeben, wann ein Prozessmodell hinsichtlich seiner Struktur, seinem Ausführungsverhalten und der Datenversorgung valide ist. Diese

Korrektheitskriterien greifen wir später auf, wenn wir erläutern, wie wir in Provop die korrekte Konfiguration von Prozessvarianten gewährleisten können.

Da Prozessmodelle schnell sehr groß und unübersichtlich werden, haben wir Graph-Vereinfachungsoperationen definiert (i.S. von Refactorings), um die Struktur der Prozessmodelle geeignet zu korrigieren und ihre Lesbarkeit zu verbessern. Die Vereinfachungsoperationen erhalten immer die Ausführungssemantik der Prozessmodelle.

4

Der Provop-Ansatz im Überblick

In Kapitel 2 haben wir zwei Fallbeispiele für variantenreiche Prozesstypen aus der Automobilindustrie präsentiert. Davon ausgehend haben wir charakteristische Anforderungen an die Modellierung und das Management von Prozessvarianten elaboriert. Weiter haben wir gezeigt, dass existierende Prozessmodellierungswerkzeuge keine oder nur unzureichende Unterstützung für das Management von Prozessvarianten bieten. In diesem Kapitel beschreiben wir nun, welche konventionellen Ansätze für das Variantenmanagement mit heutigen Systemen möglich ist. Anschließend skizzieren wir den im Rahmen dieser Arbeit entwickelten Provop-Ansatz für den Umgang mit variantenreichen Prozessen. Wir beschränken uns zunächst darauf, einen Überblick über die entwickelten Konzepte zu geben; eine detaillierte Behandlung erfolgt in den Kapiteln 5 bis 9.

Kapitel 4 gliedert sich wie folgt: Abschnitt 4.1 gibt eine kurze Motivation. Abschnitt 4.2 stellt konventionelle Ansätze für das Management von Prozessvarianten vor und zeigt, warum diese in der bisher praktizierten Form für ein umfassendes Variantenmanagement nicht ausreichen. In Abschnitt 4.3 skizzieren wir den Provop-Lösungsansatz und geben einen Überblick über die entwickelten Konzepte. Abschnitt 4.4 vergleicht den Provop-Lösungsansatz mit den vorangehend vorgestellten, konventionellen Ansätzen für das Management von Prozessvarianten. Das Kapitel schließt mit einer Zusammenfassung in Abschnitt 4.5.

4.1 Motivation

Unsere Fallstudien haben gezeigt, dass konventionelle Ansätze für das Management von Prozessvarianten nicht ausreichen. Heutige Ansätze scheitern an ihrer mangelnden Skalierbarkeit und der fehlenden Transparenz modellierter Prozessvarianten. Für die Entwicklung des Provop-Ansatzes gilt es, solche Nachteile konventioneller Methoden zu vermeiden. Gleichzeitig sollen die entwickelten Konzepte den in Abschnitt 2.3 beschriebenen Anforderungen genügen.

4.2 Konventionelle Ansätze für das Management von Prozessvarianten

Konventionelle Ansätze zur Modellierung von Prozessvarianten nutzen die Standardfunktionalität existierender Modellierungswerkzeuge, wie ARIS [IDS08] oder Innovator [MID08]. Ein möglicher Ansatz besteht darin, für jede Variante ein separates Prozessmodell (explizit) festzulegen und zu verwalten [LS06]; eine Alternative dazu ist es, alle Prozessvarianten durch dasselbe Prozessmodell abzubilden [RvdA07, RMvdT09, MCH07]. Wir sprechen in der Folge von einem *Mehr-Modell-* bzw. *Ein-Modell-Ansatz* [HBR09d, HBR09b]. Im Folgenden beschreiben wir beide Ansätze im Detail und diskutieren ihre Vor- und Nachteile entlang der fachlichen Anforderungen aus Kapitel 2.

4.2.1 Mehr-Modell-Ansatz

Ein naheliegender Ansatz für das Management von Prozessvarianten ist deren Abbildung mittels separater Prozessmodelle. Allerdings führt dies meist zu einer großen Zahl zu verwaltender Prozessmodelle. Ein stark vereinfachtes Beispiel aus dem Änderungsmanagement zeigt Abbildung 4.1. Für den Prozess zur Bearbeitung eines Produktänderungsantrags existieren drei verschiedene Prozessvarianten, abhängig von den erwarteten Umsetzungskosten und der Qualitätsrelevanz der Änderung (vgl. Abschnitt 2.1).

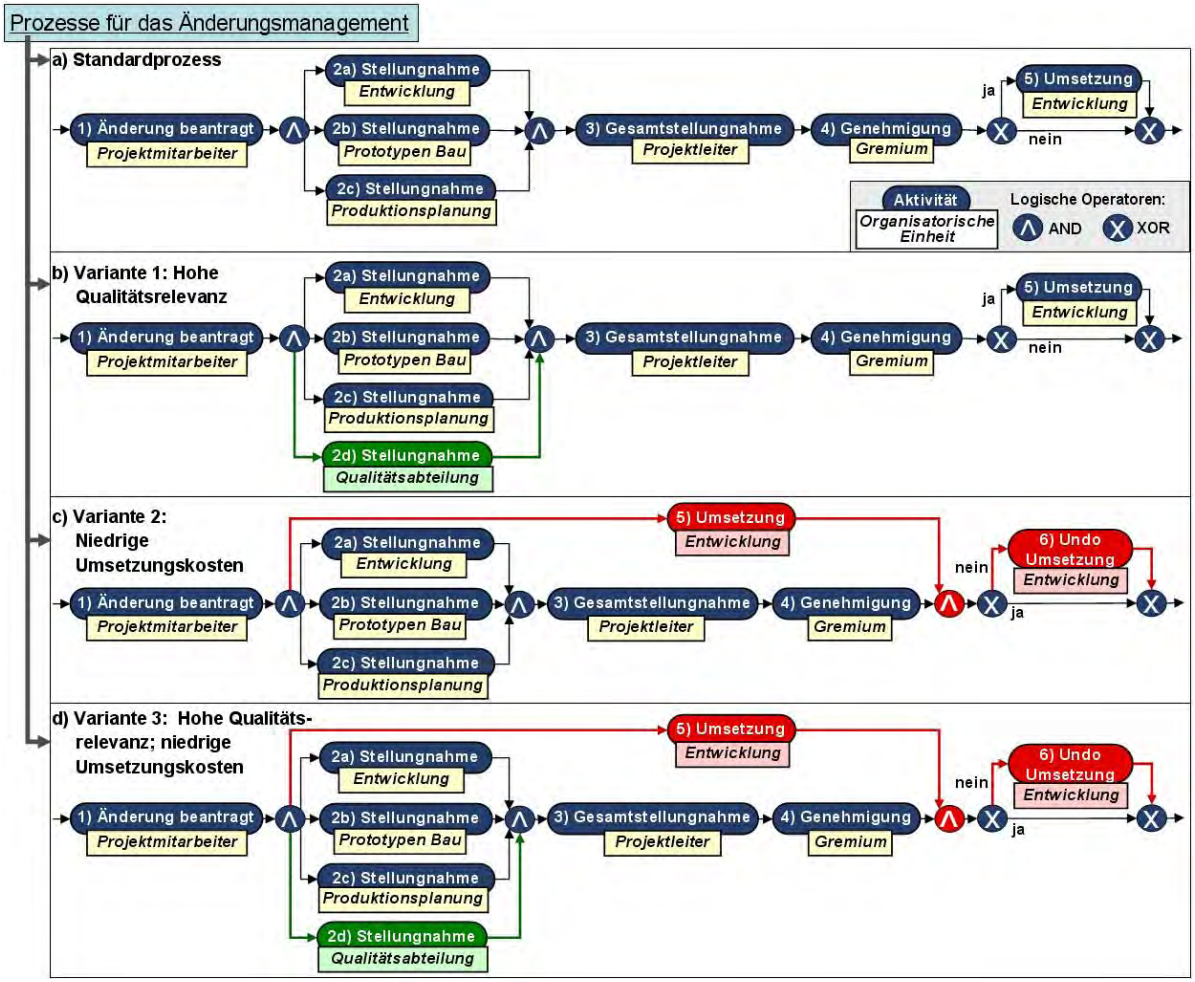


Abbildung 4.1: Verzeichnis von ausmodellierten Prozessvarianten des Änderungsmanagements

Modellierung.
Beim Mehr-Modell-Ansatz werden alle Varianten eines Prozesstyps explizit ausmodelliert, d.h. es wird für jede Variante ein separates Prozessmodell erzeugt. Um die logische Zusammengehörigkeit dieser Prozessmodelle zu dokumentieren, werden sie in existierenden Modellierungswerkzeugen meist in einem gemeinsamen Verzeichnis des Modell-Repositories abgelegt. Spezifische Algorithmen unterstützen Anwender bei der Suche nach bestimmten Prozessvarianten eines Verzeichnisses [LS06].

Dieser Ansatz wird von existierenden Prozessmodellierungswerkzeugen gut unterstützt (vgl. Abschnitt 2.4). Das Erstellen von Prozessmodellen als Kopie bzw. Weiterentwicklung existierender Modelle stellt eine Standardfunktionalität heutiger Modellierungswerkzeuge dar. Zudem ist der Mehr-Modell-Ansatz aus Modellierersicht intuitiv zugänglich, da einzelne Prozessvarianten, die für unterschiedliche Anwendungsfälle relevant sind, als separate Objekte (d.h. Prozessmodelle) gehandhabt werden.

Im Fall des in Abschnitt 2.2 vorgestellten Werkstattprozesses existieren bereits mehrere hundert Varianten. Die unabhängige Modellierung dieser Prozessvarianten bringt üblicherweise einen sehr hohen Aufwand mit sich. Insbesondere werden Prozessfragmente, die für alle Prozessvarianten unverändert anwendbar sind, redundant erfasst. Dadurch steigen die Aufwände für die Wartung und Pflege der Prozessmodelle. Sind umfangreiche Prozessänderungen durchzuführen, die mehrere oder alle Prozessvarianten betreffen, muss dem entsprechend jedes betroffene Prozessmodell separat angepasst werden.[1] So könnte es aufgrund neuer gesetzlicher Vorgaben eines Landes notwendig werden, alle Händlervarianten des Werkstattprozesses separat anzupassen. Noch aufwendiger erweisen sich Änderungen am Standardprocedere der Werkstätten, da hier weltweit zahlreiche Prozessvarianten angepasst werden müssen. Mit den dann erforderlichen redundanten Änderungen der Variantenmodelle wächst zwangsläufig auch die Wahrscheinlichkeit von Fehlern und Inkonsistenzen [WR08].

Konfiguration.
Beim Mehr-Modell-Ansatz spiegelt das Prozessmodell nicht explizit wider, welche Aspekte spezifisch für diese Variante sind. Neue Prozessvarianten können deshalb nicht einfach aus bereits bestehenden abgeleitet werden. Zum Beispiel kann Variante 3 aus Abbildung 4.1, welche eine Kombination der Varianten 1 und 2 darstellt, nicht ohne weiteres aus den Modellen dieser Varianten generiert werden.[2]

Üblicherweise werden Varianten eines Prozesstyps erforderlich, um bestimmte Rahmenbedingungen oder Anwendungsfälle abzubilden. Bezogen auf den Werkstattprozess etwa können die Varianten von Faktoren wie Fahrzeug, Land und Händler abhängen. Letztere definieren den Kontext einer Prozessvariante. Für das Management von Prozessvarianten sollte dieser Kontext genutzt werden, um die aktuell benötigte Prozessvariante zu identifizieren. Diese Kontextabhängigkeit kann im Mehr-Modell-Ansatz nur für ein ganzes Prozessmodell angegeben werden, z.B. in Form von Prozessmetadaten oder entsprechenden Benennungen der Prozessmodelle. Der Nachteil dieses Ansatzes zur Erfassung der Kontextabhängkeit von Prozessvarianten ist, dass bei einer Vielzahl von Varianten und Einflussfaktoren (z.B. Qualitätsrelevanz, Risiken, Umsetzungskosten) sehr schnell eine hohe Komplexität erreicht wird; d.h. Modellbenennungen werden sehr lang. Darüber hinaus existiert keine zentrale Datenbank

[1] Dieses Problem ist im Bereich der Softwareentwicklung auch als „multiple maintenance problem“ bekannt [Bab86].
[2] Mit Techniken des Modell-Merging konnten hierzu erste Teillösungen realisiert werden [AP03, AAAN+05, WK08].

zur Verwaltung aller Faktoren eines Prozesstyps. Daraus resultiert eine schlechte Transparenz, d.h. es ist nicht klar, welche Faktoren in einer bestimmten Domäne relevant sind und welche Werte sie annehmen können. Insgesamt ergibt sich eine heterogene Verwendung der Faktoren zur Erfassung der Kontextabhängigkeit von Prozessvarianten.

Ausführung.
Für die Ausführung einer bestimmten Prozessvariante wird im vorliegenden Fall deren Prozessmodell ausgewählt und instanziiert. Während der Ausführung einer Prozessvariante ist ein Wechsel zu einer anderen Variante jedoch nur durch Abbruch der aktuellen und Instanziierung einer anderen Prozessvariante möglich. Dadurch werden identische Prozessfragmente ggf. redundant bearbeitet was in der Praxis i.A. nicht tolerabel ist. Nachteilig sind somit die resultierende Inflexibilität sowie die zusätzlich entstehenden Aufwände bei dynamischen Variantenwechsel. Letztere könnten allerdings bei Verwendung einer adaptiven Prozess-Management-Technologie, wie ADEPT bzw. AristaFlow [DR09], reduziert werden.

Evolution.
Prozessvarianten können infolge ihrer separierten Modelle getrennt voneinander weiterentwickelt und optimiert werden. Geschieht dies jedoch ohne Berücksichtigung der (logischen) Zusammenhänge zwischen den bestehenden Prozessvarianten (d.h. ohne Kenntnis bzw. Beachtung ihrer Gemeinsamkeiten) führt dieser Ansatz auf Dauer zu einer Entartung der Variantenmodelle, die sich unterschiedlich entwickeln und immer mehr voneinander abweichen. Dies führt zu einer erschwerten Pflegbarkeit und sinkenden Qualität der Prozessmodelle. Ähnliche Beobachtungen hat man z.B. im Bereich des Software-Engineering gemacht, wo das Problem „alternder Software“ diskutiert wird [Par94].

Korrektheit.
Die Prüfung der Korrektheit von Variantenmodellen ist mit existierenden Verifikationsansätzen möglich [vdA97, vdA00, vdADG+08]. Dazu muss jedes Prozessmodell auf Korrektheit geprüft werden. Zur Gewährleistung der Korrektheit aller Varianten eines bestimmten Prozesstyps sind demnach alle konfigurierbaren Variantenmodelle zu identifizieren und zu analysieren.

Darstellung.
Die separate Modellierung von Prozessvarianten erlaubt deren spezifische Visualisierung mit verfügbaren Visualisierungstechniken [Bob08]. Insbesondere stellt ein Modell exakt eine Prozessvariante dar. Die Analyse und der Vergleich verschiedener Prozessvarianten ist mit existierenden Prozessmodellierungswerkzeugen jedoch nur eingeschränkt möglich (vgl. Abschnitt 2.4). Meist können maximal zwei Prozessmodelle gleichzeitig im Modellierungswerkzeug betrachtet werden. Das ARIS Toolset von IDS Scheer bietet z.B. einen solchen Vergleich zwischen zwei Prozessmodellen an [IDS08]. Dabei werden die Unterschiede der Prozessmodelle optisch hervorgehoben. Dieser paarweise Vergleich ist jedoch bei sehr vielen und komplexen Variantenmodellen eines Prozesstyps umständlich und aufwendig.

Skalierbarkeit.
Für eine kleine Anzahl von Prozessvarianten ist der Mehr-Modell-Ansatz noch relativ gut geeignet. Bei steigender Anzahl wird die Handhabung der Variantenmodelle dagegen schwieriger. Existieren für ein Prozessmodell viele Varianten, gelangt der Modellierer schnell an einen Punkt, an dem diese Vielzahl nicht mehr beherrschbar ist. Dies gilt insbesondere in Bezug auf Änderungen, die mehrere Varaintenmodelle betreffen.

4.2.2 Ein-Modell-Ansatz

Ein alternativer, in der Praxis ebenfalls verfolgter Lösungsansatz ist die Repräsentation aller Varianten durch ein Prozessmodell. Abbildung 4.2 zeigt dazu ein einfaches Beispiel, das die Integration des in Abbildung 4.1 vorgestellten Standardprozesses und seiner drei Varianten widerspiegelt.

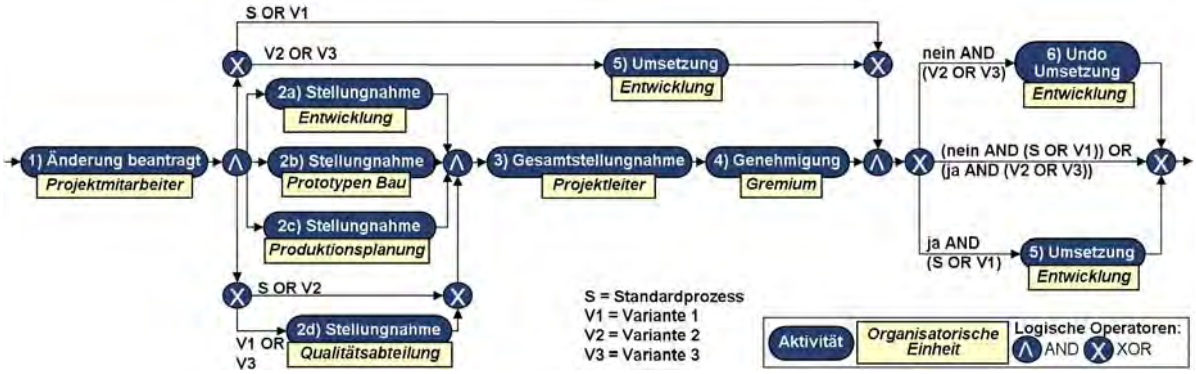


Abbildung 4.2: Bedingte Verzweigungen zur Repräsentation von Varianten

Modellierung.
Um Prozessvarianten bzw. variante Abläufe innerhalb eines Prozessmodells abzubilden, können vom Prinzip her bedingte Verzweigungen (bspw. `XORSplit`- und `XORJoin`-Knoten) verwendet werden (vgl. Abbildung 4.2). Durch Festlegung geeigneter Verzweigungsbedingungen, die sich auf Prozessdaten beziehen können, kann zur Laufzeit die jeweils gewünschte Prozessvariante determiniert werden. Der Vorteil dieses Ein-Modell-Ansatzes ist seine direkte Unterstützbarkeit durch existierende Werkzeuge. Insbesondere stellt die Modellierung bedingter Verzweigungen eine Standardfunktionalität existierender Modellierungswerkzeuge dar. Darüber hinaus ist dieser Ansatz zur Modellierung von Prozessvarianten intuitiv und erfordert keine neuen Modellierungskonstrukte. Ein Nachteil des Ein-Modell-Ansatzes besteht darin, dass Prozessvarianten nicht explizit als solche gekennzeichnet sind. Vielmehr ist die Information darüber, wann eine bestimmte Prozessvariante durchlaufen wird, in der Ablauflogik versteckt, d.h. in den entsprechenden Kantenbedingungen der varianten Ausführungspfade codiert. Dies führt dazu, dass der Modellierer nicht unmittelbar erkennen kann, ob eine bedingte Verzweigung des Prozessmodells verschiedene Varianten repräsentiert oder ob sie eine für alle Varianten gültige alternative Verzweigung darstellt, die Teil des Standardprocedere ist.

Beispiel 4.1 (Variante Abläufe durch bedingte Verzweigungen) *Im Änderungsmanagementprozess aus Abbildung 4.2 müssen wir im Prinzip für jede Prozessvariante entscheiden, ob eine Produktänderung umgesetzt werden muss oder nicht. Für den Werkstattprozess (vgl. Abbildung 2.3) findet sich ein ähnliches Beispiel für die Umsetzung einer Reparatur: Abhängig davon, ob ein Schaden gefunden wird, wird eine Reparatur durchgeführt oder aber ausgelassen. Diese Entscheidung ist nicht variantenspezifisch und muss daher in jeder Variante des Werkstattprozesses modelliert sein. Eine bedingte Verzweigung kann daher auch verwendet werden, um eine Entscheidung zwischen alternativen Ausführungspfaden abzubilden, welche für alle Prozessvarianten gleichermaßen getroffen werden muss.*

Ein weiterer Nachteil des Ein-Modell-Ansatzes ist, dass der Modellierungsaufwand sehr hoch ist, da durch unterschiedliche Reihenfolgen von Aktivitäten Redundanzen entstehen (vgl. Abbildung 4.2 Aktivität `Umsetzung`). Dies führt außerdem zu komplexen Prozessmodellen.

Konfiguration.
Beim Ein-Modell-Ansatz wird eine Prozessvariante zur Laufzeit konfiguriert und zwar dann, wenn an den bedingten Verzweigungen eine Entscheidung für oder gegen einen bestimmten varianten Ausführungspfad getroffen wird. Zur Schaffung einer Kontextabhängigkeit können Kantenbedingungen verwendet werden. Abhängig von den gegebenen Kontextinformationen kann die gewünschte Variante bzw. der zugehörige Pfad dann entweder ausgeführt werden oder nicht. Die Information über Kontextabhängigkeiten sind hier allerdings in der Ablauflogik der Prozessmodelle versteckt. Außerdem werden Entscheidungen, die bereits vor der Instanziierung und Ausführung einer Prozessvariante getroffen werden können, in die Laufzeit verschoben [GvdAJVlR07].

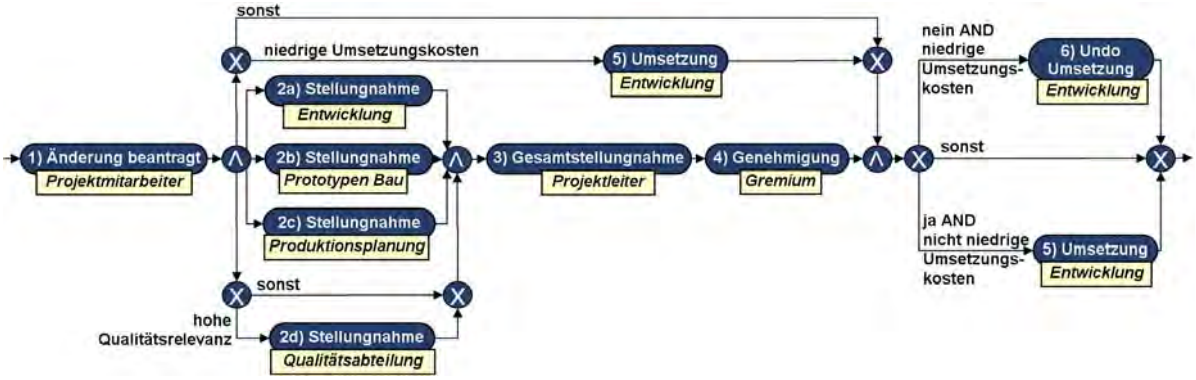


Abbildung 4.3: Kantenbedingungen basierend auf dem Kontext einer Prozessvariante

Ausführung.
Bedingte Verzweigungen erlauben es erst zur Laufzeit, die tatsächlich benötigte Prozessvariante zu determinieren. Dazu muss allerdings ein Prozessmodell instanziiert werden, dass wesentlich umfangreicher ist, als das Prozessmodell der im vorliegenden Kontext tatsächlich auszuführenden Prozessvariante [GvdAJVlR07]. Insbesondere das Monitoring von Prozessausführungen ist unter diesen Rahmenbedingungen schwierig. Ein expliziter Wechsel zwischen den Modellen verschiedener Prozessvarianten wird beim Ein-Modell-Ansatz natürlich nicht notwendig, da alle Prozessvarianten durch das Gesamtmodell abgedeckt sind.

Evolution.
Ein Vorteil des Ein-Modell-Ansatzes besteht darin, dass die Weiterentwicklung und Optimierung gemeinsamer Teile zur direkten Anpassung aller Prozessvarianten führt. Nachteilig ist, dass bei der Anpassung redundanter Teile die jeweiligen Änderungen mehrfach durchgeführt werden müssen, was leicht in Inkonsistenzen enden kann. Zum Beispiel wird in Abbildung 4.2 die Aktivität `Umsetzung` redundant modelliert. Sie kann entweder parallel zur Einholung der Stellungnahmen oder nach der Entscheidung der Umsetzung einer Produktänderung durchgeführt werden. Sind Änderungen an dieser Aktivität erforderlich, etwa wenn ein Attribut der Aktivität geändert werden soll, ist diese Änderung entsprechend mehrfach umzusetzen.

Korrektheit.
Die Korrektheit des Prozessmodells kann mit gängigen Verifikationsmethoden geprüft werden. Allerdings ist schwer nachvollziehbar, welche Prozessvariante zur Laufzeit tatsächlich ausgeführt wird und ob diese Prozessvariante semantisch und strukturell korrekt ist (vgl. Beispiel 4.2). Zum Beispiel kann eine falsche Datenversorgung erfolgen, wenn notwendige Schreiboperationen von Datenobjekten aufgrund eines Variantenwechsels nicht ausgeführt

werden (vgl. Eigenschaft 8). Je nach Metamodell kann dies zulässig sein oder zu einem schweren Fehler führen.

Beispiel 4.2 (Inkorrekte Prozessvariante zur Laufzeit) *Ein Beispiel zeigt Abbildung 4.3: Wird die Höhe der Umsetzungskosten zu Beginn mit „hoch" angegeben, wählen wir den Pfad ohne die Aktivität* `Umsetzung`*. Wird anschließend der Wert von „hoch" auf „niedrig" korrigiert, wird selbst nach einer Bewilligung der Änderung diese nicht mehr umgesetzt, da davon ausgegangen wird, dass Aktivität* `Umsetzung` *bereits ausgeführt worden ist. Das Ergebnis ist eine semantisch inkorrekte Ausführung des Prozesses, die mit existierenden Verifikationsmethoden nicht ohne weiteres detektiert werden kann.*

Darstellung.

Durch Einbinden von Prozessvarianten mittels bedingter Verzweigungen sind die varianten Ausführungspfade im Prozessmodell erkennbar, aber nicht von normalen Entscheidungen, welche für alle Prozessvarianten relevant sind, unterscheidbar. Zur Darstellung eines bestimmen Variantmodells ist eine Sichtenbildung nötig [BRB05, BRB07]. Durch geeignete Graphoperationen wie Hervorhebung, Reduktion oder Aggregation von Prozessfragmenten, kann auf diese Weise eine bestimmte Prozessvariante betrachtet werden [BBR06, BRB07]. Die Vergleichbarkeit von Prozessvarianten ist dann allerdings nur durch den Vergleich verschiedener Sichten eines Prozessmodells möglich.

Skalierbarkeit.

Bei einer geringen bzw. mittleren Anzahl von Prozessvarianten ist die skizzierte Vorgehensweise gut geeignet. Bei einer Vielzahl von Prozessvarianten skaliert der Ansatz jedoch schlecht. Müssen z.B. mehrere hundert Prozessvarianten, wie bei den Werkstattprozessen (vgl. Abschnitt 2.2), in einem Modell abgebildet werden, ist ein Ein-Modell-Ansatz nicht mehr handhabbar.

4.2.3 Zwischenfazit

Sowohl die separate Modellierung von Prozessvarianten in verschiedenen Prozessmodellen als auch die Abbildung aller Varianten in einem Prozessmodell sind in der Praxis verbreitete Methoden zur Repräsentation von Varianten. Dies ist direkte Folge der Unterstützung der hierfür nötigen Konzepte durch existierende Werkzeuge (vgl. Abschnitt 2.4), für die das Erstellen mehrerer Prozessmodelle mit unterschiedlichen Bezeichnungen bzw. das Modellieren von bedingten Verzweigungen in einem Prozessmodell lediglich Basisfunktionalitäten darstellen.

Die breite Akzeptanz und Verwendung dieser Ansätze bedeutet jedoch nicht, dass eine ausreichende Unterstützung für das Management von Prozessvarianten vorliegt. Die Modellierung von hunderten von Reparaturprozessen und deren Verwaltung in separaten Prozessmodellen ist ebenso wenig sinnvoll, wie die Modellierung aller Varianten in einem Prozessmodell. Nachteilig bei beiden Ansätze sind die fehlende Transparenz über die Prozessvarianten. Demzufolge sind die Unterschiede bestimmter Varianten in separaten Modellen (Mehr-Modell-Ansatz) bzw. in der Ablauflogik eines Prozessmodells versteckt (Ein-Modell-Ansatz). Auch die Erstellung und Verwendung von Kontextinformation wird in beiden Ansätzen nicht ausreichend berücksichtigt.

4.3 Grundlegende Aspekte des Provop-Ansatzes

Nachdem wir konventionelle Ansätze für das Management von Prozessvarianten vorgestellt haben, geben wir im Folgenden einen Überblick zu dem in dieser Arbeit entwickelten Provop-Ansatz. Dazu beschreiben wir zunächst, auf Basis welcher Beobachtungen wir Provop entwickelt haben. Danach skizzieren wir die entwickelten Konzepte und erörtern deren Zusammenhang. Abschließend vergleichen wir Provop mit den beiden vorangehend vorgestellten Ansätzen. Technische Details der entwickelten Konzepte werden in den Kapiteln 5 bis 9 erörtert.

4.3.1 Entwicklung des Provop-Ansatzes

Charakteristisch für die Modellierung von Prozessvarianten ist, dass eine bestimmte Variante aus einem bestehenden (Referenz-) Prozessmodell abgeleitet wird. Dementsprechend werden die Varianten eines Prozessmodells nicht frei entworfen, sondern stellen Anpassungen bzw. Weiterentwicklungen des Modells des betreffenden Prozesstyps dar (z.B. Werkstatt- oder Änderungsmanagementprozess). Der Ein-Modell-Ansatz etwa stellt hierfür eine Extremform dar (vgl. Abschnitt 4.2), bei der alle Varianten innerhalb eines Prozessmodells abgebildet werden. Beim Mehr-Modell-Ansatz wiederum werden existierende Prozessmodelle wiederverwendet, um Prozessvarianten abzuleiten. Das heißt, Prozessvarianten werden durch Kopieren eines existierenden Prozessmodells und anschließendem Anpassen dieser Kopie konfiguriert (vgl. Abschnitt 4.2). Folglich resultiert ein separates, an die gegebenen Rahmenbedingungen angepasstes Prozessmodell. Dieses repräsentiert eine spezifische Variante des Prozesstyps. Im Folgenden bezeichnen wir dieses Vorgehen als `Konfiguration` einer Variante auf Basis eines gegebenen Prozessmodells. Beispiel 4.3 zeigt dieses Vorgehen.

Beispiel 4.3 (Konfiguration von Varianten eines Ausgangsprozesses) *Abbildung 4.4 zeigt die übliche Vorgehensweise zur Konfiguration einer Variante aus einem gegebenen Prozess: Gegeben ist ein stark vereinfachter Werkstattprozess zur Abwicklung einer Kfz-Reparatur (vgl. Abbildung 4.4a). Ausgehend von diesem Werkstattprozess erstellen wir ein angepasstes Prozessmodell zur Abbildung einer konkreten Prozessvariante. In dieser soll keine Wartung vorgenommen werden, allerdings ist eine zusätzliche Abschlussprüfung (z.B. nach dem Vier-Augen-Prinzip) durchzuführen (vgl. Abbildung 4.4c). Bezogen auf die Modellebene bedeutet dies, dass die Aktivität* `Wartung` *des Fahrzeugs aus dem Ausgangsmodell entfernt und die Aktivität* `Prüfung` *zwischen den Aktivitäten* `Reparatur` *und* `Fzg.Übergabe` *eingefügt werden muss. Entsprechende variantenspezifische Abweichungen beschreiben wir in Abbildung 4.4b.*

Die Wiederverwendung von Teilen eines Ausgangsmodells hat den Effekt, dass abgeleitete Variantenmodelle eine mehr oder weniger starke Ähnlichkeit zu ihrem Ausgangsmodell aufweisen. Diese Ähnlichkeit oder *Modelldifferenz* kann prinzipiell durch variantenspezifische Abweichungen beschrieben werden (vgl. Abbildung 4.4) [LRWe08, WRR08]. Das heißt, eine konkrete Prozessvariante kann mit Hilfe des Ausgangsmodells und dessen variantenspezifischen Anpassungen beschrieben werden.

Eine wesentliche Erkenntnis hieraus ist, dass das anhand von Beispiel 4.3 skizzierte Vorgehen generalisierbar ist. Das heißt, prinzipiell können alle Varianten eines Prozesstyps durch Kopieren und Anpassen genau eines Prozessmodells entstehen. Provop greift diese Erkenntnis auf. Grundidee ist, die Modellierung bzw. Konfiguration der Varianten eines Prozesstyps durch Beschreibung des Ausgangsmodells und der variantenspezifischen Abweichungen zu ermöglichen.

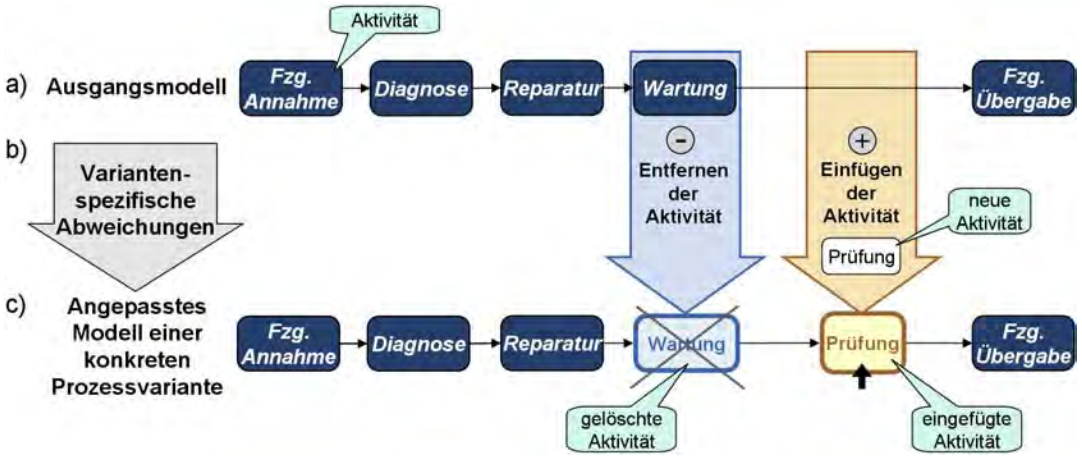


Abbildung 4.4: Ableitung einer Prozessvariante auf Basis eines Ausgangsmodells

4.3.2 Durchgängigkeit des Provop-Ansatzes

Eine wesentliche Anforderung ist, ein durchgängiges Konzept für das Management von Prozessvarianten zu entwickeln (vgl. Abschnitt 2.3). Deshalb sollen alle Phasen des Prozesslebenszyklus in die Konzeptentwicklung einbezogen werden. Abbildung 4.5 gibt eine Übersicht der vier von Provop abgedeckten Phasen des Lebenszyklus – beginnend mit der Modellierung von Prozessvarianten, über deren Konfiguration und Ausführung in einem WfMS, bis hin zu ihrer Evolution. Ziel von Provop ist es, für alle Phasen ein angemessenes Lösungskonzept zu entwickeln. Die spezifischen Lösungskonzepte für die einzelnen Phasen müssen so aufeinander abgestimmt sein, dass keine Einschränkungen im Lebenszyklus entstehen.

Abbildung 4.5: Übersicht über den Provop-Prozess-Lebenszyklus

4.3.2.1 Modellierung von Prozessvarianten

Die erste Phase des Lebenszyklus betrifft die Modellierung von Prozessvarianten. Abbildung 4.6 gibt einen Überblick über die wichtigsten Aspekte dieser Phase.

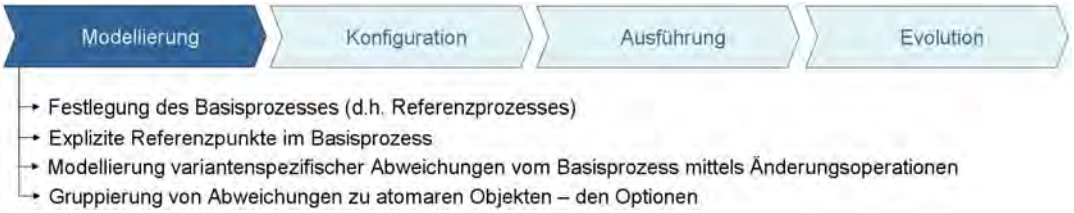


Abbildung 4.6: Modellierung von Prozessvarianten

Die Modellierung von Prozessvarianten basiert in Provop auf dem Ansatz, dass alle Varianten eines bestimmten Prozesstyps mit Hilfe eines gemeinsamen Ausgangsmodells sowie variantenspezifischer Abweichungen beschrieben werden [HBR08b]. Ein solches Ausgangsmodell nennen wir *Basisprozess*. Er stellt im Prinzip ein normales Prozessmodell dar (vgl. Definition 3.1) und kann nach unterschiedlichen Gesichtspunkten festgelegt werden. So kann damit ein Referenzprozess gemeint sein, der den typischen Standardablauf für den vorliegenden

Prozesstyp vorgibt. Ebenso denkbar sind konstruierte Basisprozesse, die z.B. als Schnitt- oder Obermenge aller konstruierbaren Prozessvarianten definiert werden.

Um eine hohe Transparenz der Prozessvarianten zu erzielen, werden in Provop variantenspezifische Abweichungen, welche die Modelldifferenz zwischen dem Basisprozess und der gewünschten Prozessvariante darstellen, explizit beschrieben. Dazu gibt es im Prinzip zwei Möglichkeiten: Wir können die Anpassungen durch höherwertige Änderungsoperationen (z.B. Einfügen, Löschen oder Verschieben von Fragmenten in einem Prozessmodell) oder durch primitive Graphoperationen (z.B. Einfügen und Entfernen von Kanten und Knoten) angeben. Im Vergleich zu höherwertigen Änderungsoperationen beziehen sich primitive Graphoperationen ausschließlich auf einzelne Graphelemente wie Knoten oder Kanten und nicht auf Aktivitäten oder gar Prozessfragmente (vgl. Abschnitt 3.2). In Provop unterstützen wir höherwertige Änderungsoperationen für ganze Prozessfragmente. Auf diese Weise wird zum einen der Modellierungsaufwand signifikant reduziert, da nun nicht für jedes einzelne Prozesselement eines Prozessfragments eine separate Änderungsoperation angegeben werden muss. Zum anderen erleichtern höherwertige Änderungsoperationen die Modellierung, da der Variantenmodellierer von primitiven Graphoperationen abstrahieren kann. Die Umsetzung von höherwertigen in primitive Änderungsoperationen bleibt verborgen.

Variantenspezifische Änderungsoperationen werden in Provop mit Bezug auf den gegebenen Basisprozess modelliert. Sie referenzieren explizite Positionen im Basisprozess, die mit Hilfe sog. *Aufsetzpunkte* markiert werden. Aufsetzpunkte stellen somit Elemente dar, die eine stabile Referenzierung einer Position im Basisprozess erlauben.

Beispiel 4.4 illustriert diesen Ansatz.

Beispiel 4.4 (Varianten des Werkstattprozesses) *Abbildung 4.7a zeigt den stark vereinfachten Standard-Werkstattprozess (vgl. Abschnitt 2.1). Er wird als Basisprozess festgelegt. Die Bereitstellung der beiden Änderungsoperationen* `Entferne Aktivität Wartung` *und* `Einfügen Aktivität Prüfung` *(vgl. Abbildung 4.7b) ermöglicht die Ableitung von insgesamt vier Prozessvarianten (vgl. Abbildung 4.7c). Im Beispiel entspricht Variante 4 dem Basisprozess selbst, der somit ebenfalls eine konkrete Prozessvariante darstellt.*

In der Praxis kann die Zahl an Änderungsoperationen ggf. sehr groß werden, etwa dann, wenn viele verschiedene Prozessvarianten abzubilden sind. In diesem Fall sind zur Ableitung einer bestimmten Prozessvariante meist mehrere Änderungsoperationen gemeinsam anzuwenden. Zur Abstraktion und besseren Handhabbarkeit erlaubt es Provop solche Änderungsoperationen zu größeren, atomaren Objekten zu gruppieren. Diese Gruppen bezeichnen wir als *Optionen* (vgl. Beispiel 4.5). Diese Art der Modellierung von Prozessvarianten durch Optionen hat auch zum Projektnamen Provop geführt – **Pro**zessvarianten mittels **Op**tionen.

Beispiel 4.5 (Option mit Änderungsoperationen) *Abbildung 4.8 zeigt ein Beispiel für eine Option mit zwei Änderungsoperationen. Die erste Änderungsoperation beschreibt eine Löschoperation. Alle Prozesselemente, welche zwischen* `Aufsetzpunkt 1` *und* `Aufsetzpunkt 2` *im Basisprozess liegen, werden bei Anwendung dieser Änderungsoperation entfernt. Die zweite Änderungsoperation nimmt das Einfügen der Aktivität* `Prüfung` *vor. Diese wird bei Anwendung der Änderungsoperation zwischen* `Aufsetzpunkt 2` *und* `Aufsetzpunkt 3` *in den Basisprozess eingefügt.*

Eine Option umfasst solche Änderungsoperationen, die stets gemeinsam auf den Basisprozess angewendet werden sollen. Gruppieren wir Änderungsoperationen jedoch nach diesem

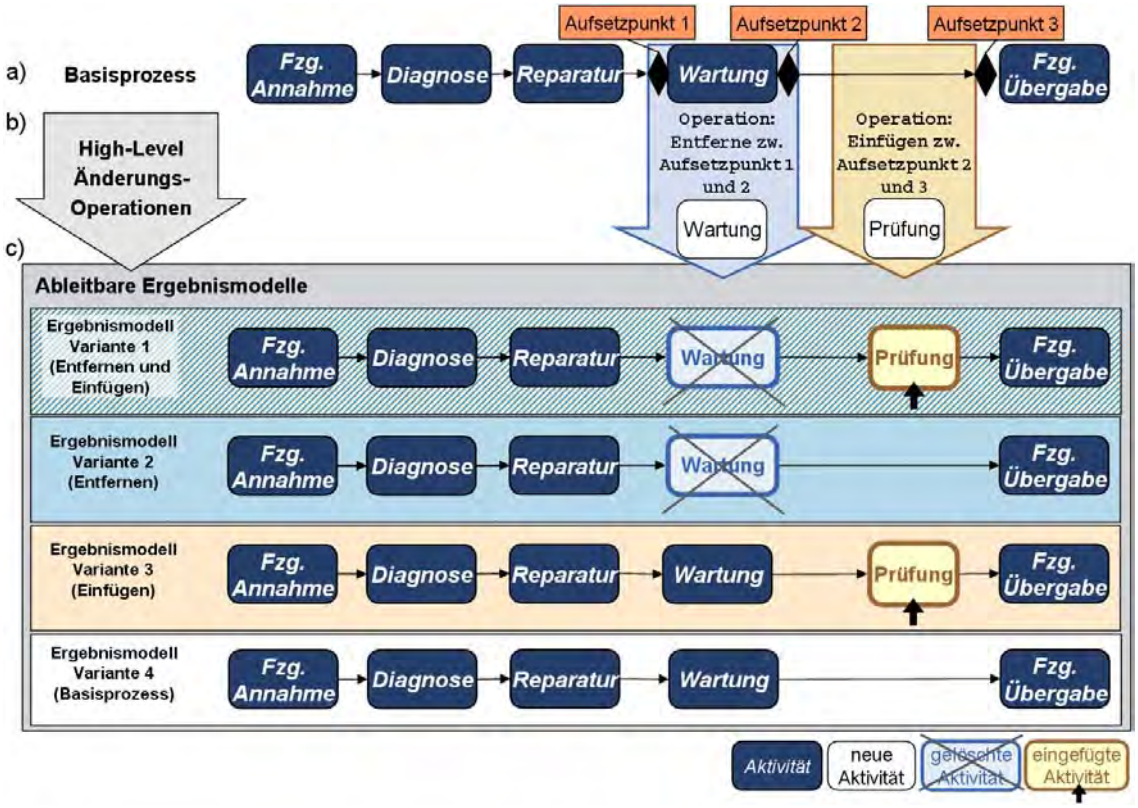


Abbildung 4.7: Komponenten des Provop-Ansatzes

Kriterium, erhalten wir unter Umständen sehr große Optionen. Änderungsoperationen, welche für verschiedene Prozessvarianten benötigt werden, müssten dann redundant modelliert werden. Dies ist i.A. nicht gewünscht. Daher definieren wir Optionen so, dass sie gemeinsam mit anderen Optionen zur Konfiguration einer Prozessvariante angewendet werden können. Dies erlaubt vor allem ein Gruppieren von Änderungsoperationen mit minimalen Redundanzen. Durch Auswahl einer Teilmenge der vorliegenden Optionen und deren Anwendung auf den Basisprozess, können wir die *Ergebnismodelle* der Prozessvarianten ableiten. Wenden wir z.B. die in Abbildung 4.8 modellierte Option auf den in Abbildung 4.7a dargestellten Basisprozess an, erhalten wir das Ergebnismodell der Variante 4 aus Abbildung 4.7c.

4.3.2.2 Automatische Konfiguration von Prozessvarianten

Die zweite Phase des Lebenszyklus betrifft die Konfiguration von Prozessvarianten. Abbildung 4.9 gibt einen Überblick zu hier relevanten Aspekten.

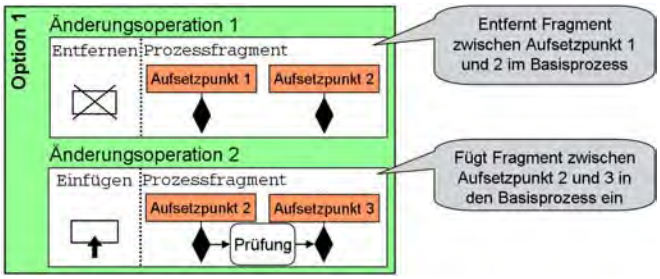


Abbildung 4.8: Beispiel für die Gruppierung von Änderungsoperationen

Abbildung 4.9: Konfiguration von Prozessvarianten

Durch Auswahl bestimmter Optionen und deren sequentiellen Anwendung auf den Basisprozess, kann der Variantenverantwortliche eine spezifische Prozessvariante *konfigurieren*. Das heißt er kann das entsprechende Ergebnismodell dieser Variante aus dem Basisprozess ableiten. Die Summe aller relevanten und auf diese Weise ableitbaren Prozessvarianten bezeichnen wir als *Prozessfamilie* des vorliegenden Prozesstyps.

Zur Gewährleistung einer korrekten Konfiguration von Prozessvarianten bietet der Provop-Ansatz verschiedene Konzepte an. Neben der rein manuellen Auswahl von Optionen kann der Variantenverantwortliche auch eine kontextabhängige Konfiguration anstoßen [HBR08c]. Dazu werden zunächst die wesentlichen Einflussfaktoren (z.B. Qualitätsrelevanz, Umsetzungskosten), welche die Auswahl bzw. Konfiguration der Variante bestimmen, in einem *Kontextmodell* gespeichert. Für jeden Faktor, im Folgenden von uns als *Kontextvariable* bezeichnet, definieren wir einen zugehörigen Wertebereich. Wir gehen davon aus, dass der Wertebereich von Kontextvariablen diskret und endlich ist.[3] Mit Hilfe des Kontextmodells kann der Kontext erfasst und dokumentiert werden.

Beispiel 4.6 (Kontextmodell des Werkstattprozesses) *Tabelle 4.1 zeigt beispielhaft einen Ausschnitt aus dem Kontextmodell des Werkstattprozesses. Für ein besseres Verständnis geben wir neben dem Namen und dem Wertebereich der Kontextvariable auch deren (optionale) Beschreibung an.*

Tabelle 4.1: Ausschnitt aus dem Kontextmodell des Werkstattprozesses

Kontextvariable	Beschreibung	Wertebereich
Wartung	Gibt an, ob die Wartung des Fahrzeugs von der Werkstatt vorgenommen wird oder nicht.	Ja, Nein
kritisch	Gibt an, ob es sich um einen kritischen Schadensfall handelt und somit eine Prüfung nach dem Vier-Augen-Prinzip zur Qualitätssicherung erforderlich ist.	Ja, Nein
Auslastung	Gibt den Grad der Auslastung einer Werkstatt an (entspricht dem Verhältnis Kapazität-Auftragslage).	niedrig, mittel, hoch

Basierend auf dem Kontextmodell kann der Variantenmodellierer beschreiben, wann eine bestimmte Option anzuwenden ist. Dies wird in Provop in Form sog. *Kontextbedingungen* realisiert. Diese Kontextbedingungen können automatisch für die gegebenen Werte der Kontextvariablen ausgewertet werden. Alle Optionen, deren Kontextbedingung im aktuellen Kontext erfüllt ist, werden anschließend auf den Basisprozess angewendet.

In der Praxis können bzw. dürfen nicht immer alle möglichen Werte-Kombinationen von Kontextvariablen auftreten. Solche Werte-Kombinationen kann der Designer des Kontextmodells ausschließen, d.h. als ungültig definieren. Mit diesem Ansatz wird insbesondere auch die Anzahl möglicher Prozessvarianten reduziert, da nun nicht mehr alle Kombinationen

[3] Im Allgemeinen sind diskrete und endliche Wertebereiche für Kontextvariablen in der Praxis ausreichend. Aus dieser Vorgabe resultiert somit in der Regel keine Einschränkung der Mächtigkeit des Lösungsansatzes oder seiner Überführbarkeit in die Praxis.

von Optionen möglich sind, sondern nur solche Optionen gemeinsam angewendet werden dürfen, die im gleichen Kontext relevant sind. In Beispiel 4.7 erläutern wir dieses Vorgehen anhand eines einfachen Szenarios aus der Domäne Werkstattprozess.

Beispiel 4.7 (Kontextabhängige Konfiguration von Prozessvarianten) *Abbildung 4.10 zeigt ein Beispiel für die kontextabhängige Konfiguration von Prozessvarianten. Der stark vereinfachte Werkstattprozess wird hier als Basisprozess festgelegt. Explizite Aufsetzpunkte zeigen an, wo im Basisprozess überhaupt Änderungen vorgenommen werden können. Das Kontextmodell umfasst die Kontextvariablen* `Wartung` *und* `kritisch`*, die jeweils die Werte* `Ja` *oder* `Nein` *annehmen können. Die Kombination* `Wartung = Nein UND kritisch = Nein` *ist als ungültig definiert. Die beiden modellierten Optionen beinhalten je eine Änderungsoperation, welche Aufsetzpunkte des Basisprozesses referenzieren. Sie führen ein Löschen der Aktivität* `Wartung` *(Option 1) sowie ein Einfügen der Aktivität* `Prüfung` *(Option 2) durch. Mit Hilfe der verknüpften Kontextbedingungen, die auf Basis der Kontextvariablen im Kontextmodell erstellt wurden, kann für einen gegebenen Kontext das entsprechende Ergebnismodell abgeleitet werden. Die Prozessfamilie besteht in diesem Fall aus drei Modellen; zu beachten ist, dass die Wertekombination* `Wartung = Nein UND kritisch = Nein` *ungültig ist, d.h. hier ist kein Ergebnismodell erforderlich.*

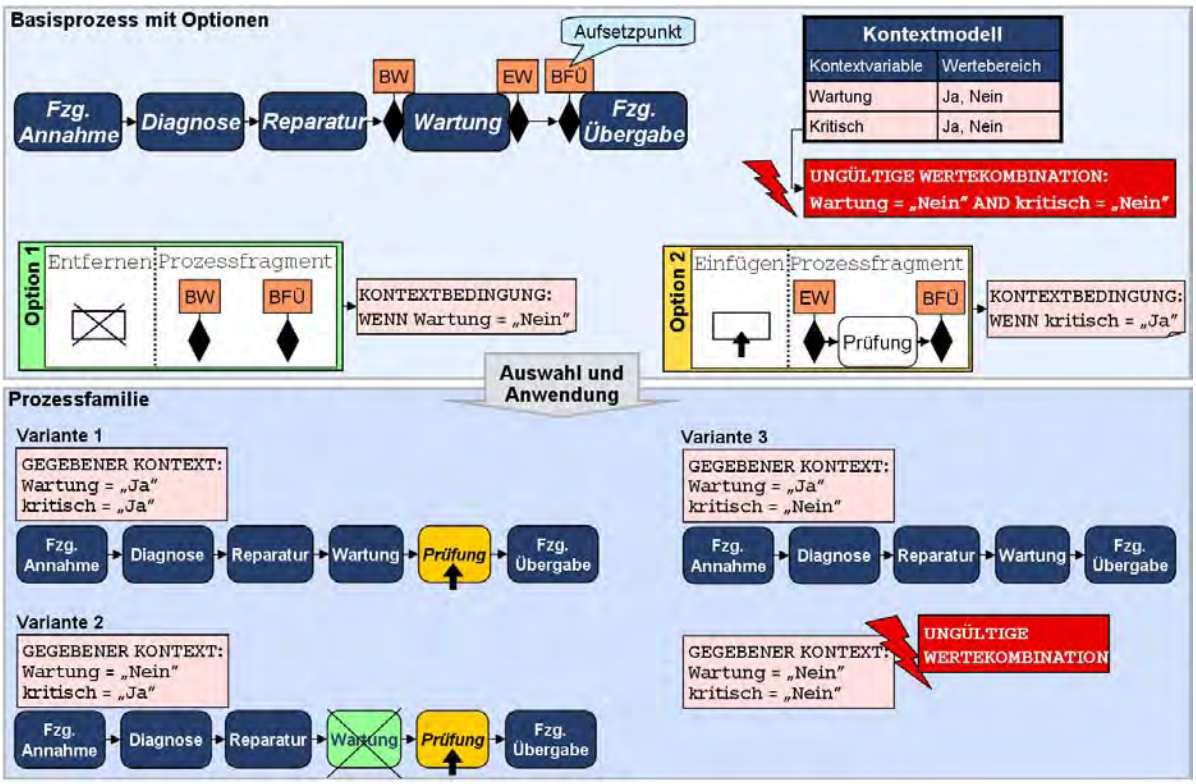


Abbildung 4.10: Kontextabhängige Konfiguration von Prozessvarianten

Ein weiterer Aspekt, der orthogonal zur Berücksichtigung des Kontextes bei der Konfiguration von Prozessvarianten ist, betrifft semantische und strukturelle Abhängigkeiten zwischen Änderungsoperationen bzw. Optionen. So kann es vorkommen, dass Änderungsoperationen aufeinander aufbauen. Bspw. kann eine Option X eine Aktivität einfügen, die anschließend von einer Änderungsoperation in Option Y modifiziert wird. Das heißt Option Y *impliziert* die Anwendung von Option X. Zur Beschreibung solcher Abhängigkeiten können noch weitere Beziehungstypen identifiziert werden, etwa der wechselseitige Ausschluss von Optionen.

Die Erfassung solcher Beziehungen und ihre Auswertung bei der Konfiguration einer Prozessvariante stellen die semantisch und strukturell korrekte Ableitung eines Ergebnismodells sicher. Außerdem reduzieren Beschränkungen die Anzahl möglicher Prozessvarianten, wie das Beispiel 4.8 zeigt. Eine wichtige Anforderung ist es daher, solche Abhängigkeiten explizit abzubilden. Um diese zu erfüllen, erlaubt Provop die explizite Modellierung von *Optionsbeziehungen*.

Beispiel 4.8 (Beziehungen zwischen Optionen) *Mit Hilfe einer Optionsbeziehung kann die gemeinsame Anwendung zweier Optionen ausgeschlossen werden. Durch den wechselseitigen Ausschluss der Optionen 1 und 2 aus Abbildung 4.11, sind in der Prozessfamilie nur drei Ergebnismodelle enthalten.*

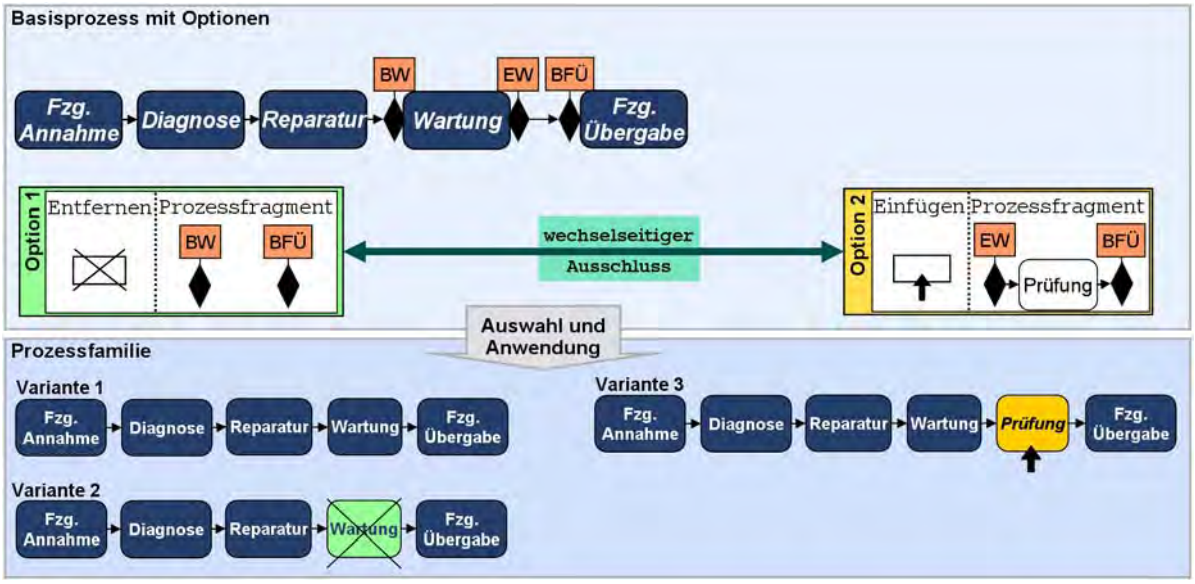


Abbildung 4.11: Berücksichtigung von Optionsbeziehungen bei der Konfiguration

4.3.2.3 Ausführung von Prozessvarianten

Die dritte Phase des Lebenszyklus betrifft die Ausführung von Prozessvarianten (vgl. Abbildung 4.12).

Abbildung 4.12: Ausführung von Prozessvarianten

Mit Hilfe eines WfMS können (formale) Prozessmodelle rechnergestützt ausgeführt werden. Dazu ist es erforderlich, ein fachlich beschriebenes Prozessmodell in ein technisches Workflow-Modell zu transformieren. Dazu sind i.A. zusätzliche Informationen erforderlich, etwa verwendete Masken und Programmaufrufe. Die Problematik der Transformation eines Geschäftsprozessmodells in ein Workflow-Modell wird aktuell in der Literatur diskutiert [DvdA04, GLK+07]. Da die Durchführung einer Transformation vom zugrunde liegenden

Prozess-Metamodell abhängt und wir uns nicht auf ein spezifisches Prozess-Metamodell festgelegt haben, fokussieren wir uns in dieser Arbeit auf die Betrachtung, wie die konfigurierten Ergebnismodelle flexibel in einem WfMS ausgeführt werden können. Die interessanten Problemstellungen ergeben sich durch Zulassen von Dynamik bei der Ausführung von Prozessvarianten. Liegen z.B. zum Zeitpunkt der Konfiguration einer Prozessvariante nicht für alle Kontextvariablen entsprechende Werte vor oder können sich zur Laufzeit dynamisch Änderungen an den gegebenen Werten ergeben, kann die konkrete Prozessvariante auch erst zur Laufzeit determiniert werden. Eine solche *dynamische Konfiguration* wird in Provop durch die dynamische Anwendung von Optionen realisiert. Sie basiert im Prinzip auf der Verwendung alternativer Verzweigungen im Ergebnismodell. Die Kantenbedingungen dieser Verzweigungen ergeben sich entsprechend aus den Kontextbedingungen der Optionen. Indem an den Verzweigungen für oder gegen die Anwendung einer Option entschieden wird, können wir die konkrete Prozessvariante zur Laufzeit bestimmen und zwar unter Berücksichtigung der aktuellen Werte der Kontextvariablen.

Beispiel 4.9 (Dynamische Konfiguration einer Prozessvariante) *In der Werkstattdomäne verhält sich die Kontextvariable* `Auslastung` *typischerweise dynamisch. Das heißt der Wert dieser Variable ist abhängig von der Anzahl der Aufträge einer Werkstatt und kann sich somit des Öfteren ändern. Optionen, deren Kontextbedingung auf dieser Kontextvariable basiert, werden dynamisch auf den Basisprozess angewendet, um so zur Laufzeit flexibel auf Änderungen der Werte von Kontextvariablen reagieren zu können. Abbildung 4.13 zeigt das Ergebnismodell, das aus der dynamischen Anwendung der Optionen 1 und 2 entsteht. Auf Basis des aktuellen Wertes für die dynamische Kontextvariable* `Auslastung` *und der entsprechenden Kontextbedingungen der Optionen, kann zur Laufzeit entschieden werden, ob die Aktivitäten* `Wartung` *und* `Prüfung` *ausgeführt werden oder nicht.*

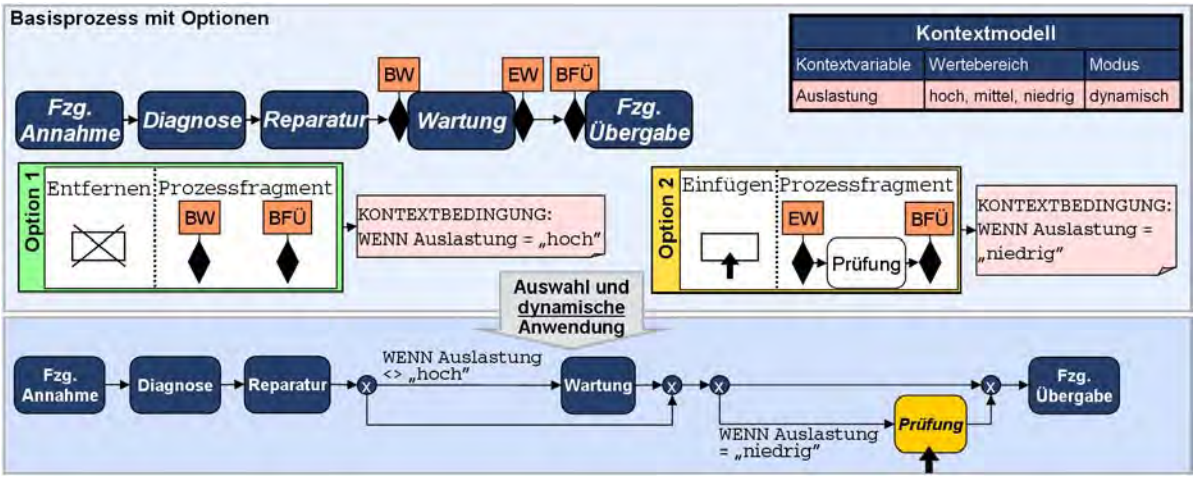


Abbildung 4.13: Dynamische Konfiguration von Prozessvarianten

Ein weiterer Aspekt der Ausführung von Prozessvarianten ist die Einhaltung explizit definierter Optionsbeziehungen zwischen dynamisch angewandten Optionen. Werden entsprechende Beziehungen definiert, z.B. ein wechselseitiger Ausschluss zwischen zwei Optionen, müssen wir natürlich auch zur Laufzeit sicherstellen, dass diese strikt eingehalten werden und die beiden Optionen nicht gemeinsam zur Anwendung kommen. Damit wir während der Ausführung von Prozessvarianten keine Optionsbeziehungen verletzen, bietet Provop eine entsprechende Prüfung an. Diese wird durchgeführt, sobald über die Ausführung einer dynamischen Option zur Laufzeit entschieden wird. Auf diese Art und Weise wird die semantische und strukturelle Korrektheit der Prozessvarianten auch zur Laufzeit gewährleistet.

4.3.2.4 Evolution von Prozessvarianten

Die vierte und letzte Phase des Lebenszyklus betrifft die Evolution von Prozessvarianten (vgl. Abbildung 4.14). Hier ist es zum einen erforderlich, die Provop-Komponenten (wie *Basisprozess* und *Optionen*) aufgrund entsprechender fachlicher Anforderungen anpassen zu können. Zum anderen ist die einfache Pfleg- und Wartbarkeit der Prozessvarianten relevant. Dazu müssen durch geeignete Refactorings vorhandene Redundanzen minimiert und nicht verwendete Objekte entfernt werden.

Abbildung 4.14: Evolution von Prozessvarianten

Bei der Anpassung einzelner Provop-Komponenten, etwa Änderungen am Basisprozess oder an einzelnen Optionen, müssen wir deren Abhängigkeiten zu anderen Komponenten berücksichtigen. Passen wir z.B. den Basisprozess an, ist auch zu prüfen, ob die auf seiner Grundlage modellierten Optionen noch korrekt anwendbar bleiben. Das heißt, wir müssen feststellen ob die Anwendung der Optionen immer noch zu korrekten Ergebnismodellen führt. Provop bietet dazu entsprechende Prüfungen und Modellierungsmethoden an. Ein wesentlicher Vorteil ist, dass durch Anpassen des Basisprozesses das gleichzeitige Anpassen aller Prozessvarianten ermöglicht wird. Diese Vorgehensweise reduziert Anpassungsaufwände deutlich.

Das Refactoring von Basisprozess bzw. Optionen hat zum Ziel, den Umgang mit den Prozessvarianten zu vereinfachen sowie deren Pfleg- und Wartbarkeit zu verbessern. Zur Identifikation geeigneter Refactorings sind verschiedene Ansätze möglich. Neben der Auswertung der modellierten Optionen und des Basisprozesses, können wir auch Ausführungsprotokolle bzw. -historien betrachten.[4] Auf Basis der gewonnenen Informationen kann der Basisprozess neu gestaltet werden, so dass er z.B. die minimale durchschnittliche Modelldistanz zu relevanten Prozessvarianten abbildet [LRWb08, LRWe08]. Auf diese Weise ist eine minimale Anzahl an Änderungsoperationen zur Ableitung aller Prozessvarianten aus dem Basisprozess erforderlich. Zur einfacheren Handhabung der modellierten Optionen können redundante Änderungsoperationen identifiziert und eliminiert werden. Redundanzen können z.B. durch ein intelligentes Neugruppieren von Änderungsoperationen zu Optionen vermieden werden. Dieses Vorgehen erläutert Beispiel 4.10.

Beispiel 4.10 (Gruppierung von Änderungsoperationen zu Optionen) *Zur Abbildung dreier Prozessvarianten des Werkstattprozesses sind vier Änderungsoperationen erforderlich. In einem* ***feingranularen Ansatz*** *werden diese vier Änderungsoperationen, wie die erste Spalte in Tabelle 4.2 zeigt, jeweils in einzelnen Optionen modelliert. Auf diese Weise entstehen bei sehr vielen Änderungsoperationen entsprechend viele Optionen. Im Vergleich dazu werden die Änderungsoperationen in einem* ***grobgranularen Ansatz*** *so gruppiert, dass alle Änderungsoperationen, die zur Konfiguration einer bestimmten Prozessvariante benötigt werden, jeweils zu einer Option zusammengefasst werden. Auf diese Weise sind hier die Änderungsoperationen* `DELETE Wartung` *und* `INSERT Prüfung` *redundant. Eine Neugruppierung von Änderungsoperationen hätte zur Folge, dass die Optionen, wie in Spalte*

[4] Während der Ausführung eines Workflows kann das WfMS protokollieren, wie bzw. welche Aktivitäten des Prozessmodells ausgeführt werden. Die so entstehenden Log-Dateien können dann durch verschiedene „Process Mining"-Techniken ausgewertet werden [vdAGRR06, vdAvDH+03].

drei in Tabelle 4.2 gezeigt, gruppiert würden. Wir erzielen auf diese Weise eine minimale Redundanz von Änderungsoperationen.

Tabelle 4.2: Optimierte Gruppierung von Änderungsoperationen zu Optionen

Feingranular:	Grobgranular:	Optimale Granularität:
Option 1: MODIFY Diagnose	Option 1: MODIFY Diagnose MODIFY Reparatur DELETE Wartung	Option 1: MODIFY Diagnose MODIFY Reparatur
Option 2: MODIFY Reparatur	Option 2: INSERT Prüfung	Option 2: DELETE Wartung
Option 3: DELETE Wartung	Option 3: DELETE Wartung INSERT Prüfung	Option 3: INSERT Prüfung
Option 4: INSERT Prüfung		
# Optionen = 4 # Operationen = 4	# Optionen = 3 # Operationen = 6	# Optionen = 3 # Operationen = 4

4.3.2.5 Korrektheit von Prozessvarianten

Ein wichtiger Aspekt für das Management von Prozessvarianten ist die Gewährleistung ihrer Korrektheit. Diese Problemstellung betrifft alle Phasen des Lebenszyklus (vgl. Abbildung 4.15): In der Phase der Modellierung müssen die Optionen so erstellt werden, dass diese korrekt auf den Basisprozess anwendbar sind. Zum Beispiel müssen alle referenzierten Prozesselemente tatsächlich im Basisprozess existieren. Des Weiteren müssen die konfigurierbaren Prozessvarianten korrekt sein, um anschließend in einem WfMS ausführbar zu sein. Diese Korrektheit betrifft sowohl strukturelle Aspekte (z.B. Ausschluss gewisser Prozessgraphmuster) als auch das Verhalten von Modellinstanzen (z.B. Verklemmungsfreiheit). Bei dynamischen Änderungen an einer Prozessvariante zur Laufzeit oder bei Änderungen im Sinne einer Weiterentwicklung bzw. Optimierung ist die Korrektheit der Prozessfamilie ebenfalls zu gewährleisten.

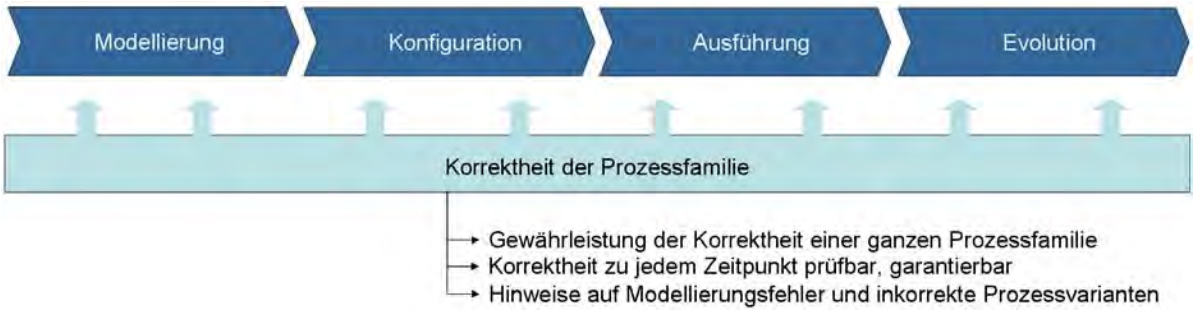


Abbildung 4.15: Korrektheit von Prozessvarianten

Die Bewertung eines Prozessmodells als inkorrekt resultiert aus den Korrektheitskriterien des jeweils zugrundeliegenden Prozess-Metamodells.[5] In Provop haben wir beispielhaft Kriterien definiert, nach denen wir die Korrektheit eines Prozessmodells beurteilen (vgl. Abschnitt 3.3).

[5] Als Beispiel seien hier die Blockstrukturierung bei ADEPT [DR09] und die bipartite Graphstruktur bei Petrinetzen [Pet81] gegeben.

Zu klären ist vor allem, wie sich inkorrekte Prozessmodelle erkennen lassen. Dabei geht es weniger um die spezifischen Algorithmen zur Identifikation struktureller Eigenschaften (z.B. Zyklenfreiheit), sondern um die Gewährleistung der Korrektheit einer Prozessfamilie in einer effizienten und zuverlässigen Weise. Zu diesem Zweck haben wir das Provop-Rahmenwerk zur Korrektheitsprüfung entwickelt [HBR09c]. Dieses gewährleistet die Korrektheit aller Varianten in einer Prozessfamilie und ist auf alle Prozess-Metamodelle übertragbar. Es ermöglicht uns sowohl statisch als auch dynamisch konfigurierte Prozessvarianten zum Zeitpunkt der Konfiguration abzusichern. Darüber hinaus überwacht dieses Rahmenwerk die Einhaltung von Optionsbeziehungen zur Laufzeit. Auf diese Weise können wir die stabile und korrekte Ausführung der Variantenmodelle gewährleisten. Details hierzu geben wir in Kapitel 7.

4.3.2.6 Provop-Rollenmodell

Wie Abbildung 4.16 zeigt, sind in den Lebenszyklus eines Prozessmodells verschiedene Personen mit bestimmten Rollen involviert.

- **Modellierer:** Untergliedert sich in drei Rollen.[6]
  - *Basisprozessmodellierer:* Er erstellt den Basisprozess und Aufsetzpunkte.
  - *Kontextmodellierer:* Er erstellt das Kontextmodell aus den Kontextvariablen und den zugehörigen Wertebereichen, sowie die Kontextregeln zur Beschreibung ungültiger Kontexte. Er legt ggf. auch die Kontextbedingungen der Optionen fest.
  - *Variantenmodellierer:* Er modelliert Änderungsoperationen und Optionen, sowie Beziehungen zwischen Optionen. Er legt ggf. auch die Kontextbedingungen der Optionen fest.
- **Variantenverantwortlicher:** Er konfiguriert die konkreten Prozessvarianten durch manuelle oder kontextbasierte Auswahl von Optionen sowie deren Anwendung auf den gegebenen Basisprozess.
- **Implementierer:** Er implementiert einen Prozess in einem IT-System bzw. erzeugt den ausführbaren Workflow.
- **Betreiber:** Er betreibt das IT-System bzw. den Workflow während der Ausführung.
- **Endanwender:** Er verwendet das IT-System bzw. ist Teilnehmer des Workflows.

Natürlich kann eine Person auch mehrere der genannten Rollen gleichzeitig ausüben. Zum Beispiel können der Variantenmodellierer und der Variantenverantwortliche in Personalunion auftreten. Auch die spezifischen Aufgaben der einzelnen Rollen müssen nicht scharf voneinander getrennt werden. So kann es notwendig sein, dass ein Variantenmodellierer auch Aufsetzpunkte im Basisprozess definieren darf.

Es ist denkbar, das Provop-Rollenmodell um ein Rechtesystem zu ergänzen und dem Variantenmodellierer dann sowohl lesenden als auch schreibenden Zugriff auf den Basisprozess zu erlauben. Zugriffsrechte und ein generelles Rechtesystem stehen allerdings nicht im Fokus dieser Arbeit und werden hier deshalb nicht weiter betrachtet.

[6] Wir verwenden im Folgenden die Kurzform „Modellierer“, wenn die konkrete Rolle aus dem Kontext klar ist.

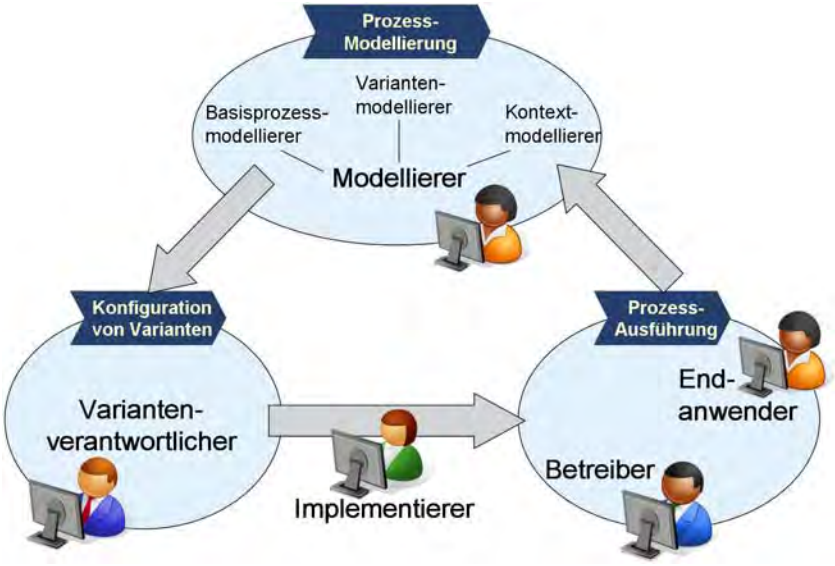


Abbildung 4.16: Rollenmodell in Provop

4.3.3 Weitere Aspekte

4.3.3.1 Darstellung von Prozessvarianten

Ein weiterer relevanter Aspekt betrifft die Darstellung von Provop-Komponenten innerhalb eines Modellierungswerkzeugs. In Provop haben wir verschiedene Konzepte zur Visualisierung von Optionen entwickelt [HBR08b]. Neben verschiedenen Konzepten zur Positionierung der Optionen in Relation zum Basisprozess betrachten wir insbesondere Ansätze zur Informationsreduktion. So können Optionen vollständig angezeigt werden, d.h. inklusive aller entsprechenden Informationen (z.B. Kontextbedingung, Name der Option, Änderungsoperationen und Optionsbeziehungen) oder komprimiert, indem bestimmte Informationen auf Wunsch des Variantenmodellierers flexibel ein- bzw. ausgeblendet werden. Unterstützende Darstellungsformen bilden zudem Ansichten, die durch geeignete Hervorhebungsmerkmale wie Farbe, Rahmen und Beschriftung, die Differenz des Ergebnismodells zum Basisprozess aufzeigen, oder die Änderungen einzelner Optionen bzw. Änderungsoperationen betonen.

4.3.3.2 Skalierbarkeit des Ansatzes

Ein weiterer Aspekt betrifft die Skalierbarkeit des Provop-Ansatzes. Unabhängig von der Anzahl abzubildender Prozessvarianten muss ein einfacher Umgang mit Varianten möglich sein. Das heißt, der Ansatz muss sowohl bei kleiner Anzahl von Prozessvarianten, mit geringer Komplexität der Variantenmodelle, als auch für eine große Anzahl von Prozessvarianten, mit komplexen Variantenmodellen, eingesetzt werden können.

Um eine gute Skalierbarkeit zu erzielen, haben wir in Provop verschiedene Konzepte entwickelt, mit deren Hilfe Prozessvarianten leicht zu handhaben sind. Diese sind z.B. das Gruppieren von Änderungsoperationen zu Optionen, die Einführung expliziter Positionen im Basisprozess in Form von Aufsetzpunkten sowie die Definition eines Kontextmodells. Durch Modellierung und Kombination mehrerer Optionen und deren Bezug auf einen gemeinsamen Basisprozess ist zudem die Wiederverwendbarkeit identischer Änderungsoperationen möglich. Auf diese Weise ist auch bei großer Anzahl an Prozessvarianten eine einfache Handhabbarkeit gegeben. Das Provop-Rahmenwerk zur Korrektheitsprüfung stellt außerdem sicher, dass die Prozessfamilie und damit jede ableitbare Prozessvariante, korrekt ist. Für den

Aspekt der kontextbasierten Konfiguration bietet Provop ein intuitives, zentrales Kontextmodell an, welches die übersichtliche Erfassung und Verwaltung auch bei einer großen Zahl von Kontextvariablen bzw. großen Wertebereichen dieser Variablen erlaubt. Des Weiteren unterstützen verschiedene Darstellungskonzepte die Vergleichbarkeit der Prozessvarianten und ermöglichen den einfachen Umgang mit einer großen Prozessfamilie bzw. vielen Prozessvarianten.

4.4 Diskussion

Das Ziel von Provop ist es, die Nachteile der konventionellen Methoden für das Management von Prozessvarianten, d.h. die eingangs dieses Kapitels diskutierten Nachteile des Mehr- und Ein-Modell-Ansatzes (vgl. Abschnitte 4.2.1 und 4.2.2), zu beheben. Des Weiteren sollen spezielle Anforderungen, z.B. die kontextabhängige Konfiguration der Prozessvarianten oder deren flexible Ausführung zur Laufzeit, erfüllt werden. Beide werden von existierenden Ansätzen bisher nicht ausreichend berücksichtigt. Der mit Provop entwickelte Lösungsansatz stellt im Prinzip eine Mischform der konventionellen Methoden dar: Durch Verwendung von Änderungsoperationen zur Beschreibung von Varianten eines Ausgangsprozesses ist Provop eine spezielle Form des konventionellen Ein-Modell-Ansatzes. Die Erzeugung und Verwaltung aller resultierenden Ergebnismodelle, d.h. der gesamten Prozessfamilie, ist mit dem Mehr-Modell-Ansatz vergleichbar. Tabelle 4.3 vergleicht die beiden konventionellen Methoden mit dem Provop-Ansatz entlang der wichtigsten Anforderungen (z.B. kontextabhängige Konfiguration und flexible Ausführung der Prozessvarianten). In Provop werden Prozessvarianten transparent und explizit, mit Hilfe von Änderungsoperationen bzw. Optionen und Aufsetzpunkten, im Basisprozess modelliert. Dies stellt einen entscheidenden Vorteil gegenüber konventionellen Methoden dar. Weitere Vorteile von Provop sind die Berücksichtigung der Rahmenbedingungen bei der Ableitung einer Prozessvariante und die Durchgängigkeit der Lösungskonzepte. Darüber hinaus stellt unser Rahmenwerk zur Prüfung der Korrektheit einer ganzen Prozessfamilie einen weiteren, wesentlichen Vorteil von Provop dar.

4.5 Zusammenfassung

Konventionelle Methoden zur Abbildung von Varianten eines Prozesstyps können grob in zwei Kategorien eingeteilt werden: Mehr-Modell-Ansätze erfordern die Ausmodellierung aller Prozessvarianten in separaten Modellen. Im Vergleich dazu bilden Ein-Modell-Ansätze alle Varianten in einem Prozessmodell ab. Beide Ansätze haben gewisse Nachteile, etwa starke Redundanz, fehlende Transparenz, fehlende Kontextabhängigkeit sowie fehlende Durchgängigkeit bezüglich der Unterstützung von Prozessvarianten.

Für ein adäquates Variantenmanagement, das die gestellten Anforderungen erfüllt und nicht unter den Nachteilen konventioneller Ansätze leidet, haben wir den Provop-Lösungsansatz entwickelt. Diesem Ansatz liegt die Idee zugrunde, das Prozessmodell einer (beliebigen) Variante durch Anpassen eines existierenden Basisprozesses zu generieren. Die Anpassungen des Basisprozess werden in Form höherwertiger Änderungsoperationen beschrieben, deren Anwendung die Ableitung eines Ergebnismodells für eine spezifische Prozessvariante ermöglicht. Dieser operationale Ansatz wird in Provop über den kompletten Prozess-Lebenszyklus betrachtet, d.h. von der Modellierung einer Prozessfamilie, über die Konfiguration einzelner Prozessvarianten und deren Ausführung in einem WfMS, bis zur Evolution der Variantenmodelle. Diese Durchgängigkeit sowie die Transparenz der Prozessvarianten durch explizite Änderungsoperationen und die kontextbasierte Konfiguration der Variantenmodelle stellen

Tabelle 4.3: Vergleich der Ansätze zum Management von Prozessvarianten

Ansatz / Aspekt	Mehr-Modell-Ansatz	Ein-Modell-Ansatz	Provop-Ansatz
Modellierung	Separate Prozessmodelle; identische Prozesselemente werden redundant in allen Variantenmodellen aufgeführt.	Ein Prozessmodell mit Varianz, ausgedrückt durch alternative Ausführungspfade; identische Prozesselemente, die für einzelne Varianten an unterschiedlichen Positionen im Prozessmodell benötigt werden, sind redundant.	Minimale Redundanzen durch Ableitung aller Prozessvarianten aus einem Prozessmodell und durch Wiederverwendung von Optionen bei der Ableitung verschiedener Prozessvarianten.
Konfiguration	Kontextabhängigkeit durch Namenskonventionen oder Modell-Metadaten erfassbar; auf das ganze Prozessmodell bezogen.	Kontextabhängigkeit durch Verzweigungsbedingungen abbildbar.	Kontextabhängige Auswahl und Konfiguration von Prozessvarianten.
Ausführung	Nur einzelne Prozessvarianten werden instanziiert, späte Determinierung und flexible Anpassung eingeschränkt möglich.	Alle Prozessvarianten werden gemeinsam instanziiert, Wechsel zwischen varianten Abläufen durch alternative Ausführungspfade zur Laufzeit möglich.	Beliebige Anzahl Prozessvarianten in einem Workflow-Modell instanziierbar; Flexibilität zur Laufzeit durch späte Determinierung von Variantenmodellen.
Evolution	Prozessvarianten sind unabhängig voneinander anzupassen.	Alle Varianten gleichzeitig anpassbar, Änderungen an redundanten Elementen sind mehrfach auszuführen.	Durch Änderungen am Basisprozess sind alle Prozessvarianten in einem Schritt anpassbar.
Korrektheit	Korrektheitsprüfung wird separat für jedes Prozessmodell angestoßen.	Es müssen immer alle Prozessvarianten gemeinsam geprüft werden.	Rahmenwerk zur effizienten Korrektheitsprüfung der gesamten Prozessfamilie.
Darstellung	Schwere Vergleichbarkeit durch getrennte Evolution der Prozessvarianten; direkte Modellvergleiche eingeschränkt möglich	Schwere Vergleichbarkeit mehrerer und Analyse einzelner Prozessvarianten; Sichtenbildung für einzelne Varianten möglich	Anwenderorientierte Sichten auf beliebige Prozessvarianten; gute Vergleichbarkeit der Prozessmodelle
Skalierbarkeit	Skaliert gut für wenige Prozessvarianten, wird jedoch bei steigender Anzahl unübersichtlich	Skaliert gut für wenige Prozessvarianten, das Prozessmodell wird jedoch schnell sehr komplex	Skaliert gut für eine beliebige Anzahl von Prozessvarianten

entscheidende Vorteile von Provop gegenüber konventionellen Ansätzen dar. In den nachfolgenden Kapiteln betrachten wir die entwickelten Konzepte des Provop-Ansatzes im Detail. Dazu orientieren wir uns am Standard-Prozesslebenszyklus. Abbildung 4.17 zeigt die Struktur dieser Beiträge im Überblick.

Abbildung 4.17: Phasen des Provop-Rahmenwerkes im Überblick

5

Modellierung von Prozessvarianten

Nachdem wir in Kapitel 4 einen Überblick zum Provop-Ansatz gegeben haben, befassen wir uns in Kapitel 5 mit der ersten Phase des Lebenszyklus von Prozessvarianten, d.h. ihrer Modellierung. Abschnitt 5.1 motiviert zunächst den in Provop gewählten Ansatz zur Modellierung von Prozessvarianten und diskutiert relevante Anforderungen. Dem folgend stellen wir die verschiedenen Komponenten des Provop-Modellierungsansatzes im Detail vor. Abschnitt 5.2 behandelt zunächst das Konzept des *Basisprozesses* aus dem sich durch Konfiguration die Prozessvarianten ableiten lassen. Abschnitt 5.3 beschreibt die eindeutige Referenzierung von Elementen und Fragmenten des Basisprozesses, die variantenspezifisch angepasst werden können. Darauf aufbauend erläutert Abschnitt 5.4 die von Provop angebotenen, höherwertigen Änderungsoperationen zur Definition variantenspezifischer Abweichungen. Anschließend präsentiert Abschnitt 5.5 ein Konzept zur Gruppierung und gemeinsamen Anwendung von Änderungsoperationen. Danach erläutert Abschnitt 5.6, wie durch Anwendung von Änderungsoperationen einzelne Variantenmodelle aus dem Basisprozess abgeleitet werden können. Abschließend diskutieren wir in Abschnitt 5.7 verwandte Arbeiten und fassen das Kapitel in Abschnitt 5.8 zusammen.

5.1 Motivation und Anforderungen

Die fachliche Beschreibung ihrer Arbeitsabläufe mittels Prozessmodellen wird für Unternehmen zunehmend wichtig, etwa um ihre Prozesslandschaft zu dokumentieren und transparent zu gestalten [BRB05, LR07, MHHR06]. Bei der Modellierung eines bestimmten Prozesstyps muss nicht nur der Standardprozess definiert werden, sondern es müssen zumeist alle relevanten Prozessvarianten erfasst und beschrieben werden. Insbesondere bildet eine explizite Modellierung die Basis für ihre Implementierung in einem IT-System sowie für ihre (kontinuierliche) Optimierung [MRB08, WSR09].

Die Modellierung einer Prozessfamilie, d.h. einer Menge ähnlicher bzw. verwandter Prozessvarianten, bildet die erste Phase im Lebenszyklus von Prozessvarianten. Die zu diesem Zweck von Provop bereitgestellten Konzepte müssen mächtig genug sein, um die fachlichen Anforderungen dieser Modellierungsphase zu erfüllen. Insbesondere müssen bei der Festlegung

eines Basisprozesses verschiedene Aspekte berücksichtigt werden, etwa die einfache Konfigurierbarkeit relevanter Prozessvarianten oder die robuste Referenzierung zu ändernder Prozessfragmente betreffend.

Variantenspezifische Anpassungen des Basisprozesses sollen mittels höherwertiger Änderungsoperationen realisiert werden. Die angebotenen Änderungsoperationen sollten alle bekannten Anwendungsfälle für die Ableitung einer Variante aus einem Basisprozess abdecken. Bei Bedarf muss aber auch eine nachträgliche Erweiterung der Menge an Änderungsoperationen möglich sein; ohne dass dadurch das grundlegende Rahmenwerk von Provop beeinträchtigt wird.

Damit auch umfangreiche Prozessfamilien abbildbar sind, benötigen wir ggf. zahlreiche Anpassungen. Zur besseren Abstraktion und Strukturierung sollte es daher möglich sein, mehrere Änderungsoperationen in ein Objekt zu gruppieren. In einer Gruppe können nur solche Änderungsoperationen zusammengefasst sein, die bei der Ableitung einer oder mehrere Prozessvarianten gemeinsam auf den Basisprozess anzuwenden sind.

Generell soll es für den Modellierer intuitiv und mit wenig Aufwand möglich sein, Prozessvarianten in einer Modellierungsumgebung, die den Provop-Ansatz unterstützt, zu erstellen. Dazu ist zum einen eine geeignete Modellierungsmethodik erforderlich und zum anderen sind Modellierungsrichtlinien, welche die korrekte Anwendung des Provop-Ansatzes erleichtern, vonnöten. Es ist nicht Ziel dieses Kapitels, allgemeine Methoden zur Modellierung von Prozessmodellen zu beschreiben [Sch98, VRM+08].

Aus den obigen Aspekten ergeben sich die Anforderungen aus Tabelle 5.1.

Tabelle 5.1: Anforderungen an die Modellierung von Prozessvarianten

Anforderung 5.1 *Für jede Prozessfamilie muss ein geeignetes Prozessmodell (im folgenden Basisprozess genannt) gewählt werden, aus dem sich die relevante Prozessvarianten einfach konfigurieren lassen.*

Anforderung 5.2 *Fragmente des Basisprozesses (inkl. Aktivitäten), die bei der Konfiguration von Prozessvarianten manipuliert oder referenziert werden können, sollen explizit gekennzeichnet werden.*

Anforderung 5.3 *Zur Beschreibung variantenspezifischer Anpassungen des Basisprozesses sollen entsprechende Änderungsoperationen einfach anwendbar und parametrisierbar sein. Die angebotenen Typen an Änderungsoperationen müssen ausreichen, um relevante Anwendungsfälle abzudecken. Eine spätere Erweiterung der Menge der Operationstypen soll möglich sein.*

Anforderung 5.4 *Komplexere Anpassungen des Basisprozesses sollen durch Zusammenfassung mehrerer Änderungsoperationen modelliert werden können. Solch gruppierte Anpassungen sollen bei der Konfiguration verschiedener Prozessvarianten wiederverwendbar sein.*

5.2 Basisprozess

Wie motiviert, können die Varianten eines Prozesstyps auf Grundlage eines bestimmten Prozessmodells (vgl. Definition 3.1) definiert werden – dem *Basisprozess*. Das heißt, das Modell des Basisprozesses lässt sich durch Anwendung variantenspezifischer Anpassungen in ein variantenspezifisches Modell transformieren (vgl. Anforderung 5.1).

Modellierung des Basisprozesses

Generell wird der Basisprozess mit den Konstrukten und Vorgaben des zugrunde gelegten Prozess-Metamodells erstellt. In dieser Arbeit verwenden wir das in Abschnitt 3.2 skizzierte Prozess-Metamodell für die Modellierung von Basisprozessen. Das heißt, wir modellieren Prozessschritte mittels Aktivitäten und beschreiben über Kanten und Strukturknoten (z.B. `ANDSplit`, `XORJoin`) den Kontrollfluss zwischen ihnen. Mit Hilfe globaler Datenobjekte sowie zugehöriger Lese- bzw. Schreiboperationen einzelner Aktivitäten können wir darüber hinaus den Datenfluss zwischen Aktivitäten definieren. Dieser legt fest, welche Daten von welcher Aktivität gelesen bzw. geschrieben werden. Abbildung 5.1 zeigt ein Beispiel für einen Basisprozess aus der Domäne „Management von Produktänderungen“.

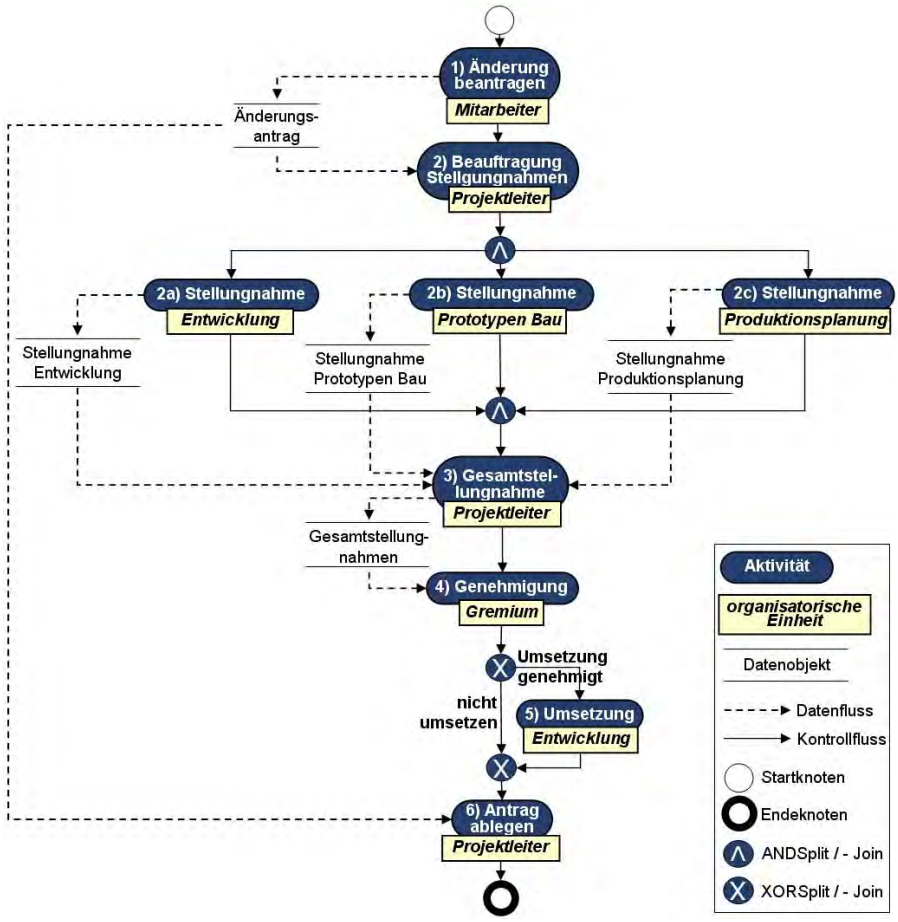


Abbildung 5.1: Darstellung eines Basisprozesses

Mögliche Strategien zur Festlegung des Basisprozesses einer Prozessfamilie

Vor Definition der Prozessvarianten durch Anwendung einer Menge von Änderungsoperationen auf einen Basisprozess muss überlegt werden, wie der Basisprozess der Prozessfamilie gewählt werden soll. Dazu muss entschieden werden, welcher Typ von Basisprozess für den aktuellen Anwendungsfall am geeignetsten ist (vgl. Anforderung 5.1). Folgende Strategien bei der Festlegung der Basisprozesse sind denkbar und können in Provop angewendet werden:

Strategie 1 (Standardprozess): Als Basisprozess kann ein domänenspezifischer Standard- oder Referenzprozess verwendet werden. In der Automobilindustrie gibt es solch einen Standardprozess z.B. für das „Engineering Change Management“ [VDA05].[1] In der Praxis

[1] Auch im betrieblichen Umfeld gibt es entsprechende Standard- bzw. Referenzprozesse [Sch97].

wird ein Standardprozess üblicherweise für bestimmte Rahmenbedingungen überarbeitet und temporär angepasst, daher entspricht die Verwendung eines Standardprozesses als Basisprozess sehr gut unserem Verständnis bezüglich der Ableitung von Prozessvarianten. Außerdem können die Prozessvarianten auf diese Weise leichter zu einem Modell, nämlich dem Standardprozess, vereinheitlicht oder gemeinsam weiterentwickelt werden. Abbildung 5.2a zeigt einen (hier vereinfacht dargestellten) Standardprozess für Werkstätten, von welchem drei Prozessvarianten abgeleitet werden.[2] Jede Prozessvariante ist dabei für bestimmte Rahmenbedingungen erforderlich.

Strategie 2 („Häufigste“ gewählte Prozessvariante): Wird eine bestimmte Prozessvariante, im Vergleich zu anderen Varianten, besonders häufig verwendet, kann es sinnvoll sein, diese als Basisprozess zu wählen. Betrachtet man dazu das Beispiel aus Abbildung 5.2, wird der Standardprozess nur in 5% der Anwendungsfälle verwendet, Variante 2 hingegen wird in 65% der Fälle eingesetzt. Wird Variante 2 als Basisprozess definiert, reduziert dies den Aufwand zur Ableitung von Prozessvarianten, da variantenspezifische Anpassungen dann seltener benötigt werden. Jedoch ist die Anzahl der zu definierenden Änderungsoperationen bei diesem Ansatz unter Umständen größer als bei der Verwendung eines Standardprozesses, insbesondere wenn der gewählte Basisprozess eine hohe Editier-Distanz [LRWe08] zu den anderen Prozessvarianten aufweist.

Strategie 3 (Minimale durchschnittliche Modelldifferenz): Durch strukturelles Mining einer bekannten Menge Prozessvarianten [GRMR+08, LRWb08] kann ein Prozessmodell erzeugt werden, für das zur Bildung aller Prozessvarianten eine minimale Anzahl von Änderungsoperationen modelliert werden muss [LRWb08, LRWe08, LRWf09]. Zusätzlich kann die Anwendungshäufigkeit der einzelnen Prozessvarianten als Gewichtung mit betrachtet werden. Ziel dieses Ansatzes ist es, von Prozessanpassungen zu lernen und Ergebnismodelle in der bestmöglichen Weise zu einem generischen Basisprozess zu verschmelzen. Nachteilig ist, dass bei späterem Hinzukommen weiterer Prozessvarianten eine Neuberechnung des Basisprozesses angestoßen werden muss.

Strategie 4 (Obermenge aller Prozessvarianten): Als Basisprozess wird bei dieser Strategie die Obermenge aller Prozessvarianten gewählt. Das heißt alle Elemente, die bei mindestens einer Prozessvariante auftreten, bilden Elemente des Basisprozesses. Bei der Anpassung des Basisprozesses zwecks Konfiguration von Prozessvarianten sind dementsprechend nur Prozesselemente zu löschen. Da die Prozesselemente prinzipiell auch redundant im Basisprozess vorgehalten werden können, sind keine Prozesselemente zu verschieben und zu verändern, und es müssen keine neuen Prozesselemente eingefügt werden. Die Anzahl notwendiger Änderungsoperationstypen wird dadurch auf Löschenoperationen reduziert. Diese Strategie hat Ähnlichkeit zur Referenzprozessmodellierung [Sch97, RvdA07, RLS+07, GvdAJVlR07, RMvdT09].

Strategie 5 (Schnittmenge aller Prozessvarianten): Bei dieser Strategie wird der Basisprozess mittels Schnittmenge aller Prozessvarianten gebildet. Das heißt ein Element, das in allen Prozessvarianten vorkommt, ist zugleich ein Element des Basisprozesses. Im Gegensatz zu Strategie 4 treten Prozesselemente aber nicht redundant auf. Für die Beispiele aus Abbildung 5.2 etwa sind dann nur die Aktivitäten `Fzg. Annahme` und `Fzg. Übergabe` im Prozessmodell der Schnittmenge enthalten. Folgt man dieser Strategie, ist zur Anpassung des Basisprozesses bei der Konfiguration von Prozessvarianten kein Entfernen

[2] Tatsächlich gibt es mehrere hundert Varianten für Werkstattprozesse, die unter anderem Länderspezifika, Fahrzeugspezifika und Werkstattspezifika abbilden.

von Prozesselementen erforderlich. Auf diese Weise wird die Anzahl notwendiger Änderungsoperationstypen auf Einfüge- und Verschiebe-Operationen reduziert. Insbesondere stellt der Basisprozess die Gemeinsamkeiten der Prozessvarianten sehr gut dar.

Die vorgestellten Strategien zur Festlegung von Basisprozessen unterscheiden sich in einem wichtigen Punkt: Die Basisprozesse der Typen 1 und 2 besitzen konkrete Anwendungsfälle, die Basisprozesse vom Typ 3 bis 5 sind hingegen konstruierte Prozesse, die nur der Bildung von Prozessvarianten dienen. Letztere sind ggf. ungültig bzw. inkorrekt, da sie aufgrund fehlender oder redundanter Aktivitäten weder fachlich relevante noch konsistente Prozessmodelle darstellen. In Provop sind solch invalide Prozesse kein Hindernis. Die Konsequenz ist jedoch, dass Basisprozesse vom Typ 3 bis 5 nicht Teil der Prozessfamilie sein können, da in dieser nur gültige und fachlich relevante Variantenmodelle enthalten sein sollen. Das heißt, bei Basisprozessen der Typen 3 bis 5 müssen ggf. immer variantenspezifische Abweichungen beschrieben und angewendet werden.

Die Wahl eines bestimmten Basisprozesstyps ist letztendlich vom jeweiligen Anwendungsfall abhängig und kann von der existierenden Prozesslandschaft bestimmt werden. Existiert z.B. in einer Domäne bereits ein Standardprozess, aber noch keine Erfahrungswerte darüber, wie häufig bestimmte Prozessvarianten ausgeführt oder wie diese Prozessvarianten gestaltet werden, stellt der Standardprozess (d.h. Strategie 1) den geeignetsten Basisprozesstyp dar.

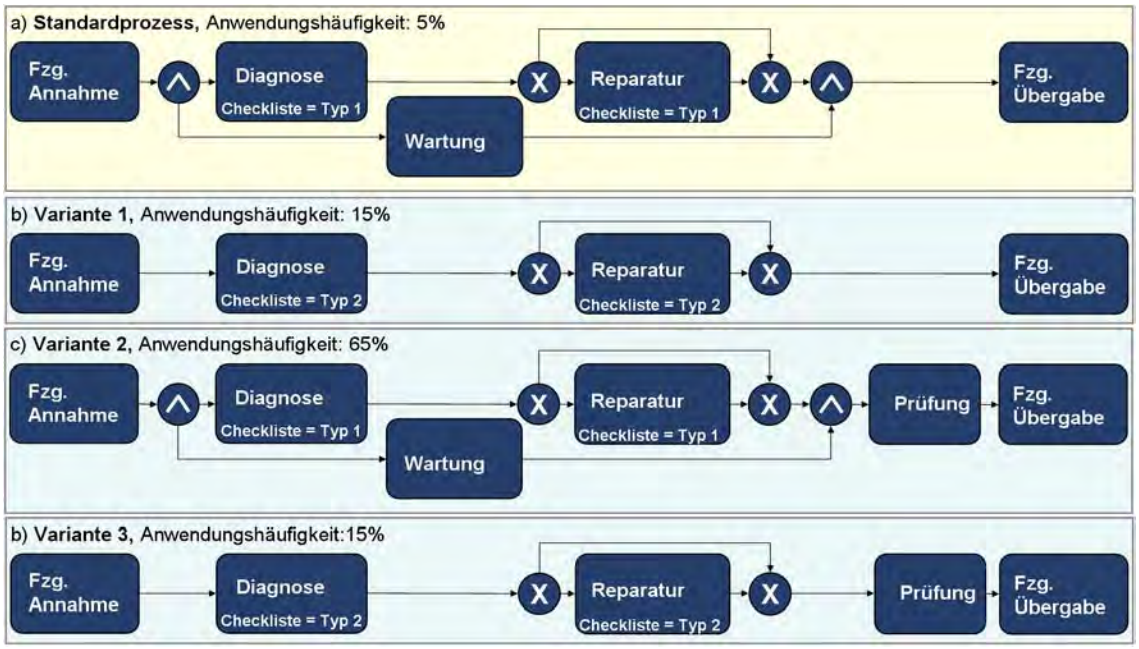


Abbildung 5.2: Standardprozess mit Prozessvarianten aus der Domäne Werkstatt

Modellierung mit mehreren Basisprozessen.

Nicht immer weisen die Prozessvarianten eines Prozesstyps hohe Ähnlichkeit zueinander auf. Vielfach lassen sich sogar verschiedene Klassen von Prozessvarianten in einer Prozessfamilie identifizieren. Die Frage, wann die Modellierung mit einem zweiten Basisprozess für eine Klasse von Prozessvarianten sinnvoll ist, kann nicht allgemeingültig beantwortet werden, sondern hängt vom spezifischen Anwendungsfall ab. Kriterien sind bspw., wie viele Prozessvarianten zu modellieren sind und wie stark sich die Prozessvarianten bzw. die Klassen von Prozessvarianten voneinander unterscheiden.

Ein Nachteil an der Modellierung mehrerer Basisprozesse ist der enorme Mehraufwand bei deren Verwaltung. Bei Anpassungen, die in allen Basisprozessen durchgeführt werden müssen, treten leicht Inkonsistenzen oder Fehler auf. Außerdem fällt die Vereinheitlichung der Prozessvarianten schwerer. Die Problematik entspricht somit derjenigen beim konventionellen Variantenmanagement mit Mehr-Modell-Ansatz (vgl. Abschnitt 4.2.1).

Letztendlich muss der Modellierer für seinen spezifischen Anwendungsfall die Vor- und Nachteile der Modellierung mit einem oder mehreren Basisprozessen abwägen. In Provop wird die Modellierung mit mehreren Basisprozessen prinzipiell mit unterstützt.

5.3 Verwendung expliziter Referenz- bzw. Aufsetzpunkte im Basisprozess

Damit variantenspezifische Anpassungen des Basisprozesses vorgenommen werden können, müssen wir explizit angeben, was geändert werden soll. Das heißt, in der Beschreibung einer Änderungsoperation müssen die betroffenen Elemente und Fragmente im Modell des Basisprozesses eindeutig referenziert werden können (vgl. Anforderung 5.2).

Einen naheliegenden Ansatz stellt die Referenzierung von Prozesselement-IDs dar. Diese sind innerhalb eines Prozessmodells eindeutig und daher als Referenz gut geeignet. Nachteilig ist jedoch, dass die referenzierten Elemente durch Änderungsoperationen geändert oder gar gelöscht werden können. Entfernen wir z.B. eine Aktivität, die als Referenz für eine Einfügeoperation verwendet wird, erhalten wir bei späterer Konfiguration ggf. einen Fehler: Die Einfügeoperation kann dann nicht durchgeführt werden, wenn eine durch sie referenzierte Aktivität nicht mehr im Basisprozess vorhanden ist. Der ID-basierte Ansatz ist daher für unseren operationalen Lösungsansatz nicht immer robust genug.

In Provop begegnen wir diesem Problem, indem wir ein neues Prozesselement einführen, den sog. *Aufsetzpunkt*. Aufsetzpunkte können von Änderungsoperationen referenziert werden, um Fragmente des Basisprozesses zu adressieren. Sie beschreiben Positionen im Basisprozess, die auch nach Löschen von Prozesselementen erhalten bleiben.[3] Sie sind somit, im Vergleich zu Prozesselement-IDs, robuster, da es nicht zu verwaisten Objektreferenzen kommen kann. Beispiel 5.1 zeigt die Verwendung von Aufsetzpunkten.

Beispiel 5.1 (Referenzierung von Aufsetzpunkten) *Abbildung 5.3 zeigt die Definition von zwei Aufsetzpunkten für einen Basisprozess. Beide Aufsetzpunkte sind als schwarze Rauten mit einem Bezeichner dargestellt. Die Einfügeoperation referenziert nun diese Aufsetzpunkte und fügt Aktivität* `3a` *an der gewünschten Position, d.h. zwischen diesen Aufsetzpunkten, in den Basisprozess ein.*

Für die Platzierung von Aufsetzpunkten im Basisprozess sind zwei Ansätze denkbar:

- **Ansatz 1 (Aufsetzpunkte an Kantenursprung, -ziel):** Eine mögliche Position von Aufsetzpunkten bilden der Ursprung bzw. das Ziel einer Kontrollfluss-Kante. Nachteilig ist, dass sich an Strukturknoten mit mehreren ein- bzw. ausgehenden Kontrollfluss-Kanten mehrere Aufsetzpunkte häufen können. In diesem Fall ist nicht intuitiv klar, welche Aufsetzpunkte zur Ableitung einer gewünschten Prozessvariante verwendet werden müssen.

[3] Wir diskutieren den Aspekt, wie Aufsetzpunkte von Änderungsoperationen manipuliert werden können, in Abschnitt 5.4.5.

Abbildung 5.3: Referenzierung von Aufsetzpunkten

- **Ansatz 2 (Aufsetzpunkte an Knotenein- und Knotenausgängen):** Eine alternative Möglichkeit zur Positionierung von Aufsetzpunkten bieten die Ein- und Ausgänge von Knoten (z.B. Aktivitäten und Strukturknoten). Diese Positionen sind vergleichbar mit dem Kantenursprung bzw. -ziel. Vorteilhaft ist jedoch, dass an Strukturknoten maximal zwei Aufsetzpunkte (jeweils an Ein- und Ausgang des Strukturknotens) liegen können und die Verwendung von Aufsetzpunkten intuitiv zugänglich ist.

Aufgrund der Vorteile von Ansatz 2 platzieren wir Aufsetzpunkte ausschließlich an Knotenein- und Knotenausgängen. Auf die zusätzliche Verwendung von Positionen an Kantenursprung oder -ziel verzichten wir dagegen.

Abbildung 5.4 zeigt die in dieser Arbeit verwendete Notation für einen Basisprozess mit Aufsetzpunkten. Wir stellen Aufsetzpunkte als schwarze Rauten dar, die direkt an den betreffenden Knoten liegen. Abbildung 5.4a zeigt alle möglichen Knotenein- und Knotenausgänge, die für die Platzierung von Aufsetzpunkten verwendet werden können. Abbildung 5.4b zeigt zwei Aufsetzpunkte, die mit eindeutigen Bezeichnern versehen sind. Diese sollten einheitlich nach bestimmten Prinzipien vergeben werden (z.B. Numerierung, Namenskonvention). Aus Platzgründen wird in dieser Arbeit gelegentlich auch die Notation aus Abbildung 5.4c verwendet.

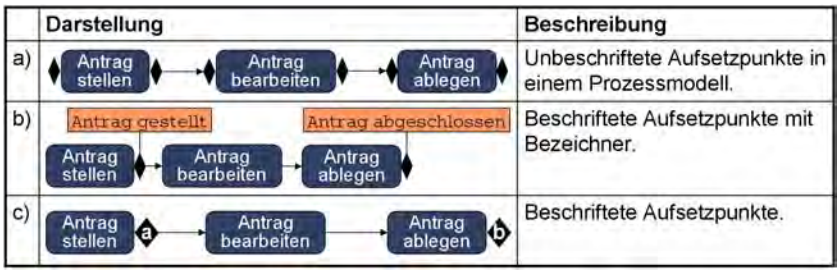


Abbildung 5.4: Darstellung von Aufsetzpunkten in einem Basisprozess

Generell sollte der Modellierer nur relevante Aufsetzpunkte im Basisprozess festlegen, d.h. nur solche Aufsetzpunkte, die als Referenz für eine Änderungsoperation benötigt werden. Dies verbessert die Übersichtlichkeit komplexer Prozessmodelle mit vielen Aufsetzpunkten, und Bereiche, die nicht geändert werden sollen, können gezielt von variantenspezifischen Anpassungen ausgeschlossen werden.

Die Erweiterung der Menge möglicher Prozesselemente in einem Basisprozess erfordert die Anpassung der Definition von Prozessmodellen bzw. Prozessfragmenten (vgl. Definition 3.1) zu Definition 5.1.

Definition 5.1 (Basisprozess)
Sei Labels eine Menge möglicher Bezeichner. Ein Basisprozess $B = (P, AP)$ *wird beschrieben durch sein Prozessmodell* $P = (N, E, NT, ET, EC) \in \mathcal{P}$ *sowie eine Menge zugehöriger Aufsetzpunkte* $AP \subseteq Labels \times N \times \{ENTRY, EXIT\}$. *Ein Aufsetzpunkt* $ap = (id, n, pos)$ *referenziert entweder den Eingang* ($pos = ENTRY$) *oder den Ausgang* ($pos = EXIT$) *eines Knotens* $n \in N$.

Aufsetzpunkte sind nicht für alle Änderungsoperationstypen gleich gut geeignet. So kann es wünschenswert sein, gezielt bestimmte Elemente zu löschen oder zu verändern. In diesem Fall stellt die Angabe der ID eines Prozesselements die bessere Alternative dar. Wir erlauben daher für die Änderungsoperationstypen Löschen und Verändern von Prozesselementen auch die direkte Referenzierung von Prozesselement-IDs.

5.4 Änderungsoperationen

Damit wir von einem Basisprozessmodell eine Prozessvariante ableiten können, müssen die dazu erforderlichen variantenspezifischen Anpassungen beschrieben werden (vgl. Anforderung 5.3). Dies erfolgt in Provop durch Angabe einer Folge höherwertiger Änderungsoperationen. Sie bewirken z.B. das Entfernen oder Einfügen von Aktivitäten oder Prozessfragmenten. Wir geben zunächst eine abstrakte Definition für Änderungen eines Prozessmodells sowie für hieraus resultierende Prozessvarianten. Konkrete Änderungsoperationen und ihre Anwendung auf den Basisprozess zeigen wir nachfolgend.

Definition 5.2 (Prozessänderung und -variante)
Beschreibe $\mathcal{P}$ *die Menge möglicher Modelle eines Basisprozesses und* C *die Menge möglicher Änderungsoperationen. Seien ferner* $S, S' \in \mathcal{P}$ *zwei Prozessmodelle, sei* $\Delta \in C$ *eine Änderungsoperation und sei* $\sigma = \langle \Delta_1, \Delta_2 \ldots \Delta_n \rangle \in C^*$ *eine Sequenz von Änderungsoperationen. Dann:*

- $S[\Delta\rangle S' :\Leftrightarrow \Delta$ *ist anwendbar auf* S *und* S' *ist das Ergebnismodell, das aus der Anwendung von* Δ *auf* S *resultiert.*
- $S[\sigma\rangle S' :\Leftrightarrow \exists\, S_1, S_2, \ldots S_{n+1} \in \mathcal{P}$ *mit* $S = S_1, S' = S_{n+1}$ *und* $S_i[\Delta_i\rangle S_{i+1}$ *für* $i \in \{1, \ldots n\}$.

In beiden Fällen wird S' *als* ***Variante*** *von* S *bezeichnet (kurz: Prozessvariante).*

In der Literatur wird eine Vielzahl von Änderungsoperationen für Prozessmodelle diskutiert. So werden in [WRR07, WRR08, RMRW08a] Änderungsmuster (sog. Change Patterns) beschrieben. In Provop beschränken wir uns auf die wichtigsten Änderungsoperationstypen: `INSERT`, `DELETE` und `MOVE` von Prozessfragmenten sowie `MODIFY` von Attributwerten einzelner Prozesselemente. Durch Kombination mehrerer Änderungsoperationen sind zudem komplexe Anpassungen des Basisprozesses möglich (z.B. Ersetzen eines Prozessfragmentes). Generell reichen bereits diese vier Änderungsoperationen aus, um die für die Praxis relevanten Anwendungsfälle abzudecken. Die Menge angebotener Änderungsoperationen ist in Provop allerdings erweiterbar.

Eine wichtige Fragestellung ist, ob eine Änderungsoperation stets ein korrektes Ausgangsmodell erfordern soll bzw. zu einem korrekten Ergebnismodell führen muss (vgl. Tabelle 3.4).

Da wir in Provop auch „inkorrekte“ (z.B. unvollständige) Basisprozessmodelle zulassen (vgl. Abschnitt 5.2), können wir nicht fordern, dass durch eine Änderungsoperation alle Inkorrektheiten behoben werden. Es ist zudem unser Ziel, die Provop-Änderungsoperationen nicht zu restriktiv zu gestalten. Um dennoch die Korrektheit der konfigurierbaren Ergebnismodelle sicherzustellen, haben wir ein Rahmenwerk für Korrektheitsprüfungen einer Prozessfamilie geschaffen [HBR09e, HBR09c]. Dieses Rahmenwerk stellen wir in Abschnitt 6.5 vor.

Aus Gründen der Übersichtlichkeit verzichten wir im Folgenden bei der Darstellung von Ergebnismodellen auf die Anzeige der Aufsetzpunkte. Diese sind nach Anwendung aller notwendigen Änderungsoperationen auf den Basisprozess auch nicht mehr relevant und sollten deshalb vor den Anwendern verborgen bleiben.

5.4.1 `INSERT`-Operation

Zur Bildung bestimmter Prozessvarianten kann es erforderlich werden, zusätzliche Aktivitäten in den Basisprozess aufzunehmen. Zum Beispiel wird zur Bildung einer bestimmten Variante des Änderungsmanagementprozesses (siehe Abbildung 5.5a) eine zusätzliche Stellungnahme der Qualitätsabteilung erforderlich. Dies entspricht auf Modellebene dem Einfügen der Aktivität `2d` in den gegebenen Basisprozess. Die entsprechende Änderungsoperation zeigt Abbildung 5.5b. Das resultierende Ergebnismodell der Prozessvariante illustriert Abbildung 5.5c. Neben dem Einfügen einzelner Aktivitäten ist auch das Einfügen mehrerer zusammenhängender Aktivitäten (d.h. von Prozessfragmenten) ein wichtiger Anwendungsfall. Wir unterstützen solche Prozesserweiterungen in Provop durch eine spezielle `INSERT`-Operation.

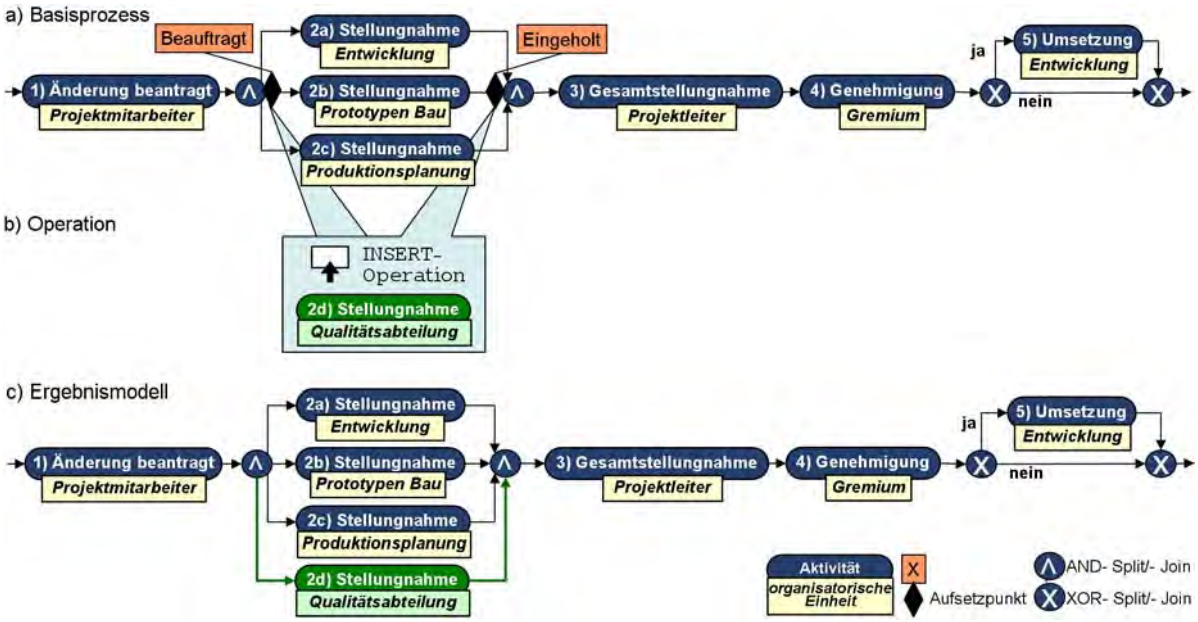


Abbildung 5.5: Einfügen eines Prozessfragments in den Basisprozess

Im Folgenden stellen wir vor, welche Aufruf-Parameter die `INSERT`-Operation benötigt (z.B. Position an der eingefügt werden soll, einzufügende Fragmente), welche Vorbedingungen für ihre Anwendung erfüllt sein müssen und welche Nachbedingungen gelten.

Einfügbare Prozessfragmente. Die Durchführung einer `INSERT`-Operation benötigt Informationen darüber, *was* in den Basisprozess eingefügt werden soll. Dies ist durch Angabe des einzufügenden Prozessfragments gegeben (vgl. Definition 5.3). Generell kann ein beliebiges

Prozessfragment eingefügt werden. Das heißt wir sind auch in der Lage, Prozessfragmente mit mehreren Ein- und Ausgängen einzufügen. Das Prozessfragment kann Verzweigungen und Zusammenführungen von Ausführungspfaden aller Typen (d.h. `AND`, `OR` und `XOR`), komplexe Schleifenkonstrukte, und zusätzliche Start- und Endknoten enthalten. Außerdem ist es vom Prinzip her möglich, zusätzliche Aufsetzpunkte einzufügen. Abbildung 5.6 zeigt Beispiele für einfügbare Prozessfragmente.

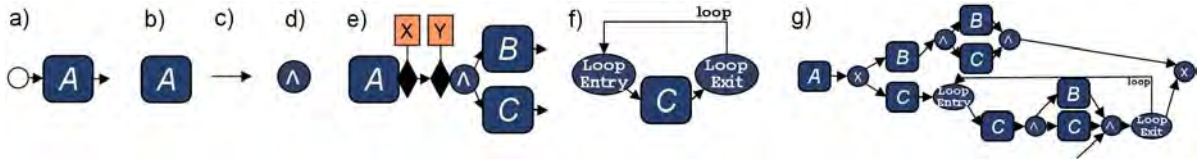


Abbildung 5.6: Beispiele für einfügbare Prozessfragmente bzw. -elemente

Wir fordern nicht, dass einzufügende Prozessfragmente korrekt bzw. vollständig sind (vgl. Abschnitt 3.3). Diese Forderung wäre zu restriktiv, da ggf. nachfolgend angewendete Änderungsoperationen zu einer Auflösung des Problems führen. Ein ungültiger Rücksprung (vgl. Eigenschaft 6) kann z.B. durch eine anschließende Löschen- oder eine Verschiebeoperation behoben werden.

Angabe der Position, an welcher das Fragment eingefügt werden soll. Die Frage nach der Position, an der ein Prozessfragment in den Basisprozess eingefügt werden soll, ist mit Aufsetzpunkten beantwortbar. Dazu werden die Aufsetzpunkte von der `INSERT`-Operation referenziert. Um einerseits eine Unterscheidung zwischen referenzierten und durch die Änderungsoperation neu eingefügten Aufsetzpunkten (vgl. Abbildung 5.6e) zu ermöglichen und andererseits eine möglichst generische Beschreibung zu erlauben, realisieren wir diese Referenzierung der Aufsetzpunkte in einer `INSERT`-Operation nicht direkt. Stattdessen verwenden wir sog. *Anker*. Diese markieren alle Ein- und Ausgänge des einzufügenden Prozessfragments und stellen im Prinzip das Pendant zu den Aufsetzpunkten des Basisprozesses dar. Damit wir auch Prozessfragmente in den Basisprozess einfügen können, die aus einer Kontrollfluss-Kante bestehen oder mit einer Kontrollfluss-Kante beginnen bzw. enden (vgl. Abbildung 5.6a, 5.6b und 5.6e), dürfen Anker, im Unterschied zu Aufsetzpunkten, auch an Kantenursprung bzw. -ziel positioniert werden. Zur Unterscheidung von Ankern, welche an den Eingängen platziert werden, von solchen, die an Ausgängen liegen, bezeichnen wir diese entsprechend als Start- bzw. Ende-Anker. Bei Anwendung der `INSERT`-Operation werden die Anker auf Aufsetzpunkte des Basisprozesses abgebildet. Wir sprechen im Folgenden von *Aufsetzpunkt-Mapping* (kurz: *Mapping*).

Notation der `INSERT`-Operation. Abbildung 5.7 zeigt die im Folgenden verwendete Notation einer `INSERT`-Operation: Der Änderungsoperationstyp wird durch ein spezielles `INSERT`-Symbol gekennzeichnet. Wir verwenden dieses Symbol (und die entsprechenden Symbole weiterer Änderungsoperationstypen) im Folgenden, um eine vereinfachte Darstellung der Änderungsoperationen zu erlauben. Gleichzeitig unterstützt die Symbolik die leichte Verständlichkeit der Modelle. Neben dem Änderungsoperationstyp ist das einzufügende Prozessfragment anzugeben. An seinen Ein- bzw. Ausgängen sind jeweils Start- bzw. Ende-Anker (`S` bzw. `E` in Abbildung 5.7) festgelegt. Das Mapping der Anker auf die Aufsetzpunkte `Beauftragt` bzw. `Eingeholt` zeigt die Position, an welcher das Prozessfragment in den Basisprozess eingefügt werden soll. Das entsprechende Ergebnismodell zeigt Abbildung 5.5c.

Vorgehensweise zum (parallelen) Einfügen von Fragmenten in den Basisprozess. Für die `INSERT`-Operation ist zu definieren, *wie* ein Prozessfragment in den Basisprozess eingefügt

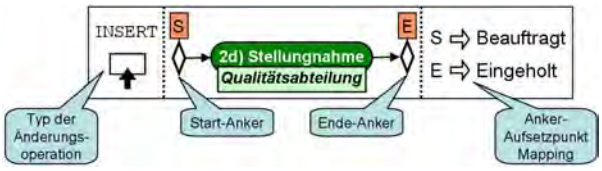


Abbildung 5.7: Beschreibung einer INSERT-Operation

werden soll. So könnte dieses z.B. ein anderes Fragment ersetzen, parallel zu einem anderen Fragment, sequentiell zwischen zwei direkt aufeinander folgenden Fragmenten oder als bedingter Pfad innerhalb einer XOR-Verzweigung eingefügt werden [Rei00, RMRW08a]. Der Algorithmus der INSERT-Operation fügt Prozessfragmente immer parallel in den Basisprozess ein. Eine spezielle INSERT-Operation für das sequentielle Einfügen ist nicht erforderlich. Dies wird durch Vereinfachungsoperationen erreicht, die nach Ableitung einer Prozessvariante auf das Ergebnismodell angewendet werden (vgl. Abschnitt 3.4). Das bedingte Einfügen wird durch Einfügen eines Prozessfragments ermöglicht, in welchem entsprechende Strukturknoten und Kantenbedingungen modelliert sind. Das ersetzende Einfügen stellt eine Kombination der INSERT- und DELETE-Operation dar.

Ein Vorteil dieser Vorgehensweise ist, dass der Variantenmodellierer nicht zwischen verschiedenen INSERT-Operationstypen unterscheiden muss. Ein weiterer Vorteil besteht darin, dass parametrisierte Einfügeoperationen auf diese Weise kompatibler zu vorhergehenden Anpassungen des Basisprozesses sind. Werden z.B. mehrere Aktivitäten durch unterschiedliche INSERT-Operationen in den gleichen Bereich eines Basisprozesses eingefügt, ist das sequentielle Einfügen dieser Aktivitäten ggf. nicht mehr korrekt, sondern es liegt stattdessen ein paralleles Einfügen vor. Da die sequentielle Änderungsoperation somit falsch spezifiziert wäre, könnte sie nicht auf den Basisprozess angewendet werden.

Das parallele Einfügen führt dazu, dass das einzufügende Fragment als paralleler Ausführungspfad in den angegebenen Bereich des Basisprozesses eingefügt sowie der Kontrollfluss des Basisprozesses mit dem eingefügten Fragment verbunden bzw. synchronisiert wird. In diesem Kontext ist zu definieren, wie das neu eingefügte Prozessfragment mit dem Basisprozess verknüpft werden soll, d.h. welcher Strukturknotentyp (AND, OR oder XOR) zur Aufsplittung und Zusammenführung der Ausführungspfade verwendet werden soll. Da wir ein paralleles Einfügen vornehmen, ist die Verwendung von ANDSplit- und ANDJoin-Knoten aufgrund ihrer Semantik ein nahe liegender Ansatz. In diesem Fall wird das eingefügte Prozessfragment durch den Basisprozess am ANDSplit aktiviert. Umgekehrt wartet der Basisprozess am ANDJoin-Knoten auf die Beendigung des eingefügten Prozessfragments. Dieser Ansatz führt allerdings zum Abbruch der Prozessausführung, wenn die ausgehende Kontrollfluss-Kante des eingefügten Prozessfragments im Zustand false_signaled ist. Beispiel 5.2 erläutert diesen Fall.

Beispiel 5.2 (Fehlschlagende Synchronisation durch ANDJoin-Knoten) *Abbildung 5.8 zeigt einen Basisprozess und eine INSERT-Operation. Durch deren Anwendung auf den Basisprozess soll Aktivität E eingefügt werden. Dazu verwenden wir ANDSplit- und ANDJoin-Knoten. Die Aktivität D kann anschließend nur dann ausgeführt werden, wenn zuvor der alternative Pfad mit Aktivität C ausgewählt wurde. Ist dies nicht der Fall, wird Aktivität D als geskippt markiert, d.h. es findet eine sog. Dead Path Elimination statt.*

Damit ein stabiles und vorhersehbares Ausführungsverhalten zur Laufzeit gewährleistet ist, muss dieses Synchronisationsverhalten vermieden werden. Eine Aufsplittung mit einem

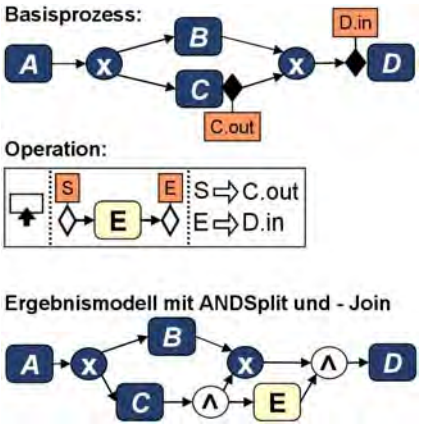


Abbildung 5.8: Aufspaltung und Zusammenführung durch `ANDSplit`- und `ANDJoin`-Knoten

`ANDSplit`-Knoten ist zwar korrekt, aber die Verwendung eines `ANDJoin`-Knoten ist nicht immer geeignet. Ein besserer Ansatz ist es daher, anstelle eines `ANDJoins` einen `ORJoin`-Knoten zu verwenden (vgl. Beispiel 5.3).

Beispiel 5.3 (Synchronisation durch `ORJoin`-Knoten) *Abbildung 5.9 zeigt die Verwendung eines `ORJoin`-Knotens anstelle eines `ANDJoin`-Knotens. Vorteilig ist hier, dass auch bei Auswahl des oberen Pfades (mit Aktivität `B`) eine Ausführung von Aktivität `D` möglich ist (vgl. Beispiel 5.2).*

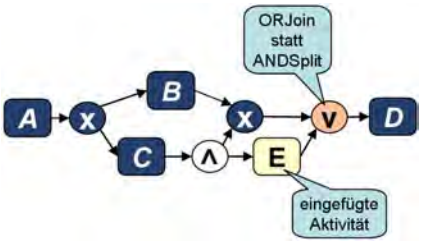


Abbildung 5.9: Zusammenführung der Ausführungspfade durch `ORJoin`-Knoten

Ein Problem dieser Lösung ist, dass die Ablauflogik des Basisprozesses geändert werden kann. So können abgewählte (d.h. `geskippte`) Aktivitäten des Basisprozesses mit Hilfe einer entsprechenden `INSERT`-Operation trotzdem ausgeführt werden (vgl. Beispiel 5.4). Das heißt die `INSERT`-Operation ändert das Ausführungsverhalten des Prozesses signifikant und das ggf. über ihren eigentlichen Anwendungsfall, das Erweitern des Basisprozesses, hinaus. Somit stellt auch die Verwendung eines `ORJoin`-Knotens keine optimale Lösung dar.

Beispiel 5.4 (Fehlschlagende Synchronisation durch `ORJoin`-Knoten) *Abbildung 5.10 zeigt ein unerwünschtes Ausführungsverhalten nach Anwendung einer `INSERT`-Operation: Sind Kontrollfluss-Kante `x` des Basisprozesses im Zustand `false_signaled` und Kontrollfluss-Kante `y` aus dem eingefügten Prozessfragment im Zustand `true_signaled`, kann Aktivität `D`, nach der Synchronisation dieser Kanten am `ORJoin`-Knoten, ausgeführt werden. Dies ist jedoch nicht wünschenswert, da ggf. semantische Abhängigkeiten zwischen Aktivitäten `B`,`C` und `D` vorliegen können. Das heißt, die Ausführung von `D`, ohne die vorherige Ausführung der Aktivitäten `B` und `C`, ist ggf. nicht sinnvoll bzw. möglich.*

Der dritte mögliche Strukturknotentyp, der `XORJoin`, kommt aufgrund seiner Semantik nicht infrage. Wenn alle eingehenden Kanten im Zustand `true_signaled` sind, würde hier eine

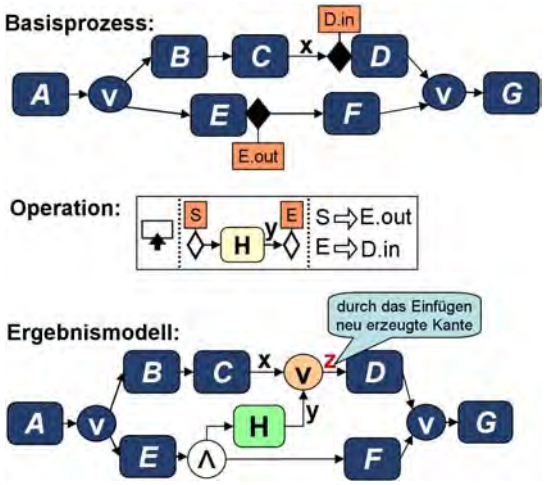


Abbildung 5.10: Unerwünschtes Ausführungsverhalten durch Zusammenführung mit `ORJoin`-Knoten

Dead-Path-Elimination erfolgen. Es kann allerdings nicht davon ausgegangen werden, dass immer entweder die Kante aus dem Basisprozess oder des eingefügten Prozessfragments in den Zustand `false_signaled` übergeht. Der `XORJoin` ist daher nicht geeignet.

Um das gewünschte Ausführungsverhalten zu erreichen, versehen wir den `ORJoin`-Knoten mit einer speziellen *Eingangsbedingung*. Diese sagt aus, dass nur für den Fall, dass der Zustand der Kontrollfluss-Kanten aus dem Basisprozess `true_signaled` ist und die ein- und ausgehenden Kontrollfluss-Kanten des eingefügten Prozessfragments nicht undefiniert sind (d.h. sie befinden sich im Zustand `true_signaled` oder `false_signaled`), die ausgehende Kante des `ORJoin`-Knotens in den Zustand `true_signaled` versetzt wird. Die Auswertung der Eingangsbedingung am `ORJoin`-Knoten erfordert somit eine Unterscheidung der eingehenden Kantentypen. Kontrollfluss-Kanten, die zwingend `true_signaled` sein müssen, bezeichnen wir daher im Folgenden als *reguläre Kontrollfluss-Kanten*. Sie sind weiterhin vom Kantentyp `ControlFlow`. Kontrollfluss-Kanten, die bei der Synchronisation am `ORJoin`-Knoten nicht im Zustand `not_signaled` sein dürfen, bezeichnen wir im Folgenden als *Operationskanten*. Entsprechend weisen wir diesen Kanten einen neuen Typ `Operation` zu.

Wir definieren für eine `INSERT`-Operation, dass die Kanten des Basisprozesses reguläre Kontrollfluss-Kanten sind. Die ein- und ausgehenden Kontrollfluss-Kanten des eingefügten Prozessfragments spezifizieren wir als Operationskanten. Sie werden im folgenden als gestrichelte Pfeile dargestellt.

Damit wir die Semantik eines `ORJoin`-Knotens mit zusätzlicher Eingangsbedingung von „normalen“ `ORJoin`-Knoten unterscheiden können, führen wir einen neuen Knotentyp ein - den `OPJoin`-Knoten. Dieser wird als eingekreistes „größer-als“ Zeichen dargestellt (vgl. Abbildung 5.11). Für eine bessere Verständlichkeit der Ergebnismodelle und für eine leichtere Identifikation eingefügter Prozessfragmente wird in Provop der `ANDSplit`-Knoten durch einen `OPSplit`-Knoten ersetzt. Dieser wird, analog zum `OPJoin`-Knoten, als eingekreistes „kleiner-als“ Symbol dargestellt (vgl. Abbildung 5.11). Die Semantik entspricht jedoch weiterhin der Semantik eines `ANDSplit`-Knotens. Die neuen Kanten- und Knotentypen erfordern eine entsprechende Erweiterung der Kanten- und Knotenmenge bei Definition des Basisprozesses (vgl. Definition 5.1).

Bei Anwendung des Provop-Ansatzes können `OPSplit`- und `OPJoin`-Knoten, je nach zugrunde liegendem Prozess-Metamodell, mit den jeweiligen Basiselementen abgebildet werden.

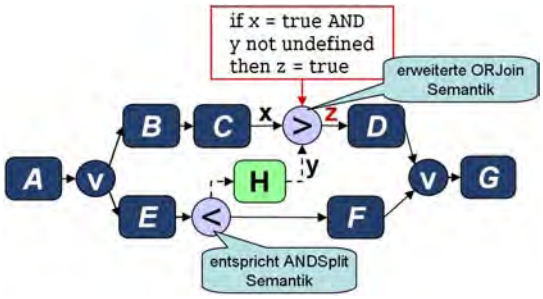


Abbildung 5.11: Erweiterung der `ORJoin`-Knoten Semantik

Hier weisen die Prozess-Metamodelle allerdings große Unterschiede auf. Zum Beispiel können Operationkanten in ADEPT mittels *Sync-Kanten* realisiert werden [Rei00]. Ein spezieller Join-Knoten ist hier nicht erforderlich. In der *Business Process Execution Language* (BPEL) [OAS07] wiederum können wir zur Abbildung des `OPJoin`-Knotens entsprechende Kantenbedingungen für die eingehenden Kanten eines normalen `ORJoin`-Knoten definieren. Wir verwenden daher in dieser Arbeit die neu eingeführten `OPSplit`- und `OPJoin`-Knoten, um die gewünschte Semantik sprachunabhängig beschreiben und visualisieren zu können.

Nachbedingungen der `INSERT`-Operation. Als Ergebnis einer `INSERT`-Operation erhält man ein parallel eingefügtes Prozessfragment, dessen Eingänge im Basisprozess durch einen `OPSplit`-Knoten (`ANDSplit`-Semantik) aufgespaltet und dessen Ausgänge durch einen `OPJoin`-Knoten (erweiterte `ORJoin`-Semantik) mit dem Basisprozess wieder zusammengeführt werden. Die `INSERT`-Operation stellt sicher, dass nach ihrer Anwendung im Einfügebereich eine korrekte Anordnung von Knoten und Kanten resultiert. Um diesen Nachbedingungen zu genügen, wird jeder Anker einer `INSERT`-Operation durch ein entsprechendes *Ersetzungskonstrukt* aus Kanten und Strukturknoten ersetzt. Durch die vier möglichen Positionen eines Ankers (Ein-/ Ausgänge von Knoten bzw. Ursprung/ Ziel von Kanten) ergeben sich für die Abbildung eines Ankers auf einen Aufsetzpunkt vier Möglichkeiten. Für jede von ihnen wird ein anderes Ersetzungskonstrukt gewählt, welches die Verknüpfung des eingefügten Prozessfragments mit dem Basisprozess korrekt übernimmt. Beispiel 5.5 zeigt die Ersetzungskonstrukte für alle möglichen Kombinationen von Ankern und Aufsetzpunkten an Aktivitäten.

Beispiel 5.5 (Ersetzungskonstrukte für Aufsetzpunkte an Aktivitäten) *Abbildung 5.12 zeigt vier `INSERT`-Operationen. Ihre Anker bzw. das Mapping der Anker auf Aufsetzpunkte des Basisprozesses ist so gewählt, dass alle möglichen Positionen von Aufsetzpunkten (d.h. an Ein- und Ausgängen von Aktivitäten) abgebildet sind. Die resultierenden Ersetzungskonstrukte sind in Abbildung 5.12c jeweils hervorgehoben. Das Neuverbinden existierender Kanten wird durch einen Kreis an der Quelle bzw. am Ziel der Kanten dargestellt.*

Bei der Bildung von Ersetzungskonstrukten sind auch Aufsetzpunkte an Strukturknoten zu betrachten (vgl. Beispiel 5.6). Wird ein Start-Anker auf einen Aufsetzpunkt am Ausgang eines Split-Knotens abgebildet, sind die ausgehenden Kanten nicht neu zu verbinden. Das gleiche gilt für einen Ende-Anker, welcher auf einen Aufsetzpunkt am Eingang eines Join-Knotens abgebildet wird. Auch hier werden die eingehenden Kanten des Knotens nicht neu verbunden. Beispiel 5.6 zeigt die verschiedenen Fälle für Aufsetzpunkte an Strukturknoten.

Beispiel 5.6 (Ersetzungskonstrukte für Aufsetzpunkte an Strukturknoten) *Abbildung 5.13 zeigt die Ersetzungskonstrukte für das Einfügen an Strukturknoten. Eine Unterscheidung der einzelnen Strukturknotentypen (d.h. `AND`, `OR` oder `XOR`) ist hier nicht erforderlich. In den Beispielen aus*

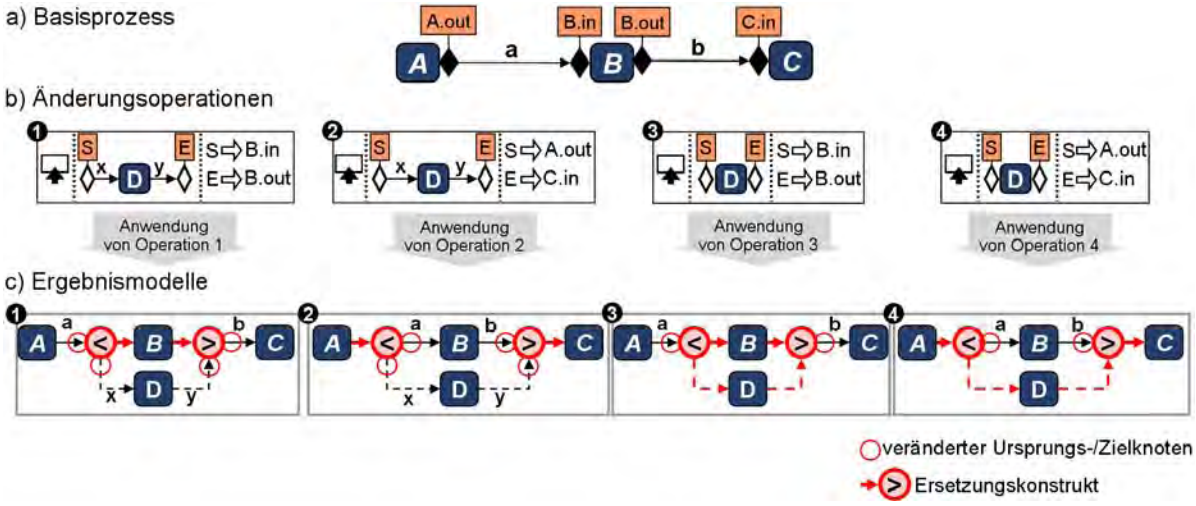


Abbildung 5.12: Ersetzungskonstrukte für Aufsetzpunkte an Aktivitäten

Abbildung 5.13c① und 5.13c② muss eine neue reguläre Kante generiert werden, um den `OPJoin` *korrekt zu verwenden. Wir können die Ergebnismodelle aus Abbildung 5.13c① und 5.13c② vereinfachen zu den jeweils darunter angegebenen Modellen. Die Ersetzungskonstrukte für das Ergebnismodell 3 sind identisch zu den Ersetzungskonstrukten für Aufsetzpunkte an Aktivitäten.*

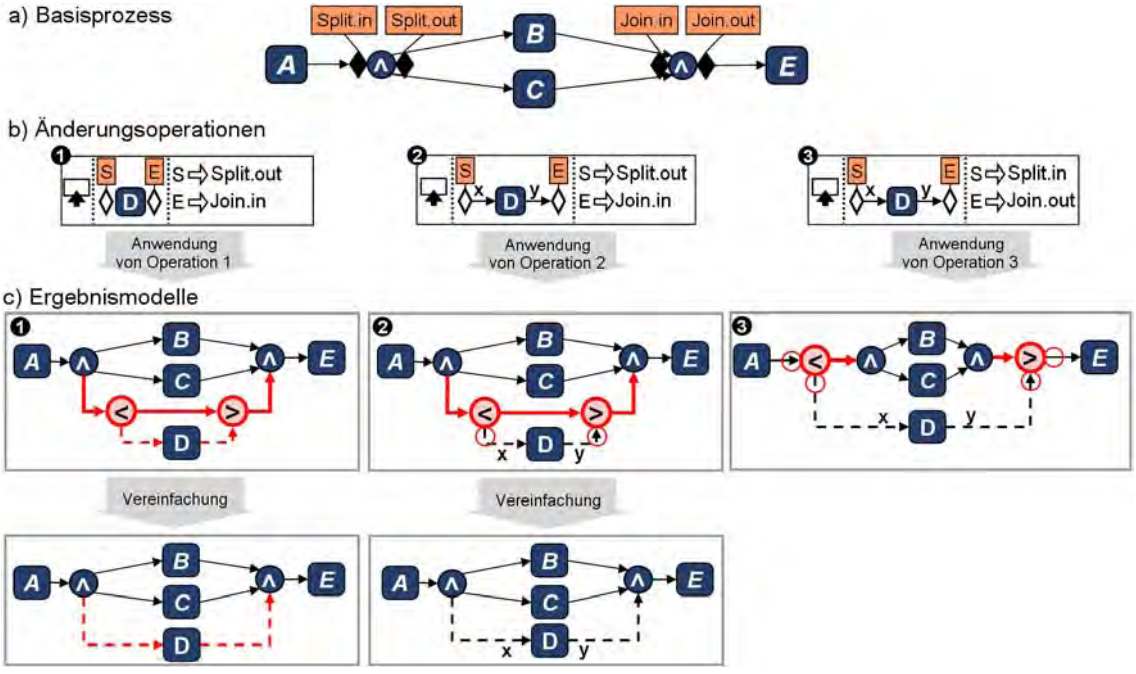


Abbildung 5.13: Ersetzungskonstrukte für Aufsetzpunkte an Strukturknoten

Einen weiteren Fall, den wir bei der Bildung von Ersetzungskonstrukten berücksichtigen müssen, sind Aufsetzpunkte an `Loop_Entry`- und `Loop_Exit`-Knoten. Ebenso wie Strukturknoten, besitzen auch Schleifenknoten zwei eingehende Kanten (`Loop_Entry`) bzw. zwei ausgehende Kanten (`Loop_Exit`). Eine dieser Kanten ist eine spezielle Kontrollfluss-Kante vom Typ `Loop`. Diese wird nicht mit einem `OPSplit`-/ `OPJoin`-Knoten verbunden, da Schleifenkanten nur zwischen `Loop_Entry`- und `Loop_Exit`-Knoten definiert werden können (vgl. Beispiel 5.7).

Beispiel 5.7 (Ersetzungskonstrukte für Aufsetzpunkte an Schleifenknoten) *Abbildung 5.14a zeigt einen Basisprozess mit Schleife. Abbildung 5.14b enthält vier Änderungsoperationen, welche jeweils Aufsetzpunkte referenzieren, die an Schleifen-Knoten liegen. Die nach Anwendung dieser Änderungsoperationen resultierenden Ergebnismodelle sind in Abbildung 5.14c dargestellt. Wir verzichten hier auf die Betrachtung von Ankern an ein- und ausgehenden Kanten des einzufügenden Prozessfragments. In diesen Fällen sind die Operationskanten nicht neu zu generieren, sondern analog zu obigen Beispielen nur neu mit dem* `OPSplit`*-/* `OPJoin`*-Knoten zu verbinden.*

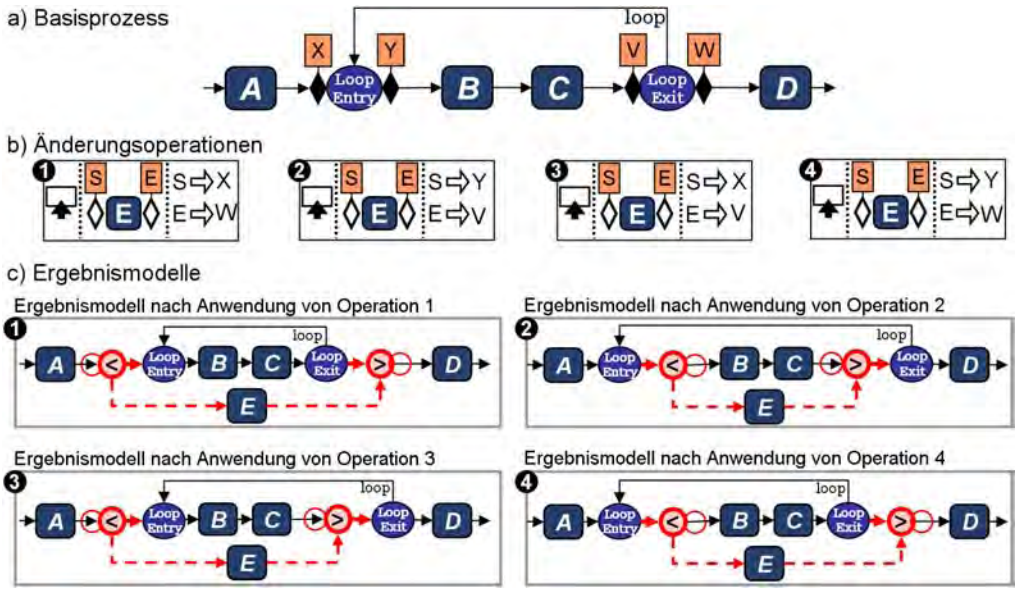


Abbildung 5.14: Ersetzungskonstrukte für Aufsetzpunkte an Schleifenknoten

Ebenfalls zu betrachten ist die Abbildung mehrerer Anker auf den gleichen Aufsetzpunkt. In Provop unterstützen wir diese Art des Mappings und ersetzen den mehrfach referenzierten Aufsetzpunkt durch ein `OP`-Knotenkonstrukt. Das heißt, es wird sowohl ein `OPSplit` als auch ein `OPJoin`- Knoten an dieser Stelle in den Basisprozess eingefügt. Diese Art des Einfügens, in Kombination mit Vereinfachungsoperationen, realisiert im Prinzip ein sequentielles Einfügen. Eine Einschränkung ist jedoch, dass Aufsetzpunkte am Ausgang eines Split-Knotens oder am Eingang eines Join-Knotens nicht mehrfach referenziert werden dürfen, da in diesem Fall keine Sequenz vorliegt und es nicht klar ist, wie die ein- bzw. ausgehenden Kanten der Join- bzw. Split-Knoten mit dem `OP`-Konstrukt verbunden werden müssen. Beispiel 5.8 erläutert die Mehrfachverwendung von Aufsetzpunkten exemplarisch.

Beispiel 5.8 (Ersetzungskonstrukt für die Mehrfachverwendung von Aufsetzpunkten) *In Abbildung 5.15 fügen wir Aktivität* `E` *sequentiell in den Basisprozess ein, indem Start- und Ende-Anker jeweils den gleichen Aufsetzpunkt referenzieren. Die Ergebnismodelle aus Abbildung 5.15c① und 5.15c② zeigen, welche Ersetzungskonstrukte gebildet werden müssen. Die resultierenden Ergebnismodelle sind anschließend vereinfachbar (vgl. Abschnitt 5.6): Die* `OPSplit`*- und* `OPJoin`*-Knoten sind überflüssig, da der Basisprozess auch ohne dieses Konstrukt immer korrekt ausgeführt werden kann. Die beschriebenen Synchronisationsprobleme können nach Vereinfachung somit nicht auftreten.*

Ablauf der `INSERT`-Operation. Nach Darlegung der notwendigen Vor- und Nachbedingungen, können wir den Ablauf der `INSERT`-Operation skizzieren (vgl.Abbildung 5.16):

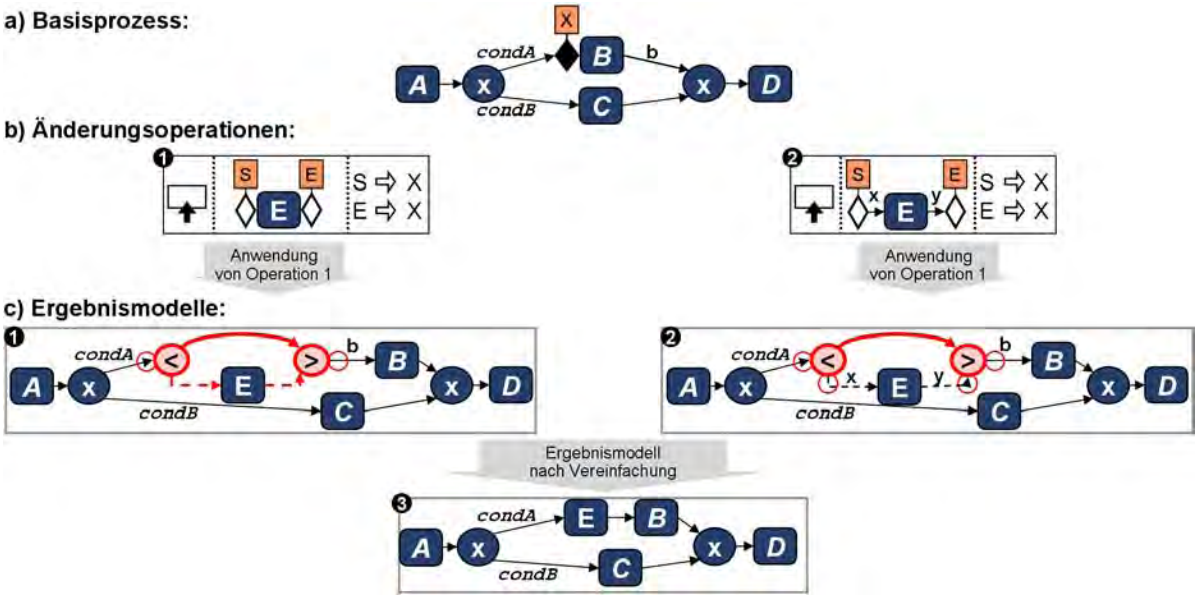


Abbildung 5.15: Mehrfache Referenzierung eines Aufsetzpunktes durch Anker

1. **Prüfung der Vorbedingungen:** Zunächst prüfen wir, ob alle Vorbedingungen (z.B. korrekte Angabe der notwendigen Parameter) zur Anwendung der Änderungsoperation erfüllt sind. Ist dies der Fall, fahren wir mit Schritt 2 fort. Andernfalls wird die Anwendung der Änderungsoperation mit entsprechender Fehlermeldung abgebrochen.

2. **Ersetzungskonstrukte identifizieren:** In diesem Schritt identifizieren wir für jeden Anker, ob dieser ein Start- oder Ende-Anker ist. Anschließend wird untersucht, auf welchen Aufsetzpunkt der Anker abgebildet werden soll und an welcher Position dieser Aufsetzpunkt im Basisprozess liegt. Auf Basis der Ergebnisse dieser Untersuchungen ermitteln wir das entsprechende Ersetzungskonstrukt.

3. **Einfügen und Verbinden der Ersetzungskonstrukte:** Wir fügen nun das entsprechende Ersetzungskonstrukt in den Basisprozess ein und verbinden es mit dem Basisprozess sowie dem eingefügten Prozessfragment.

4. **Prüfung der korrekten Verwendung des `OPJoin`-Knoten:** Nach der Anwendung der Änderungsoperation prüfen wir, ob alle eingefügten `OPJoin`-Knoten mit einer regulären Kante versorgt werden. Ist dies nicht der Fall, fügen wir nachträglich alle fehlenden regulären Kanten von den `OPSplits` zu den `OPJoins` ein.

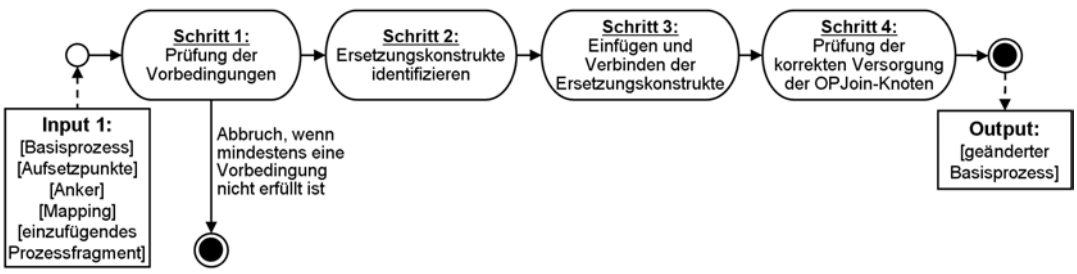


Abbildung 5.16: Ablauf der `INSERT`-Operation

Formale Definition der INSERT-Operation. Wir können nun die Fragmente (vgl. Definition 5.3), die durch die INSERT-Operation in den Basisprozess eingefügt werden sollen sowie die INSERT-Operation selbst (vgl. Tabelle 5.2) wie folgt definieren:

Definition 5.3 (Einfügefragment)
Ein Einfügefragment ist ein Tupel $E = (PF, AP_{PF}, A_{PF})$ *mit:*

- $PF = (N', E', NT', ET', EC')$ *ist ein Prozessfragment (mit* $N' \neq \emptyset \vee E' \neq \emptyset$*).*
- AP_{PF} *ist die Menge der PF zugeordneten Aufsetzpunkte und es gilt:* $AP_{PF} \subseteq Labels \times N' \times \{ENTRY, EXIT\}$.
- A_{PF} *ist die Menge der Anker des Einfügefragments mit* $A_{PF} \subseteq (N' \cup E') \times \{ENTRY, EXIT\}$.
  Ein Anker ist logischer Ein- bzw. Ausgang eines Knotens $n \in N'$*, wobei dieser keine eingehende bzw. keine ausgehende Kante haben darf, oder logische Quelle bzw. Senke einer Kante* $e \in E'$*, wobei dieser keinen Quell- bzw. Zielknoten haben darf. Formal:*
  - $a = (x, ENTRY) \in A_{PF} :\Leftrightarrow x \in N'$ *und x hat keine einmündende Kante oder* $x \in E'$ *und die Kante x hat keine Quellknoten.*
  - $a = (x, EXIT) \in A_{PF} :\Leftrightarrow x \in N'$ *und x kat keine ausgehende Kante oder* $x \in E'$ *und die Kante x hat keinen Zielknoten.*

Tabelle 5.2: Definition der Provop INSERT-Operation

INSERT-Operation Δ_{INS}
Beschreibung:
Die Provop INSERT-Operation fügt ein Prozessfragment in einen Basisprozess ein.
Aufrufparameter:
- Basisprozess $B = (P, AP)$ mit $P = (N, E, NT, ET, EC) \in \mathcal{P}$ und Menge der P zugeordneten Aufsetzpunkte AP. - Einfügefragment $E = (PF, AP_{PF}, A_{PF})$ - Abbildungsfunktion $map : A_{PF} \mapsto AP$. Sie ordnet jedem Anker $a \in A_{PF}$ einen Aufsetzpunkt $map(a) \in AP$ des Basisprozesses zu.
Vorbedingung:
Jeder Anker $a \in A_{PF}$ muss genau auf einen Aufsetzpunkt $ap \in AP$ abgebildet werden, wobei die Abbildungsfuntkion $map(a)$ weder surjektiv noch injektiv sein muss.
Nachbedingung:
$B[\Delta_{INS}\rangle B'$ mit $B' = (P', AP')$ - P' ist das Prozessmodell, dass nach Einfügen von PF in P resultiert. - $AP' = AP \cup AP_{PF}$

5.4.2 DELETE-Operation

Zur Ableitung bestimmter Prozessvarianten aus dem Basisprozess kann es erforderlich werden, diesen zu verkleinern. So kann für Varianten des Änderungsmanagementprozesses (vgl. Abschnitt 2.1) eine bestimmte Stellungnahme nicht erforderlich sein. Für diese Varianten ist daher der Basisprozess in Abbildung 5.17a durch Entfernen der Aktivität 2c entsprechend anzupassen. Das resultierende Ergebnismodell der Variante zeigt Abbildung 5.17c. Mit der

DELETE-Operation geben wir dem Modellierer die Möglichkeit, den Basisprozess in diesem Sinne anzupassen. Dabei können sowohl einzelne Elemente (z.B. einzelne Aktivitäten) als auch komplexe Prozessfragmente aus dem Basisprozess entfernt werden.

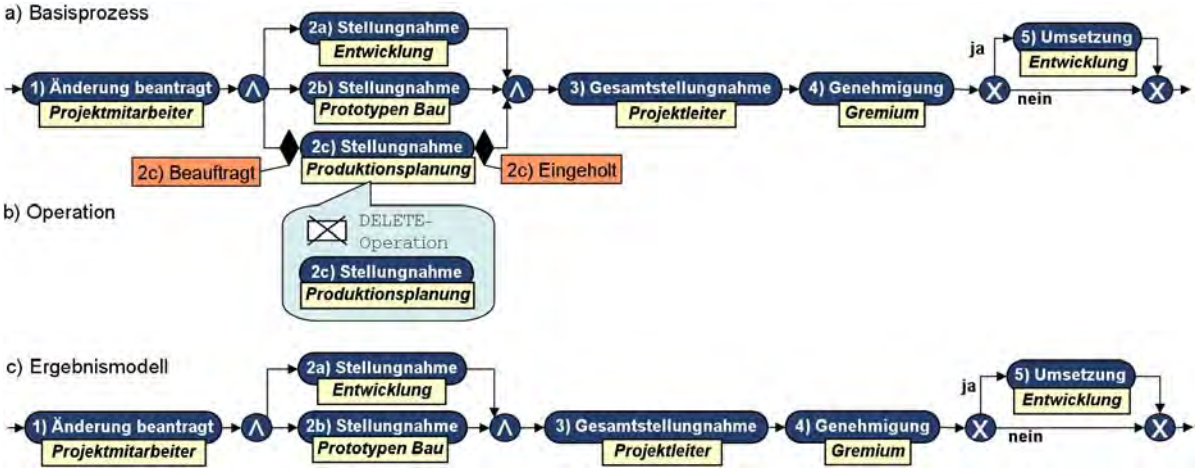


Abbildung 5.17: Entfernen eines Prozessfragments aus dem Basisprozess

Analog zur INSERT-Operation beschreiben wir im Folgenden, welche Aufruf-Parameter die DELETE-Operation benötigt (z.B. zu löschende Fragmente), welche Vorbedingungen für ihre Anwendung erfüllt sein müssen und welche Nachbedingungen gelten.

Festlegung zu löschender Elemente bzw. Fragmente. Damit wir die DELETE-Operation auf einen Basisprozess (bzw. eine davon abgeleitete Variante) anwenden können, müssen wir zunächst die zu löschenden Prozesselemente des Basisprozesses eindeutig identifizieren. Ein naheliegender Ansatz ist die Referenzierung mittels Prozesselement-IDs. Alternativ können wir Aufsetzpunkte verwenden, mittels derer ein zu löschendes Fragment spezifiziert werden kann. Dazu geben wir Anfangs- und Ende-Aufsetzpunkte an, zwischen denen alle Prozesselemente gelöscht werden sollen. Abbildungen 5.18 und 5.19 zeigen dazu ein Beispiel.

- **Ansatz 1 (Angabe der Prozesselement-ID):** Durch Angabe von Prozesselement-IDs lassen sich einzelne Prozesselemente gezielt aus dem Basisprozess entfernen. Bei mehreren zu löschenden Prozesselementen müssen entsprechend alle Prozesselement-IDs angegeben werden. Abbildung 5.18 zeigt ein Beispiel.

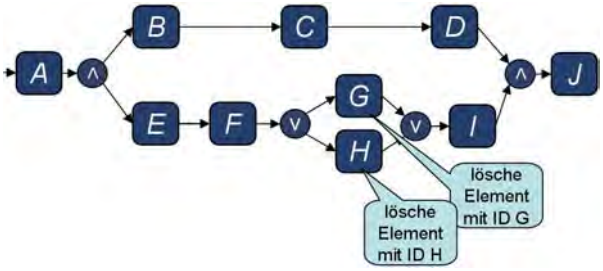


Abbildung 5.18: Angabe der Prozesselement-ID zwecks Löschen eines Fragmentes

- **Ansatz 2 (Angabe von Aufsetzpunkten):** Durch Angabe von Aufsetzpunkten im Basisprozess lässt sich ein zu entfernendes Prozessfragment markieren. Somit werden all diejenigen Prozesselemente entfernt, die auf einem Ausführungspfad zwischen den

gegebenen Anfangs- und Ende-Aufsetzpunkten liegen.[4] Intern werden zunächst die zu löschenden Prozesselemente ermittelt und anschließend ein Löschen anhand der Prozesselement-IDs durchgeführt. Abbildung 5.19 zeigt ein Beispiel.

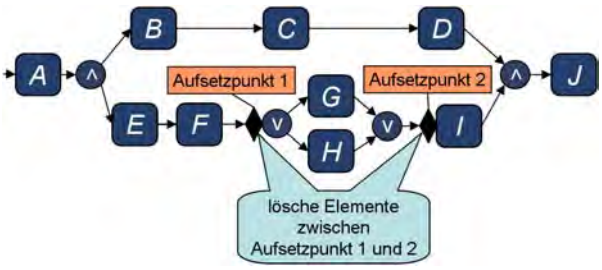


Abbildung 5.19: Angabe von Aufsetzpunkten zwecks Löschen eines Fragmentes

Ist eine Vielzahl von zusammenhängenden Prozesselementen aus dem Basisprozess zu entfernen, ist die Angabe von Prozesselement-IDs aufwändig. In diesem Fall ist die Verwendung von Aufsetzpunkten vorteilhaft. Mittels einer Löschoperation kann damit ein zusammenhängendes Prozessfragment gelöscht werden. Die beiden Ansätze unterscheiden sich zudem darin, wann die zu löschenden Prozesselemente identifiziert werden. Bei Angabe von Prozesselement-IDs stehen die zu löschenden Prozesselemente bereits zur Modellierungszeit fest. Im Fall von Aufsetzpunkten ist dies erst zum Zeitpunkt der Anwendung einer `DELETE`-Operation der Fall, d.h. während der Konfiguration der Prozessvariante. Der wesentliche Unterschied besteht darin, dass bei Angabe von Aufsetzpunkten ggf. auch Prozesselemente entfernt werden, welche zuvor (z.B. durch eine `INSERT`-Operation) in den zu löschenden Bereich eingefügt worden sind (siehe Beispiel 5.20). Dieses Verhalten kann in manchen Anwendungsfällen gewünscht sein, wenn z.B. semantisch zusammenhängende Blöcke entfernt werden sollen. Umgekehrt existieren auch Anwendungsfälle für das gezielte Löschen einzelner Prozesselemente. Wir unterstützen daher in Provop beide Ansätze.

Beispiel 5.9 (Unterschiede des Löschens mit Prozesselement-ID und mit Aufsetzpunkten) *Die `DELETE`-Operationen in Abbildung 5.20c werden modelliert, um die Aktivitäten `B`, `C` und `D` aus dem Basisprozess in Abbildung 5.20a zu entfernen. Wenden wir diese Änderungsoperationen auf den geänderten Basisprozess aus Abbildung 5.20b an, welcher nach Einfügen von Aktivität `F` resultiert, erhalten wir unterschiedliche Ergebnismodelle: Beim Löschen mit Prozesselement-IDs verbleibt die neu eingefügte Aktivität im Prozessmodell. Geben wir ein Fragment durch Aufsetzpunkte an, wird Aktivität `F`, die zwischen diese Aufsetzpunkte in den Basisprozess eingefügt worden ist, durch Anwendung der `DELETE`-Operation wieder entfernt.*

Notation der `DELETE`-Operation. Abbildung 5.21 zeigt die Notation der `DELETE`-Operation für unser Beispiel aus Abbildung 5.17: Die `DELETE`-Operation in Abbildung 5.21a entfernt eine Aktivität durch Angabe ihrer Prozesselement-ID. Die `DELETE`-Operation in Abbildung 5.21b hingegen entfernt die gleiche Aktivität unter Verwendung von Aufsetzpunkten. Die Ergebnismodelle, welche durch Anwendung jeweils einer der beiden Änderungsoperationen entstehen, sind identisch und entsprechen dem in Abbildung 5.17c dargestellten Prozessmodell.

Vorbedingungen der `DELETE`-Operation mittels Aufsetzpunkten. Bei Verwendung von Aufsetzpunkten ist zu prüfen, ob diese im Basisprozess existieren. Anschließend ist festzustellen,

[4] Abschnitt 5.4.5 zeigt, wie wir Aufsetzpunkte handhaben, die in einem zu löschenden Prozessfragment liegen.

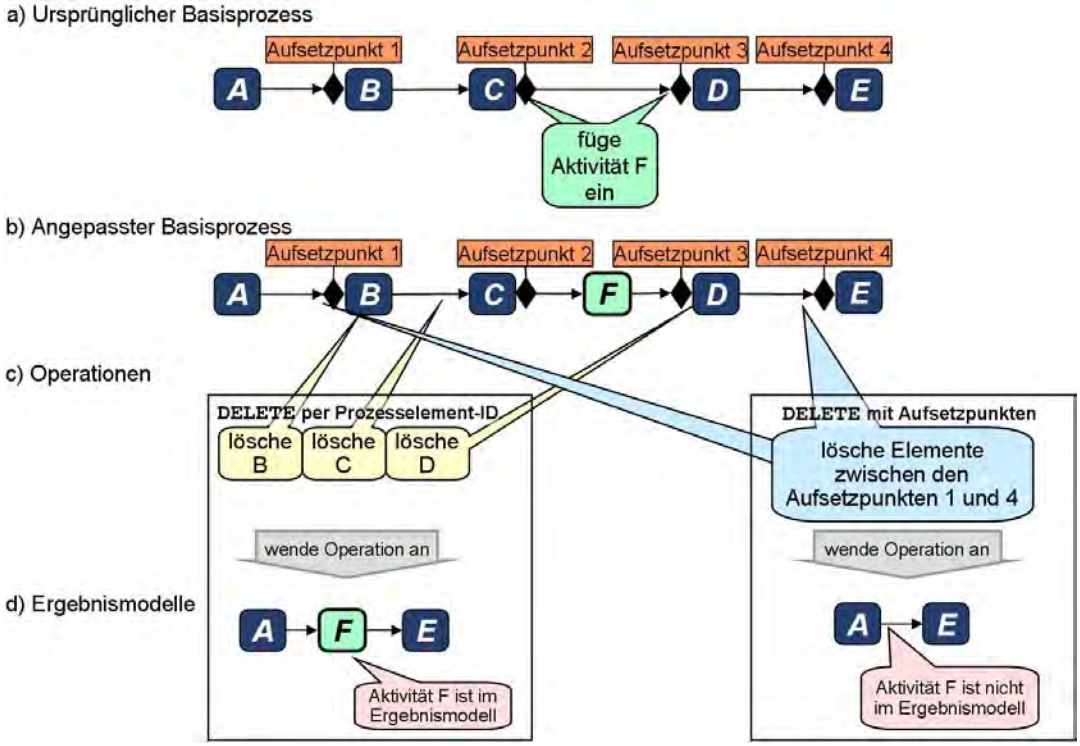


Abbildung 5.20: Unterschiede des Löschens mit Prozesselement-ID und mit Aufsetzpunkten

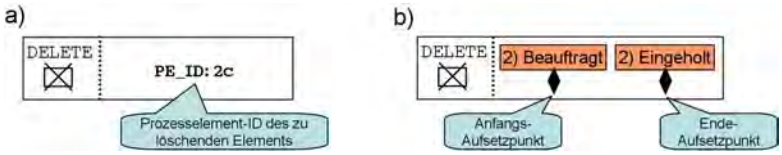


Abbildung 5.21: DELETE-Operation mit Angabe a) einer Prozesselement-ID und b) Aufsetzpunkten

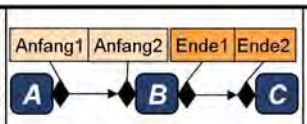
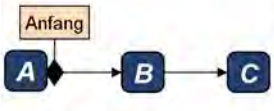
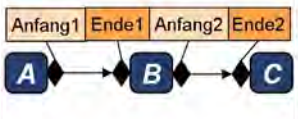
ob sich das zu löschende Fragment auf Grundlage der angegebenen Aufsetzpunkte eindeutig identifizieren lässt. Generell muss für jeden Anfang-Aufsetzpunkt mindestens ein Ende-Aufsetzpunkt angegeben werden, der auf dem gleichen Ausführungspfad liegt. Andernfalls liegt eine unzulässige Angabe vor. Tabelle 5.3 zeigt Beispiele.[5] Angaben, die eine eindeutige Identifikation der zu löschenden Prozesselemente erlauben, die aber fehlerhaft sind (z.B. werden in Tabelle 5.3a überflüssige Aufsetzpunkte verwendet), sind entsprechend zu korrigieren. In diesen Fällen erfolgt das Löschen und es wird eine Warnung ausgegeben. Unzulässige Fälle (vgl. Tabelle 5.3b und 5.3c) führen zum Abbruch der DELETE-Operation, da mit den gegebenen Aufsetzpunkten die zu löschenden Elemente nicht eindeutig identifiziert werden können.

Sind die Aufsetzpunkte korrekt angegeben, können die zu löschenden Elemente im angegebenen Prozessfragment eindeutig identifiziert werden. Beispiel 5.10 zeigt jedoch, dass nicht alle Elemente, welche zwischen zwei Aufsetzpunkten liegen, tatsächlich durch die Änderungsoperation gelöscht werden sollten. So müssen beim Löschen bestimmte Strukturknoten ggf. erhalten bleiben, um weiter eine korrekte Ausführung zu ermöglichen.

Beispiel 5.10 (Identifizieren zu löschender Prozessfragmente) *Abbildung 5.22 zeigt Beispiele für das Identifizieren zu löschender Prozessfragmente. In den Abbildungen 5.22b① und 5.22c①*

[5] Wir zeigen die möglichen Verteilungen hier anhand einfacher Prozessfragmente, die Aspekte sind jedoch entsprechend auf komplexere Prozessmodelle übertragbar.

Tabelle 5.3: Verteilung der Aufsetzpunkte im Basisprozess

a		Zulässige Angabe, aber fehlerhaft modelliert, da Aufsetzpunkt Anfang2 und Ende1 überflüssig sind	b		Unzulässige Angabe, da kein Ende-Aufsetzpunkt angegeben
			c		Unzulässige Angabe, da unklar, ob das Prozessfragment zwischen den Aufsetzpunkten Ende 1 und Anfang2 ebenfalls gelöscht werden muss

wird ein Prozessfragment gelöscht, das zwischen den angegebenen Aufsetzpunkten liegt. In Abbildung 5.22b② und 5.22c② liegt ein Ende-Aufsetzpunkt in einem parallelen Ausführungspfad. Entsprechend darf der Strukturknoten nicht gelöscht werden. Er verbleibt im Ergebnismodell (vgl. Abbildung 5.22c).

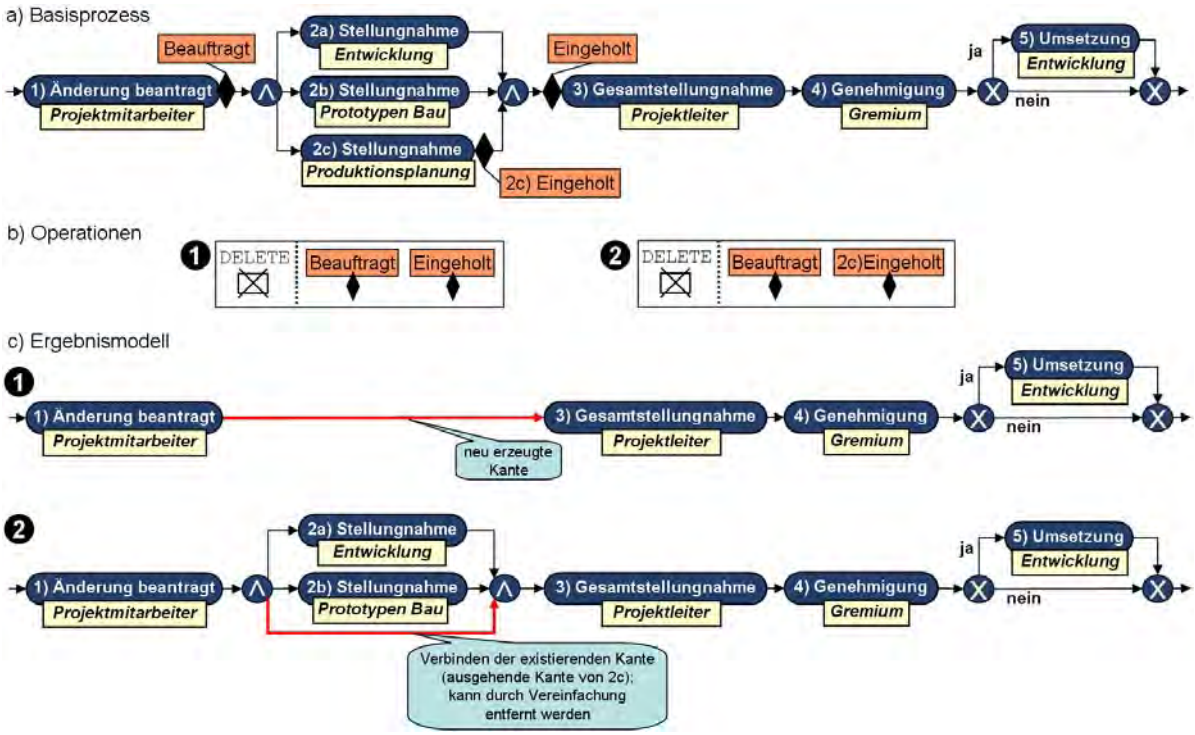


Abbildung 5.22: Identifizieren der zu löschenden Prozesselemente für zwei beispielhafte DELETE-Operationen

Spezielle Anwendungsfälle für das Entfernen von Elementen bzw. Fragmenten. In Abbildung 5.23 entsteht nach Löschen des Prozessfragments ein unzusammenhängender Prozessgraph, da die ursprünglich ein- bzw. ausgehenden Kanten des Fragments keinen Ziel- bzw. Ursprungsknoten mehr besitzen (vgl. Abbildung 5.23c). Das heißt es entstehen Ausführungspfade, die nicht mit einem Endknoten verbunden sind. Dem entsprechend muss nach Durchführung einer DELETE-Operation das Prozessmodell ggf. wieder korrekt zusammengeführt werden können.

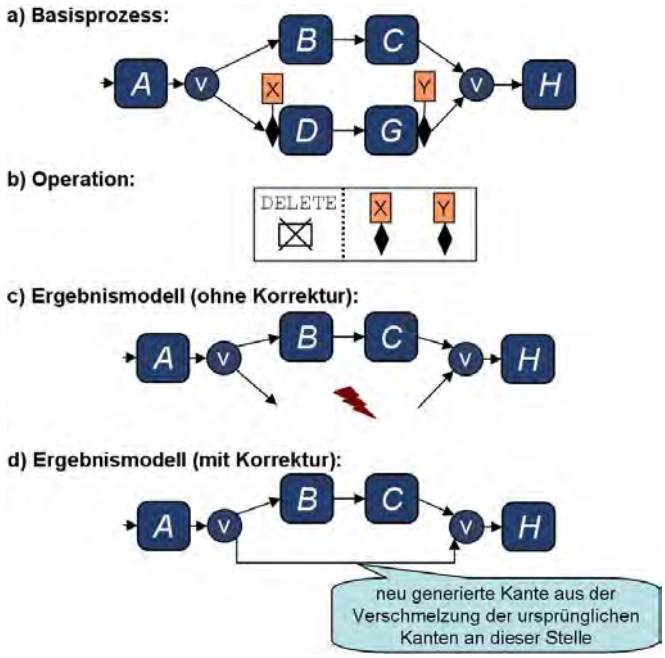


Abbildung 5.23: Anwendungsfall der Kontrollfluss erhaltenden DELETE-Operation

Einen anderen Anwendungsfall zeigt Abbildung 5.24. Die Kontrollfluss-Kante mit Prozesselement-ID 1 sorgt dafür, dass Aktivität C erst nach den parallel ausgeführten Aktivitäten B und D ausführbar ist. Diese Kante löschen wir nun mit Hilfe einer DELETE-Operation aus dem Basisprozess. Würde die Änderungsoperation wie im vorangehenden Beispiel arbeiten, d.h. würden wir das Prozessmodell wieder zusammenführen, erhalten wir das Ergebnismodell aus Abbildung 5.24c. Dieses ist mit dem Basisprozess identisch, abgesehen von der Prozesselement-ID der neu generierten Kontrollflusskante, und entspricht deshalb nicht dem erwarteten Prozessmodell. Letzteres ist in Abbildung 5.24d dargestellt. An dieser Stelle ist das Zusammenführen des Prozessgraphen durch die Änderungsoperation nicht erwünscht. Vielmehr sind solche „Lücken“ Ziel der Änderungsoperation und sollen daher nicht automatisch geschlossen werden.

Damit wir in Provop zwischen diesen beiden Alternativen unterscheiden können, kann der Variantenmodellierer bei der Modellierung einer DELETE-Operation diese als `Kontrollfluss erhaltend` oder `Kontrollfluss löschend` spezifizieren. Dazu wird ein entsprechendes Flag bei der Modellierung gesetzt. Eine Alternative dazu stellen DELETE-Operationen mit unterschiedlicher Semantik dar. Dies verwerfen wir hier, da die Änderungsoperationen bis auf die Korrekturen nach ihrer Anwendung identisch sind.

In Definition 3.1 haben wir gefordert, dass ein Prozessmodell einen zusammenhängenden Graph bildet. Die `Kontrollfluss löschende` DELETE-Operation kann daher zu inkorrekten Prozessmodellen führen (siehe hierzu auch Eigenschaft 2 und 3).[6] Im Folgenden betrachten wir die `Kontrollfluss erhaltende` DELETE-Operation. Wir geben nur für `Kontrollfluss löschende` DELETE-Operationen eine explizite Parametrisierung an.

Nachbedingungen der DELETE-Operation. Die durch Aufsetzpunkte oder IDs spezifizierten Prozesselemente werden aus dem Prozessmodell entfernt. Bei einer `Kontrollfluss erhaltenden` Löschung folgen keine Kanten auf Kanten oder Knoten auf Knoten. Das heißt

[6] Wie wir sicherstellen, dass alle Prozessvarianten einer Prozessfamilie korrekt sind, erläutern wir in Abschnitt 6.5.

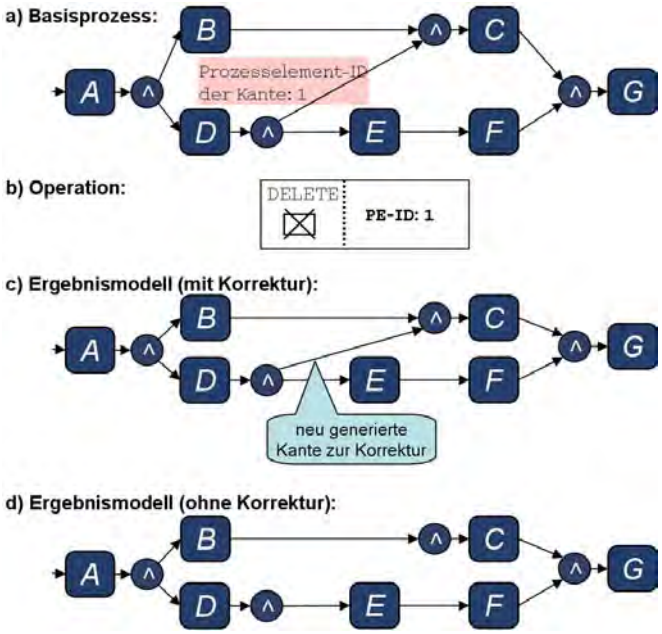


Abbildung 5.24: Anwendungsfall der Kontrollfluss löschenden DELETE-Operation

es werden ggf. neue Kanten konstruiert, existierende Kanten zu einer neuen Kante zusammengeführt oder Kanten neue Quell- bzw. Zielknoten zugewiesen. Eine Übersicht zeigt Abbildung 5.25.

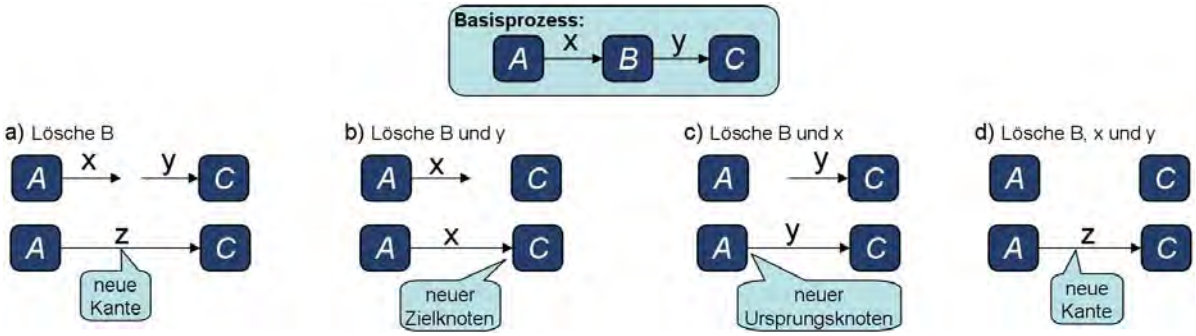


Abbildung 5.25: Korrekturen nach der Anwendung der Kontrollfluss erhaltenden DELETE-Operation

Bei einer `Kontrollfluss löschenden` DELETE-Operation werden die Prozesselemente entfernt. Es werden keine korrigierenden Maßnahmen getroffen.

Ablauf der `DELETE`-Operation. Abbildung 5.26 zeigt den Ablauf der DELETE-Operation.

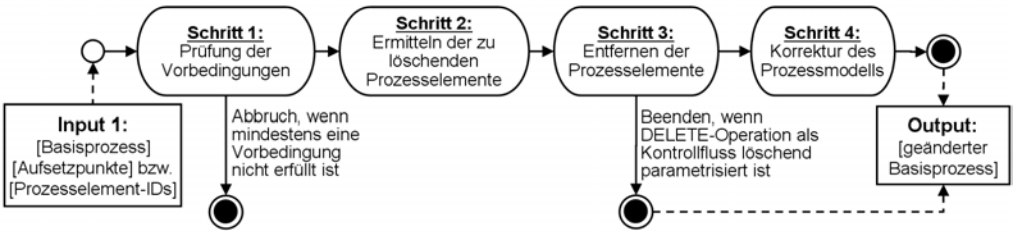


Abbildung 5.26: Ablauf der DELETE-Operation

1. **Prüfung der Vorbedingungen:** Zunächst prüfen wir, ob alle Vorbedingungen zur Anwendung der Löschoperation erfüllt sind. Ist dies der Fall, fahren wir mit Schritt 2 fort. Andernfalls wird mit einer entsprechenden Fehlermeldung abgebrochen.
2. **Ermitteln der zu löschenden Prozesselemente:** Wir identifizieren die zu löschenden Prozesselemente. Sind diese direkt per Prozesselement-ID spezifiziert, fahren wir mit Schritt 3 fort. Bei Angabe von Aufsetzpunkten ermitteln wir zuvor die IDs der zu löschenden Elemente.
3. **Entfernen der Prozesselemente:** Wir entfernen in Schritt 3 jedes Prozesselement (gegeben durch seine Prozesselement-ID).
4. **Korrektur des Prozessmodells:** Ist die `DELETE`-Operation als `Kontrollfluss erhaltend` angegeben, wird das Prozessmodell nach dem Löschen ggf. wieder zusammengeführt.

Formale Definition der `DELETE`-Operation. Wir können die `DELETE`-Operationen mittels Aufsetzpunkten bzw. Prozesselement-IDs wie folgt definieren:

Tabelle 5.4: Definition der Provop `DELETE`-Operation mit Aufsetzpunkten

`DELETE`-Operation Δ_{DEL} mit Aufsetzpunkten
Beschreibung:
Diese Variante der Provop `DELETE`-Operation entfernt ein Prozessfragment mittels Aufsetzpunkten aus dem Basisprozess.
Aufrufparameter:
- Basisprozess $B = (P, AP)$ mit $P = (N, E, NT, ET, EC) \in \mathcal{P}$ und der P zugeordneten Menge der Aufsetzpunkte AP. - Menge $AP_{DEL} \subseteq AP$ mit den Lösch-Aufsetzpunkten, zwischen denen Prozesselemente aus dem Basisprozess gelöscht werden sollen.
Vorbedingung:
Durch die Lösch-Aufsetzpunkte muss ein Prozessfragment gegeben sein. Dazu muss jeder angegebene Lösch-Aufsetzpunkt über einen Pfad vorwärts oder rückwärts von mindestens einem anderen Lösch-Aufsetzpunkt aus erreichbar sein. $\forall\, ap_i \in AP_{DEL}\ \exists\, ap_j \in AP_{DEL}$ mit $ap_i \neq ap_j$ und $p \equiv ap_i = n_1, n_2, \ldots, n_k = ap_j$ oder $p \equiv ap_j = n_1, n_2, \ldots, n_k = ap_i$ ist ein Pfad in (N, E) vom Knoten n_1 zum Knoten n_k mit $n_l \in N$ für $l = 1 \ldots k$ und $k \in \mathbb{N}$.
Nachbedingung:
$B[\Delta_{DEL}\rangle B'$ mit $B' = (P', AP')$ - P' ist das Prozessmodell, das nach dem Löschen des Prozessfragments in P resultiert. - AP' ist die Menge der P' zugeordneten Aufsetzpunkte mit $AP' \subseteq AP$.

5.4.3 `MOVE`-Operation

Für bestimmte Varianten kann es erforderlich sein, die im Basisprozess definierte Ausführungsreihenfolge der Aktivitäten zu verändern. Zum Beispiel ist die Umsetzung einer Produktänderung in gewissen Fällen auch parallel zur Einholung der Stellungnahmen durchführbar (vgl. Abbildung 5.27). Mit der `MOVE`-Operation ermöglichen wir das Verschieben von Prozessfragmenten innerhalb eines Prozessmodells.

Im Folgenden stellen wir vor, welche Aufruf-Parameter die `MOVE`-Operation benötigt (z.B. Start- und Zielpositionen der Verschiebung), welche Vorbedingungen für ihre Anwendung erfüllt sein müssen und welche Nachbedingungen gelten.

Tabelle 5.5: Definition der Provop DELETE-Operation per ID

DELETE-Operation $\Delta_{DEL'}$ per ID
Beschreibung:
Diese Variante der Provop DELETE-Operation entfernt ein einzelnes Prozesselement per ID aus dem Basisprozess.
Aufrufparameter:
- Basisprozess $B = (P, AP)$ mit $P = (N, E, NT, ET, EC) \in \mathcal{P}$ und der P zugeordneten Menge der Aufsetzpunkte AP. - ID eines Prozesselementes $x \in N \cup E$
Vorbedingung:
x muss Kante oder Knoten des Basisprozesses sein, d.h. $x \in N \cup E$.
Nachbedingung:
$B[\Delta_{DEL'}\rangle B'$ mit $B' = (P', AP')$ - P' ist das Prozessmodell, das nach dem Löschen von x in P resultiert. - AP' ist die Menge der P' zugeordneten Aufsetzpunkte mit $AP' \subseteq AP$.

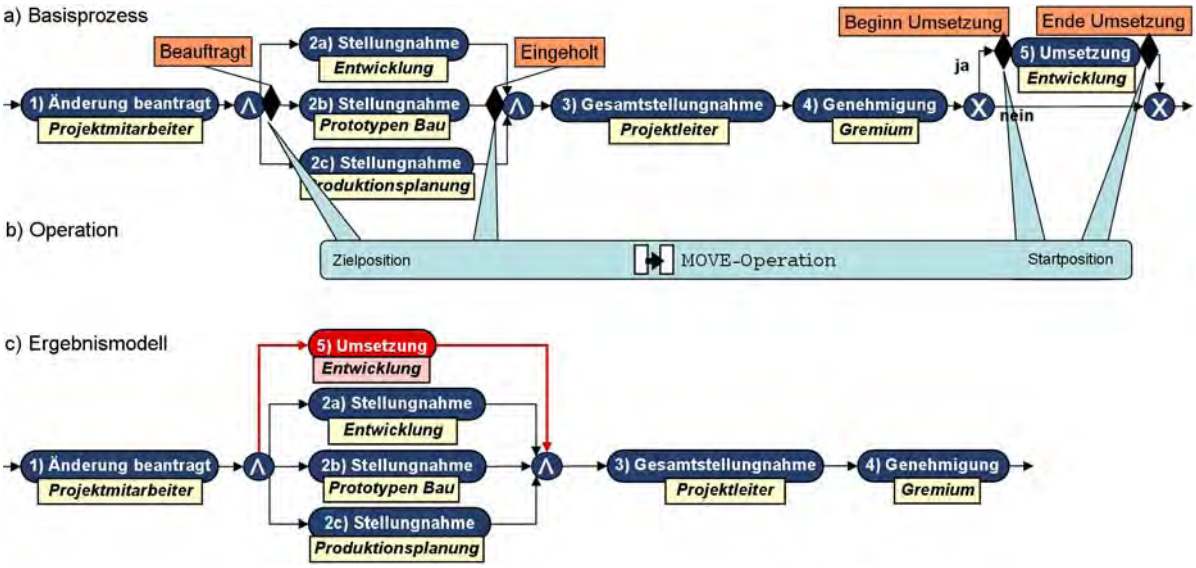


Abbildung 5.27: Verschieben eines Prozessfragments innerhalb des Basisprozesses

Notwendigkeit einer expliziten MOVE-Operation. Die MOVE-Operation stellt eine in der Praxis häufige Änderungsoperation dar. Sie kann prinzipiell auch durch Kombination der DELETE- und INSERT-Operation umgesetzt werden. Nachteilig hierbei ist jedoch, dass durch das Entfernen und anschließende Einfügen neue Prozesselement-IDs für die verschobenen Elemente vergeben werden. Damit auch nachfolgende ID-basierte Änderungsoperationen mit den ursprünglichen IDs der verschobenen Prozesselemente arbeiten können, bietet Provop eine explizite MOVE-Operation an. Diese erlaubt uns nun die Prozesselement-IDs verschobener Elemente i.S. einer robusteren Referenzierung zu erhalten.

Identifikation des zu verschiebenden Fragments und der neuen Position im Basisprozess. Der Modellierer identifiziert zunächst das zu verschiebende Prozessfragment durch Angabe von Aufsetzpunkten (analog zu einer DELETE-Operation mittels Aufsetzpunkten). Das so fixierte Fragment kann analog zur INSERT-Operation beliebige Prozesselemente und

Konstrukte (z.B. Schleifen) enthalten. Die Zielposition der Verschiebung wird wie bei einer INSERT-Operation mit Hilfe von Aufsetzpunkten spezifiziert.

Wir müssen prüfen, ob die angegebenen Aufsetzpunkte im Basisprozess existieren. Außerdem fordern wir, dass sie in einer korrekten Reihenfolge vorgegeben werden, analog zur Verteilung der Aufsetzpunkte im Zusammenhang mit der DELETE-Operation (vgl. Tabelle 5.3). Das Mapping der Aufsetzpunkte der Startposition auf die Aufsetzpunkte der Zielposition muss vollständig sein, d.h. jeder Aufsetzpunkt einer Startposition muss auf einen Aufsetzpunkt an der Zielposition abgebildet werden.

Notation der MOVE-Operation. Abbildung 5.28 zeigt ein Beispiel für eine MOVE-Operation. Die Umsetzung einer Produktänderung soll in diesem Fall bereits während der Stellungnahme und des Entscheidungsprozesses durchgeführt werden. Durch Angabe der Aufsetzpunkte `Beginn Umsetzung` und `Ende Umsetzung` wird Aktivität `Umsetzung` zum Verschieben markiert. Anschließend wird die Zielposition durch die Aufsetzpunkte `Beauftragt` und `Eingeholt` angegeben. Durch das paarweise Abbilden der Start- und Ziel-Aufsetzpunkte sowie ein Neuverbinden der Kanten (d.h. das Zuweisen neuer Ursprungs- und Zielknoten) wird die Aktivität an der ursprünglichen Position ausgeschnitten und an der neuen Position eingefügt (vgl. Abbildung 5.27c).

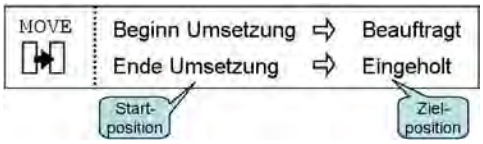


Abbildung 5.28: Beschreibung einer MOVE-Operation

Nachbedingungen der MOVE-Operation. Nach Verschiebung befindet sich ein Prozessfragment an einer neuen Position. Dabei werden, analog zu einer den `Kontrollfluss erhaltenden` DELETE-Operation, Lücken im Kontrollfluss geschlossen und ggf. Strukturknoten erhalten. Darüber hinaus wird das Prozessfragment an der neuen Position (parallel) eingefügt und mit dem Basisprozess synchronisiert. Dazu werden, analog zur INSERT-Operation, entsprechende Ersetzungskonstrukte gewählt.

Ablauf der MOVE-Operation. Der Ablauf der MOVE-Operation ist wie folgt:

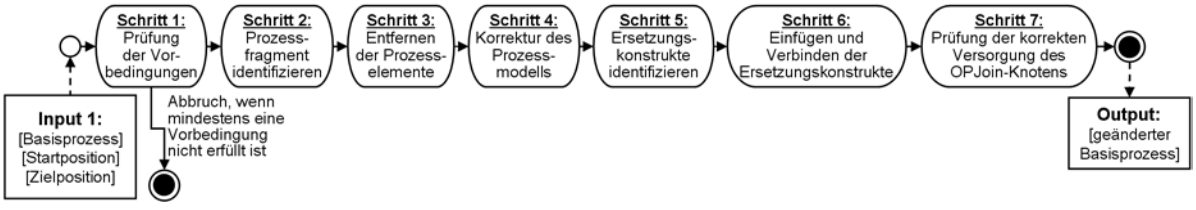


Abbildung 5.29: Ablauf der MOVE-Operation

1. **Prüfung der Vorbedingungen:** Zunächst prüfen wir, ob alle Vorbedingungen (z.B. korrekte Angabe der notwendigen Parameter) zur Anwendung der Verschiebeoperation erfüllt sind. Ist dies der Fall, fahren wir mit Schritt 2 fort. Andernfalls wird mit einer entsprechenden Fehlermeldung abgebrochen.

2. **Prozessfragment identifizieren:** Im nächsten Schritt identifizieren wir das zu verschiebende Prozessfragment (analog zur DELETE-Operation). Es wird als Aufruf-Parameter der INSERT-Operation verwendet.

3. **Entfernen der Prozessfragmente:** Analog zur `DELETE`-Operation wird das Prozessfragment aus dem Basisprozess entfernt.
4. **Korrektur des Prozessmodells:** Das Prozessmodell wird wieder korrekt zusammengeführt, so dass keine Lücke durch das Verschieben entsteht.
5. **Ersetzungskonstrukte identifizieren:** Entsprechend der Aufsetzpunkte werden die Ersetzungskonstrukte für das korrekte Einfügen identifiziert.
6. **Einfügen und Verbinden der Ersetzungskonstrukte:** Wir fügen das Prozessfragment an neuer Position in den Basisprozess ein. Die Ersetzungskonstrukte werden mit dem Basisprozess und dem verschobenen Prozessfragment verbunden.
7. **Prüfung der korrekten Verwendung des `OPJoin`-Knoten:** Nach Anwendung der Änderungsoperation prüfen wir, ob alle eingefügten `OPJoin`-Knoten mit einer regulären Kante versorgt werden. Ist dies nicht der Fall, fügen wir nachträglich alle fehlenden regulären Kanten von den `OPSplits` zu den `OPJoins` ein.

Formale Definition der `MOVE`-Operation. Wir können nun die `MOVE`-Operation wie folgt formal definieren:

Tabelle 5.6: Definition der Provop `MOVE`-Operation

`MOVE`-Operation Δ_{MOV}
Beschreibung:
Die Provop `MOVE`-Operation verschiebt ein Prozessfragment des Basisprozesses an eine neue Position innerhalb des Basisprozesses.
Aufrufparameter:
- Basisprozess $B = (P, AP)$ mit $P = (N, E, NT, ET, EC) \in \mathcal{P}$ und der Menge der P zugeordneten Aufsetzpunkte AP. - Menge von Start-Aufsetzpunkten AP_{Source}, die das zu verschiebende Prozessfragment im Basisprozess beschreibt. - Menge von Ziel-Aufsetzpunkten AP_{Target}, welche die Zielposition des zu verschiebenden Prozessfragments angibt. - Abbildungfunktion $map : AP_{Source} \mapsto AP_{End}$, die jedem Start-Aufsetzpunkt $ap_{St} \in AP_{St}$ einen zugehörigen Ziel-Aufsetzpunkt $map(ap_{Source}) \in AP_{End}$ zuordnet.
Vorbedingung:
- $AP_{St} \subseteq AP$; $AP_{Source} \neq \emptyset$ - $AP_{End} \subseteq AP$; $AP_{End} \neq \emptyset$ - Durch die Menge der Start-Aufsetzpunkte AP_{Source} muss ein Prozessfragment gegeben sein. Das heißt, jeder Aufsetzpunkt $ap_{Source} \in AP_{Source}$ ist über einen Pfad vorwärts oder rückwärts von mindestens einem Aufsetzpunkt $ap'_{Source} \in AP_{Source}$ aus erreichbar. $\forall\ ap_{Source} \in AP_{Source}\ \exists\ ap'_{Source} \in AP_{Source}$ mit $ap_{Source} \neq ap'_{Source}$ und $p \equiv ap_{Source} = n_1, n_2, \ldots, n_k = ap'_{Source}$ oder $p \equiv ap'_{Source} = n_1, n_2, \ldots, n_k = ap_{Source}$ ist ein Pfad in (N, E) vom Knoten n_1 zum Knoten n_k mit $n_l \in N$ für $l = 1 \ldots k$ und $k \in \mathbb{N}$. - $\forall ap_{St} \in AP_{St}$ muss ein $ap_{End} \in AP_{End}$ definiert sein (d.h. $ap_{Target} = map(ap_{Source})$) und die Abbildungsfunktion $map(ap_{St})$ muss surjektiv sein.
Nachbedingung:
$B[\Delta_{MOV}\rangle B'$ mit $B' = (P', AP')$ - P' ist das Prozessmodell, dass nach dem Verschieben des Prozessfragments in P resultiert. - $AP' = AP$

5.4.4 MODIFY-Operation

Eine weitere für die Konfiguration von Prozessvarianten relevante Änderungsoperation stellt die Anpassung der Attribute von Prozesselementen dar. Im Änderungsmanagement kann z.B. zur Abgabe einer Stellungnahme der zuständige Bereich von „Produktionsplanung“ zu „Entwicklung“ abgeändert werden, d.h. für Aktivität `2c` (vgl. Abbildung 5.30) ist das Attribut `Zuständigkeit` entsprechend anzupassen.

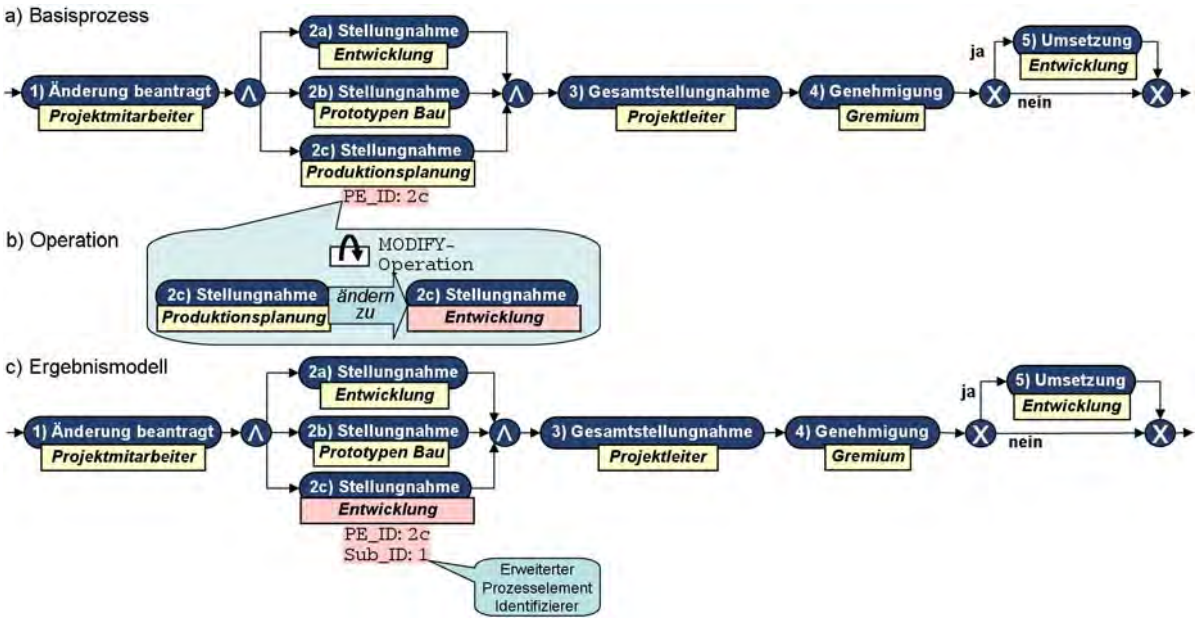


Abbildung 5.30: Verändern eines Attributs einer Aktivität des Basisprozesses

Im Folgenden diskutieren wir, welche Aufruf-Parameter die `MODIFY`-Operation benötigt, welche Vorbedingungen für ihre Anwendung erfüllt sein müssen und welche Nachbedingungen gelten.

Vorbedingungen der `MODIFY`-Operation Vor Anwendung einer `MODIFY`-Operation muss feststehen, welche Attribute eines Prozesselements veränderbar sein sollen. Anpassbare Attribute einer Aktivität sind z.B. Dauer oder Bearbeiterrolle. Darüber hinaus kann die Weitergabe von Prozessdaten zwischen Aktivitäten durch Ändern von Ein- und Ausgabedaten angepasst werden. Tabelle 5.7 gibt eine Übersicht über erlaubte Attributänderungen für die verschiedenen Prozesselementtypen.

Bestimmte Attribute dürfen von einer `MODIFY`-Operation nicht verändert werden. Ein Beispiel ist die Prozesselement-ID, die sich nicht ändern darf, um Seiteneffekte durch falsche Referenzierungen zu vermeiden. Um zu verhindern, dass bestimmte Attribute geändert werden, kann der Modellierer diese als „unveränderlich“ kennzeichnen. Tabelle 5.7 gibt eine Übersicht über (un-)veränderliche Attribute der Prozesselementtypen.

Identifikation des zu ändernden Attributs. Zum Ändern eines Prozesselements benötigt die `MODIFY`-Operation dessen ID. Außerdem muss der eindeutige Attributbezeichner angegeben werden sowie der Wert auf den das Attribut gesetzt werden soll. Wir müssen sicherstellen, dass das referenzierte Prozesselement auch tatsächlich im Basisprozess existiert. Außerdem muss dieses Prozesselement dem erwarteten Typ entsprechen. Das heißt, das zu ändernde Attribut muss für das Prozesselement spezifiziert worden sein. Wir müssen zusätzlich prüfen, ob das Attribut nicht als unveränderlich definiert ist. Abschließend müssen wir sicherstellen, dass

Tabelle 5.7: Übersicht möglicher Änderungen, solange vom Modellierer nicht anders angegeben

Attribut Prozesselementtyp	Veränderbare Attribute	Unveränderliche Attribute
Aktivität	Name, Dauer, Rolle, benutzerdefinierte Attribute	Prozesselement-ID
Strukturknoten	Logischer Operator (`AND`, `OR`, `XOR`, `OP`), benutzerdefinierte Attribute	Prozesselement-ID, Split bzw. Join Eigenschaft
Kontrollflusskante	Name, Kantenbedingung, benutzerdefinierte Attribute	Prozesselement-ID, Kantentyp (`ControlFlow`, `DataFlow`, `Loop`, `Operation`)
Datenobjekt	Name, benutzerdefinierte Attribute	Prozesselement-ID
Datenflusskante	Name, benutzerdefinierte Attribute	Prozesselement-ID, Kantentyp, lesender- bzw. schreibender Zugriff

der neue Wert, auf den das Attribut gesetzt werden soll, im Wertebereich des Attributs liegt und dem erwarteten Datentyp entspricht. Zum Beispiel müssen wir bei einem Zahlenwert prüfen, ob dieser als Zahl oder als Text angegeben werden soll (z.B. „2“ oder „zwei“).

Notation der `MODIFY`-Operation. Abbildung 5.31 zeigt beispielhaft die Notation einer `MODIFY`-Operation für unser Beispiel aus Abbildung 5.30: Neben dem Typ der Änderungsoperation ist mit `PE-ID: 2c` die Prozesselement-ID der zu ändernden Aktivität angegeben. Das betroffene Attribut `Zuständigkeit` soll bei Anwendung der `MODIFY`-Operation auf den Wert `Entwicklung` gesetzt werden.

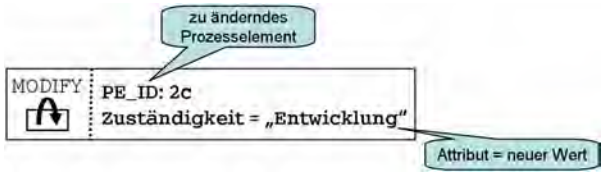


Abbildung 5.31: Beschreibung der `MODIFY`-Operation

Nachbedingungen der `MODIFY`-Operation. Nach Anwendung einer `MODIFY`-Operation befindet sich das geänderte Prozesselement im Ergebnismodell. Um allerdings zwischen Original und geändertem Prozesselement unterscheiden zu können, vergeben wir eine zusätzliche *Sub-Prozesselement-ID* bzw. kurz *Sub-ID* (vgl. Abbildung 5.30).[7]

Ablauf der `MODIFY`-Operation. Abbildung 5.32 zeigt den Ablauf der `MODIFY`-Operation.

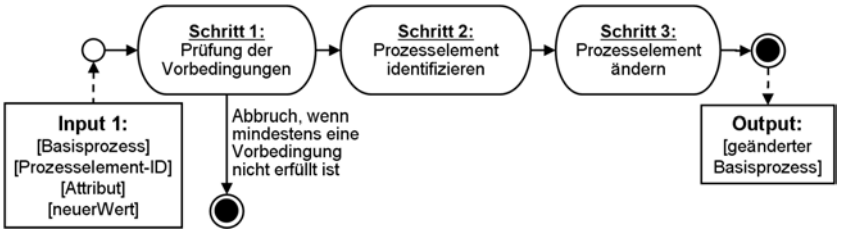


Abbildung 5.32: Ablauf der `MODIFY`-Operation

1. **Prüfung der Vorbedingungen:** Sind alle Vorbedingungen zur Anwendung der Änderungsoperation erfüllt, fahren wir mit Schritt 2 fort. Andernfalls wird mit einer entsprechenden Fehlermeldung abgebrochen.

[7] Wir werden in Abschnitt 7.2.2.2 zeigen, dass unter dem Aspekt der dynamischen Konfiguration von Prozessvarianten diese Sub-ID äußerst relevant für unseren Provop-Lösungsansatz ist.

2. **Prozesselement identifizieren:** Wir identifizieren das zu ändernde Prozesselement bzw. Attribut.

3. **Prozesselement ändern:** Wir ändern das betreffende Attribut des gewählten Prozesselements und kennzeichnen das geänderte Element mit einer Sub-ID.

Formale Definition der `MODIFY`-Operation. Wir können nun die `MODIFY`-Operation wie folgt formal definieren:

Tabelle 5.8: Definition der Provop `MODIFY`-Operation

`MODIFY`-Operation Δ_{MOD}
Beschreibung:
Die Provop `MODIFY`-Operation ändert ein Prozesselement des Basisprozesses.
Aufrufparameter:
- Basisprozess $B = (P, AP)$ mit $P = (N, E, NT, ET, EC) \in \mathcal{P}$ und der P zugeordneten Menge der Aufsetzpunkte AP. - ID eines Prozesselements $x \in N \cup E$ - $Attr$ ist das zu verändernde Attribut von x - $NewValue$ ist der für das Attribut $Attr$ zu setzende Wert
Vorbedingung:
- x muss Kante oder Knoten des Basisprozesses sein, d.h. $x \in N \cup E$. - $NewValue$ liegt im Wertebereich von $Attr$, d.h. $NewValue \in Domain(Attr)$ - $Attr$ ist ein existierendes und veränderbares Attribut des Prozesselements x
Nachbedingung:
$B[\Delta_{MOD}\rangle B'$ mit $B' = (P', AP')$ - P' ist das Prozessmodell, dass nach dem Ändern von x in P resultiert. - $AP' = AP$

5.4.5 Veränderung der Aufsetzpunkte durch Änderungsoperationen

Bei Anwendung von Änderungsoperationen wird der Basisprozess sukzessive zum Ergebnismodell transformiert. Dabei ist sicherzustellen, dass nach Anwendung einzelner Änderungsoperationen ein Zustand resultiert, der die korrekte Anwendung weiterer Änderungsoperationen erlaubt. Eine wichtige Rolle nehmen in diesem Zusammenhang Aufsetzpunkte ein, die z.B. nicht gelöscht werden sollten. Generell nehmen `INSERT`- und `MODIFY`-Operationen keine Veränderung an existierenden Aufsetzpunkten vor. `DELETE`- und `MOVE`-Operationen hingegen manipulieren die Struktur der bereits im Basisprozess vorhandenen Elemente. Dadurch können prinzipiell auch Knoten vor bzw. nach denen Aufsetzpunkte platziert sind gelöscht oder verschoben werden. Wir müssen daher definieren, welche Auswirkungen diese Änderungsoperationen auf die verknüpften Aufsetzpunkte haben. Es gibt grundsätzlich zwei mögliche Ansätze:

Ansatz 1 (Enge Kopplung der Aufsetzpunkte): Sollen gezielt Änderungen an den Aufsetzpunkten vorgenommen werden, ist es wünschenswert, dass durch eine `DELETE`- bzw. `MOVE`-Operation eines Knotens, auch die Aufsetzpunkte an diesem Knoten aus dem Prozessmodell gelöscht bzw. verschoben werden. Dies entspricht dem erwarteten Verhalten dieser Änderungsoperationen. Der Ansatz führt jedoch auch zu unerwünschten Seiteneffekten. So ist z.B. das Löschen eines Knotens mitsamt zugehöriger Aufsetzpunkte problematisch, wenn andere

Änderungsoperationen, die zur Konfiguration der Prozessvariante noch angewandt werden sollen, die gelöschten Aufsetzpunkte referenzieren. In Verbindung mit der MOVE-Operation können ebenfalls unerwünschte Seiteneffekte auftreten. Dies ist der Fall, wenn nachfolgende Änderungsoperationen die verschobenen Aufsetzpunkte referenzieren. Durch die vertauschte Reihenfolge der Aufsetzpunkte können z.B. unerwünschte Zyklen oder Verklemmungen entstehen (vgl. Eigenschaft 6 und 11).

Ansatz 2 (Fixe Position der Aufsetzpunkte): Um die Probleme der engen Kopplung zu vermeiden, können Aufsetzpunkte auch fix in einem Prozessmodell positioniert werden. Das heißt, die Struktur der Aufsetzpunkte wird nach Anwendung von Änderungsoperationen bewahrt. Dazu werden für alle Knoten, die mit Aufsetzpunkten behaftet sind, leere Knoten erstellt, auf welche die Aufsetzpunkte dann übertragen werden können. Die leeren Knoten verbleiben dann an der ursprünglichen Position im Basisprozess, während die ehemals aufsetzpunktbehafteten Knoten verschoben bzw. gelöscht werden. Beispiel 5.11 zeigt diesen Ansatz.

Beispiel 5.11 (Fixe Position der Aufsetzpunkte) *Abbildung 5.33 zeigt einen Basisprozess mit den fix positionierten Aufsetzpunkten* `X` *und* `Y`*. Nach Anwendung von Operation 1 werden die Aktivitäten* `D` *und* `E` *gelöscht. Um die Aufsetzpunkte an ihrer Position zu bewahren, werden für das Zwischenmodell (vgl. Abbildung 5.33c) zwei leere Knoten erzeugt. An diesen werden entsprechend die Aufsetzpunkte positioniert. Die anschließend auf das Zwischenmodell angewandte Operation 2 kann nun Aktivität* `G` *in den alternativen Pfad des Prozessmodells einfügen. Die leeren Knoten werden, mitsamt den an ihnen positionierten Aufsetzpunkten, aus dem finalen Ergebnismodell entfernt.*

Bei Ansatz 2 können unerwünschte Seiteneffekte auftreten. Dies ist der Fall, wenn der verschobene Bereich einen semantisch sinnvollen Block darstellt, in den ein Fragment, welches ebenfalls inhaltlich bzw. fachlich in diesen Block eingefügt werden muss, nun an der alten Position eingefügt wird. Auf diese Weise werden semantische Abhängigkeiten verletzt.

Fazit. Die vorgestellten Konzepte zur Behandlung von Aufsetzpunkten besitzen ihre spezifischen Vor- und Nachteile (vgl. Tabelle 5.9), d.h. keiner der beiden Ansätze ist wirklich überlegen. Um eine hohe Mächtigkeit zu erzielen, unterstützt Provop daher beide Konzepte. Dazu parametrisieren wir die Aufsetzpunkte. Der Modellierer kann dann für jeden Aufsetzpunkt ein bestimmtes Verhalten (enge Kopplung oder fixe Positionierung) festlegen. Da abhängig von der Änderungsoperation (DELETE- oder MOVE-Operation) unterschiedliche Verhaltensweisen sinnvoll sind (vgl. Tabelle 5.9), kann der Modellierer explizit festlegen, welches Verhalten bei welcher Änderungsoperation gewünscht ist. Um den Aufwand bei der Erstellung von Aufsetzpunkten gering zu halten, kann das Verhalten der Aufsetzpunkte per default festgelegt werden. Auf diese Weise muss die Entscheidung für einen Ansatz nicht für jeden Aufsetzpunkt explizit getroffen werden. Für die Wahl des Default-Verhaltens empfehlen wir eine enge Kopplung der Aufsetzpunkte bei DELETE-Operationen und eine fixe Positionierung bei MOVE-Operationen.[8]

5.5 Gruppieren von Änderungsoperationen zu Optionen

Wie unsere Fallstudien gezeigt haben, werden i.A. mehrere Änderungsoperationen benötigt, um aus dem Basisprozess die gewünschte Prozessvariante abzuleiten. Dabei sind oftmals bestimmte Änderungsoperationen bei der Konfiguration einer Prozessvariante in Kombination

[8] Die Parametrisierung der Aufsetzpunkte muss in den Algorithmen der DELETE- und MOVE-Operation entsprechend berücksichtigt werden.

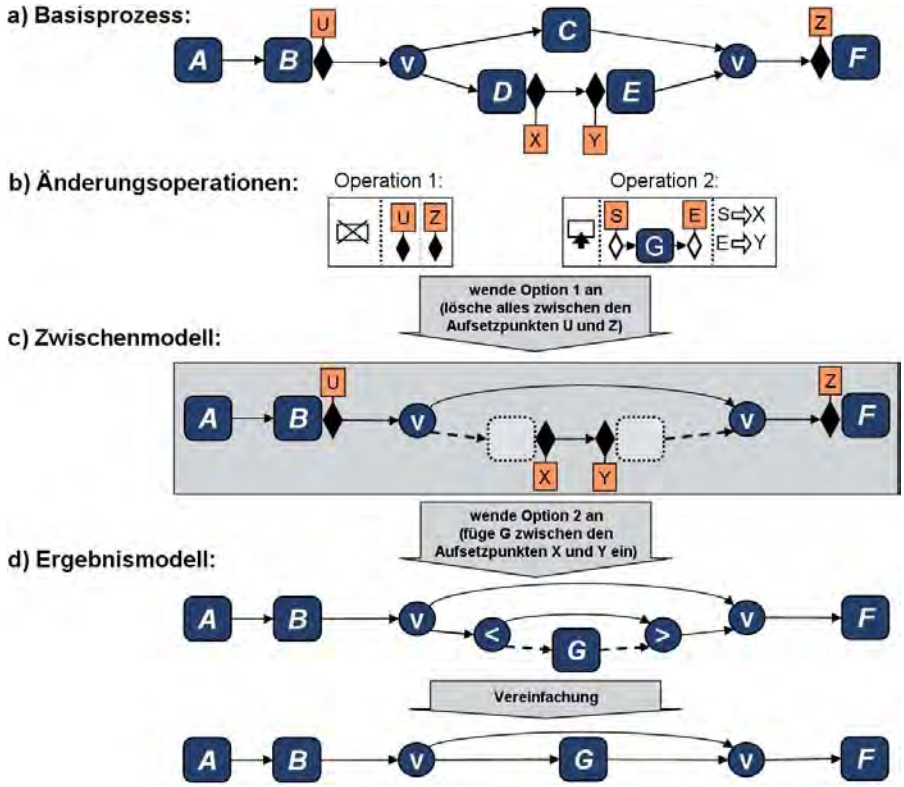


Abbildung 5.33: Fixe Aufsetzpunkte

Tabelle 5.9: Vergleich der Ansätze zur Manipulation der Aufsetzpunkte durch Änderungsoperationen

Ansatz / Aspekt	Enge Kopplung an Knoten	Fixe Positionierung an leeren Knoten
Vorteile	Das Löschen bzw. Verschieben von Knoten, mitsamt verknüpften Aufsetzpunkten, entspricht dem erwarteten Verhalten und ist für Variantenmodellierer bzw. -verantwortliche logisch nachvollziehbar.	Die Aufsetzpunkte bleiben an ihrer ursprünglichen Position im Basisprozess. Dadurch bleibt die Struktur der Aufsetzpunkte im Prozessmodell erhalten. Nachfolgende Änderungsoperationen können die Aufsetzpunkte weiterhin korrekt referenzieren.
Nachteile	Die Struktur der Aufsetzpunkte im Prozessmodell wird verändert. Dies kann zu inkorrekten Prozessmodellen oder zum Abbruch der Anwendung führen. Letzteres ist z.B. der Fall, wenn referenzierte Aufsetpunkte nicht mehr vorhanden sind.	Das Konstrukt der leeren Knoten ist für den Variantenverantwortlichen nicht sichtbar. Daraus kann ein unerwartetes Verhalten resultieren, wenn z.B. ein Einfügen an der ursprünglichen Positionen der Aufsetzpunkte vorgenommen wird, statt an der neuer Position nach einem Verschieben.
Einsatzzweck	Für `MOVE`-Operationen sinnvoll, da Aufsetzpunkte garantiert erhalten bleiben und es keine fehlenden Referenzen gibt. Nachfolgende Änderungsoperationen wirken dann auch auf die neue Position der verschobenen Prozesselemente.	Für Löschoperationen sinnvoll, da Aufsetzpunkte erhalten bleiben und weiterhin referenziert werden können.

miteinander anzuwenden. Ein Ersetzen von Prozessfragmenten wird z.B. durch Kombination der Provop-Änderungsoperationen `DELETE` und `INSERT` umgesetzt. Diese Tatsache macht eine Gruppierung von mehreren Änderungsoperationen erforderlich (vgl. Anforderung 5.4). Das heißt Änderungsoperationen, die immer gemeinsam angewendet werden sollen, müs-

sen zu einem eigenständigen Objekt zusammengefasst werden. In Provop erfüllen wir diese Anforderung durch Verwendung sog. *Optionen*.

Beispiel 5.12 (Gruppierung von Änderungsoperationen zu Optionen) *Abbildung 5.34b zeigt ein Beispiel für eine Option mit zwei Änderungsoperationen. Diese sind als Sequenz gespeichert und entsprechend nummeriert. Bei Anwendung der Option bzw. ihrer Änderungsoperationen wird aus dem Basisprozess aus Abbildung 5.34a, Aktivität* `2c` *durch Aktivität* `2d` *ersetzt. Das Ergebnismodell zeigt Abbildung 5.34c.*

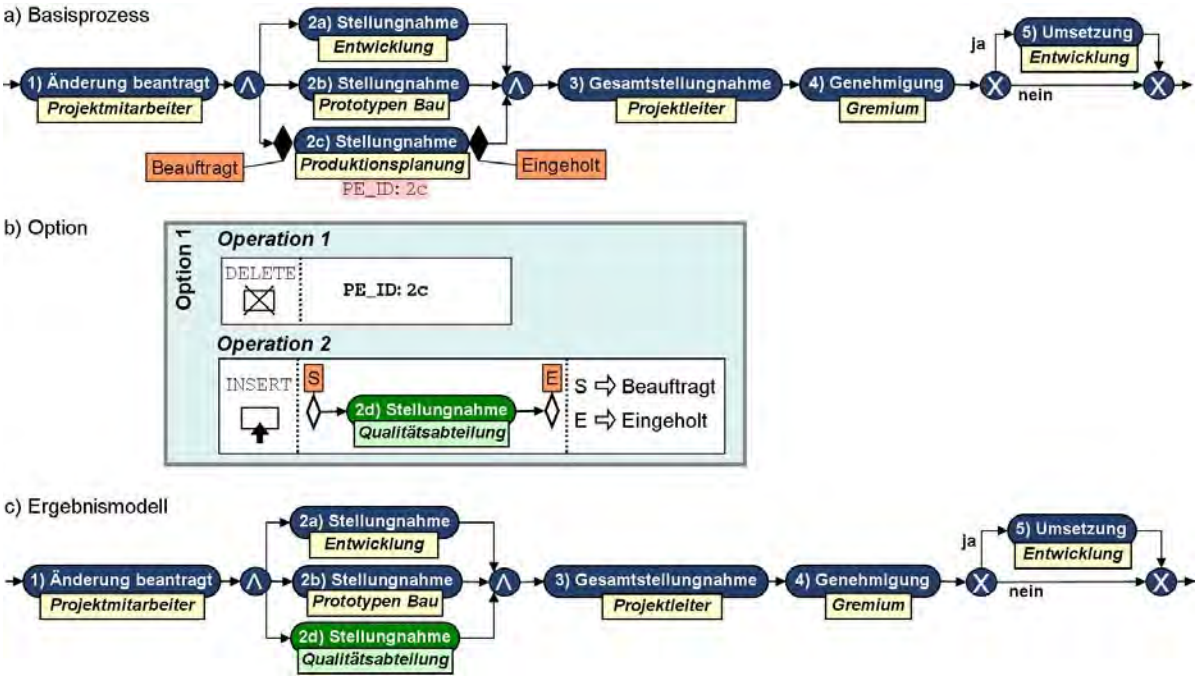


Abbildung 5.34: Gruppierung von Änderungsoperationen zu Optionen

Sequentielle Anordnung von Änderungsoperationen in Optionen. Änderungsoperationen werden in einer Option sequentiell angeordnet (vgl. Beispiel 5.12). Eine deterministische Reihenfolge ist erforderlich, da Änderungsoperationen i.A. nicht kommutativ sind. Das heißt, die Anwendung der gleichen Menge an Änderungsoperationen, mit jeweils geänderter Anwendungsreihenfolge, führt ggf. zu unterschiedlichen Ergebnismodellen.[9] Im Folgenden verzichten wir weitestgehend auf die explizite Angabe einer Reihenfolge. Stattdessen sortieren wir die Änderungsoperationen innerhalb einer Option graphisch in Form einer Liste.

Atomarität von Optionen. Wird eine Option zur Anwendung auf den Basisprozess ausgewählt, werden die Änderungsoperationen in der angegebenen Reihenfolge nacheinander auf den Basisprozess angewendet. Dabei gilt für eine Option das Prinzip der „Atomarität", d.h. es werden immer alle Änderungsoperationen der Option auf den Basisprozess angewendet oder keine.[10] Dies ist sinnvoll, da Optionen solche Änderungsoperationen gruppieren, die eine strukturelle oder semantische Einheit bilden.

Granularität von Optionen. Die Änderungsoperationen zum Erzeugen einer bestimmten Prozessvariante werden i.A. nicht in einer einzelnen Option modelliert; auch die Kombination mehrerer Optionen zur Ableitung einer Prozessvariante ist sinnvoll und möglich. Bei der

[9] Details zu diesem Aspekt erläutern wir in Abschnitt 6.4.

[10] Wir werden in Abschnitt 7.2.2.3 zeigen, dass wir auch zur Laufzeit, unter dem Aspekt der Dynamik von Prozessmodellen, das Prinzip der Atomarität einhalten.

Definition von Optionen muss der Modellierer daher überlegen, wie er die definierten Änderungsoperationen sinnvoll zu Optionen gruppieren kann. Dabei muss er diverse Aspekte berücksichtigen, etwa die Wartbarkeit der Optionen und deren Wiederverwendbarkeit bei der Ableitung unterschiedlicher Prozessvarianten.

Zwei Extreme bilden fein- bzw. grobgranulare Optionen. Feingranulare Optionen bestehen jeweils aus einer bzw. sehr wenigen Änderungsoperationen. Diese können sehr leicht zur Ableitung verschiedenster Prozessvarianten verwendet werden. Allerdings erhöht sich die Anzahl der Optionen, die zur Bildung aller relevanten Prozessvarianten nötig sind, signifikant. Grobgranulare Optionen bündeln dagegen eine größere Menge an Änderungsoperationen. Im Extremfall könnten alle Änderungsoperationen in einer Option gruppiert werden, die zur Ableitung einer bestimmten Prozessvariante benötigt werden. Allerdings sind solch grobgranulare Optionen kaum wiederverwendbar und aufgrund ihrer oft unvermeidlichen Redundanz schwer wartbar.

Beispiel 5.13 (Granularität von Optionen) *Ein Beispiel für das fein- und grobgranulare Zusammenfassen von Änderungsoperationen zeigt der Werkstattprozess aus Abbildung 5.2. Die spezifischen Anpassungen der Prozessvarianten 1 bis 3 können mit Hilfe von vier Änderungsoperationen definiert werden: Einfügen der Aktivität* `Prüfung`*, Ändern des Attributs „Checkliste“ (für die Aktivitäten* `Diagnose` *und* `Reparatur`*) und Löschen der Aktivität* `Wartung`*. Die feingranulare Gruppierung dieser Änderungsoperationen umfasst entsprechend vier Optionen (vgl. Tabelle 5.10), von denen jede jeweils genau eine Änderungsoperation beinhaltet. Ein grobgranulares Zusammenfassen der Änderungsoperationen, orientiert an den abzuleitenden Prozessvarianten, führt zu drei Optionen. Die Änderungsoperationen zum Einfügen der Aktivität* `Prüfung` *und zum Löschen der Aktivität* `Wartung` *sind in diesem Ansatz redundant modelliert.*
Keiner der beiden Aufteilungen bietet ein optimales Ergebnis. Stattdessen ist es sinnvoller, die Änderungsoperationen zum Ändern des Attributs „Checkliste“ (für die Aktivitäten `Diagnose` *und* `Reparatur`*) in einer Option zusammenfassen und diese Option zur Bildung zweier Prozessvarianten wiederzuverwenden.*

Tabelle 5.10: Granularität von Optionen bzw. Änderungsoperationen

Feingranular:	Grobgranular:	Optimale Granularität:
Option 1: MODIFY Diagnose	Option 1: MODIFY Diagnose MODIFY Reparatur DELETE Wartung	Option 1: MODIFY Diagnose MODIFY Reparatur
Option 2: MODIFY Reparatur	Option 2: INSERT Prüfung	Option 2: DELETE Wartung
Option 3: DELETE Wartung	Option 3: DELETE Wartung INSERT Prüfung	Option 3: INSERT Prüfung
Option 4: INSERT Prüfung		
# Optionen = 4 # Operationen = 4	# Optionen = 3 # Operationen = 6	# Optionen = 3 # Operationen = 4

Die optimale Granularität liegt oftmals zwischen den beiden genannten Extremen. Für das sinnvolle Gruppieren von Änderungsoperationen zu Optionen lassen sich daher Faustregeln definieren. Diese haben zum Ziel, redundante Änderungsoperationen zu vermeiden und gleichzeitig die Anzahl der Optionen zu minimieren. Änderungsoperationen sollten immer dann in einer Option gruppiert werden, wenn sie aufgrund semantischer oder

struktureller Abhängigkeiten, stets gemeinsam zur Ableitung einer Prozessvariante verwendet werden. Wird z.B. ein Ersetzen von Fragmenten durch Kombination der Provop-Änderungsoperationen `DELETE` und `INSERT` modelliert, ist es sinnvoll, diese Änderungsoperationen gemeinsam in einer Option zu kapseln und nicht auf zwei Optionen zu verteilen. Die Anzahl der Optionen sinkt dadurch, ohne den Nachteil einer geringeren Wiederverwendbarkeit.

Weitere Aspekte. In den nachfolgenden Kapiteln zeigen wir, dass die Verwendung von Optionen über eine reine Gruppierung von Änderungsoperationen hinausgeht. Zum Beispiel wird die kontextabhängige Konfiguration von Varianten auf Basis von Optionen durchgeführt, indem für eine Option Regeln definiert werden, welche die vorausgesetzten Rahmenbedingungen für ihre Anwendung beschreiben. Außerdem können strukturelle und semantische Beziehungen zwischen Änderungsoperationen auf Ebene der Optionen explizit definiert werden.

Formale Definition von Optionen. Wir können nun Optionen wie folgt definieren:

Definition 5.4 (Option)
Sei $B = (P, AP)$ ein Basisprozess mit Prozessmodell $P = (N, E, NT, ET, EC) \in \mathcal{P}$ und Aufsetzpunkten AP. Sei ferner C die Menge der auf P oder einer Variante von P anwendbaren Änderungsoperationen, die sich auf Aufsetzpunkte der Menge AP beziehen können.

- *Eine **Option** $o =< \Delta_1, \Delta_2 \ldots \Delta_n > \in C^*$ entspricht einer Sequenz von Änderungsoperationen, deren Anwendung auf das Prozessmodell P von B ($P[o\rangle P'$) bzw. einer Variante davon zu einem neuen Prozessmodell P' führt. Insgesamt resultiert ein modifizierter Basisprozess $B' = (P', AP')$.*[a]
- *Die B insgesamt zugeordnete Menge von Optionen bezeichnen wir als Optionsmenge O_B von B.*

[a] Durch Einfügen oder Löschen von Fragmenten kann sich die Menge der Aufsetzpunkte AP von B zu AP' verändern.

5.6 Ergebnismodell und Prozessfamilie

Durch Anwendung von Optionen bzw. ihrer Änderungsoperationen wird der Basisprozess sukzessive zu einem anderen Prozessmodell transformiert.[11] Dieses Modell stellt das Ergebnismodell einer konkreten Prozessvariante dar.

Ein Ergebnismodell kann ggf. strukturell vereinfacht werden, ohne dabei sein Verhalten (d.h. seine Ausführungssemantik) zu ändern. Wir erweitern daher an dieser Stelle die Vereinfachungsoperationen (vgl. Abschnitt 3.4) um Provop-spezifische. Diese betreffen die neuen Strukturknotentypen `OPSplit` und `OPJoin`. Abbildung 5.35 zeigt diese Vereinfachungsoperationen exemplarisch.

Prinzipiell kann durch Anwendung von Optionen eine Vielzahl von Prozessvarianten abgeleitet werden. Die Menge aller tatsächlich generierten Ergebnismodelle, die in der Praxis relevant und korrekt sind, bezeichnen wir als *Prozessfamilie*. [12] Der Basisprozess selbst kann ebenfalls eine relevante und korrekte Prozessvariante der Prozessfamilie darstellen.

Wir definieren *Ergebnismodell* und *Prozessfamilie* wie folgt:

[11] Zur Anwendung der Optionen verwenden wir die Funktion `calculateVariant` (vgl. Anhang A).
[12] Ein Beispiel für eine Prozessfamilie ist dargestellt in Abbildung 4.7.

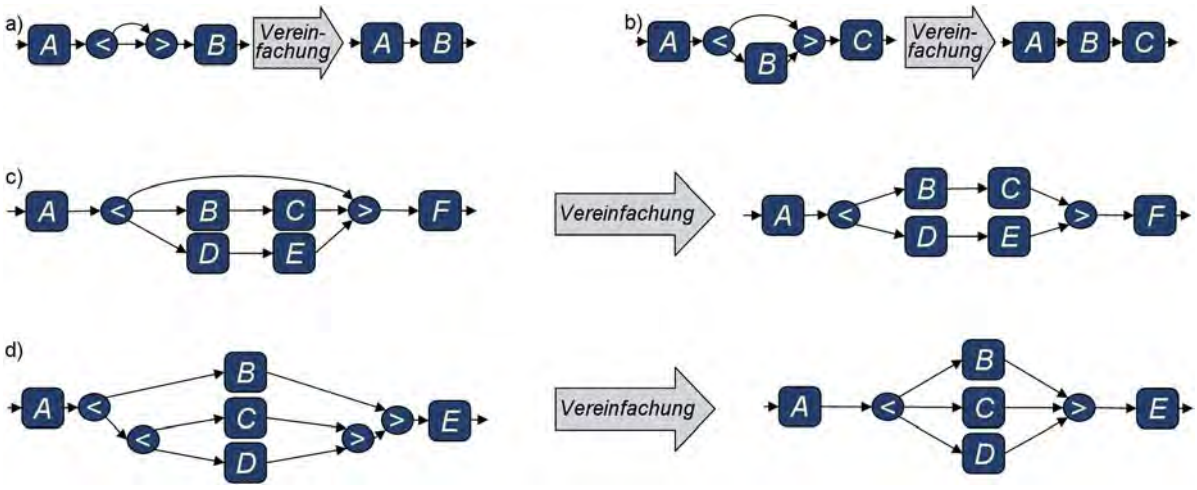


Abbildung 5.35: Beispiele für die Anwendung von Vereinfachungsregeln auf Ergebnismodelle

Definition 5.5 (Ergebnismodell und Prozessfamilie)
Sei $B = (P, AP)$ ein Basisprozess mit Optionsmenge O_B.

- *Sei weiter $\Sigma =< o_1, \ldots, o_n >$ eine Sequenz von Optionen mit $\{o_1, \ldots, o_n\} \subseteq O_B$ und $B[\Sigma\rangle B'$ mit $B' = (P', AP')$. Dann bezeichnen wir B' als **Prozessvariante**; das aus P durch Anwendung von Σ resultierende Prozessmodell P' nennen wir **Ergebnismodell**.*
- *Die Menge aller Prozessvarianten, die aus B bei gegebener Optionsmenge O_B konfiguriert werden können, bezeichnen wir als **Prozessfamilie** $\mathcal{PF}(B)$ von B.*
  $\mathcal{PF}(B) = \{B^* | B[\Sigma\rangle B^* \text{ mit } \Sigma =< o_1, \ldots, o_k >, k \in \mathbb{N} \wedge \{o_1, \ldots, o_k\} \subseteq \mathfrak{P}(O_B)\}$

Wie wir in Kapitel 6 noch sehen werden, erlaubt Provop auch Beschränkungen bezüglich (gemeinsam) anwendbarer Optionen.

5.7 Diskussion

In der wissenschaftlichen Literatur und in der Praxis existieren einige Ansätze zur Modellierung von Prozessvarianten. Nachfolgend stellen wir die wichtigsten von ihnen vor und grenzen sie gegenüber Provop ab.

Abbildung von Prozessvarianten nach dem Mehr-Modell Ansatz

Der in [LS06] beschriebene Ansatz vereinfacht die Handhabung ausmodellierter Prozessvarianten. Diese Varianten werden in einem sog. Varianten-Repository verwaltet. Mit Hilfe von Such-Algorithmen können dann diejenigen Prozessvarianten gefunden werden, die der Struktur der jeweils gegebenen Suchanfrage (in Form eines Prozessfragments) ähneln. Dabei werden die Suchanfrage und die Varianten im Repository auf Ähnlichkeit hin untersucht und die ähnlichsten Modelle im Repository geordnet nach dem Grad dieser Ähnlichkeit ausgegeben. Im Gegensatz dazu werden in Provop die Prozessvarianten nicht ausmodelliert (vgl. Abschnitt 4.2.1). Außerdem werden Prozessvarianten in Provop nicht durch Suchen und Anpassen gefundener Modelle erstellt, sondern basierend auf aktuellen Kontextinformationen konfiguriert.

Abbildung von Prozessvarianten nach dem Ein-Modell Ansatz

Ebenso wie Provop beschäftigt sich die Referenzprozessmodellierung mit der Konstruktion von Modellen ausgehend von einem Prozessmodell [Sch97, Bro03, FL04, vdADG+05]. Referenzprozessmodelle sind allgemein gültige Prozessmodelle, die für eine Klasse von Anwendungsfällen verwendet werden. Außerdem besitzen sie einen Empfehlungscharakter, d.h. „sie dienen als Ausgangslösung, aus der sich unternehmensspezifische Konkretisierungen ableiten lassen“ [Tho06].

Je nachdem, wie der Modellierer den Basisprozess vorgibt, sind die Eigenschaften und Merkmale von Referenzmodellen auf Provop übertragbar. So hat ein Standardprozess, der als Basisprozess dient, durchaus Empfehlungscharakter. Das Merkmal der Allgemeingültigkeit von Referenzprozessen kann in Provop für alle Prozessvarianten betrachtet werden, die in Summe eine Art „Klasse von Anwendungsfällen“ repräsentieren. In manchen Veröffentlichungen zur Referenzprozessmodellierung wird gefordert, dass ein Basisprozess für mindestens einen Anwendungsfall unverändert eingesetzt werden kann. Durch die verschiedenen Ansätze zur Festlegung des Basisprozesses in Provop, ist dies nicht immer gegeben. So sind Basisprozesse, die als Schnittmenge aller Prozessvarianten definiert werden oder für einen „minimalen Modellierungsaufwand“ optimiert sind (vgl. Abschnitt 5.2), konstruierte Prozesse ohne spezifischen Anwendungsfall.

Ein konkreter Ansatz zur Referenzprozessmodellierung bietet das Konzept der *Configurable Event-driven Process Chain (cEPC)* [RvdA07]. Er ermöglicht das Erstellen von Prozessvarianten auf Grundlage eines konfigurierbaren Referenzprozesses. Insbesondere können Funktionen und Strukturknoten bei der Prozessvariantenbildung so konfiguriert werden, dass sie entweder fester Bestandteil der Prozessvariante sind, ausgelassen werden müssen oder optional sind. Um im Prozessmodell zu kennzeichnen, welche Funktionen und Strukturknoten konfigurierbar sind und welche nicht, besitzen konfigurierbare Elemente einen stärkeren Rahmen. Des weiteren werden sie mit Labels versehen. Diese repräsentieren bestimmte Anforderungen, die bei der Konfiguration einer konkreten Prozessvariante berücksichtigt werden müssen, damit diese semantisch und strukturell korrekt gebildet werden kann. Diese Anforderungen werden durch teilweise komplexe Regeln beschrieben und können mehreren Elementen gleichzeitig zugeordnet werden (z.B., wenn für einen Split-Knoten und den zugehörigen Join-Knoten die gleiche Anforderung gilt).

Einen mit der Referenzprozessmodellierung verwandten Ansatz stellt [RMvdT09] vor. Er beschreibt, wie durch *Aggregation* von Prozessvarianten ein übergeordnetes Modell erzeugt wird. Dazu werden variantenspezifische Elemente in ein Prozessmodell integriert. Dieses Prozessmodell ist durch eine sog. *aggregate Event driven Process Chain (aEPC)* repräsentiert. Um in den aggregierten Prozessmodellen eindeutig kennzeichnen zu können, welches Element zu welcher Prozessvariante gehört, verwendet der Ansatz einfache Labels, die jeweils an Ereignisse und Funktionen der aEPCs geknüpft werden. Die einzelnen Prozessmodelle können durch Auswertung dieser Labels aus dem aggregierten Prozessmodell erzeugt werden.

Alle diese Ansätze der Referenzprozessmodellierung stellen im Prinzip Optimierungen des Ein-Modell Ansatzes dar (vgl. Abschnitt 4.2.2) und entsprechen der Modellierung des Basisprozesses als Obermenge aller seiner Prozessvarianten in Provop (vgl. Abschnitt 5.2). Im Gegensatz zu Provop bieten diese Ansätze bei der Variantenbildung jedoch keine Möglichkeit zum Erweitern des generischen Referenzprozesses. Auch das Verschieben von Prozesselementen ist nur durch die redundante Modellierung von Prozesselementen an verschiedenen Positionen möglich. Außerdem berücksichtigen diese Ansätze bisher nur den Kontrollfluss, d.h. Änderungen von Attributwerten sind ebenfalls nicht vorgesehen. Sie sind daher in erster

Linie für sehr ähnliche Prozessvarianten geeignet, die eine geringe Modelldifferenz aufweisen.

In der Literatur werden weiter Ansätze diskutiert, mittels denen aus verschiedenen Modellen ein Prozessmodell abgeleitet werden kann. Neben der bereits beschriebenen Aggregation [RMvdT09], sind dies Ansätze zum Berechnen eines optimalen Referenzprozesses, der eine minimale Editier-Distanz zu einer Menge von Prozessvarianten aufweist [LRWe08, LRWb08] sowie Ansätze zum Mergen von Graphstrukturen [ZWR01, AP03, DBDK04, AAAN+05, KPP06]. [AP03] erläutert z.B. das sog. *3-Wege-Mergen* von Graphen. Dabei wird zunächst das Delta der zu verschmelzenden Modelle zu einem gemeinsamen Ausgangsprozess in Form von Änderungsoperationen beschrieben. Anschließend kann durch Anwenden des Deltas (d.h. der identifizierten Sequenz von Änderungsoperationen) auf das jeweils andere Modell ein neues Modell abgeleitet werden. Dieses stellt dann die Verschmelzung der Modelle dar. [AP03] diskutiert ähnliche Problemstellungen, wie sie auch in Provop existieren, etwa die Nicht-Kommutativität von Änderungsoperationen.

Ansätze zur Beschreibung von Änderungsoperationen in der Prozessmodellierung

Generell kann jedes Prozessmodell aus einem anderen abgeleitet werden, indem bestimmte Anpassungen vorgenommen werden. Solche Anpassungen entsprechen verschiedenen Änderungsmustern, wie dem Löschen von Kanten oder Knoten. Eine detaillierte Betrachtung möglicher Änderungsmuster präsentiert [WRR08, RMRW08a].[13] Die hier vorgestellten Änderungsmuster werden in zwei Kategorien eingeteilt. Änderungsmuster zur Anpassung von Prozessmodellen und solche für planbare Abweichungen. Tabelle 5.11 gibt einen Überblick.

Tabelle 5.11: Änderungsmuster nach [WRR08]

CHANGE PATTERNS			
ADAPTATION PATTERNS (AP)			
Pattern Name	**Scope**	**Pattern Name**	**Scope**
AP1: Insert Process Fragment(*)	I / T	**AP8:** Embed Process Fragment in Loop	I / T
AP2: Delete Process Fragment	I / T	**AP9:** Parallelize Process Fragment	I / T
AP3: Move Process Fragment	I / T	**AP10:** Embed Process Fragment in Conditional Branch	I / T
AP4: Replace Process Fragment	I / T	**AP11:** Add Control Dependency	I / T
AP5: Swap Process Fragment	I / T	**AP12:** Remove Control Dependency	I / T
AP6: Extract Sub Process	I / T	**AP13:** Update Condition	I / T
AP7: Inline Sub Process	I / T		
PATTERNS FOR PREDEFINED CHANGES (PP)			
Pattern Name	**Scope**	**Pattern Name**	**Scope**
PP1: Late Selection of Process Fragments	I / T	**PP3:** Late Composition of Process Fragments	I / T
PP2: Late Modeling of Process Fragments	I / T	**PP4:** Multi-Instance Activity	I / T

I… Instance Level, T … Type Level
(*) A process fragment can either be an atomic activity, an encapsulated sub process or a process (sub) graph

Das Management von Prozessvarianten, wie in Provop vorgenommen, stellt im Prinzip eine Anwendungsdomäne für diese Änderungsmuster dar. Allerdings wird in Provop nur ein Teil der bekannten Änderungsmuster direkt umgesetzt. Zum Beispiel bieten wir keine explizite Ersetzungsoperation an. Diese kann aber z.B. durch Kombination von `INSERT`- und

[13] Eine formale Definition der Änderungsmuster finden sich in [RMRW08a].

DELETE-Operationen abgebildet werden. Nicht durch Provop-Änderungsoperationen umgesetzte Änderungsmuster beziehen sich auf die Verwendung von Sub-Prozessen (vgl. AP6 und AP7 in Tabelle 5.11).[14] Außerdem bietet Provop keine expliziten Änderungsoperationen für ungeplante Änderungen zur Laufzeit an (vgl. PP1 bis PP4 in Tabelle 5.11), und wir wenden Änderungsoperationen immer auf den Basisprozess an, d.h. wir implementieren Änderungsmuster nur für Prozesstypen nicht aber für Prozessinstanzen. Änderungsoperationen, die auf laufende Prozessinstanzen anwendbar sind, werden in [WRR08, RRKD05, Rei00, Dad05] beschrieben. Solche Ad-hoc Änderungen werden in erster Linie zur Beschreibung von ungeplanten und ggf. nur einmalig relevanten Abweichungen verwendet. Es ist nicht Ziel dieser Änderungsoperationen, dauerhafte und wiederverwendbare Prozessvarianten zu dokumentieren.

Ansätze zur Abbildung von (Software-) Produktvarianten

In der Softwareentwicklung nehmen Prozessvarianten traditionell eine wichtige Rolle ein. Neben grundlegenden Charakterisierungen von Variabilität [BB01] werden im Kontext von Software-Architekturen und Software-Produktfamilien auch Prozessvarianten behandelt [HP03, BGGB01]. So finden sich z.B. bei *PESOA* [BBG+05, PSWW05] Konzepte zur Modellierung von Prozessvarianten in einem UML-Modell. Es werden verschiedene Variabilitätsmechanismen (z.B. Vererbung, Parametrisierung, Erweiterungspunkte) vorgestellt und beispielhaft in verschiedenen UML-Modelltypen angewendet. Diese Variabilitätsmechanismen werden durch entsprechende UML-Konstrukte direkt in einem Prozess modelliert. Aufsetzpunkte sind in ähnlicher Form auch in der Literatur zu Softwareproduktfamilien beschrieben. Hier werden sie jedoch als *Variation Points* (dt. Variationspunkte) bezeichnet, über deren Typ (angegeben durch Kardinalitäten) bestimmt wird, wie die im Variation Point modellierten Prozessvarianten verwendet werden können [SvGB05, DSNB04, BBG+05].[15] Im Gegensatz zu Provop findet in diesen Ansätzen keine logische oder modellierungstechnische Trennung von Basisprozessen und Prozessvarianten statt. Auch beschränken sich in Provop die zusätzlich notwendig werdenden Konstrukte auf Optionen. Die unterschiedlichen Variabilitätsmechanismen werden durch intuitive Änderungsoperationen beschrieben. Dadurch wird die Darstellung von Basisprozessen und Optionen insgesamt übersichtlicher und verständlicher, und die Komplexität für den Variantenmodellierer reduziert.

Ein ebenfalls verwandter Bereich aus der Software-Entwicklung ist die Versionierung von Komponenten eines Softwareprodukts bzw. einer Software-Konfiguration [LDC+89, CW97, CW98, Jar05]. In [CW98] beschreiben die Autoren verschiedene Ansätze zur Verwaltung von Versionen, wie das *Change Oriented Versioning (COV)*. Hierbei werden Versionen durch Anwendung einer Sequenz von Änderungen repräsentiert, die auf eine gemeinsame Basis angewendet werden. Abbildung 5.36a stellt dar, wie den einzelnen Versionen (v1 bis v3) verschiedene Änderungen (a1 bis a4) zugeordnet werden können. Eine graphische Repräsentation dieser Matrix zeigt Abbildung 5.36b. Analog dazu stellt Provop ein *änderungsbasiertes Variantenmanagement* dar. Eine Darstellung der Prozessvarianten in einer Varianten-Optionen-Matrix bzw. einem Variantengraph, angelehnt an Abbildung 5.36, ist in Provop prinzipiell denkbar.

Ein anderer Bereich, in dem das Management von Varianten ein wichtiges Thema darstellt, ist die Produktentwicklung [Zen06, SG08]. Hier werden Stücklisten verwaltet, die angeben, welche Einzelteile mit welcher Zahl in einem Produkt verbaut werden. Eine Form der Stückliste, ist die Gleichteilestückliste. Diese führt alle Teile bzw. Teilegruppen auf, die in allen Produktvarianten gemeinsam auftreten. Eine andere Form ist die Plus-Minus-Stückliste. In

[14] Ein Sub-Prozess wird innerhalb eines Prozesses aufgerufen. Nach dessen Terminierung wird im aufrufenden Prozess mit der Ausführung fortgefahren.

[15] Eine gute Übersicht der Ansätze gibt [Ver07].

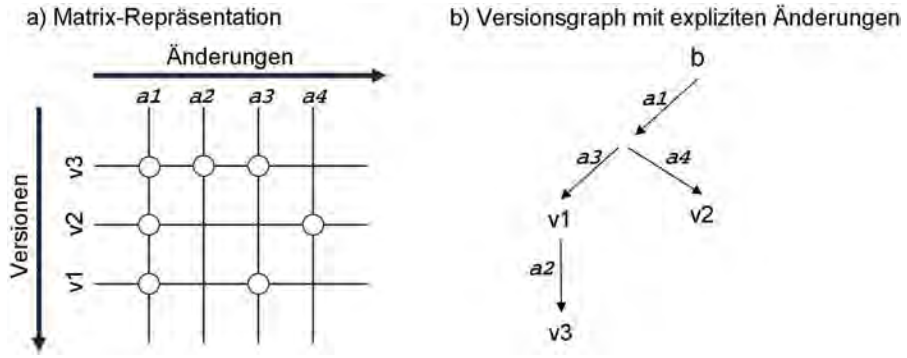


Abbildung 5.36: Änderungsbasiertes Versionieren nach [CW98]

ihr werden Varianten mit Hilfe von positiven bzw. negativen Vorzeichen abgebildet. Diese Vorzeichen zeigen an, welche Komponenten hinzukommen bzw. entfallen. Diese Ansätze zur Beschreibung von Stücklisten ähneln denen zur Festlegung eines Basisprozesses (z.B. als Schnitt- oder Obermenge aller Prozessvarianten). Stücklisten stellen im Prinzip Baumstrukturen dar. Dadurch sind für sie nicht alle möglichen Änderungsoperationen, wie in Provop vorgestellt, relevant. Zum Beispiel sind keine `MOVE`-Operationen erforderlich.

5.8 Zusammenfassung

Der Provop-Ansatz für das Management von Prozessvarianten basiert auf der Anwendung expliziter Änderungsoperationen. Diese werden auf einen Basisprozess angewendet, der vom Prinzip her ein „normales“ Prozessmodell darstellt. Ein solcher Basisprozess kann nach unterschiedlichen Aspekten festgelegt werden. Zum Beispiel kann er als Referenzprozess einer bestimmten Domäne dienen.

Im Gegensatz zu normalen Prozessmodellen kann ein Basisprozess mit speziellen Referenzpunkten (sog. Aufsetzpunkten) angereichert werden. Diese werden verwendet, um im Basisprozess explizit zu kennzeichnen, wo variantenspezifische Anpassungen vorgenommen werden können. Anpassungen werden in Form von Änderungsoperationen beschrieben. In Provop werden die Änderungsoperationstypen `INSERT`, `DELETE` und `MOVE` unterstützt. Diese erlauben eine Veränderung des Kontrollflusses und können sowohl auf einzelne Prozesselemente als auch auf komplexe Prozessfragmente angewendet werden. Zusätzlich wird mit der `MODIFY`-Operation eine vierte Änderungsoperation angeboten, mit deren Hilfe die Attribute von Prozesselementen angepasst werden können. Die Provop-Änderungsoperationen stellen einen mächtigen Satz an Basisoperationen dar, mit denen alle relevanten Anwendungsfälle abgedeckt werden können. Der Ansatz kann zudem flexibel um zusätzliche Änderungsoperationstypen erweitert werden.

Mehrere Änderungsoperationen können zu Gruppen zusammengefasst werden, die wir als Optionen bezeichnen. Zum Gruppieren von Änderungsoperationen in Optionen gibt es Richtlinien, die zu einer geeigneten Granularität führen, d.h. die Redundanz der Änderungsoperationen wird vermieden und die Wiederverwendbarkeit erhöht, bei gleichzeitig minimierter Anzahl an Optionen. Eine Option wird als atomares Objekt betrachtet. Das heißt es werden bei der Anwendung einer Option immer alle ihre Änderungsoperationen auf den Basisprozess angewendet. Das durch Anwendung von Optionen resultierende Prozessmodell wird als Ergebnismodell einer spezifischen Prozessvariante betrachtet. Es ist Teil einer Prozessfamilie, die der Menge aller ableitbaren und in der praxisrelevanten Prozessvarianten eines Prozesstyps entspricht.

Basierend auf den Grundlagen des Provop-Ansatzes wird im folgenden Kapitel beschrieben, wie Optionen abhängig von einem Kontext zur Anwendung auf den Basisprozess ausgewählt werden können und wie semantische und strukturelle Beziehungen zwischen Optionen dokumentiert und ausgewertet werden können.

6

Konfiguration von Prozessvarianten

Nachdem wir im vorherigen Kapitel die Modellierung von Basisprozessen und darauf basierenden Optionen beschrieben haben, diskutieren wir nun, wie durch Anwendung einer bestimmten Menge von Optionen auf den Basisprozess, die Ergebnismodelle von Prozessvarianten konfiguriert werden können. Kapitel 6 gliedert sich wie folgt: Abschnitt 6.1 motiviert die Konfiguration von Prozessvarianten. Abschnitt 6.2 beschreibt, wie Varianten manuell oder kontextabhängig konfiguriert werden können. Abschnitt 6.3 erläutert die Abbildung semantischer und struktureller Abhängigkeiten zwischen Optionen. Abschnitt 6.4 beschreibt die Sortierung der zur Anwendung auf den Basisprozess ausgewählten Optionen in die gewünschte Anwendungsreihenfolge, und Abschnitt 6.5 stellt vor, wie die Korrektheit der gesamten, konfigurierbaren Prozessfamilie gewährleistet werden kann. Abschließend diskutiert Abschnitt 6.6 verwandte Arbeiten; Abschnitt 6.7 fasst das Kapitel zusammen.

6.1 Motivation und Anforderungen

Die Konfiguration von Prozessvarianten folgt einem definierten Ablauf (vgl. Abbildung 6.1): In der Modellierungsphase beschreiben wir durch einen Basisprozess und zugehörige Anpassungen (Optionen) eine ganze Prozessfamilie. Zur Konfiguration einer konkreten Prozessvariante dieser Familie, muss der Variantenmodellierer angeben, welche der modellierten Optionen auf den Basisprozess anzuwenden sind. Das heißt, er muss eine Auswahl aus den zuvor modellierten Optionen treffen und die Reihenfolge ihrer Anwendung auf den Basisprozess fixieren. Dabei sollte der Modellierer den Kontext der gewünschten Variante berücksichtigen. Beispielsweise muss er beachten, ob eine Variante des Produktentwicklungsprozesses für PKWs oder LKWs benötigt wird. Entsprechend sind ggf. andere Optionen auf den Basisprozess anzuwenden, um die jeweilige Variante zu konfigurieren. Darüber hinaus müssen die ausgewählten Optionen strukturell und semantisch kompatibel sein, damit sie gemeinsam angewendet werden können.

Offensichtlich stellt die Auswahl der Optionen (vgl. Abbildung 6.1a) eine nicht triviale Aufgabe dar. Um den Variantenverantwortlichen bei ihrer Durchführung bestmöglich zu unterstützen, muss eine automatische Auswahl der Optionen möglich sein und zwar basierend auf

Abbildung 6.1: Ablauf der Konfiguration von Prozessvarianten in Provop

dem Prozesskontext: Ist der mögliche Kontext eines Prozesstyps bekannt, muss der Kontextmodellierer diesen in einem Modell erfassen können. Diese Information sollte ausgewertet werden können, so dass die Menge der im gegebenen Kontext relevanten Optionen möglichst automatisch bestimmt werden kann. Dazu müssen der Kontext und der Zusammenhang zwischen Kontext und Optionen in einer maschinenlesbaren Form beschrieben werden.

Bei der Auswahl von Optionen spielen strukturelle und semantische Abhängigkeiten zwischen den Optionen eine wichtige Rolle (vgl. Abbildung 6.1b). So kann der Variantenmodellierer angeben, dass bestimmte Optionen nicht gemeinsam angewendet werden dürfen oder eine Option die Anwendung einer anderen voraussetzt. Solche Beziehungen zwischen Optionen muss der Variantenmodellierer in Provop explizit dokumentieren können. Außerdem muss diese Information genutzt werden können, um die semantische Kompatibilität einer Menge ausgewählter Optionen automatisch prüfen zu können. Das System kann dann, für eine gegebene Auswahl, feststellen, ob alle Beschränkungen eingehalten werden oder, ob Verletzungen auftreten. Diese „Kompatibilitätsprüfung“ darf nicht abhängig sein von der Art und Weise, wie die Menge von Optionen bestimmt wird. Mit anderen Worten, sie muss sowohl bei der manuellen als auch bei der kontextbasierten Konfiguration anwendbar sein.

Nach Feststellung der Kompatibilität von Optionen, müssen diese zu einer Optionsfolge sortiert werden (vgl. Abbildung 6.1c). Dies wird erforderlich, da Optionen i.A. nicht kommutativ sind. Um entsprechenden Konflikten vorzubeugen, muss der Variantenmodellierer die Anwendungsreihenfolge der Optionen explizit angeben können.

Ist die Anwendungsreihenfolge der (kompatiblen) Optionen bekannt, werden diese auf den Basisprozess angewendet (vgl. Abbildung 6.1d). Dabei müssen jeweils alle Änderungsoperationen der Optionen, in der angegebenen Reihenfolge, auf den Basisprozess wirken und diesen sukzessive zum Ergebnismodell transformieren. Dies ist jedoch nur möglich, wenn die Änderungsoperationen tatsächlich auf den Basisprozess angewendet werden können, d.h. ihre jeweiligen Vorbedingungen erfüllt sind.

Abschließend ist die Sicherstellung der Korrektheit der Ergebnismodelle wesentlich, um deren Ausführbarkeit in einem Workflow-Management-System (WfMS) zu ermöglichen (vgl. Abbildung 6.1e). Dazu müssen wir prüfen, ob das Ergebnismodell den Korrektheitskriterien des diesem WfMS zugrunde liegenden Prozess-Metamodells genügt. Diese Eigenschaften müssen nicht nur für ein einzelnes Modell geprüft werden, sondern für die gesamte Prozessfamilie.

Daraus ergeben sich im Wesentlichen die in Tabelle 6.1 aufgeführten Anforderungen.

6.2 Auswahl von Optionen für eine konkrete Prozessvariante

Wie kann nun eine Teilmenge von Optionen zur Anwendung auf den Basisprozess bestimmt werden. Abschnitt 6.2.1 stellt zunächst eine rein manuelle Vorgehensweise vor. Für eine kontextabhängige Konfiguration beschreibt Abschnitt 6.2.2, wie Informationen über den Kontext zur automatischen Auswahl von Optionen verwendet werden.

Tabelle 6.1: Anforderungen an die Konfiguration von Prozessvarianten

Manuelle Auswahl von Optionen

Anforderung 6.1 *Der Variantenverantwortliche muss aus allen modellierten Optionen manuell diejenigen auswählen können, die im gegebenen Kontext auf den Basisprozess angewendet werden sollen.*

Kontextbasierte Auswahl von Optionen

Anforderung 6.2 *Der Anwendungskontext einer spezifischen Prozessvariante muss in einfacher Art und Weise angegeben werden können. Dazu muss ein intuitives Modell zur Erfassung und Beschreibung des Kontextes von verschiedenen Prozesstypen definiert werden können.*

Anforderung 6.3 *Die Beziehungen zwischen modellierten Optionen und Kontextmodell müssen in maschinenlesbarer Form beschrieben werden, so dass die Bewertung der Gültigkeit von Optionen in einem bestimmten Kontext automatisch möglich ist.*

Beziehungen zwischen Optionen

Anforderung 6.4 *Strukturelle und semantische Abhängigkeiten zwischen zwei oder mehreren Optionen müssen explizit beschrieben werden können.*

Reihenfolge der Anwendung von Optionen

Anforderung 6.5 *Für nicht-kommutative Optionen muss eine explizite Anwendungsreihenfolge der Optionen festgelegt werden können.*

Korrektheit der Prozessfamilie

Anforderung 6.6 *Für eine Prozessfamilie muss die Korrektheit aller Variantenmodelle, unter Berücksichtigung des Kontextmodells sowie semantischer und struktureller Abhängigkeiten zwischen den Optionen, gewährleistet werden.*

6.2.1 Manuelle Auswahl

Zur Auswahl von Optionen ist dem Variantenverantwortlichen eine einfache Methode anzubieten, die ihm erlaubt, direkt aus einer entsprechenden Optionsliste die relevanten Optionen für eine konkrete Prozessvariante zu bestimmen (vgl. Anforderung 6.1). Dies ist vor allem dann erforderlich, wenn der Anwendungskontext der Ergebnismodelle nicht bekannt oder nicht (in maschinenlesbarer Form) dokumentiert ist.

Beispiel 6.1 (Manuelle Anpassung eines Referenzprozesses zur Bestellabwicklung) *Der SAP-Referenzprozess zur Bestellabwicklung [Hel07] wird in verschiedenen Unternehmen, wie z.B. BMW, VW, Audi oder BASF, und somit in sehr unterschiedlichem Kontext eingesetzt. Um hier die gewünschte Variante zu konfigurieren, kann der Kunde aus einer Liste von Alternativen auswählen (vgl. Tabelle 6.2). Auf diese Weise kann das Domänenwissen zur Konfiguration einer Variante verwendet werden.*

Tabelle 6.2: Manuelle Auswahl von Prozessfragmenten (exemplarisch)

Unternehmen /Variante	BMW	VW	Audi	BASF
elektronische Verarbeitung	x	-	x	x
manuelle Verarbeitung	-	x	-	x

Die manuelle Auswahl gibt dem Variantenverantwortlichen die Möglichkeit, sich bewusst für oder gegen bestimmte Optionen zu entscheiden. Dabei wird im Prinzip auch eine „se-

mantische“ Qualitäts- und Korrektheitskontrolle durch den Variantenverantwortlichen vorgenommen. Der Ansatz ist jedoch nicht für alle Szenarien gleich gut geeignet. Zum Beispiel steigt großer Zahl von Optionen das Fehlerrisiko durch Auswahl falscher bzw. Auslassen relevanter Optionen. Es sind daher weitergehende Verfahren erforderlich, die dem Variantenverantwortlichen entsprechende Funktionalität zur Absicherung seiner Auswahl bieten oder eine automatisierte Auswahl von Optionen erlauben. Nichtsdestotrotz unterstützt Provop eine manuelle Optionsauswahl, um die benötigte Flexibilität zu erzielen.

6.2.2 Kontextabhängige Auswahl

Bei der (manuellen) Konfiguration von Varianten kann beobachtet werden, dass die Entscheidung, ob eine spezifische Option in eine Auswahl übernommen werden soll oder nicht, meist aus denselben Gründen getroffen wird (z.B. abhängig von bestimmten Rahmenbedingungen). Das heißt der Variantenmodellierer muss zur Konfiguration einer Prozessvariante meist ähnliche Entscheidungsprozesse durchlaufen. Um ihn bei dieser Aufgabe zu unterstützen bzw. um diese Konfiguration zu automatisieren, können Auswahlkriterien definiert werden, welche die Gültigkeit von Optionen explizit beschreiben. Das heißt, es kann direkt in den Optionen dokumentiert werden, wann diese anzuwenden sind. Eine Prozessvariante wird demzufolge konfiguriert, indem die Auswahlkriterien der Optionen vom System ausgewertet werden. In Provop basieren die Auswahlkriterien der Optionen und deren Auswertung auf der Betrachtung des domänenspezifischen Kontextes, für den eine Variante erstellt werden soll. Wir bezeichnen eine solche Konfiguration als *kontextbasiert* bzw. *kontextabhängig*. Abschnitt 6.2.2.1 beschreibt, wie wir den domänenspezifischen Kontext erfassen und dokumentieren können. Anschließend stellt Abschnitt 6.2.2.2 vor, wie wir die Kontextabhängigkeit von Optionen erzielen.

6.2.2.1 Erfassung des Kontextes in einem Kontextmodell

Um eine kontextbasierte Konfiguration von Prozessvarianten zu erlauben, müssen wir den aktuell gegebenen Kontext erfassen und beschreiben (vgl. Anforderung 6.2). Darüber hinaus müssen diese Kontextinformationen in einer maschinenlesbaren Form vorliegen, damit wir eine automatische Konfiguration der Prozessvarianten vornehmen können. Neben der Erfassung aktuell relevanter Kontextinformationen muss es außerdem möglich sein, zentral in einem *Kontextmodell* alle möglichen Kontexte, die zur Bildung der Gesamtheit möglicher Varianten eines Prozesstyp erforderlich sind, in einer intuitiven Art und Weise zu verwalten. Diese zentrale Verwaltung ermöglicht es allen Variantenmodellierern, auf die gleichen Rahmenbedingungen Bezug zu nehmen und somit einheitlich den für eine Prozessvariante gegebenen Kontext zu beschreiben.

Um die Erfassung des Kontextes intuitiv und einfach zu gestalten, beschreiben wir in Provop diesen mit Hilfe einer Menge von *Kontextvariablen*. Jede Kontextvariable beschreibt genau eine Rahmenbedingung bzw. eine Dimension des Kontextes, und wird mit ihren Attributen (d.h. Name, Beschreibung und Wertebereich) in einer *Kontexttabelle* vorgehalten (vgl. Tabelle 6.3). Wir gehen davon aus, dass der Wertebereich einer Kontextvariable endlich ist und nur diskrete Werte umfasst.[1]

Tabelle 6.3 zeigt einen Ausschnitt aus dem Kontextmodell der Produktentwicklung bei der Daimler AG. Dieses Kontextmodell ist spezifisch für diese Domäne bzw. diesen Unternehmensbereich. Ein anderes Beispiel bieten Werkstattprozesse. Typische Kontextvariablen für

[1] Dies ist in der Praxis i.A. auch ausreichend, d.h. es ergibt sich hier keine wesentliche Einschränkung der Mächtigkeit des Ansatzes. Bei stetigen Wertebereichen genügt zudem oftmals eine diskrete Aufteilung in Teilintervalle.

Tabelle 6.3: Ausschnitt aus dem Kontextmodell der Produktentwicklung (vereinfacht)

Variablenname	Beschreibung	Wertebereich
Geschäftsbereich	Geschäftsbereiche der Daimler AG	Daimler Trucks (DT), Daimler Buses (DB), Mercedes-Benz Vans (MV), Mercedes-Benz Cars (MBC)
Fahrzeugtyp	Bezeichnung des Fahrzeugtyps	LKW, Van, Bus, PKW
Marke	Bezeichnung der Marke	Mercedes Benz, Smart, Fuso, [..]
Phase	Entwicklungsphase des Fahrzeugprojekts	Vorentwicklung, Entwicklung, Anlauf, Produktion

einen Reparaturprozess in einer Werkstatt etwa sind das Land, in welchem sich die Werkstatt befindet, und der Typ des zu reparierenden Fahrzeugs. Somit sind für verschiedene Unternehmensbereiche jeweils unterschiedliche Kontextvariablen relevant. Entsprechend können die Kontextmodelle, je nach Prozesstyp, variieren.

Wie motiviert sollen durch die Dokumentation eines Kontextmodells alle möglichen Kontexte eines Prozesstyps erfasst werden. Dies ist durch Angabe einer entsprechenden Kontexttabelle möglich. Einen konkreten Kontext, für den eine Prozessvariante konfiguriert werden soll, spezifizieren wir auf Grundlage solcher Tabellen, und zwar durch Belegung der Kontextvariablen mit konkreten Werten. Ein möglicher Kontext für das Kontextmodell der Produktentwicklung aus Tabelle 6.3 etwa ist:

```
{<Geschäftsbereich, Daimler Buses (DB)>, <Fahrzeugtyp, Bus>,
<Marke, Mercedes Benz>, <Phase, Produktion>}
```

Im Folgenden bezeichnen wir eine solche Wertekombination von Variablen eines Kontextmodells als *Kontextbeschreibung (kurz: Kontext)*. Bezogen auf ein gegebenes Kontextmodell lässt sich feststellen, dass nicht immer alle möglichen Kontextbeschreibungen (d.h. alle Kombinationen von Wertebelegungen für Kontextvariablen) für die Praxis relevant sind [CW98]. Zum Beispiel ist bei der Produktentwicklung der Geschäftsbereich „Daimler Trucks“ nicht an der Entwicklung von PKWs beteiligt. Die Kontextbeschreibung:

```
{<Geschäftsbereich, Daimler Trucks>, <Fahrzeugtyp, PKW>}
```

macht deshalb wenig Sinn. Dementsprechend benötigen wir für diese Kontextbeschreibung auch keine gültige Prozessvariante. Im Gegensatz zu solch *ungültigen Kontextbeschreibungen* muss für jede *gültige Kontextbeschreibung* eine Prozessvariante definiert sein.

Die Unterscheidung von gültigen und ungültigen Wertekombinationen für Kontextvariablen ist in Provop sehr wichtig, da auf diese Weise der tatsächliche Bedarf an Prozessvarianten für einen Prozesstyp deutlich wird. Des Weiteren kann dadurch der Aufwand für die Modellierung und Konfiguration der Prozessvarianten reduziert werden (vgl. [CW98, Mun96]). Um eine entsprechende Unterscheidung vornehmen zu können, müssen die möglichen Wertekombinationen der Kontextvariablen des zugrunde liegenden Kontextmodells betrachtet werden. Im obigen Beispiel etwa werden die Kontextvariablen `Geschäftsbereich` und `Fahrzeugtyp` betrachtet, deren Werte „Daimler Trucks“ bzw. „PWK“ in Kombination miteinander eine ungültige Kontextbeschreibung darstellen. Solche ungültigen Wertekombinationen sollen durch den Kontextmodellierer explizit ausgeschlossen werden können.

Ein nahe liegender Ansatz zur Bewertung der Gültigkeit bietet eine n-dimensionale Matrix der Werte aller n Variablen eines Kontextmodells (vgl. [Sci94]). Für die Kontextvariablen der Produktentwicklung aus Tabelle 6.3, d.h. für die Variablen `Geschäftsbereich`, `Fahrzeugtyp` und `Marke`, entsteht auf diese Weise ein *Kontextwürfel* (vgl. Abbildung 6.2). Seine Achsen entsprechen den drei Kontextvariablen aus dem gegebenen Kontextmodell. Die möglichen

Werte der Kontextvariablen unterteilen den Kontextwürfel derart, dass kleinere „Teil-Würfel“ entstehen. Jeder Teil-Würfel repräsentiert exakt eine mögliche (d.h. gültige oder ungültige) Kontextbeschreibung. Die Bewertung der Gültigkeit wird dann auf Basis der Teil-Würfel vorgenommen. In Abbildung 6.2 ist diese Bewertung durch eine entsprechende Einfärbung der Teil-Würfel visualisiert.

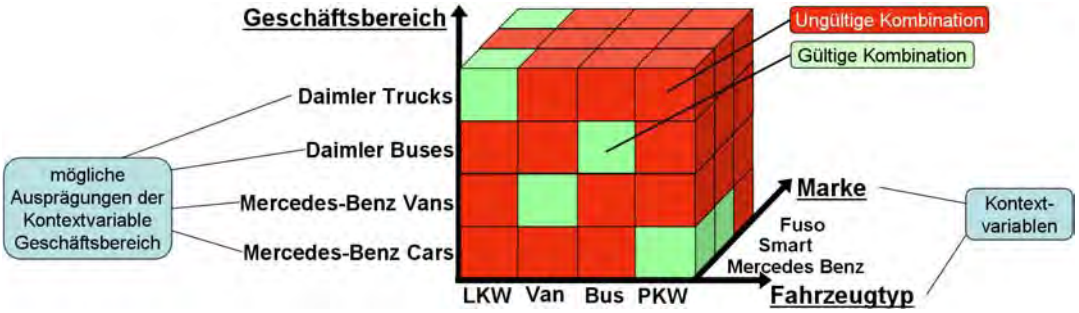


Abbildung 6.2: Kontextwürfel mit gültigen (grün) und ungültigen (rot) Kontextkombinationen

Der dargestellte Kontextwürfel zeigt anschaulich, welche Wertekombinationen für Kontextvariablen gültig sind und welche nicht. Für ein komplexeres Kontextmodell mit vielen Kontextvariablen und großen Wertebereichen ist ein Kontextwürfel bzw. -körper jedoch unpraktikabel. Unter der Annahme, dass alle n Kontextvariablen mindestens m verschiedene Werte annehmen können, gibt es entsprechend mindestens m^n mögliche und auf ihre Gültigkeit hin zu untersuchende Wertekombinationen. Bereits für 5 Kontextvariablen mit je 10 Werten gibt es somit 100.000 Teil-Körper. Die Bewertung aller Teil-Körper als gültig bzw. ungültig würde einen sehr hohen Pflegeaufwand bedeuten. Das entstehende mehrdimensionale Gebilde wäre zudem für den Modellierer nicht mehr handhabbar. Des Weiteren ist zu bedenken, dass wir bisher nur Kontextvariablen mit endlichen Wertebereichen betrachten; jedoch können in der Praxis auch unendliche Wertebereiche auftreten (z.B. für Zahlen). In diesem Fall entstehen unendlich viele Teil-Körper, deren (Un-)Gültigkeit praktisch nicht mehr bewertbar ist.[2]

Provop bietet eine benutzerfreundlichere Bewertung zahlreicher Wertekombinationen an. Dazu drücken wir die (Un-)Gültigkeit von Kontextbeschreibungen durch explizite *Kontextregeln* aus. Diese werden als einfache `WENN-DANN`-Formeln spezifiziert (vgl. Beispiel 6.2). Um nicht für jede gültige bzw. ungültige Kontextbeschreibung eine Regel definieren zu müssen, können sich Kontextregeln auch auf ganze Gruppen von Kontextbeschreibungen beziehen (z.B. durch Ausblenden einer Kontextvariable). Darüber hinaus fordern wir nicht, dass Kontextregeln disjunkt spezifiziert sind, solange die (Un-)Gültigkeit einer Kontextbeschreibung eindeutig ermittelt werden kann.

Beispiel 6.2 (Beschreibung der (Un-)Gültigkeit von Kontexten durch Kontextregeln) *Für das Kontextmodell aus Tabelle 6.3 seien folgende drei Kontextregeln gegeben:*

Regel 1:
WENN Fahrzeugtyp = „LKW“ DANN Geschäftsbereich = „DT“ UND Marke IN {Mercedes Benz, Fuso}
Regel 2:
WENN Fahrzeugtyp = „Bus“ DANN Geschäftsbereich = „DB“
Regel 3:

[2] Zur Handhabung unendlicher Wertebereiche können die relevanten Grenzwerte betrachtet werden, was eine Abbildung auf Intervalle (und somit auf endliche Werte) ermöglicht. Dies bedeutet jedoch einen hohen zusätzlichen Analyseaufwand für die Identifikation dieser Grenzwerte.

```
WENN Fahrzeugtyp = „PKW“ DANN Geschäftsbereich = „MBC“ UND Marke IN
{Mercedes Benz, Smart}
```

Die obigen Regeln sind positiv formuliert, d.h. sie beschreiben, welche Kontextbeschreibungen gültig sind. Alle drei Regeln beziehen sich nur auf einen Teil der spezifizierten Kontextvariablen des zugrunde liegenden Kontextmodells. So gibt Regel 2 an, dass unabhängig von den konkreten Werten der Kontextvariablen `Marke` *und* `Phase`*, eine Kontextbeschreibung mit* `Fahrzeugtyp = Bus` *nur dann gültig ist, wenn der Geschäftsbereich als* `DB`*' (DB = Daimler Buses) angegeben ist. Tabelle 6.4 zeigt Beispiele für die Auswertung der Gültigkeit von Kontextbeschreibungen basierend auf den obigen Kontextregeln 1 bis 3. Die Kontextbeschreibungen 2 und 3 sind hierbei ungültig, da sie gegen die definierten Kontextregeln verstoßen. Kontextbeschreibung 4 ist zwar gültig, allerdings semantisch nicht sinnvoll. Zum Ausschließen dieser Kontextbeschreibung sollte daher eine zusätzliche Regel formuliert werden.*

Tabelle 6.4: Prüfen der Gültigkeit von Kontextbeschreibungen

Nr.	Kontextbeschreibung	Gültig
1	<Geschäftsbereich, Daimler Buses (DB)>, <Fahrzeugtyp, Bus>, <Marke, Mercedes Benz>, <Phase, Produktion>	ja
2	<Geschäftsbereich, Daimler Buses (DB)>, <Fahrzeugtyp, LKW>	nein (Regel 1)
3	<Fahrzeugtyp, PKW>, <Marke, Fuso>, <Phase, Entwicklung>	nein (Regel 4)
4	<Geschäftsbereich, Daimler Vans (DV)>, <Marke, Fuso>, <Phase, Anlauf>	ja
5	<Marke, Mercedes Benz>	ja

Die Gültigkeit einer Kontextbeschreibung wird in Provop durch die Funktion `contextDescriptionValid` (vgl. Anhang A) geprüft. Ihr Ergebnis ist `true`, wenn die übergebene Kontextbeschreibung alle Kontextregeln eines Kontextmodells erfüllt.

Nach den vorangegangenen Erläuterungen zum Kontext in Provop, definieren wir die Begriffe Kontextvariable, Kontextmodell und Kontextbeschreibung wie folgt formal:

Definition 6.1 (Kontextvariable, -beschreibung, -modell)

a) *Eine Kontextvariable ist ein Tupel* $v = (VariableName, ValueRange)$ *mit:*

- $VariableName$ *ist der eindeutige Bezeichner der Kontextvariable* v*.*
- $ValueRange$ *ist der endliche Wertebereich der Kontextvariable* v*.*

b) *Eine Kontextbeschreibung ist ein Tupel* $cd = (V, Value)$ *mit:*

- $V = \{v_1, \ldots, v_n\}$ *ist eine Menge von Kontextvariablen.*
- $Value : V \rightarrow definedValue$ *ordnet jeder Kontextvariable* $v_i = < VariableName_i, VariableRange_i >$ *einen Wert* $Value(v_i) \in ValueRange_i$ *zu.*

c) *Ein Kontextmodell ist ein Tupel* $CM = (V, Validity)$ *mit:*

- $V = \{v_1, \ldots, v_n\}$ *ist eine Menge von CM zugeordneten Kontextvariablen.*
- *Für die Menge* $CD = \{cd_1, \ldots, cd_n\}$ *der möglichen Kontextbeschreibungen definiert die Funktion* $Validity : CD \rightarrow \{true, false\}$ *für jede Kontextbeschreibung* cd_i*, ob diese Wertekombination erlaubt ist. Jede Kontextbeschreibung* cd_i *legt hierbei die Werte* $Value(v_i)$ *für alle Kontextvariablen* $v_i \in V$ *des Kontextmodells fest. Das heißt* $cd = (V, Value)$*.*

6.2.2.2 Beschreibung der Kontextabhängigkeit von Optionen

Nach Festlegung des Kontextmodells für einen bestimmten Prozesstyp, bleibt die Frage, wie eine konkrete Prozessvariante kontextbasiert konfiguriert werden kann, noch unbeantwortet. So ist noch nicht klar, wie Prozessvarianten einer bestimmten Kontextbeschreibung zugeordnet werden können. Abbildung 6.3 zeigt beispielhaft die Zuordnung von Prozessvarianten zu Teil-Würfeln des Kontextwürfels der Produktentwicklung (vgl. Abbildung 6.2).[3]

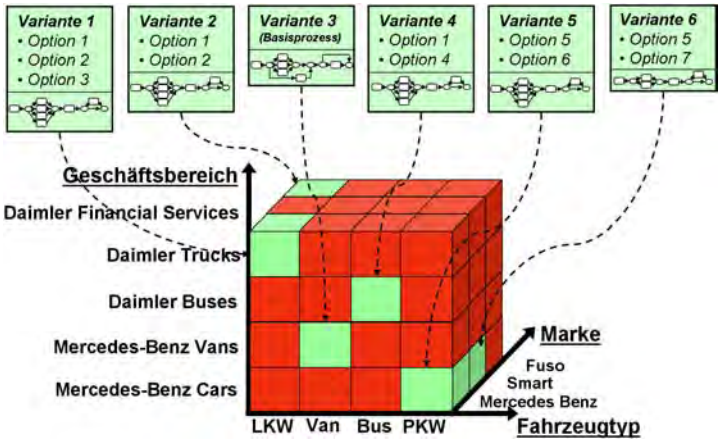


Abbildung 6.3: Gültigkeit von Varianten (bzw. Optionen) in bestimmten Kontexten

Für jede der in Abbildung 6.3 dargestellten Varianten ist die *Optionsmenge* angegeben, die jeweils zu ihrer Bildung erforderlich ist. Diese Optionsmengen müssen bei einer kontextbasierten Konfiguration, abhängig von den vorliegenden Kontextinformationen, ermittelt werden (vgl. Anforderung 6.3). Dazu ist für jede modellierte Option zu entscheiden, ob sie in der aktuell betrachteten Kontextbeschreibung erforderlich ist oder nicht. Die Zuordnung von Varianten zu einem bestimmten Kontext findet somit auf Ebene der Optionen statt.

Indem wir für einzelne Optionen definieren, dass diese in verschiedenen Kontextbeschreibungen erforderlich werden, können diese Optionen zur Konfiguration verschiedener Prozessvarianten verwendet werden. Ein Beispiel hierzu zeigt Abbildung 6.3. Hier wird Option 1 zur Bildung der Varianten 1, 2 und 4, jeweils in Kombination mit anderen Optionen und für unterschiedliche Kontextbeschreibungen, relevant.

Die Bewertung der Gültigkeit einer Option bei vorliegender Kontextbeschreibung wird durch *Kontextbedingungen* realisiert. Eine solche Bedingung bezieht sich, ebenso wie die Kontextregeln, auf das Kontextmodell des zugehörigen Basisprozesses bzw. den entsprechenden Prozesstyp. Im Unterschied zu Kontextregeln werden Kontextbedingungen jedoch nur als `WENN`-Formel beschrieben. Der `DANN`-Teil ist mit den beschriebenen Änderungsoperationen gegeben. Jede Option kann mit maximal einer (ggf. komplexen) Kontextbedingung verknüpft werden, d.h. die Kontextbedingung einer Option ist eindeutig. Dies bedeutet, dass die Kontextbedingung immer für die gesamte Option gilt, d.h. für alle in der Option beschriebenen Änderungsoperationen.[4]

Beispiel 6.3 (Optionen mit Kontextbedingungen) *In Abbildung 6.4a definieren wir für einen gegebenen Basisprozess zwei Optionen. Die Kontextbedingung von Option 1 sagt aus, dass diese Option*

[3] Im Folgenden verwenden wir zur anschaulichen Darstellung von Kontextmodellen und (un-)gültigen Kontextbeschreibungen meist einen entsprechenden (eingefärbten) Kontextwürfel.

[4] Auf diese Weise wird die geforderte Atomarität von Optionen bewahrt, welche besagt, dass alle Änderungsoperationen, die in einer Option gruppiert werden, als ein Objekt gehandhabt werden müssen (vgl. Abschnitt 5.5).

nur dann angewendet werden darf, wenn der betrachtete Kontext die Bedingung `Fahrzeugtyp = LKW` *erfüllt. Analog legen wir für Option 2 fest, dass diese zur Anwendung kommen soll, falls die Kontextbedingung* `Marke = Mercedes Benz` *erfüllt ist. Bei Betrachtung der unterschiedlichen Kontexte kann nun jeweils die entsprechende Optionsmenge bestimmt und auf den Basisprozess angewendet werden. Daraus resultiert eine Prozessfamilie aus vier Ergebnismodellen (vgl. Abbildung 6.4b).*

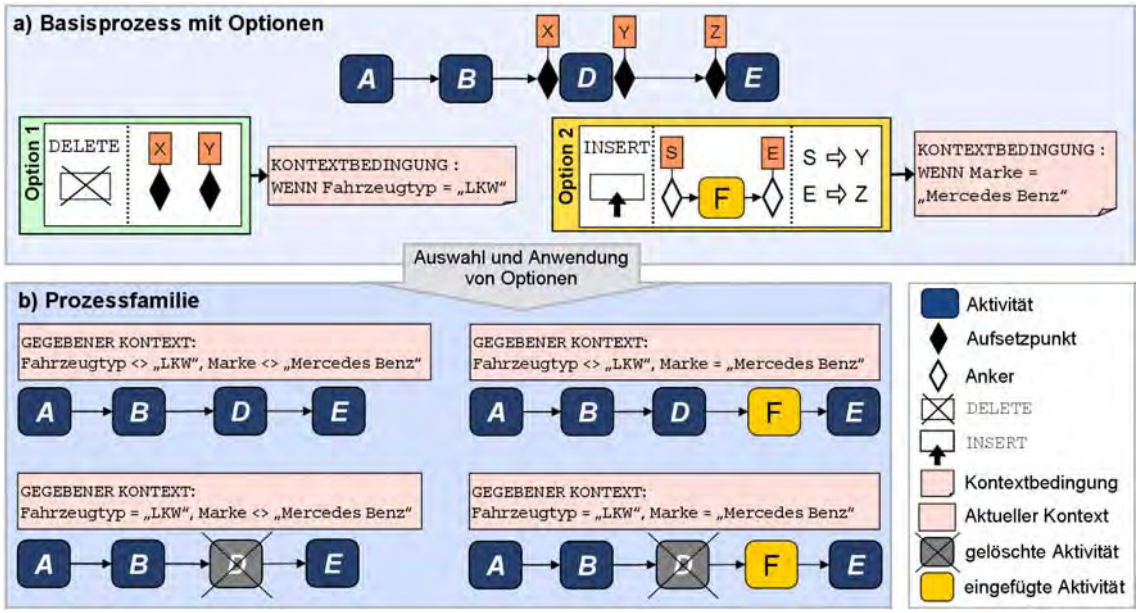


Abbildung 6.4: Kontextbasierte Auswahl und Anwendung von Optionen zur Ableitung der Prozessfamilie

Zur Auswertung der Kontextbedingungen einer Option verwenden wir die Funktion `contextRuleValid` (vgl. Anhang A). Der Rückgabewert der Funktion ist `true`, wenn die Kontextbedingung einer Option in einer gegebenen Kontextbeschreibung erfüllt ist. Andernfalls ist ihr Rückgabewert `false`. Eine Option, für die keine Kontextbedingung angegeben ist, wird immer (d.h. in jeder gültigen Kontextbeschreibung) auf den Basisprozess angewendet. Die Funktion `contextRuleValid` gibt in diesem Fall den Wert `true` zurück.

Für die Auswertung der Kontextbedingungen ist die Angabe einer konkreten Kontextbeschreibung erforderlich. Eine Fragestellung dabei ist, wie wir mit Kontextvariablen umgehen, für die kein Wert angegeben ist (d.h. es liegen nur unvollständige Angaben über den Kontext vor). Hier sind im Prinzip drei Ansätze denkbar:

- **Ansatz 1 (Default-Werte):** Für jede Kontextvariable wird ein Default-Wert definiert, der bei fehlender Angabe verwendet wird. In diesem Fall müsste diese Information im Kontextmodell ergänzt werden. Nachteilig ist jedoch, dass für den Variantenmodellierer bzw. -verantwortlichen ggf. nicht transparent wird, wie sich dieser Default-Wert bei der Auswertung der Bedingungen auswirkt.

- **Ansatz 2 (Nichtauswertbarkeit):** Kontextbedingungen, die auf Basis einer Kontextvariable definiert wurden, für die kein Wert spezifiziert ist, sind nicht auswertbar. Entsprechend wird dem Variantenverantwortlichen ein Fehler gemeldet (vgl. Tabelle 6.5). Auf diese Weise wird zwar transparent, welche notwendigen Angaben zur Auswertung fehlen, allerdings ist ein manuelles Eingreifen des Variantenverantwortlichen erforderlich.

- **Ansatz 3 (Gesamter Wertebereich):** Für jede undefinierte Variable wird ihr gesamter Wertebereich als möglicher Wert der Variable angenommen. In diesem Fall können die Kontextbedingungen der Optionen ausgewertet werden. Dieser Ansatz ist semantisch fragwürdig, da ggf. Optionen gleichzeitig relevant werden, die nicht gemeinsam angewendet werden sollen und daher für unterschiedliche Kontextbeschreibungen definiert wurden.[5]

Um eine möglichst hohe Transparenz für den Variantenverantwortlichen zu schaffen, verfolgen wir für die statische Konfiguration von Prozessvarianten in Provop den zweiten Ansatz.

Beispiel 6.4 (Auswertung von Kontextbedingungen) *Basierend auf dem Kontextmodell aus Tabelle 6.3 zeigt Tabelle 6.5 eine gültige Kontextbeschreibung sowie die Optionen 1 bis 5, deren Gültigkeit in einem bestimmten Kontext durch Angabe einer entsprechenden Kontextbedingung definiert ist. Auf jede Option wenden wir nun die Funktion* `contextRuleValid` *(vgl. Anhang A) an, um die Gültigkeit der Kontextbedingungen im angegebenen Kontext zu bestimmen. Auf diese Weise können wir erkennen, dass im gegebenen Kontext nur die Optionen 2 und 4 zur Anwendung kommen sollen, während Option 1 nicht ausgewählt wird.*
Da für die Kontextvariable „Phase" kein Wert spezifiziert ist, ist der angegebene Kontext unvollständig. Entsprechend sind die Kontextbedingungen der Optionen 3 und 5 nicht auswertbar und führen zur dargestellten Fehlermeldung. Die Bedingung von Option 4 basiert zwar ebenfalls auf der Variable „Phase", allerdings ist sie durch Angabe eines Wertes für die Variable „Fahrzeugtyp" bereits eindeutig auswertbar.

Tabelle 6.5: Auswertung von Kontextbedingungen

Gegebene Kontextbeschreibung: <Fahrzeugtyp, Bus>, <Marke, Mercedes Benz>		
Nr.	Kontextbedingung	Erfüllt
Option 1	WENN Marke ∈ {Fuso, Smart}	nein
Option 2	WENN Fahrzeugtyp = Bus	ja
Option 3	WENN Phase = Produktion	undefiniert
Option 4	WENN Fahrzeugtyp <> PKW ODER Phase = Produktion	ja
Option 5	WENN Fahrzeugtyp = Bus UND Phase = Produktion	undefiniert
Ergebnis: Phase undefiniert! Option 3 konnte nicht bewertet werden Phase undefiniert! Option 5 konnte nicht bewertet werden		

Können die Kontextbedingungen aller Optionen entweder als gültig oder ungültig bewertet werden, besteht eine weitere Herausforderung darin, zu überprüfen, ob alle strukturellen und semantischen Abhängigkeiten zwischen den Optionen eingehalten werden. Das heißt wir müssen feststellen, ob die Anwendung der Optionen *kompatibel* ist. Darüber hinaus müssen die anzuwendenden, kompatiblen Optionen sortiert werden, um anschließend sequentiell auf den Basisprozess angewendet werden zu können. In den folgenden Abschnitten beschreiben wir, wie wir diese Herausforderungen in Provop bewältigen.

6.3 Explizite Beziehungen zwischen Optionen

Bei der Konfiguration von Prozessvarianten muss die gemeinsame Anwendung von Änderungsoperationen, die semantisch oder strukturell voneinander abhängig sind, gewährleistet

[5] Wir werden später noch zeigen, dass zur Visualisierung und besseren Vergleichbarkeit von mehreren Variantenmodellen, sowie zur dynamischen Konfiguration von Prozessvarianten, Ansatz 3 durchaus relevant und sinnvoll ist.

werden. Bisher realisieren wir dies durch Modellierung abhängiger Änderungsoperationen innerhalb (atomarer) Optionen. Die Gruppierung gemeinsam anzuwendenden Änderungsoperationen kann jedoch zu grob granularen Optionen führen (vgl. Abschnitt 5.5), welche nur eingeschränkt zur Konfiguration verschiedener Prozessvarianten verwendbar sind. Um eine feinere Granularität der Optionen und somit eine möglichst hohe Wiederverwendbarkeit der Änderungsoperationen zu schaffen, sind zusätzliche Konzepte erforderlich, welche die Abbildung struktureller und semantischer Abhängigkeiten auf Ebene von Optionen erlauben. Zum Beispiel muss der Optionsmodellierer beschreiben können, dass eine Option immer auch die Anwendung einer zweiten Option auf den Basisprozess (semantisch oder strukturell) impliziert. Das Erzwingen einer gemeinsamen Anwendung von Optionen ist jedoch nur eine mögliche Beschränkung, die zwischen Optionen abgebildet werden muss. So können auch andere Beziehungstypen identifiziert werden, die für die Konfiguration von Prozessvarianten relevant sind (vgl. Beispiel 6.5).

Beispiel 6.5 (Strukturelle und semantische Abhängigkeiten zwischen Optionen)
Abbildung 6.5 zeigt einen Basisprozess mit Aufsetzpunkten, für den mehrere Optionen modelliert sind. Zwischen diesen existieren sowohl strukturelle als auch semantische Beziehungen: Option 1 führt eine `INSERT`*-Operation durch. Dabei wird Aktivität „Änderung durchführen" eingefügt. Aktivität „Änderung dokumentieren" wird durch Option 2 in den Basisprozess eingefügt. Das Dokumentieren einer Änderung ist allerdings nur dann erforderlich, wenn diese tatsächlich durchgeführt worden ist, also wenn zuvor die Aktivität „Änderung durchführen" in den Basisprozess eingefügt worden ist. Daraus folgt, dass Option 2 nur dann angewendet werden darf, wenn auch Option 1 auf den Basisprozess angewendet wird. Option 2* impliziert *somit Option 1. Die Gruppierung der Änderungsoperationen von Option 1 und 2 zu einer einzigen Option ist hier nicht unbedingt sinnvoll, da unter bestimmten Rahmenbedingungen die alleinige Anwendung von Option 1 relevant sein kann (z.B. wenn keine Zeit für die Dokumentation der Änderung eingeplant ist).*
Ein weiteres Beispiel ist durch Optionen 3 und 4 gegeben. Diese beziehen sich auf das gleiche Attribut der Aktivität „Änderung dokumentieren" (`PE-ID: 5`*). Es ist daher wünschenswert, die gemeinsame Anwendung der Optionen* wechselseitig auszuschließen *oder die Anzahl der anzuwendenden Optionen auf maximal eine zu beschränken.*

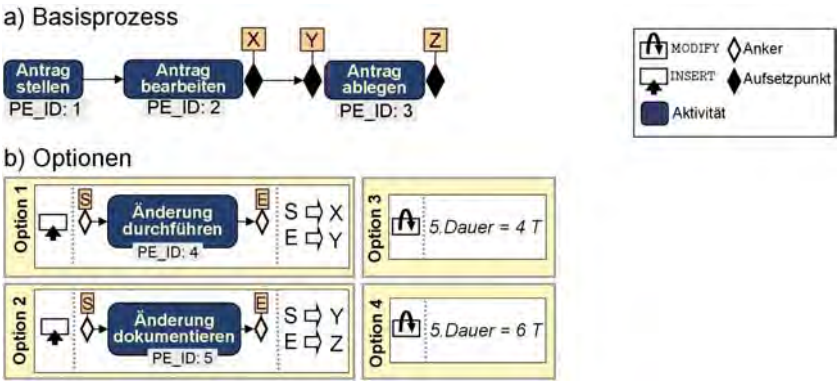


Abbildung 6.5: Strukturell und semantisch voneinander abhängige Optionen

Die explizite Beschreibung semantischer und struktureller Abhängigkeiten stellt in Provop ein wichtiges Hilfsmittel für die korrekte Konfiguration der Prozessvarianten dar. Dementsprechend unterstützen wir die Verwendung der Beziehungstypen *Implikation*, *Wechselseitiger Ausschluss* und *Auswahl: n-aus-m* (vgl. Anforderung 6.4). Eine Erweiterung um andere Beziehungstypen ist prinzipiell möglich, wird von uns aber an dieser Stelle nicht weiter verfolgt.

In der wissenschaftlichen Literatur finden sich bereits zahlreiche Arbeiten zur Verwendung und Beschreibung von strukturellen und semantischen Abhängigkeiten (engl. *Constraints*) [DKRR98, Pes08]. Dazu werden Sprachen bzw. Logiken, wie z.B. die Object Constraint Language (OCL), die Linear Temporal Logic (LTL) oder die Computation Tree Logic (CTL), verwendet. Für diese Sprachen existieren dedizierte Verifikationsverfahren (z.B. „Model Checking" [CGP00]), mit deren Hilfe die Einhaltung von Constraints überprüft werden kann. Optionsbeziehungen sind in Provop zwar wichtig, allerdings nicht Fokus unserer Arbeit. Dementsprechend motivieren und beschreiben wir die grundsätzliche Verwendung und Notwendigkeit von Optionsbeziehungen, verzichten aber auf eine formale Darstellung.

6.3.1 Beziehungstypen

Implikation. Implikationsbeziehungen zwischen Optionen werden verwendet, wenn infolge semantischer oder struktureller Abhängigkeiten eine bestimmte Option stets die Anwendung einer anderen Option erfordert. Durch explizite Beschreibung dieser Implikation kann die Anwendung mehrerer Optionen gekoppelt werden. Die Implikationsbeziehung ist unidirektional. Zur Abbildung einer wechselseitigen Implikation sind entsprechend zwei Implikationsbeziehungen (zyklisch) zu definieren. Im Folgenden wird die explizite Beschreibung der wechselseitigen Implikationsbeziehung zusätzlich angeboten, um den Aufwand bei der Modellierung von Abhängigkeiten zu reduzieren. Tabelle 6.6 zeigt die Darstellung einer (wechselseitigen) Implikationsbeziehung.

Wechselseitiger Ausschluss Sich wechselseitig ausschließende Optionen dürfen bei der Konfiguration einer Prozessvariante nicht gemeinsam ausgewählt bzw. niemals gemeinsam auf den Basisprozess angewendet werden. Der wechselseitige Ausschluss ermöglicht es somit, echt alternative Optionen zu definieren. Tabelle 6.6 zeigt die Darstellung des wechselseitigen Ausschlusses zweier Optionen X und Y.

Auswahl n-aus-m Bei diesem Beziehungstyp gibt ein Vergleichsoperator der Menge $\{<, >, =, \neq, \geq, \leq\}$ an, wie die Menge der ausgewählten Optionen hinsichtlich der Menge aller auswählbaren Optionen beschaffen sein muss. Durch Kombination dieser Auswahlbeziehungen können z.B. Intervalle definiert werden. Einen typischen Anwendungsfall der Auswahl n-aus-m Beziehung erläutert Beispiel 6.6. Die Darstellung der n-aus-m Beziehung zeigt Tabelle 6.6. Der angezeigte Vergleichsoperator (im Beispiel $\geq$) kann flexibel ausgetauscht werden.

Beispiel 6.6 (Anwendungsfall der Auswahl n-aus-m Beziehung) *Ein Anwendungsfall für eine Auswahl von genau einer Option aus einer Menge von m Optionen ist z.B. das Einfügen eines Prüfschrittes. Dieser kann zu verschiedenen Zeitpunkten während der Ausführung des Prozesses durchgeführt werden. Durch verschiedene* `INSERT`*-Operationen, die jeweils in einer anderen Option modelliert werden, kann er an verschiedenen Positionen eingefügt werden. Die Auswahlbeziehung stellt dann sicher, dass der Prüfschritt genau einmal eingefügt wird, d.h. genau eine Option aus der Menge der m Optionen die jeweils das Einfügen vornehmen.*

6.3.2 Kompatibilität von Optionen einer Optionsmenge

Wird bei der Konfiguration einer Variante eine Optionsmenge durch manuelle oder kontextbasierte Auswahl erstellt, kann mit Hilfe der explizit beschriebenen Optionsbeziehungen die Korrektheit dieser Optionsmenge überprüft werden. Mit Hilfe der Implikationsbeziehungen kann festgestellt werden, ob die getroffene Auswahl vollständig ist, also ob alle Optionen, von denen Optionen in der Optionsmenge abhängig sind, ebenfalls in dieser

Tabelle 6.6: Übersicht der Beziehungstypen

Beziehungstyp	Darstellung	Ausdruck
Implikation		$\neg(Option_x \wedge \neg Option_y)$
Wechselseitige Implikation		$\neg(Option_x \; xor \; Option_y)$
Wechselseitiger Ausschluss		$\neg(Option_x \wedge Option_y)$
Auswahl n-aus-m*		$\min_n(Option_1,\ldots,Option_m)$

* $n \leq m, n \in \mathbb{N}_0, m \in \mathbb{N}$
** bel. Relationssymbol

Menge enthalten sind. Die Kompatibilitätsprüfung wird in Provop, mit Hilfe der Funktion `checkOptionConstraints`, durchgeführt (vgl. Anhang A). Diese prüft, ob eine Optionsmenge alle Optionsbeziehungen erfüllt. Ist dies nicht der Fall, wird dieser Fehler für spätere Auswertungen (z.B. zwecks Optimierungen) dokumentiert und der Wert `false` zurückgegeben. Andernfalls ist der Rückgabewert der Funktion `true`.

6.4 Reihenfolge der Anwendung ausgewählter Optionen

Nachdem wir eine Menge von Optionen zwecks Konfiguration einer Prozessvariante manuell oder kontextbasiert ausgewählt sowie die Kompatibilität dieser Menge hinsichtlich expliziter Optionsbeziehungen geprüft haben, müssen wir die Optionen sequentiell auf den Basisprozess anwenden. Zu diesem Zweck ist eine entsprechende Anwendungsreihenfolge festzulegen (vgl. Anforderung 6.5).[6]

Ein naiver Ansatz zur Festlegung dieser Anwendungsreihenfolge der gewählten Optionen, ist die Anordnung entsprechend des Zeitpunktes der Optionserstellung. Ausgewählte Optionen können dann entsprechend dieses „Zeitstempels“ (beginnend mit der ältesten Option) sortiert werden. Die Anwendungsreihenfolge der Optionen entspricht somit ihrer Modellierungsreihenfolge. Diese einfache Sortierung reicht jedoch oftmals nicht aus, da Optionen i.A. nicht kommutativ sind. Das heißt abhängig von der Reihenfolge ihrer Anwendung können unterschiedliche Ergebnismodelle resultieren oder sogar Fehler auftreten (vgl. Beispiel 6.7).

Beispiel 6.7 (Nicht-kommutative Optionen) *Ein Beispiel für nicht kommutative Optionen zeigt Abbildung 6.6: Gegeben ist ein Basisprozess, für den zwei Optionen modelliert wurden. Option 1 fügt die Aktivität `M` zwischen den Aufsetzpunkten `Y` und `Z` ein. Option 2 löscht alle Prozesselemente zwischen den Aufsetzpunkten `X` und `Z`. Dabei verbleibt der Aufsetzpunkt `Y`, der als „fix positioniert“ parametrisiert ist, im Prozessmodell. Die Zeitstempel der Optionen zeigen, dass Option 1 vor Option 2 modelliert wurde. Werden die Optionen in dieser Reihenfolge auf den Basisprozess angewendet (vgl. Abbildung 6.6c), wird das Einfügen von `M` aufgehoben. Dies ist im gegebenen Beispiel nicht wünschenswert. Stattdessen soll das Ergebnismodell aus Abbildung 6.6d abgeleitet werden, indem Operation 2 vor Operation 1 angewendet wird.*

Die Sortierung und Anwendung einer Menge von Optionen entsprechend ihres Zeitstempels ist nur dann ausreichend, wenn der Modellierer alle nicht-kommutativen Optionen in der exakt richtigen Reihenfolge modelliert. Dies ist nicht immer praktikabel, da die Menge modellierter Optionen nachträglich erweiterbar sein sollte, jedoch ohne dazu den intern gesetzten

[6] Wie Abschnitt 5.5 erläutert, werden Änderungsoperationen in einer Option bereits als Sequenz gespeichert. An dieser Stelle ist daher nur die Sequentialisierung von Optionen erforderlich.

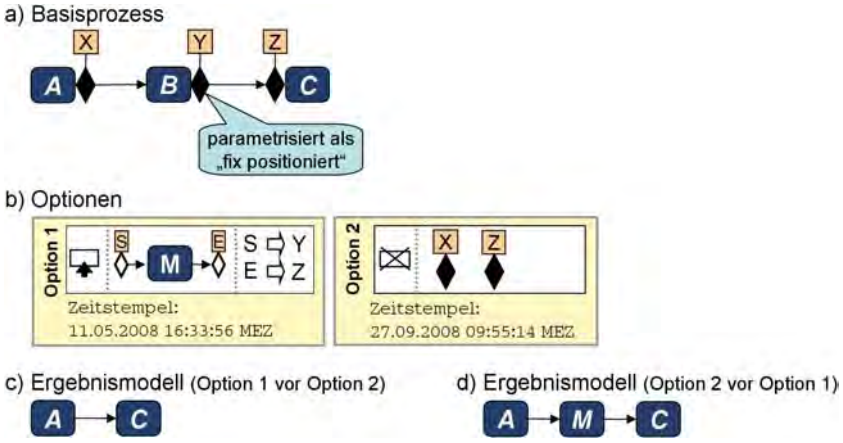


Abbildung 6.6: Nicht-kommutative Änderungsoperationen

Zeitstempel zu manipulieren. Im Folgenden beschreiben wir, wie die Reihenfolge von Optionen in Provop explizit angeben werden kann (vgl. Abschnitt 6.4.1) und wie wir solche Angaben zur automatischen Sortierung einer Optionsmenge verwenden (vgl. Abschnitt 6.4.2).

6.4.1 Explizite Reihenfolgebeziehungen zwischen Optionen

Um das Problem nicht-kommutativer Optionen zu beheben muss die Anwendungsreihenfolge der Optionen durch den Variantenmodellierer definiert werden. Provop bietet die Möglichkeit an, diese Reihenfolge der Optionen mittels expliziter *Reihenfolgebeziehungen* zwischen Optionen, zu beschreiben. Die Darstellung von Reihenfolgebeziehungen zeigt Tabelle 6.7.

Tabelle 6.7: Übersicht der expliziten Reihenfolgebeziehungen

Beziehungstyp	Darstellung	Ausdruck
Reihenfolge	- - - - - →	$Option_x \Rightarrow Option_y$
Hierarchie	—(H)→	$\neg(Option_x \wedge \neg Option_y) \wedge (Option_x \Rightarrow Option_y)$

Infolge struktureller Abhängigkeiten sind bei Implikationsbeziehungen oftmals auch Reihenfolgebeziehungen gegeben. Um diese in einer Optionsbeziehung abzubilden und somit den Modellierungsaufwand zu reduzieren, werden von uns auch *Hierarchiebeziehungen* unterstützt.[7]

Neben der Aufwandsreduktion ist ein weiterer Vorteil dieses Beziehungstyps, dass wir die Menge der modellierten Optionen logisch strukturieren können. So sind auf diese Weise auch Strukturen wie „Varianten von Varianten“ abbildbar (vgl Abbildung 6.7). Dabei nehmen Vateroptionen allgemeinere Änderungen am Basisprozess vor, während die verschiedenen Kindoptionen spezialisiertere Änderungen realisieren.

Diese Vorgehensweise erlaubt das Bilden von *Klassen von Prozessvarianten* für einen Prozesstyp. Hierarchiebeziehungen können somit auch die Problematik der Modellierung mit mehreren Basisprozessen für einen Prozesstyp lösen (vgl. Abschnitt 5.2).

[7] Die Hierarchiebeziehung wird auch bei der Kompatibilitätsprüfung, beschrieben in Abschnitt 6.3.2, betrachtet. Sie wird dazu auf eine Implikationsbeziehung abgebildet.

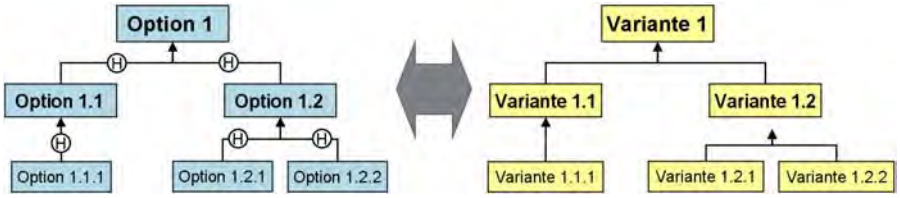


Abbildung 6.7: Hierarchie von Optionen und Varianten

6.4.2 Generierung einer sortierten Optionsmenge

Um die manuell oder kontextbasiert ausgewählten Optionen zur Ableitung einer spezifischen Prozessvariante sequentiell auf den Basisprozess anwenden zu können, müssen wir die Optionsmenge zu einer Folge von Optionen sortieren. Dazu werten wir die expliziten Reihenfolgebeziehungen der Optionen aus. Dadurch erhalten wir jeweils Paare von Optionen, die nacheinander auf den Basisprozess angewendet werden müssen. Wir fordern nicht, dass die Nachfolger unmittelbar nach ihren Vorgängern angewendet werden. Stattdessen können zwischen diesen Optionen noch weitere Optionen liegen. Um nicht für alle Optionen der gegebenen Optionsmenge entsprechende Reihenfolgebeziehungen definieren zu müssen, können wir den Zeitstempel der Optionen als zusätzliche Sortierinformation nutzen. Auf diese Weise ist auch bei einer unvollständigen Angabe von Reihenfolgebeziehungen eine eindeutige Sortierung der Optionen zu einer *Optionsfolge* (d.h. einer Sequenz von Optionen) möglich.

Beispiel 6.8 (Sortierung einer Optionsmenge zu einer Optionsfolge) *Abbildung 6.8a zeigt fünf Optionen, zwischen denen explizite Reihenfolge- bzw. Hierarchiebeziehungen definiert sind. Der Bezeichner der Option gibt die Modellierungsreihenfolge bzw. den Zeitstempel der Optionen wider. Abbildung 6.8b zeigt die einzelnen Vorgänger-Nachfolger-Paare, die sich durch die Beziehungen ergeben. Da diese Information nicht ausreicht, um eine eindeutige Anwendungsreihenfolge zu identifizieren, verwenden wir zusätzlich die Zeitstempel-Sortierung aus Abbildung 6.8c. Wir können dann eine eindeutige Reihenfolge für die Optionen festlegen (vgl. Abbildung 6.8d).*

Die oben beschriebe Sortierung ist allerdings nicht immer möglich. Werden zyklische Reihenfolgebeziehungen zwischen den Optionen einer Optionsmenge definiert (z.B. Option 1 < Option 2 < Option 1), muss die Sortierung abgebrochen werden bzw. der Variantenverantwortliche muss korrigierend eingreifen.

Zur Sortierung einer Optionsmenge verwenden wir in Provop die Funktion `sortOptionSet` (vgl. Anhang A). Ist eine Sortierung erfolgreich, gibt sie den Wert `true` zurück. Andernfalls ist der Rückgabewert `false`.

6.5 Korrektheit einer Prozessfamilie

Bei der Konfiguration einer Prozessvariante sollte in jedem Fall ein Prozessmodell erzeugt werden, das den Geschäftsprozess vollständig und korrekt beschreibt. Nur dann macht die Dokumentation Sinn bzw. kann das Prozessmodell später in ein ausführbares Workflow-Modell überführt werden. In diesem Zusammenhang geforderte Korrektheitskriterien betreffen z.B. die Terminierung des Workflows (z.B. Verklemmungsfreiheit) und die korrekte Versorgung der Prozessaktivitäten mit Daten [RD98, KR96, vdA97, vdA00].

Die Herausforderung besteht darin, die Konsistenz einer ganzen Prozessfamilie (d.h. einer Menge von aus einem Basisprozess ableitbaren Prozessmodellen) sicherzustellen und zwar

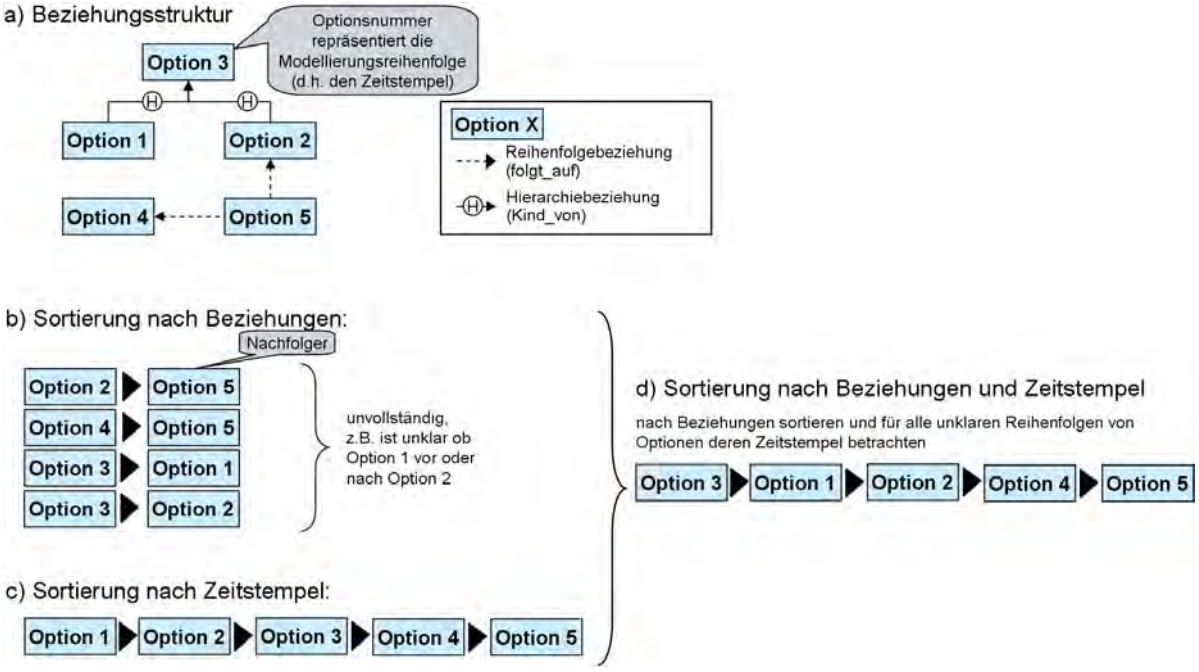


Abbildung 6.8: Sortierung von Optionen nach Beziehungen und Zeitstempel

unter Berücksichtigung der explizit definierten Optionsbeziehungen (vgl. Anforderung 6.6). Das Ziel von Provop ist daher nicht eine Lösung zur Überprüfung der Korrektheit eines einzelnen Prozessmodells zu bieten – hierzu gibt es für existierende Formalismen bereits hinreichend Literatur [Wes07, vdA97, SZ06]. Stattdessen wollen wir ein Prozess-Metamodell unabhängiges Rahmenwerk für die semantisch und strukturell korrekte Konfiguration einer ganzen Prozessfamilie aufbauen. Dieses Rahmenwerk kann auf verschiedene Metamodelle übertragen werden.

6.5.1 Grundlegende Ansätze zur Sicherstellung von Korrektheit

Prinzipiell sind die nachfolgenden Ansätze zur Sicherstellung der Korrektheit einer Prozessfamilie möglich:

Ansatz 1 (Korrekter Basisprozess):
Die Korrektheit konfigurierter Prozessvarianten kann sichergestellt werden, indem die Konfiguration, ausgehend von einem korrekten Prozessmodell, gestartet wird und jeweils nach Anwendung einer Änderungsoperation wiederum ein korrektes Prozessmodell erzwungen wird. Daraus ergibt sich, dass die Anwendung einer Menge von Änderungsoperationen bzw. Optionen wieder in einem korrekten Prozessmodell resultiert.

Der Vorteil von Ansatz 1 ist, dass wir nicht jedes Ergebnismodell einzeln auf Korrektheit prüfen müssen, sondern nur den Basisprozess. Nachteilig ist, dass wir nur solche Änderungsoperationen zulassen, welche die Korrektheit des Prozessmodells bei der Konfiguration erhalten. Dadurch verlagern wir die Korrektheitsprüfungen in die Vor- bzw. Nachbedingungen von Änderungsoperationen. Wir fordern in Provop, im Gegensatz zu existierenden Konfigurationstechniken (vgl. [vdADG+08]), nicht notwendigerweise ein korrektes Prozessmodell als Ausgangspunkt für die Konfiguration einer bestimmten Prozessvariante. Stattdessen soll dem Modellierer größtmögliche Flexibilität bei der Modellierung eines Basisprozesses gegeben werden. Dazu unterstützen wir verschiedene Modellierungsstrategien für Basisprozesse

(vgl. Abschnitt 5.2). Zum Beispiel kann ein Basisprozess als Obermenge aller Varianten definiert werden oder er stellt ein minimales Modell dar, d.h. eine Schnittmenge aller Varianten.

Beispiel 6.9 (Forderung eines korrekten Basisprozesses) *In Abbildung 6.9a sind die Auto- und Bus-spezifischen Prozessvarianten eines bestimmten Prozesstyps dargestellt. Definieren wir den Basisprozess als Schnittmenge der beiden Variantenmodelle, erhalten wir das im unteren Teil von Abbildung 6.9a dargestellte Prozessmodell. Dieser Basisprozess enthält die gemeinsamen Aktivitäten der Prozessvarianten (*`GA1`, `GA2` *und* `GA3`*), nicht aber die für ein Auto bzw. einen Bus spezifischen Aktivitäten (*`Auto` *und* `Bus`*). Dieser Basisprozess ist nicht korrekt, da das Datenobjekt, welches von Aktivität* `GA3` *gelesen wird, nicht von den gemeinsamen Aktivitäten geschrieben wird (vgl. Korrektheitskriterium 8). Da bei der Modellierung des Basisprozesses davon ausgegangen werden kann, dass die entsprechende Schreiboperation durch Variantenkonfiguration eingefügt wird und somit ein korrektes Prozessmodell resultiert, ist dies ein akzeptabler Basisprozess in Provop.*

Das „Erzwingen" eines korrekten Basisprozesses ist im vorliegenden Szenario nicht optimal. Würde z.B. das Prozessmodell von Variante 1 als Basisprozess gewählt, käme es zur Verletzung von Sichtbarkeitsregeln: Variante 1 wäre dann für den Eigner von Variante 2 sichtbar (vgl. Abbildung 6.9b). Darüber hinaus muss der Eigner von Variante 2 Änderungen am Prozessmodell von Variante 1 mitverfolgen, um Variante 2 bzw. die auf den Basisprozess (d.h. Variante 1) bezogenen Optionen aktuell zu halten. Ein ebenfalls ungeeigneter Ansatz zur Herstellung der Korrektheit des Basisprozesses wäre es, eine „virtuelle" Platzhalteraktivität einzufügen, die das betreffende Datenobjekt schreibt. Dies würde jedoch den Modellierungsaufwand unnötigerweise erhöhen, da zusätzliche Anpassungen zur Ableitung der Varianten erforderlich werden.

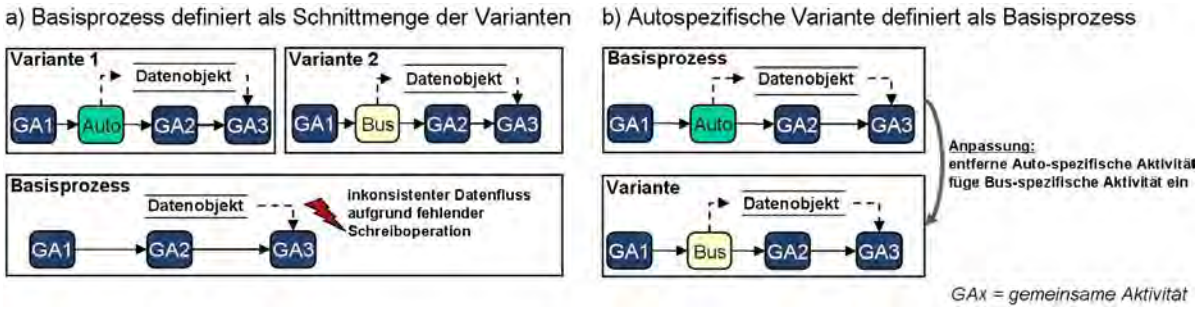


Abbildung 6.9: Inkorrekter Basisprozess in (a) verändert zu korrektem Modell in (b)

Ansatz 2 (Abschließende Korrektheitsprüfung):

Eine alternative Möglichkeit zur Sicherstellung der Korrektheit konfigurierter Prozessvarianten besteht darin, zunächst einmal die gewünschten Optionen auf den Basisprozess anzuwenden und erst danach die Korrektheit des Ergebnismodells zu überprüfen. Um die Korrektheit der gesamten Prozessfamilie zu garantieren, müssen wir alle theoretisch möglichen Kombinationen von Optionen auf den Basisprozess anwenden und anschließend die Ergebnismodelle auf ihre Korrektheit überprüfen.

Der Vorteil dieses Ansatzes ist, dass der Modellierer eine größere Flexibilität bei der Festlegung des Basisprozesses hat. Allerdings ist dieser Ansatz bereits bei einer geringen Anzahl von Optionen aufwändig: Unter der Annahme, dass Optionen nicht kommutativ sind, müssten bei drei Optionen bereits 16 verschiedene Optionsfolgen auf den Basisprozess angewendet

und anschließend die entsprechenden Ergebnismodelle auf ihre Korrektheit geprüft werden.[8] Offensichtlich ist dieser Ansatz für ein Szenario mit dutzenden von Optionen nicht geeignet.

Zwischenfazit: Da Ansatz 1 durch die gegebenen Einschränkungen bei der Festlegung des Basisprozesses in Provop nicht praktikabel ist, verfolgen wir Ansatz 2. Die Herausforderung dabei ist, die Anzahl der zu überprüfenden Prozessmodelle bestmöglich zu reduzieren und nur solche Ergebnismodelle auf Konsistenz zu prüfen, die entsprechend des gegebenen Kontextmodells sowie unter Berücksichtigung der modellierten Optionsbeziehungen auch relevant sind.

6.5.2 Rahmenwerk zur Korrektheitsprüfung

Die Sicherstellung der Korrektheit konfigurierbarer Prozessvarianten erfolgt in nachfolgendem Schema (vgl. Abbildung 6.10):

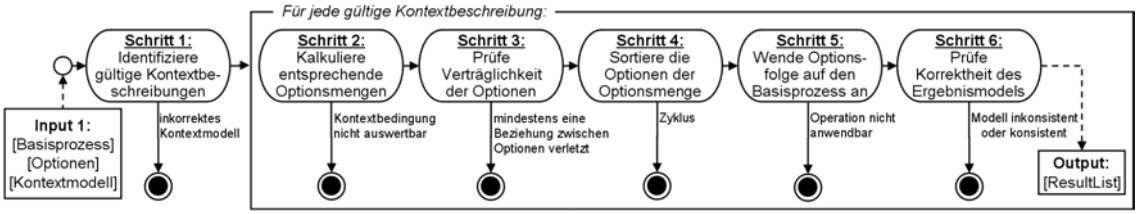


Abbildung 6.10: Übersicht über das Provop Rahmenwerk zur Korrektheitsprüfung (UML-Aktivitätendiagramm)

- **Schritt 1:** Um alle relevanten Prozessvarianten zu prüfen, identifizieren wir zunächst für welche Kontextbeschreibungen jeweils eine Variante konfigurierbar sein soll. Wir betrachten daher die Menge gültiger Kontextbeschreibungen des spezifischen Kontextmodells der Prozessfamilie. Ist das Kontextmodell nicht korrekt spezifiziert, brechen wir die Prüfung ab.
- **Schritt 2:** Für jede gültige Kontextbeschreibung wird die zugehörige Menge, der auf den Basisprozess anzuwendenden Optionen ermittelt. Wir brechen die Prüfung in Schritt 2 ab, falls die Kontextbedingung einer Option, z.B. aufgrund fehlender Kontextinformationen, nicht ausgewertet werden kann (vgl. Abschnitt 6.2.2.2).
- **Schritt 3:** In diesem Schritt prüfen wir, ob die ermittelten Optionsmengen hinsichtlich der definierten Optionsbeziehungen kompatibel sind, d.h. ob alle Beziehungen eingehalten werden (vgl. Abschnitt 6.3). Ist eine Optionsmenge nicht gültig, d.h. mindestens eine Optionsbeziehung ist verletzt, wird ein Fehler gemeldet und abgebrochen. Andernfalls fahren wir mit Schritt 4 fort.
- **Schritt 4:** Die Optionen werden, entsprechend der beschriebenen Reihenfolgebeziehungen, zu einer Optionsfolge sortiert. Sind die Optionen nicht sortierbar, d.h. wurden zyklische Reihenfolgebeziehungen beschrieben, brechen wir an dieser Stelle mit einer entsprechenden Fehlermeldung ab.
- **Schritt 5:** Nach erfolgreicher Sortierung aller Optionsmengen zu „Optionsfolgen“, wenden wir diese jeweils auf den Basisprozess an und leiten somit die einzelnen Variantenmodelle der Prozessfamilie ab. Ist eine Option bzw. Änderungsoperation nicht anwendbar (d.h. ist eine Vorbedingung verletzt), brechen wir die Prüfung ab.

[8] Das Ergebnis ergibt sich bei 3 Optionen durch alle Permutationen von je 3, 2 und 1 Option(en) plus den Basisprozess (0 Optionen); d.h. $1*3!+3*2!+3*1!+1=16$

- **Schritt 6:** Abschließend erfolgt in Schritt 6 die Korrektheitsprüfung der Ergebnismodelle auf Basis der spezifischen Korrektheitskriterien des zugrunde liegenden Prozess-Metamodells.

Im Folgenden werden diese Schritte ausführlich beschrieben. Dabei erörtern wir die Vorgehensweise der Provop-Korrrektheitsprüfung anhand des in Beispiel 6.10 beschriebenen Szenarios.

Beispiel 6.10 (Szenario für die Korrektheitsprüfung) *Abbildung 6.11a zeigt einen abstrakten Basisprozess. Dieser ist nach den Vorgaben einiger Prozess-Metamodelle nicht korrekt, da ein doppeltes Schreiben des Datenobjektes durch die Aktivitäten* `B1` *und* `B2` *vorgenommen wird. In Provop ist dieser Prozess dennoch ein möglicher Basisprozess für die Beschreibung von Änderungsoperationen. Des Weiteren ist in Abbildung 6.11b das zugehörige Kontextmodell als dreidimensionaler Kontextwürfel gegeben. Die Teil-Würfel 3, 6, 12 und 15 stellen ungültige Kontextbeschreibungen dar. In Abbildung 6.11c sind die Optionen des Basisprozesses spezifiziert. Dabei repräsentiert die Optionsnummer die Modellierungsreihenfolge (d.h. den Zeitstempel) der Optionen. Für jede Option ist eine Kontextbedingung angegeben, basierend auf dem in Abbildung 6.11b dargestellten Kontext. Abbildung 6.11d zeigt die Beziehungen der definierten Optionen.*

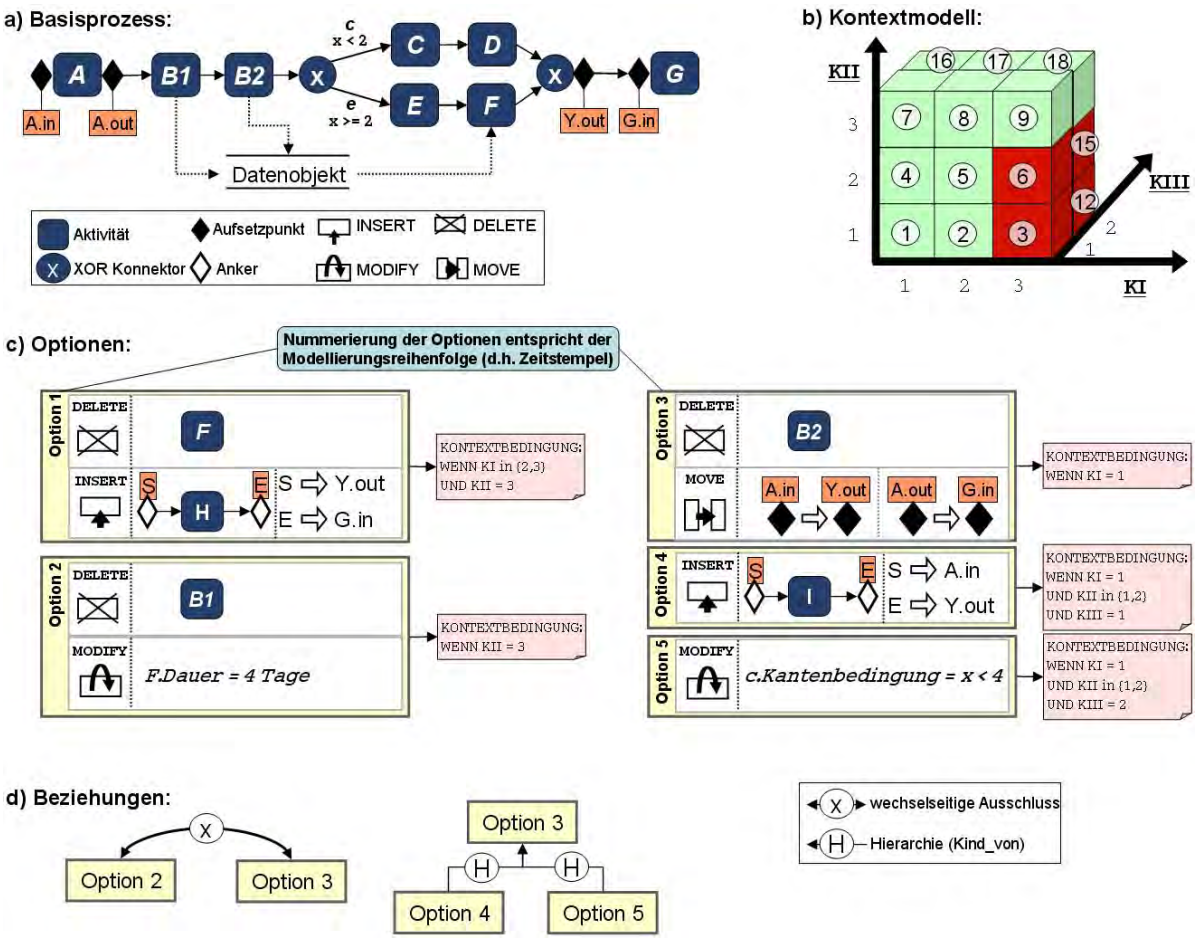


Abbildung 6.11: Beispiel-Szenario mit Basisprozess a), Kontextmodell b), Optionen c) und expliziten Beziehungen zwischen den Optionen d)

6.5.2.1 Schritt 1: Identifiziere gültige Kontextbeschreibungen

Im ersten Schritt der Korrektheitsprüfung werden alle gültigen Kontextbeschreibungen für das gegebene Kontextmodell ermittelt, für die dann eine (korrekte) Prozessvariante konfiguriert werden muss. Hierbei können wir erkennen, ob das Kontextmodell fehlerfrei beschrieben ist. Mögliche Fehler sind z.B. unvollständige Angaben (d.h. es ist kein Name oder Wertebereich für eine Kontextvariable spezifiziert) oder nicht eindeutige Benennungen von Kontextvariablen (d.h. es gibt mehrere Kontextvariablen mit gleichem Namen).

Ist das Kontextmodell fehlerfrei beschrieben, fassen wir alle gültigen Kontextbeschreibungen in der Menge `ValidCtxtDescr` zusammen. Provop prüft dazu alle möglichen Wertekombinationen der Kontextvariablen, d.h. jeden Teil-Würfel eines Kontextwürfels (vgl. Abbildung 6.11b), hinsichtlich der beschriebenen Kontextregeln. Dazu wird die Funktion `contextDescriptionValid` aufgerufen (vgl. Anhang A). Für eine gegebene Kontextbeschreibung gibt die Funktion `true` zurück, wenn die Kontextbeschreibung keine Kontextregel des zugrunde liegenden Kontextmodells verletzt und somit „gültig“ ist. In diesem Fall wird die Kontextbeschreibung in die Menge `ValidCtxtDescr` aufgenommen. Ist der Rückgabewert der Funktion `false`, wird die Kontextbeschreibung nicht weiter berücksichtigt.

// Schritt 1: Identifiziere gültige Kontextbeschreibungen
ValidCtxtDescr = ∅ *// Initialisierung der Menge gültiger Kontextbeschreibungen*
// Erzeuge Kontextbeschreibungen durch Simulation aller möglichen Werte der Wertebereiche jeder im Kontextmodell definierten Kontextvariablen $\texttt{CtxtVar}_i$*, i=1,..,n. Es wird angenommen, dass die Wertebereiche der Variablen* $\texttt{ValueRange(CtxtVar}_i\texttt{)}$ *diskret und endlich sind.*
for all CtxtDescription ∈ (ValueRange($CtxtVar_1$) × ... × ValueRange($CtxtVar_n$)) **do**
  // prüfe die Gültigkeit der aktuell betrachteten Kontextbeschreibung
  if contextDescriptionValid(CtxtDescription) = true **then**
    ValidCtxtDescr := ValidCtxtDescr ∪ {CtxtDescription}

Beispiel 6.11 (Schritt 1 anhand von Beispiel 6.10) *Für das Kontextmodell unseres Szenarios aus Abbildung 6.11b ist die Menge gültiger Kontextbeschreibungen durch*

$$\texttt{ValidCtxtDescr} = \{1, 2, 4, 5, 7, 8, 9, 10, 11, 13, 14, 16, 17, 18\}$$

gegeben. Die Nummern beziehen sich auf die entsprechenden Teil-Würfel im Kontextmodell aus Abbildung 6.11b.

6.5.2.2 Schritt 2: Kalkuliere Optionsmengen

Für jede gültige Kontextbeschreibung aus der in Schritt 1 bestimmten Menge `ValidCtxtDescr`, berechnen wir in Schritt 2 die zugehörige Optionsmenge, d.h. die Menge der im gegebenen Kontext anzuwendenden Optionen. Diese Menge ergibt sich aus den Optionen, die in der aktuell betrachteten Kontextbeschreibung erforderlich sind, um die gewünschte Variante aus dem Basisprozess abzuleiten.[9] Zu diesem Zweck wird die Funktion `contextRuleValid` verwendet (vgl. Anhang A). Sie prüft, ob die Kontextbedingung einer Option im gegebenen Kontext zutrifft. Sie wird für jede modellierte Option aufgerufen. Gibt sie für eine Option den Wert `true` zurück, wird diese der Optionsmenge der aktuell betrachteten Kontextbeschreibung hinzugefügt. Falls die Funktion den Wert `false` liefert (d.h. die Kontextbedingung nicht erfüllt ist), ist die Option im aktuellen Kontext nicht relevant und wird daher nicht weiter betrachtet. Wir brechen die Prüfung in Schritt 2 ggf. ab, falls die Kontextbedingung einer Option nicht ausgewertet werden kann.

[9] Die Optionsmenge kann auch leer sein. In diesem Fall stellt der Basisprozess selbst die Variante dar.

Wie Abbildung 6.12 zeigt, können verschiedene Kontextbeschreibungen zu einer identischen Optionsmenge führen. Dies ist zum Beispiel für die Teil-Würfel 8 und 9 der Fall. Die berechnete Optionsmenge besteht in beiden Fällen jeweils aus den Optionen 1 und 2. Um diese Optionsmengen nicht mehrfach behandeln bzw. prüfen zu müssen, werden die zugehörigen Kontextbeschreibungen in einer sog. *Kontextgruppe* zusammengefasst. Jeder Gruppe von Kontextbeschreibungen wird dann genau eine Optionsmenge zugewiesen. Als Ergebnis von Schritt 2 erhalten wir somit eine Menge von <Kontextgruppe, Optionsmenge>-Paaren. Dieses Objekt wird als `ProcessVariantCandidates` bezeichnet.[10]

```
// Schritt 2: Kalkuliere entsprechende Optionsmengen
// verwende Menge der gültigen Kontextbeschreibungen ValidCtxtDescr, die in Schritt 1 erstellt wurde
// und Menge AllDefinedOptions aller modellierter Optionen
ProcessVariantCandidates = ∅
for each CtxtDescription ∈ ValidCtxtDescr do
    CalculatedOptions := ∅
    // prüfe Gültigkeit der Kontextbedingungen aller definierten Optionen
    for each Option ∈ AllDefinedOptions do
        if contextRuleValid(Option,CtxtDescription) = true then
            CalculatedOptions := CalculatedOptions ∪ {Option}
    // prüfe in ProcessVariantCandidates ob die Optionsmenge bereits existiert
    if hasOptSet(ProcessVariantCandidates,CalculatedOptions) = true then
        // füge neue Kontextgruppe mit gleicher Optionsmenge in ProcessVariantCandidates ein
        insertCtxtGroup(ProcessVariantCandidates,CtxtDescription,CalculatedOptions)
    else // erweitere existierende Kontextgruppe um die aktuell betrachtete Kontextbeschreibung
        extendCtxtGroup(ProcessVariantCandidates,CtxtDescription,CalculatedOptions)
```

Beispiel 6.12 (Schritt 2 anhand von Beispiel 6.10) *Bezogen auf das Beispiel aus Abbildung 6.11 zeigt Tabelle 6.8 die einzelnen Zwischenschritte bei der Erstellung des Objektes* `ProcessVariantCandidates`*. Dabei durchlaufen wir alle 18 Teil-Würfel (Spalte 1) und ermitteln die Optionsmenge (Spalte 2). Ist eine Optionsmenge bereits im Objekt der* `ProcessVariantCandidates` *enthalten, fügen wir die Nummer des Teil-Würfels in die zugehörige Gruppe der Kontextbeschreibungen dieser Optionsmenge ein. Falls die Optionsmenge noch nicht existiert, erstellen wir ein neues <Kontextgruppe, Optionsmenge>-Paar. Als Ergebnis von Schritt 2 erhalten wir:*

```
ProcessVariantCandidates = <{1, 4}, {Option 3, Option 4}>,
<{2, 5, 11, 14},{ }>, <{7, 16},{Option 2, Option 3}>,
<{8,9,17,18},{Option 1, Option 2}>, <{10, 13},{Option 3, Option 5}>.
```

Abbildung 6.12 visualisiert dieses Ergebniss mit optisch gruppierten Teil-Würfeln.

6.5.2.3 Schritt 3: Prüfe Kompatibilität der Optionen einer Optionsmenge

Schritt 3 der Korrektheitsprüfung untersucht für jede gültige Kontextgruppe bzw. jedes Element der Menge `ProcessVariantCandidates`, ob die Optionen der zugeordneten Optionsmenge kompatibel zueinander sind, d.h. ob sie den explizit definierten Optionsbeziehungen genügen. Für diese Kompatibilitätsprüfung wird die Funktion `checkOptionConstraints` verwendet. Sie gibt für eine gegebene Optionsmenge den Wert `true` zurück, wenn die Optionsmenge kompatibel ist, d.h. keine Beziehung zwischen Optionen verletzt wird. Andernfalls

[10] Aufgrund von Fehlerkorrekturen nach einer Korrektheitsprüfung, ergeben sich ggf. Änderungen an den Prozessvarianten einer Prozessfamilie. Wir sprechen daher im Zusammenhang mit der Korrektheitsprüfung von „Kandidaten“ für Prozessvarianten.

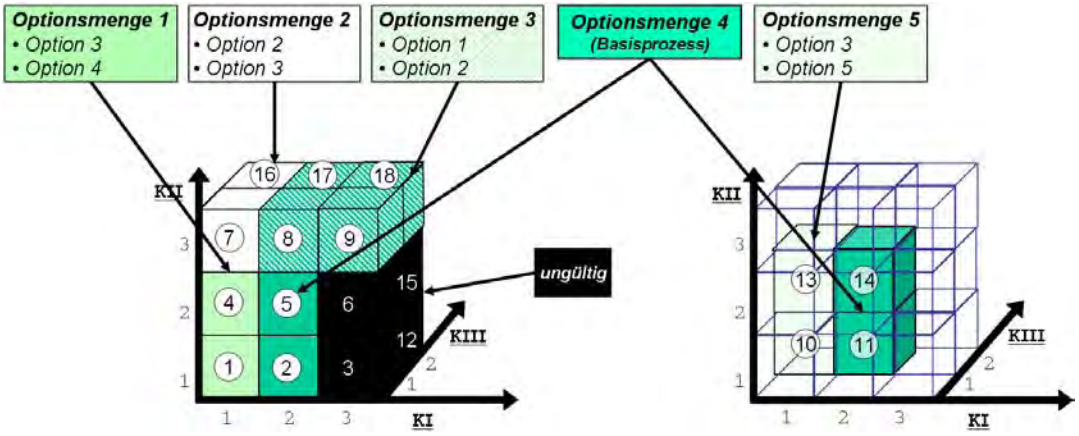


Abbildung 6.12: Gruppen gültiger Kontextbeschreibungen und entsprechende Optionsmengen

Tabelle 6.8: Sukzessive Erstellung des Objektes `ProcessVariantCandidates`

Nr.	Optionsmenge	ProcessVariantCandidates
1	{Option 3, Option 4}	**{<1,{Option 3, Option 4}>}**
2	{}	{<1,{Option 3, Option 4}>, **<{2},{}>**}
4	{Option 3, Option 4}	{<{1,**4**},{Option 3, Option 4}>, <{2},{}>}
5	{}	{<{1,4},{Option 3, Option 4}>, <{2,**5**},{}>}
7	{Option 1, Option 2, Option 3}	{<{1,4},{Option 3, Option 4}>, <{2,5},{}>, **<7,{Option 2, Option 3}>**}
8	{Option 1, Option 2}	{<{1,4},{Option 3, Option 4}>, <{2,5},{}>, <7,{Option 2, Option 3}>, **<8,{Option 1,Option 2}>}**
9	{Option 1, Option 2}	{<{1,4},{Option 3, Option 4}>, <{2,5},{}>, <7,{Option 2, Option 3}>, <{8,**9**}{Option 1, Option 2}>}
10	{Option 3, Option 5}	{<{1,4},{Option 3, Option 4}>, <{2,5},{}>, <7,{Option 2, Option 3}>, <{8,9}{Option 1, Option 2}>,**<10,{Option 3, Option 5}>**}
11	{}	{<{1,4},{Option 3, Option 4}>, <{2,5,**11**},{}>, <7,{Option 2, Option 3}>, <{8,9}{Option 1, Option 2}>,<10,{Option 3, Option 5}>}
13	{Option 3, Option 5}	{<{1,4},{Option 3, Option 4}>, <{2,5,11},{}>, <7,{Option 2, Option 3}>, <{8,9}{Option 1, Option 2}>, <{10,**13**},{Option 3, Option 5}>}
14	{}	{<{1,4},{Option 3, Option 4}>, <{2,5,11,**14**},{}>, <7,{Option 2, Option 3}>, <{8,9}{Option 1, Option 2}>, <{10,13},{Option 3, Option 5}>}
16	{Option 1, Option 2, Option 3}	{<{1,4},{Option 3, Option 4}>, <{2,5,11,14},{}>, <{7,**16**},{Option 2, Option 3}>, <{8,9}{Option 1, Option 2}>, <{10,13},{Option 3, Option 5}>}
17	{Option 1, Option 2}	{<{1,4},{Option 3, Option 4}>, <{2,5,11,14},{}>, <{7,16},{Option 2, Option 3}>, <{8,9,**17**}, {Option 1, Option 2}>, <{10,13},{Option 3, Option 5}>}
18	{Option 1, Option 2}	{<{1,4},{Option 3, Option 4}>, <{2,5,11,14},{}>, <{7,16},{Option 2, Option 3}>, <{8,9,17,**18**},{Option 1, Option 2}>, <{10,13},{Option 3, Option 5}>}

ist der Rückgabewert `false`, da kein Ergebnismodell abgeleitet werden kann. Das fehlerhafte <Kontextgruppe, Optionsmenge>-Paar wird dann dem Variantenmodellierer gemeldet und die Ausführung der Korrektheitsprüfung an dieser Stelle abgebrochen.

```
// Schritt 3: Prüfe Verträglichkeit der Optionen einer Optionsmenge
for each <CtxtBlock,OptionSet> ∈ ProcessVariantCandidates do
    if checkOptionConstraints(OptionSet) = false then
        // schreibe Eintrag in Fehlerdokument mit verletzten Constraints
```

```
writeErrorLog(OptionsSet,...)
break // Abbruch
```

Beispiel 6.13 (Schritt 3 anhand von Beispiel 6.10) *Abbildung 6.11d unseres Szenarios zeigt die explizit definierten Beziehungen der Optionen 1 bis 5. Zwischen den Optionen 2 und 3 haben wir einen wechselseitigen Ausschluss definiert, d.h. wir schließen die gemeinsame Anwendung dieser Optionen bei der Konfiguration einer Prozessvariante aus. Da die Kontextbedingung der Optionen 2 und 3 keine disjunkten Kontexte beschreiben, ergeben sich zwei Teil-Würfel (7 und 16), in denen beide Optionen gültig sind. Bei den Kompatibilitätsprüfungen der in Abbildung 6.12 gegebenen Optionsmengen wird festgestellt, dass Optionsmenge 2 (welche sich für die Teil-Würfel 7 und 16 ergibt) gegen diese Beziehung verstößt. Die Korrektheitsprüfung bricht an dieser Stelle mit einer entsprechenden Fehlermeldung ab. Bevor wir mit Schritt 1 erneut beginnen, muss der identifizierte Fehler behoben werden. Der Variantenmodellierer kann dazu entweder:*

- *die Kontextbedingungen der Optionen anpassen, so dass die dann resultierende Optionsmenge für die Teil-Würfel 7 und 16 hinsichtlich der Optionsbeziehungen kompatibel sind,*
- *die definierten Optionsbeziehungen ändern, so dass die ermittelte Optionsmenge kompatibel ist, oder*
- *das Kontextmodell anpassen, so dass die Kontextbeschreibung entweder gar nicht mehr existiert (d.h. der Wertebereich von Kontextvariablen wurde verändert) oder durch eine entsprechende Kontextregel als ungültig definiert wird.*

In diesem Szenario gehen wir davon aus, dass der Kontextmodellierer eine zusätzliche Kontextregel definiert, welche die Kontextwürfel 7 und 16 invalidiert. Diese Regel lautet wie folgt:

```
WENN KI = 1 DANN KII in {1,2}
```

Nach einer Neuberechnung der gültigen Kontextwürfel und der entsprechenden Optionsmengen wiederholen wir Schritt 3. Die korrigierten Kontextgruppen und Optionsmengen, welche nun alle hinsichtlich der definierten Beziehungen kompatibel sind, zeigt Abbildung 6.13.

6.5.2.4 Schritt 4: Sortiere die Optionen einer Optionsmenge

Nach erfolgreicher Durchführung von Schritt 3 sind alle relevanten und kompatiblen Optionsmengen identifiziert. Da Änderungsoperationen i.A. nicht kommutativ sind, müssen die Optionen noch zur gewünschten Anwendungsreihenfolge sortiert werden. Wie in Abschnitt 6.4 beschrieben, verwenden wir zur Sortierung der Optionsmengen die Funktion `sortOptionSet`

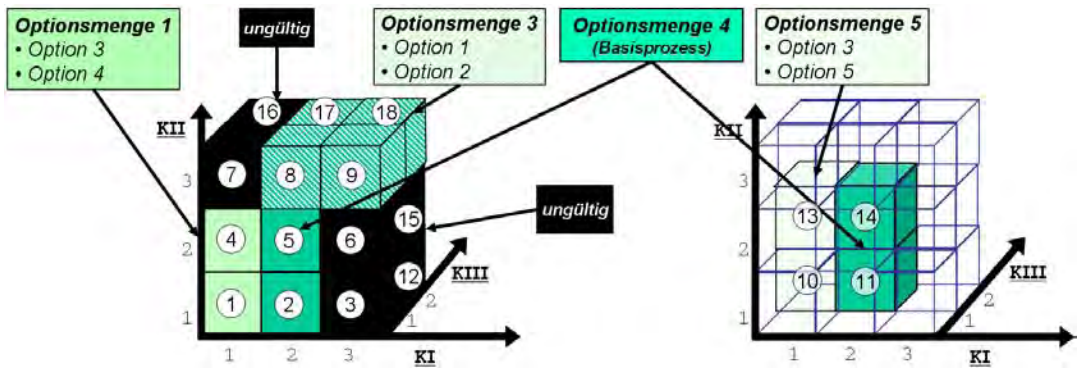


Abbildung 6.13: Gruppen gültiger Kontextbeschreibungen und entsprechende Optionsmengen

(vgl. Anhang A). Tritt dabei, z.B. aufgrund zyklischer Reihenfolgebeziehungen, ein Fehler auf, werden die Sortierung abgebrochen und eine entsprechende Fehlermeldung generiert.

Beispiel 6.14 (Schritt 4 anhand von Beispiel 6.10) *Zur Sortierung der Optionsmengen werden die Zeitstempel und expliziten Reihenfolge- bzw. Hierarchiebeziehungen der Optionen berücksichtigt. Für unser Beispielszenario entsprechen die Optionsfolgen den Reihenfolgen der Optionen in den in Abbildung 6.13 dargestellten Optionsmengen.*

```
// Schritt 4-6: Sortiere Optionsmengen und wende sie auf den Basisprozess an. Prüfe anschließend die
Korrektheit der Ergebnismodelle.
for each <CtxtBlock,OptionSet> ∈ ProcessVariantCandidates do
   // Schritt 4: Sortiere die Optionsmenge OptionSet zu SortedOptionSet
   if sortOptionSet(OptionSet,SortedOptionSet) = true then
      // Schritt 5: Leite das Ergebnismodell durch Anwendung der Optionsfolge auf den Basisprozess ab
      if calculateVariant(BaseProcess,SortedOptionSet,ResultingModel) = true then
      // Schritt 6: Prüfe Konsistenz des Ergebnismodells
         if checkSoundness(ResultingModel) = true then // Ergebnismodell ist konsistent
            storeResult(OptionSet,true,..) // Dokumentiere das Ergebnis der Prüfung
         else // Ergebnismodell ist nicht konsistent
            storeResult(OptionSet,false,..)
            writeErrorLog(OptionsSet,...)
            break // Abbruch
      else
      writeErrorLog(OptionsSet,...)
      break // Abbruch
   else
   writeErrorLog(OptionsSet,...)
   break // Abbruch
// Alle Optionsmengen sind korrekt und führen zu einem konsistenten Ergebnismodell
```

6.5.2.5 Schritt 5: Wende Optionen auf den Basisprozess an

Nach Sortierung der Optionsmengen zu Optionsfolgen, werden diese in Schritt 5 mit Hilfe der Funktion `calculateVariant` (vgl. Anhang A) auf den Basisprozess angewendet. Dabei transformieren die Optionen und die darin beschriebenen Änderungsoperationen den Basisprozess sukzessive zum Ergebnismodell. Dieses Vorgehen wird abgebrochen, wenn eine Option bzw. eine Änderungsoperation nicht auf den Basisprozess anwendbar ist. Dies ist zum Beispiel dann der Fall, wenn Objektreferenzen fehlen (z.B. für Aufsetzpunkte oder Elemente) oder unvollständige bzw. fehlerhafte Angaben, für die zur Durchführung einer Änderungsoperation notwendigen Parameter, vorliegen. Solche syntaktisch inkorrekten Änderungsoperationen sind nicht auf den Basisprozess anwendbar. Entsprechend ist die Optionsmenge als „inkonsistent“ einzustufen. Diese Information wird in einem zentralen Ergebnisdokument bzw. Protokoll abgespeichert.

Beispiel 6.15 (Schritt 5 anhand von Beispiel 6.10) *In Schritt 5 wenden wir die Optionsfolgen auf den Basisprozess an. Dabei resultiert in unserem Beispiel bei Anwendung von Optionsfolge 3 ein Fehler (vgl. Abbildung 6.14): Die `MODIFY`-Operation der Option 2 kann wegen fehlender Referenzen (Aktivität `F` wurde zuvor von Option 1 aus dem Basisprozess entfernt) nicht angewendet werden. Die Korrektheitsprüfung bricht daher ab und der Modellierer muss entsprechend „nachbessern“. Dabei stehen ihm die selben Änderungsmöglichkeiten, wie in Schritt 3 zur Verfügung. Zusätzlich kann er*

die Option bzw. ihre Änderungsoperationen anpassen, so dass keine strukturelle Abhängigkeit mehr existiert. In diesem Fall definieren wir mit Hilfe einer expliziten Reihenfolgebeziehung, dass Option 2 vor Option 1 angewendet werden soll (vgl. Abbildung 6.14). Daraus resultiert eine geänderte Optionsfolge, die nun ohne Probleme auf den Basisprozess angewendet werden kann.

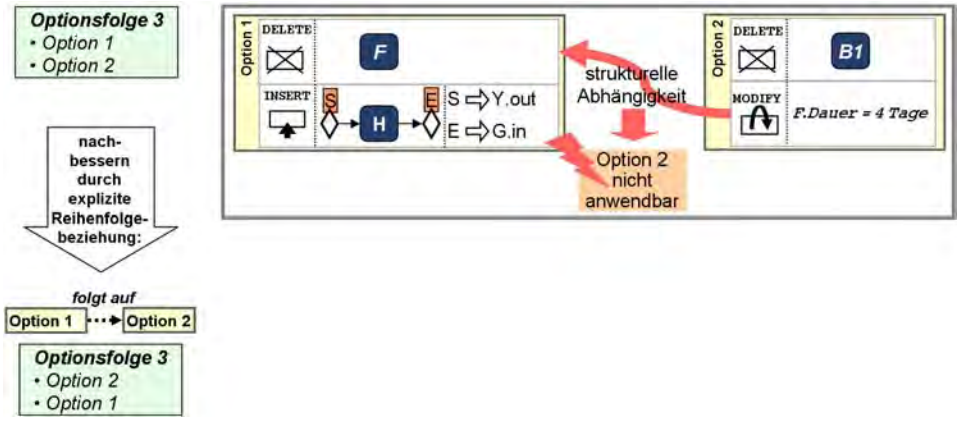


Abbildung 6.14: Fehler bei der Anwendung einer Optionsfolge

6.5.2.6 Schritt 6: Prüfe Korrektheit der Ergebnismodelle

Sind alle Optionen einer Optionsfolge erfolgreich auf den Basisprozess angewendet worden, muss in Schritt 6 die Korrektheit der entsprechenden Ergebnismodelle geprüft werden. Dazu werden die spezifischen Eigenschaften des zugrunde liegenden Prozess-Metamodells betrachtet (z.B. Verklemmungsfreiheit). Erfüllt ein Ergebnismodell alle Eigenschaften, ist es *korrekt*. Ist mindestens eine Eigenschaft des Metamodells nicht erfüllt, ist das Ergebnismodell *inkorrekt*. Das Ergebnis dieser Prüfung wird ebenfalls in einem Ergebnisdokument bzw. Protokoll gespeichert.

Beispiel 6.16 (Schritt 6 anhand von Beispiel 6.10) *Basierend auf unserem Szenario in Beispiel 6.10 können wir in Schritt 5 die in Abbildung 6.15 dargestellten Ergebnismodelle ableiten. Abhängig von den Eigenschaften des zugrunde liegenden Prozess-Metamodells können wir anschließend in Schritt 6 die Korrektheit der Ergebnismodelle bewerten. Im Folgenden führen wir die Diskussion der Korrektheit beispielhaft für die abgeleiteten Ergebnismodelle 1 bis 5 aus Abbildung 6.15 und mit den in Abschnitt 3.3 eingeführten Eigenschaften korrekter Prozessmodelle durch.*

- ***Optionsfolge 1** kann erfolgreich auf den Basisprozess angewendet werden (vgl. Abbildung 6.15a). In Schritt 6 wird jedoch eine Verklemmung erkannt: Nachdem durch Option 3 die Aktivität `A` verschoben wurde, liegen die Aufsetzpunkte nicht mehr in der ursprünglichen Reihenfolge vor. Dies führt nach dem Einfügen der Aktivität `I` zu einer Verklemmung, da die Eingangsbedingung des `OPJoin`-Knotens nicht ausgewertet werden kann. Eine solche Verklemmung ist aufgrund der Forderung nach der korrekten Terminierung von Prozessmodellen nicht in allen Prozess-Metamodellen erlaubt. Um für diese Fälle ein korrektes Prozessmodell zu erzeugen, muss der Modellierer korrigierend eingreifen. Mögliche Nachbesserungen sind hier z.B.:*
  - *die Anpassung der Aufsetzpunkte,*
  - *das Anpassen der Anwendungsreihenfolge der Optionen, oder*
  - *die Markierung der Aufsetzpunkte `A.in` und `A.out` im Basisprozess aus Abbildung 6.11a als „fix positioniert“ (vgl. Abschnitt 5.4.5).*

Nachbesserungen haben ggf. einen massiven Einfluss auf bereits als korrekt erkannte Prozessvarianten, daher erfordern Nachbesserungen eine erneute Korrektheitsprüfung.

- ***Optionsfolge 3*** *kann erfolgreich auf den Basisprozess angewendet werden. Das Ergebnismodell entspricht den von uns genannten Korrektheitskriterien (vgl. Abschnitt 3.3).*
- ***Optionsfolge 4*** *ist leer. Das heißt, der Basisprozess selbst stellt die gewünschte Variante dar. Entsprechend der Eigenschaften des zugrunde liegenden Prozess-Metamodells, kann das doppelte Schreiben des Datenobjektes durch die Aktivitäten* `B1` *und* `B2` *hier als Inkonsistenz bewertet werden. Eine Nachbesserung wäre es z.B. zu garantieren, dass ein Löschen der Aktivität* `B1` *oder* `B2` *immer vorgenommen wird. Der Modellierer müsste dazu entweder die Anwendung von Option 2 oder Option 3 für alle gültigen Kontextbeschreibungen ermöglichen. Dazu kann er entsprechende Kontextbedingungen und zusätzlich explizite Beziehungen zwischen diesen Optionen definieren, welche dies absichern (z.B. eine Auswahl n-aus-m Beziehung mit der Ausprägung* `1-aus-2`*). Die Korrektheitsprüfung der Prozessfamilie ist anschließend zu wiederholen.*
- ***Optionsfolge 5*** *führt nach Anwendung aller Optionen zu einem inkonsistenten Ergebnismodell: Nach Anpassung der Kantenbedingung ist der Strukturknotentyp ungültig. Anstelle des* `XOR`*-Knotens ist ein logischer* `ORSplit`*-Knoten erforderlich. Zum Nachbessern ist die Anpassung des* `XORSplit`*- und* `XORJoin`*-Knotens durch entsprechende* `MODIFY`*-Operationen denkbar.*

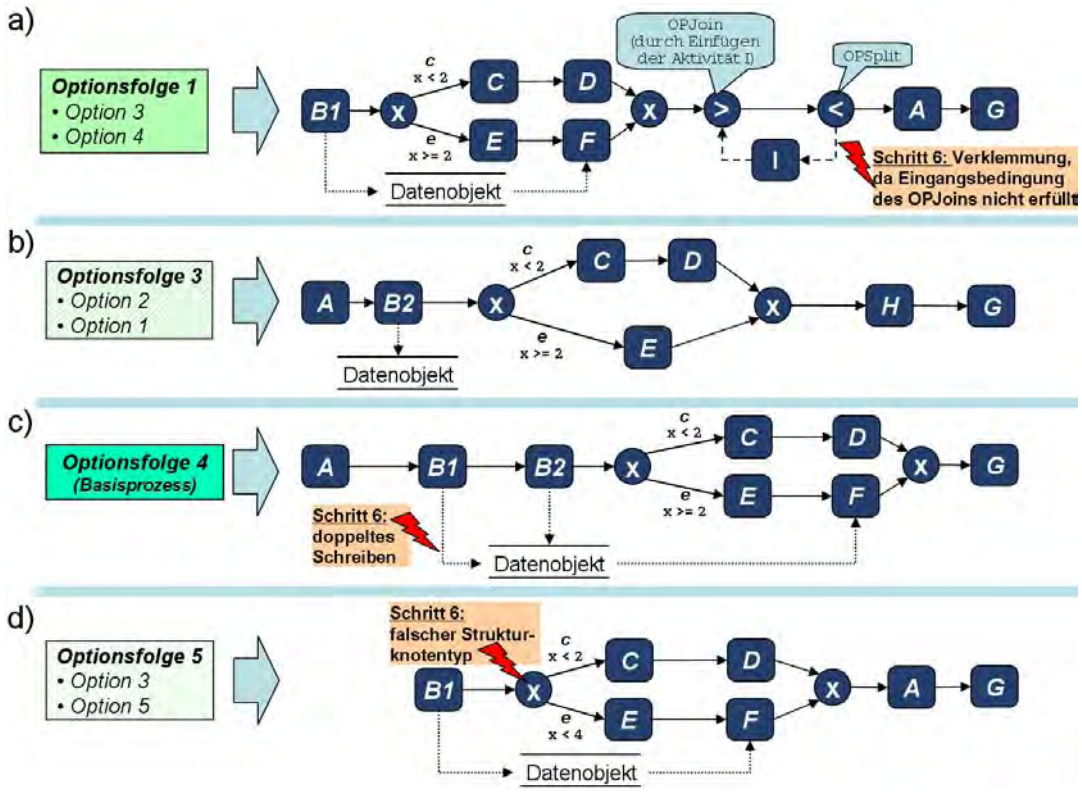


Abbildung 6.15: Ergebnismodelle der Varianten

6.5.2.7 Optimierungsaspekte

Die oben beschriebene Provop-Korrektheitsprüfung kann noch weiter optimiert werden. Zum Beispiel können die Schritte 1 bis 3 der Prüfung aus Effizienzgründen auch zusammengefasst werden. Zwecks einer besseren Erläuterung haben wir sie in dieser Arbeit einzelnen beschrieben.

Bisher brechen wir die Provop-Korrektheitsprüfung ab, sobald ein Fehler erkannt wird, der ein manuelles Eingreifen erfordert. Ein Optimierungsaspekt ist daher, das Fortsetzen einer Korrektheitsprüfung trotz Nachbesserungsbedarf zu ermöglichen. In diesem Fall werden zunächst alle Fehler dokumentiert und dem Modellierer nach Durchlaufen der Korrektheitsprüfung gemeldet. Auf diese Weise kann das wiederholte Anstoßen der Prüfung nach einzelnen Nachbesserungen vermieden werden. Nach entsprechenden Korrekturen kann die Prüfung dann wiederholt werden, beginnend bei Schritt 1.

Um die Aufwände zur Wiederholung vorangehender Prüfschritte einzusparen, können wir ggf. auch ein gezieltes Wiederaufsetzen der Prüfung an einem anderen Prüfschritt als Schritt 1 erlauben. Werden z.B. keine Änderungen am Kontextmodell vorgenommen, bleibt die Menge der gültigen Kontextbeschreibungen konstant und ein Start der Prüfung bei Schritt 1 ist nicht erforderlich. Wir können die Prüfung daher effizienter gestalten, wenn wir zunächst feststellen, welche Nachbesserungen der Modellierer vorgenommen hat und anschließend zum entsprechenden Schritt der Korrektheitsprüfung springen. Dieser Zwischenschritt erlaubt uns auch, bei jedem Anstoß der Korrektheitsprüfung, den spät möglichsten Schritt für ein Wiederaufsetzen der Prüfung zu ermitteln. Dazu untersuchen wir, welche Änderungen seit der letzten Prüfung vorgenommen wurden und ob diese Auswirkungen auf das Ergebnis der Prüfung haben. Als Grundlage für diese Analyse sind für die Objekte Basisprozess, Kontextmodell und Optionen jeweils Änderungshistorien erforderlich. Diese dokumentieren, wann welche Änderung an einem Objekt vorgenommen wurde. Auf Basis dieser Informationen können wir entscheiden, mit welchem Schritt wir die Korrektheitsprüfung beginnen müssen.

6.6 Diskussion

Im Folgenden stellen wir die verwandten Arbeiten zur Konfiguration von Prozessvarianten vor. Dabei betrachten wir neben Vorschlägen aus dem Prozessmanagement auch Ansätze aus dem Bereich der Softwareentwicklung.

Ansätze zur Variantenkonfiguration durch Prozessblöcke

In [RRvHZ00, Rup02] wird ein Bottom-up Ansatz für die (semi-) automatische Konfiguration projekt-spezifischer Modelle vorgestellt. Entsprechende Prozesse werden durch Kombination generischer Prozessblöcke erstellt. Dabei wird der projekt-spezifische Kontext (z.B. beschrieben durch Anforderungen, welche die Prozesskonfiguration beeinflussen) berücksichtigt. Dieser Ansatz erfordert einen vollständigen und umfangreichen Regelsatz, um die Prozesse konsistent und ausführbar aus den Prozessblöcken zusammenzusetzen. Die einzelnen Schnittstellen der Prozessblöcke müssen dokumentiert werden. Dieser modulare Ansatz ist in Provop ebenfalls abbildbar, wenn Optionen als Prozessblöcke betrachtet werden. Schnittstellen zwischen Prozessblöcken können dann in Form von Optionsbeziehungen abgebildet werden. Der wesentliche Unterschied zwischen den Ansätzen besteht jedoch darin, dass in Provop Bezug auf einem übergeordneten Basisprozess genommen wird und somit der grundsätzliche Ablauf einen Prozesses dokumentiert werden kann.

Ansätze zur Konfiguration von Referenzprozessmodellen

Ein Top-down Ansatz zur Prozesskonfiguration ist die Konfiguration eines Referenzprozessmodells. Generell dient ein Referenzprozessmodell als Vorlage zur Ableitung aller Varianten und stellt alle möglichen Kontrollfluss-Alternativen dar. In [RvdA07] wird z.B. beschrieben, wie solche Referenzprozesse durch *configurable Event-driven Process Chains (cEPCs)* dargestellt werden können und wie Konfigurationsdaten in Form von Labels den konfigurierbaren

Prozesselementen annotiert werden können (vgl. Abschnitt 5.7). Dabei können auch strukturelle und semantische Abhängigkeiten zwischen den durchzuführenden Anpassungen konfigurierbarer Elementen formuliert werden. Im Vergleich zu Provop können die verschiedenen Beziehungstypen allerdings nicht durch entsprechende Symbole und Kanten im Prozessmodell unterschieden werden; die graphische Modellierung der Beziehungen ist nicht möglich. Konfigurierbare Prozesselemente, die miteinander in Beziehung stehen, sind nur durch Analyse der annotierten Labels identifizierbar.

Eine Weiterentwicklung der cEPCs betrifft die Konfiguration von Prozessvarianten mittels einer Fragebogenmethode [RLS+07, RGDvdA07, Ros09]: Ein Fragebogen umfasst Fragen in natürlicher Sprache und wird auf Basis eines Fragebogenmodells erstellt. Der Variantenmodellierer wird dann durch den Fragebogen geführt, d.h. die Fragen werden entsprechend einer im Fragebogenmodell beschriebenen Reihenfolge gestellt und irrelevante Fragen, die aufgrund vorheriger Antworten des Modellierers überflüssig sind, bleiben außen vor. Die Antworten dienen anschließend als Konfigurationsdaten. Dabei wird jeder Antwort ein entsprechender Entscheidungspunkt im Referenzprozessmodell zugewiesen und führt entweder zur Ausführung, Auslassung oder optionalen Auslassung (d.h. abhängig von einer Bedingung) von Prozesselementen. Die Fragebogenmethode erlaubt eine Abstraktion von der eigentlichen Konfiguration von Prozessvarianten und ähnelt einem kontextbasierten Ansatz. So könnten die Werte der Kontextvariablen in Provop auch durch einen Fragebogen erhoben werden. Das Auslassen von bestimmten Fragen, abhängig von bereits gegebenen Antworten, entspricht dem Ansatz der Kontextregeln. Allerdings gehen die Autoren hier über den Provop-Ansatz hinaus, da sie nicht nur die Abbildung solcher Abhängigkeiten erlauben, sondern auch explizit betrachten, wie diese Regeln bei der Erhebung von Konfigurationsdaten berücksichtigt werden können. Die Vorgabe einer Reihenfolge, für das Erheben der Konfigurationsdaten, stellt dabei eine zusätzliche Erleichterung für den Variantenmodellierer dar. Diese Ansätze können ohne Weiteres in Provop adaptiert werden.

Ein zu cEPCs ähnlicher Ansatz wird in [GvdAJVlR07, Ros09] vorgestellt. Hier werden die Konzepte der Konfiguration eines Referenzprozessmodells auf Workflow-Modelle übertragen. Die Autoren erläutern dazu die Erweiterung der *Yet Another Workflow Language (YAWL)* [vdAtH05] zu *configurable YAWL (cYAWL)*. Zur Konfiguration der Referenzprozessmodelle in cYWAL beschreibt [GvdAJVlR07, Ros09] die Ansätze *(optionales) Blockieren* (engl. blocking) und *(optionales) Verstecken* (engl. hiding). Das Blockieren erlaubt, ganze Ausführungspfade auszulassen. Wird eine Aktivität blockiert, werden die nachfolgenden Aktivitäten des gleichen Pfades ebenfalls blockiert. Das Verstecken von Aktivitäten erlaubt im Gegensatz dazu eine einzelne Aktivität auszulassen, ohne dass davon nachfolgende Aktivitäten beeinflusst werden. Auf diese Weise können semantische und strukturelle Abhängigkeiten zwischen einer Aktivität und ihren Nachfolgern beim Auslassen berücksichtigt werden.

Über die Betrachtung des Kontrollflusses, bei der Konfiguration von Prozessmodellen, geht [LRDtH+08] hinaus. Hier werden bei der Konfiguration von Prozessmodellen auch Rollen und Objekte berücksichtigt. Der Ansatz verwendet cEPCs und überträgt die Konfigurationskonzepte entsprechend. Außerdem ist neben dem (optionalen) Entfernen einer Rolle bzw. eines Objektes auch eine Spezialisierung des Objektes möglich. Dazu werden zusätzlich Modelle zur Abbildung von Rollen- bzw. Objekt-Hierarchien erstellt. Im Augenblick liegt der Schwerpunkt von Provop auf dem Kontrollfluss, allerdings berücksichtigen wir bei der Variantenbildung ebenfalls die Anpassung von Prozesselementattributen. Auf diese Weise können z.B. auch Rollen als Attribute aufgefasst und mit Hilfe der `MODIFY`-Operation in Provop angepasst werden.

Prinzipiell können bei der Konfiguration eines Referenzmodells keine Prozesselemente verschoben oder neu hinzugefügt werden. Dies stellt einen wesentlichen Nachteil der Referenprozessmodellierung im Vergleich zum Provop-Ansatz dar.

Ansätze zur Konfiguration von Prozessmodellen durch Spezialisierung

Es gibt verschiedene Arbeiten zur Frage, wie Prozessmodellvariabilität durch *Spezialisierung von Prozessmodellen* abgebildet werden kann [MCH07, vdAB02]. Im Rahmen des MIT-Prozess-Handbuchs zeigt [MCH07], wie eine Spezialisierung für einfache Zustands- und Datenflussdiagramme eingesetzt werden kann. Für beide Diagrammtypen wird eine Menge von Transformationsregeln angeboten, deren Ergebnis die Spezialisierung eines bestimmten Modells ist.[11] Abbildung 6.16a zeigt ein Beispiel für einen generischen Bestellprozess in einem Restaurant. Durch Spezialisierung können daraus nun verschiedene Prozessvarianten abgeleitet werden: Abhängig von der Art des Restaurants (z.B. „Fast Food"-Restaurant, „Buffet"-Restaurant), werden die Schritte in einer anderen Reihenfolge ausgeführt (vgl. Abbildung 6.16b und 6.16c). Um diese Sequenzen zu erreichen, werden bestimmte Zustandsübergänge aus dem generischen Prozessmodell entfernt.

Abbildung 6.16: Konfiguration von Prozessvarianten durch Spezialisierung

Offensichtlich können Spezialisierungen verwendet werden, um zu einem gewissen Grad Prozessvarianten abzubilden. Im Gegensatz zu diesen Ansätzen, folgt Provop einem operationalen Ansatz, der unabhängig vom zugrunde liegenden Prozess-Metamodell ist. Provop erlaubt nicht nur Spezialisierungen durch das Entfernen von Prozesselementen, sondern ermöglicht es darüber hinaus, den generischen Referenzprozess (d.h. den Basisprozess) zur Abbildung von Prozessvarianten zu erweitern oder Attribute der Prozesselemente anzupassen. Nicht betrachtet wird in Provop eine Verfeinerung oder Aggregation, um Prozessvarianten abzuleiten. Dies ist allerdings indirekt, über die Hierarchisierung von Optionen, realisierbar. Zum Beispiel können Vateroptionen Platzhalter einfügen, die von Kindoptionen durch ein verfeinertes Prozessfragment ersetzt werden.

Analogien zwischen der Spezialisierung von Prozessen und Provop ergeben sich auch durch die *Prozesstaxonomien* bzw. *Spezialisierungshierachien*. So zeigt [MCH07], wie durch Spezialisierung eine Taxonomie von Prozessen generiert werden kann, welche die Untersuchung von Design-Alternativen und die Wiederverwendung existierender Prozessmodelle erleichtert. Abbildung 6.17 zeigt die entsprechende Prozesstaxonomie für das Beispiel des Bestellprozesses in einem Restaurant.

Eine ähnliche Strukturierung der Prozessfamilie präsentiert [RMvdT09] im Kontext aggregierter Prozessmodelle (vgl. Abschnitt 5.7). Hier entsteht eine Prozesstaxonomie durch hierarchische Strukturierung der Labels, die in einem aggregierten Prozessmodell (d.h. einer aEPC) dokumentieren, welches Prozesselement zu welcher spezifischen Prozessvariante bzw. welchem Teilbaum von Prozessvarianten gehört.

[11] Ähnlich dazu ist der in [vdAB02] beschriebene Ansatz basierend auf der Spezialisierung von Petri-Netzen.

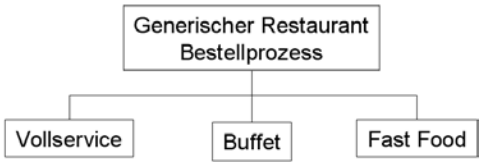


Abbildung 6.17: Prozesstaxonomie für den Bestellprozess

Prozesstaxonomien sind vergleichbar mit den Kontextmodellen in Provop. Im Unterschied zu Provop sind diese Taxonomien jedoch ausschließlich streng hierarchisch aufgebaut. Aus [MCH07] geht auch nicht hervor, wie eine Abhängigkeit zwischen Prozessvariante und Kontext hergestellt werden kann. Darüber hinaus können in beiden Ansätzen (d.h. [MCH07, RMvdT09]) keine ungültigen Kontextbeschreibungen definiert werden, wie es in Provop der Fall ist.

Verwandte Ansätze aus der Softwareentwicklung

Sehr ähnliche Problemstellungen zum Management von Prozessvarianten finden sich im Bereich des Versionsmanagements von Softwareprodukten bzw. -konfigurationen [LDC+89, CW97, CW98]. Hier müssen die Versionen und Varianten der Komponenten eines Softwareproduktes (z.B. Quellcode, kompilierte und ausführbare Dateien, Dokumentationen) gehandhabt werden. Versionen bilden im Vergleich zu Varianten eine zeitliche Dimension ab. Sie sind Weiterentwicklungen eines Produktes, während Varianten koexistierende Alternativen darstellen. Für die Konfiguration einer Software müssen die einzelnen Komponenten zu einem Produkt zusammengestellt werden. Durch die Vielzahl der einzelnen Komponenten und ihrer möglichen Versionen, ergibt sich hier eine hohe Komplexität.

Der Ansatz der *Feature-Diagramme* bzw. *Feature-Bäume* ermöglicht es, die Varianz innerhalb einer (Software-)Produktfamilie zu beschreiben [SHTB07, CE00, vGBS01]. Ein *Feature* beschreibt in diesem Zusammenhang eine bestimmte Systemeigenschaft, die ein Produkt genauer spezifiziert. Ein Produkt kann somit durch Dekomposition hierarchisch in verschiedene Features zerlegt werden. In Feature-Diagrammen wird ebenfalls das Konzept der *Variation Points* eingesetzt. Über diese Punkte wird beschrieben, welche Features zwingend, optional oder alternativ in einem Produkt vorhanden sein müssen. Auf diese Weise können verschiedene Produktkonfigurationen in einem Diagramm abgebildet werden.

Feature-Diagramme beziehen sich nicht auf Prozessmodelle. Eine Analogie zwischen diesen Diagrammtypen und Provop ist allerdings gegeben, wenn wir Feature-Diagramme als Repräsentation der modellierten Optionen bzw. der Optionsbeziehungen betrachten. So bilden Feature-Diagramme z.B. Hierarchien, wechselseitige Ausschlüsse und Implikationsbeziehungen ab, sowie eine Auswahl n-aus-m. Nicht abgebildet sind Reihenfolgeabhängigkeiten. Außerdem erfordern Feature Diagramme immer eine streng hierarchische Betrachtung. Netze von Features bzw. Optionsbeziehungen, wie in Provop erforderlich, sind nicht abbildbar.

Einen Ansatz zur einfacheren Handhabung von Software-Konfigurationen und Versionen ist der *HICoV*- Ansatz [Mun96]. Hier wird der Versionsraum (engl. version space) durch sog. „Optionen“ erfasst. Diese repräsentieren jeweils eine spezifische Eigenschaft eines Produktes.[12] Eine Option kann dann z.B. das Betriebssystem oder die Oberflächensprache der Benutzerschnittstelle der Software sein. Versionen und Optionen sind so miteinander verknüpft, dass das gewünschte Produkt anschließend durch Auswahl von Optionen erstellt

[12] Damit entsprechen Optionen in [Mun96] den Features bzw. Merkmalen in sog. Feature-Bäumen (vgl. Abschnitt 5.7).

werden kann. Darüber hinaus erlaubt [Mun96] verschiedene Regeln zwischen Optionen zu definieren, welche die Korrektheit der Auswahl sicherstellen sollen. Solche Beziehungen sind wechselseitiger Ausschluss, Implikation und Hierarchie. Beispiele hierzu sind nachfolgend angegeben:

`mutex operatingsystem: unix msdos os2`
– es werden drei Betriebssysteme angeboten, von denen aber nur eins ausgewählt werden darf
`subopt unix: aix ultirx sunos`
– die Option `unix` hat diverse Suboptionen, für die es Versionen geben kann
`mutex language: english deutsch norsk`
– die Software unterstützt drei Sprachen, von denen aber nur eine ausgewählt werden darf
`dep: help language`
– die Anzeige der Hilfe ist abhängig von der ausgewählten Sprache

Die Optionen in [Mun96] entsprechen in Provop den möglichen Werten von Kontextvariablen. Dabei erlaubt Provop durch Kontextregeln ebenfalls einen wechselseitigen Ausschluss und eine Implikation von bestimmten Werten einer Kontextvariable. Der Vorteil von Provop ist an dieser Stelle allerdings, dass auch Abhängigkeiten zwischen Kontextvariablen abgebildet werden können.

Einen ähnlichen Ansatz zur Erfassung des Versionsraumes präsentiert [Sci94]. Hier werden in einem objekt-orientierten Datenmodell die Objekte und ihre zugehörigen Versionen gespeichert. Mit Hilfe einer entsprechenden Anfrage (engl. Query) sollen aus dem Datenmodell genau die Versionen von Objekten herausgefiltert werden, die bestimmte Eigenschaften erfüllen. Diese Eigenschaften beschreibt der Autor als Kontextvariablen, die – gemeinsam betrachtet – ein multidimensionales Koordinatensystem ergeben. Durch Angabe spezifischer Werte für diese Kontextvariablen und eine entsprechend formulierte Query nach dem Schema `RETRIEVE - FROM - WHERE - INCONTEXT` können alle Versionen eines Datenobjektes ausgegeben werden, welche die beschriebenen Eigenschaften erfüllen.

Der Ansatz hat Ähnlichkeit mit dem Kontextwürfel in Provop. Auch hier stellen wir Kontextvariablen in einem mehrdimensionalen Gebilde dar. In Provop erlauben wir allerdings zusätzlich die Beschreibung von Kontextregeln, welche bestimmte Wertekombination von Kontextvariablen ausschließen. Darüber hinaus betrachten wir den Kontext einer Prozessvariante nicht als die Beschreibung spezifischer Eigenschaften des Prozessmodells, sondern als dessen Anwendungskontext. Dies ist für das Management von Prozessvarianten, die für bestimmte Rahmenbedingungen benötigt werden, auch sinnvoller. Eigenschaften eines Prozessmodells wären z.B. die Anzahl der Aktivitäten, die Laufzeit und verwendete Datenobjekte.

Ansätze zur Gewährleistung der Korrektheit konfigurierter Prozessmodelle

Die Prüfung der Korrektheit eines Prozessmodells ist abhängig vom zugrunde liegenden Prozess-Metamodell. So erlauben EPKs z.B. zyklische Strukturen bzw. Verklemmungen. Dies würde bei der Ausführung des Prozesses in einem WfMS zu einem Fehler führen; daher sind die Terminiation und die Zyklenfreiheit wichtige Korrektheitseigenschaften von Workflow Beschreibungssprachen (wie z.B. WS-BPEL, YAWL).[13] Aufgrund dieser Abhängigkeit zwischen Prozess-Metamodell und seinen Korrektheitseigenschaften existieren zahlreiche Ansätze bzw. Algorithmen, die jeweils die Korrektheit für ein bestimmtes Prozess-Metamodell

[13] Man unterscheidet in diesem Zusammenhang auch zwischen struktureller Korrektheit und einem korrekten Ausführungsverhalten eines Workflow-Modells (engl. structural and behavioural soundness).

prüfen können [vdA97, vdA00, KR96, VvdAtH07, AP08, TBG08, TBB08]. So können inkorrekte Prozessmodelle, z.B. durch Reduktion und anschließende Analyse des verbleibenden Prozessmodells, einfach identifiziert werden [TBB08, TBG08]. Ein weiterer Ansatz sucht nach verschiedenen Mustern in den Prozessmodellen, die typischerweise inkorrekte Strukturen darstellen [vDMvdA06]. In [vdA00] werden dazu Petri-Netze und dynamische Analysen auf Petri-Netzen verwendet.

In der Literatur finden sich nur wenige Ansätze, die sich mit der fundamentalen Frage beschäftigen, wie Prozessvarianten basierend auf einem (Referenz-) Prozessmodell konfiguriert werden können, so dass am Ende ein korrektes und konsistentes Ausführungsverhalten garantiert werden kann [vdADG+08, RvdA07, RLS+07, GvdAJVlR07]. Dabei steht die Korrektheit eines einzelnen Modells meist im Vordergrund, d.h. die Korrektheit wird jeweils nur für eine mögliche Variante betrachtet. In [vdADG+08, Men08] wird z.B. gezeigt, wie Referenzprozessmodelle inkrementell so konfiguriert werden können, dass die Korrektheit der Varianten sichergestellt werden kann. Dabei gehen die Autoren davon aus, dass der Referenzprozess selbst korrekt ist und jeder Konfigurationsschritt wieder zu einem korrekten Prozessmodell führt. Auf diese Weise kann die Korrektheit des abgeleiteten Prozessmodells ohne weitere Prüfungen gewährleistet werden. In Provop ist es nicht immer wünschenswert von einem korrekten Basisprozess ausgehend eine Prozessvariante zu konfigurieren. Die Beschreibung von Korrektheits-erhaltenden Änderungsoperationen würde außerdem die Mächtigkeit dieser Operationen einschränken. Darüber hinaus betrachten die Autoren nicht, wie die Korrektheit einer ganzen Prozessfamilie effizient sichergestellt werden kann.

Ein Aspekt, welcher bei der Korrektheitsprüfung häufig vernachlässigt wird, ist die *semantische Korrektheit* der Prozessmodelle. Hier existieren nur wenige Ansätze, die sich mit der Frage beschäftigen, wie semantische Abhängigkeiten (sog. Constraints) modelliert und die Einhaltung dieser Constraints für Prozessmodelle und -instanzen geprüft werden können [Sin95, CGP00, Esh02, LRD06, Pes08, LRMGD09, LRD08b]. Solche Ansätze können verwendet werden, um die Kompatibilität von Optionsbeziehungen in Provop zu prüfen.

Ansätze zur Abbildung von Regeln und Abhängigkeiten

In diesem Kapitel haben wir verschiedene Regeltypen eingeführt: Kontextregeln zur Bewertung der Gültigkeit von Kontextbeschreibungen, Kontextbedingungen zur Bewertung der Relevanz einer Option in einer gegebenen Kontextbeschreibung und Optionsbeziehungen zur Abbildung von strukturellen und semantischen Abhängigkeiten zwischen Optionen. In der wissenschaftlichen Literatur finden sich zahlreiche Arbeiten zur Verwendung und Beschreibung von Regeln und Abhängigkeiten. Dabei wird zwischen verschiedenen Klassen von Regeln unterschieden [End04, VH01]. Beispiele hierfür sind: Integritätsregeln, Berechnungsvorschriften, Restriktionen, Heuristiken, Stimulus/ Response, Ableitungsregeln und Aktionsregeln [HKMS94, VH01, Ros03, Gro00]. Diese Regeln unterscheiden sich in erster Linie in den betroffenen Objekten, den Möglichkeiten zur Prüfung der Einhaltung der Regeln und dem Umgang mit Fehlern oder Konflikten.

Allgemeine Anforderungen an die Repräsentation von Regeln identifiziert [End04]. So sollten Regeln u.A. „angelehnt an die natürliche Sprache“, nachvollziehbar, strukturiert und atomar sein. In Provop erfüllen wir diese Anforderungen für alle drei Regeltypen. Darüber hinaus formuliert [End04] Anforderungen an eine Regelmenge. Diese muss vollständig, schnittfremd (d.h. es gibt keine überlappenden Regeln), konsistent sowie aktiv und mit eindeutiger Verantwortlichkeitszuordnung verwaltet werden. Diese Anforderungen bzw. ihre Erfüllung, halten

wir auch in Provop für die Definition von Regeln als sinnvoll, auch wenn wir im Rahmen dieser Arbeit nicht all diese Kriterien im Detail betrachtet haben oder strikt fordern.[14]

Zur technischen Abbildung von Regeln und Abhängigkeiten werden in der Literatur einige Ansätze vorgestellt. [HP99, End04, MGR03] verwenden z.B. sog. *Event-Condition (EC)*, *Event-Condition Action (ECA)* und *Event-Condition-Action-Alternative-Action (ECAA)*-Regeln zur regelbasierten Modellierung von Geschäftsprozessen. Abbildung 6.18 zeigt hierzu beispielhaft, wie aus einer solchen Regel ein Prozessmodell abgeleitet werden kann. Ein weiteres Beispiel für ein WfMS, das zur Spezifikation von Arbeitsabläufen eine Regelsprache verwendet, ist *ULTRAflow* [FRF01].

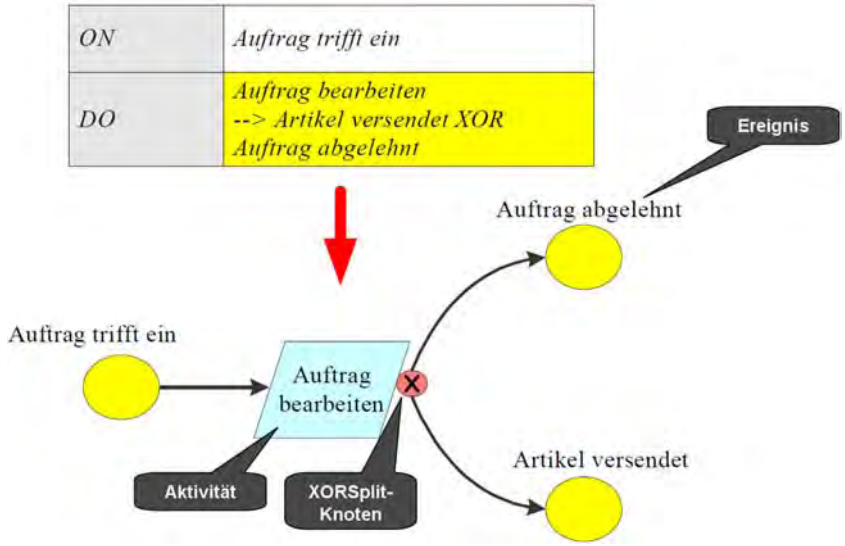


Abbildung 6.18: Regelbasierte Modellierung von Geschäftsprozessen mit ECAA-Regeln nach [End04]

Darüber hinaus finden sich in der Literatur Ansätze zur Beschreibung struktureller und semantischer Abhängigkeiten durch eine *Event Algebra* [Sin95] oder Sprachen bzw. Logiken, wie z.B. die *Object Constraint Language (OCL)*, *Computation Tree Logic (CTL)* oder die *Linear Temporal Logic (LTL)* [Pes08, CGP00, LRMGD09]. Die Einhaltung der so beschriebenen Abhängigkeiten kann mit Hilfe von Model-Checking-Techniken verifiziert werden [CGP00].

6.7 Zusammenfassung

Nach der Modellierung eines Basisprozesses und seiner Optionsmenge können die Ergebnismodelle der Varianten einer Prozessfamilie konfiguriert werden. Ein wesentlicher Aspekt dabei ist die Unterstützung des Variantenverantwortlichen bei der Auswahl der anzuwendenden Optionen. Provop bietet dem Variantenverantwortlichen neben einer rein manuellen Auswahl die Möglichkeit, den Kontext einer Prozessvariante bei deren automatischen Konfiguration zu berücksichtigen. Zur Sicherstellung der Kompatibilität (manuell oder kontextbasiert) ausgewählter Optionsmengen kann der Variantenmodellierer außerdem verschiedene Beziehungen zwischen Optionen beschreiben, wie z.B. wechselseitiger Ausschluss oder Implikation. Die Kompatibilitätsprüfung stellt dann die Einhaltung dieser Beziehungen bei der Konfiguration einer Prozessvarianten sicher. In diese Betrachtung eingeschlossen sind auch Beziehungen, welche die Anwendungsreihenfolge von Optionen explizit beschreiben. Diese Reihenfolge ist erforderlich, da Optionen sequentiell auf den Basisprozess angewendet

[14] Zum Beispiel fordern wir keine schnittfremde Kontextregeln, solange die Gültigkeit aller Kontextbeschreibungen eindeutig ist.

werden müssen. Schließlich prüfen wir mit der Provop-Korrektheitsprüfung die Korrektheit der gesamten Prozessfamilie. Dazu verwenden wir ein Prozess-Metamodell unabhängiges Rahmenwerk.

In Kapitel 7 werden wir diese Ansätze weiter vertiefen. Wir werden unter anderem zeigen, wie wir in Provop mit dynamisch veränderlichen Kontextvariablen umgehen und in diesem Zusammenhang die Korrektheit einer Prozessfamilie sicherstellen.

7

Ausführung von Prozessvarianten

In Kapitel 6 haben wir beschrieben, wie zur Ableitung einer Prozessvariante mehrere Optionen kontextabhängig ausgewählt und sukzessive auf den Basisprozess angewendet werden. Bei dieser kontextabhängigen Auswahl relevanter Optionen sind wir bisher von konstanten Werten der Kontextvariablen ausgegangen; d.h., wir haben angenommen, dass diese Werte zum Konfigurationszeitpunkt vorliegen und sich während der Laufzeit von Instanzen des konfigurierten Prozesses nicht mehr ändern können. In der Praxis werden die Werte von Kontextvariablen jedoch oft dynamisch an geänderte Rahmenbedingungen angepasst oder wichtige Kontextinformationen liegen erst zur Laufzeit eines Prozessmodells vor. Das vorliegende Kapitel erläutert, wie Provop mit solchen Laufzeit-Kontextänderungen verfährt und die korrekte Ausführung von Prozessvarianten jederzeit gewährleistet. Relevant sind Aspekte, welche die Atomarität von Optionen, die Kontextabhängigkeit der Optionen sowie Optionsbeziehungen betreffen. Kapitel 7 gliedert sich wie folgt: Abschnitt 7.1 motiviert fachliche Anforderungen an die Ausführung von Prozessvarianten. Abschnitt 7.2 stellt dar, wie Kontextänderungen in Provop abgebildet werden können. Anschließend erläutert Abschnitt 7.3, wie die Korrektheit einer Prozessfamilie auch für einen dynamisch veränderten Kontext gewährleistet werden kann. Desweiteren präsentiert Abschnitt 7.3 einen Ansatz zur Prüfung der Kompatibilität ausgewählter Optionen mit allen Optionsbeziehungen zur Laufzeit. Abschließend diskutiert Abschnitt 7.4 verwandte Arbeiten. Abschnitt 7.5 fasst das Kapitel zusammen.

7.1 Motivation und Anforderungen

Je länger die Ausführungsdauer eines Prozesses ist, desto wahrscheinlicher werden Prozessanpassungen, die noch während der Ausführung zu berücksichtigen sind [RRD04b]. Ein Auslöser für solche Prozessanpassungen sind Kontextänderungen zur Laufzeit, etwa Änderungen an Gesetzestexten, Neubewertungen von Kosten oder andere sich verändernde Kontextinformation. Um diese Dynamik von Kontextvariablen abbilden zu können, muss im Kontextmodell für jede Variable vermerkt werden, wann ihr endgültiger Wert feststeht. Das heißt, wir müssen jeweils definieren, ob dieser Wert bereits zum Konfigurationszeitpunkt oder erst während der Ausführung des Prozessmodells bestimmt wird. Bezogen auf Prozessvarianten bedeutet dies zum einen, dass ggf. erst zur Laufzeit entschieden werden kann, welche

Prozessvariante konkret auszuführen ist. Zum anderen kann es erforderlich werden, dass die Ausführung der aktuellen Prozessvariante unterbrochen und dynamisch zu einer anderen Prozessvariante gewechselt wird.

Die skizzierte Dynamik des Kontextes, und damit die Relevanz unterschiedlicher Prozessvarianten bei der Ausführung einer Prozessinstanz zur Laufzeit, spiegelt sich in der Dynamik der Auswahl und Anwendung von Optionen in Provop wider. Bereits bei der statischen Konfiguration von Prozessvarianten werden Optionen nur dann angewendet, wenn ihre Kontextbedingung erfüllt ist. Bei der dynamischen Konfiguration wollen wir diese Kontextabhängigkeit beibehalten. Mögliche Änderungen an den Werten einzelner Kontextvariablen erfordern jedoch, dass die aktuellen Kontextinformationen zur Laufzeit abgefragt und die Kontextbedingungen der Optionen dynamisch ausgewertet werden. Dabei streben wir in Provop an, bereits zum Zeitpunkt der Konfiguration alle Ausführungsvarianten in einem Modell abzubilden (z.B. zu Visualisierungszwecken). Zu diesem Zweck sollen (dynamische) Optionen als vorgeplante Anpassungen des Basisprozesses, die nicht notwendigerweise alle wirksam werden, bereits bei der Modellierung und Konfiguration der Prozessvarianten berücksichtigt werden. Dadurch kann zur Laufzeit, in Abhängigkeit vom aktuellen Kontext, über die Anwendung der vorgeplanten Anpassungen entschieden werden. Dabei ist die Atomarität der Optionen in jedem Fall zu bewahren. Das heißt, es müssen entweder immer alle Änderungsoperationen einer Option gemeinsam angewendet werden oder aber es darf keine von ihnen wirken. Dies gilt auch nach Kontextänderungen, die Einfluss auf die „Gültigkeit“ einer dynamischen Option haben. In diesem Fall ist es nicht möglich, eine Option nur teilweise anzuwenden, etwa nur eine bestimmte Teilmenge aller in ihr beschriebenen Änderungsoperationen.

Ein anderer Aspekt von sich dynamisch änderndem Kontext betrifft die Einhaltung von Optionsbeziehungen (d.h. Beschränkungen) zur Laufzeit. Entscheiden wir über die Anwendung bestimmter Optionen erst zur Laufzeit, resultiert entsprechend eine dynamische Optionsmenge. Diese ist dann ggf. nicht mehr kompatibel mit den definierten Optionsbeziehungen. Das heißt es können Konflikte zwischen der Kontextabhängigkeit von Optionen und ihren Optionsbeziehungen entstehen. Solche Konflikte sollten keine manuellen Eingriffe des Endanwenders erfordern, um aufgelöst zu werden. Stattdessen muss bereits bei der Modellierung definiert werden, ob zur Laufzeit Kontextabhängigkeit oder Optionsbeziehungen höher priorisiert werden sollen.

Eine weitere wichtige Anforderung ist die Korrektheit einer Prozessfamilie im dynamischen Fall. So müssen die durch dynamische Konfiguration entstehenden Variantenmodelle weiterhin ausführbar sein und den Korrektheitskriterien des zugrunde liegenden Prozess-Metamodells genügen. Eine entsprechende Korrektheitsprüfung ist unerlässlich.

Es ergeben sich die in Tabelle 7.1 aufgeführten Anforderungen an die Ausführung von Prozessvarianten mit teilweise dynamischem Kontext.

7.2 Abbildung der dynamischen Konfiguration in Provop

Wie werden Kontextänderungen in Provop erfasst? –Abschnitt 7.2.1 beschreibt, wie Laufzeitänderungen von Kontextvariablen im Kontextmodell abgebildet werden. Anschließend erläutert Abschnitt 7.2.2, wie sich diese Kontext-Dynamik auf die Anwendung von Optionen auswirkt.

Tabelle 7.1: Anforderungen an die Ausführung von Prozessvarianten

Abbildung der Dynamik im Kontextmodell

Anforderung 7.1 *Für jede Kontextvariable aus dem Kontextmodell eines Prozesstyps muss angegeben werden, wann ihr endgültiger Wert feststehen soll (zum Zeitpunkt der Modellierung/ Konfiguration oder erst zur Laufzeit).*

Dynamische Anwendung von Optionen

Anforderung 7.2 *Zur Laufzeit muss auf Basis aktueller Werte der Kontextvariablen über die Anwendung bestimmter Anpassungen entschieden werden können. Die Änderungsoperationen sollen als vorgeplante Änderungen bereits bei der Konfiguration des Variantenmodells berücksichtigt werden.*

Anforderung 7.3 *Bei der Anwendung von Optionen muss deren Atomarität stets gewährleistet werden. Insbesondere muss auch nach Kontextänderungen gewährleistet sein, dass immer alle oder gar keine Änderungsoperationen einer Option wirksam werden.*

Anforderung 7.4 *Bei der Modellierung von Prozessvarianten muss spezifiziert werden, wie zur Laufzeit auftretende Konflikte bzgl. Kontextabhängigkeit von Optionen auf der einen Seite und Optionsbeziehungen auf der anderen Seite automatisch aufgelöst werden sollen.*

Korrekte Ausführung zur Laufzeit

Anforderung 7.5 *Die Korrektheit aller dynamisch konfigurierbaren Prozessvarianten muss bereits zum Zeitpunkt der Konfiguration abgebildet werden.*

Anforderung 7.6 *Die semantische Kompatibilität der ausgewählten Optionen mit definierten Optionsbeziehungen muss zur Laufzeit gewährleistet werden.*

7.2.1 Statische vs. dynamische Kontextvariablen

Generell gilt, dass die Werte der Kontextvariablen eines Kontextmodells zu unterschiedlichen Zeitpunkten feststehen können (vgl. Anforderung 7.1). Während für manche Kontextvariablen bereits zum Zeitpunkt der Konfiguration konstante Werte festliegen, werden andere Kontextvariablen erst zur Laufzeit geschrieben; darüber hinaus können sich letztere während der Prozessausführung ggf. noch dynamisch ändern. In Provop unterscheiden wir entsprechend zwischen *statischen* und *dynamischen Kontextvariablen* (angezeigt durch das Attribut „Mode“ der Kontextvariablen). Der Kontextmodellierer kann daher explizit angeben, ob der Wert einer Kontextvariable erst zur Laufzeit ober bereits zum Zeitpunkt der Konfiguration (endgültig) festliegt (vgl. Beispiel 7.1). Wir erweitern entsprechend die bisherige Definition von Kontextvariablen (vgl. Definition 6.1):

Definition 7.1 (Erweiterte Definition der Kontextvariablen)
Eine Kontextvariable ist ein Tupel $V = (VariableName, ValueRange, Mode)$.

- *VariableName und ValueRange sind – wie in Definition 6.1 – der eindeutige Bezeichner und Wertebereich einer Kontextvariablen.*
- $Mode \in \{\texttt{static}, \texttt{dynamic}\}$ *legt für die Kontextvariable fest, ob ihr Wert bereits zur Konfigurationszeit endgültig festliegen muss (Mode = static) oder ob er erst zur Laufzeit fixiert wird (Mode = dynamic).*

Beispiel 7.1 (Dynamische Kontextvariablen) *In Tabelle 6.3 haben wir den Kontext des Produktentwicklungsprozesses vorgestellt. Die darin beschriebene Kontextvariable* `Fahrzeugtyp` *ist statisch; d.h. sie wird bei der Konfiguration der Prozessvariante einmalig festgelegt und ändert sich während der Ausführung von Instanzen der Prozessvarianten nicht mehr. Anders verhält sich die Variable* `Phase`*. Ihr Wert kann sich zur Laufzeit ändern, sobald eine Entwicklungsphase abgeschlossen wird und die nachfolgende Phase beginnt. Kontexttabelle 7.2 zeigt die entsprechende Erweiterung von Kontexttabelle 6.3.*

Tabelle 7.2: Ausschnitt aus dem Kontextmodell der Produktentwicklung

Variablenname	Beschreibung	Wertebereich	Modus
Geschäftsbereich	Geschäftsbereich der Daimler AG	Daimler Trucks (DT), Daimler Buses (DB), Mercedes-Benz Vans (MV), Mercedes-Benz Cars (MBC)	statisch
Fahrzeugtyp	Name des Fahrzeugtyps	LKW, Van, Bus, PKW	statisch
Marke	Bezeichnung der Marke	Mercedes Benz, Smart, Maybach, Fuso	statisch
Phase	Entwicklungsphase des Fahrzeugprojekts	Vorentwicklung, Entwicklung, Anlauf, Produktion	dynamisch

7.2.2 Statische vs. dynamische Optionen

In Provop ist durch Modellierung des Basisprozesses und seiner Optionen generell bereits vor der Prozessausführung bekannt, welche Prozessvarianten konfigurierbar sind. Kontextänderungen zur Laufzeit stellen daher einen Wechsel zwischen zwei a-priori bekannten bzw. konfigurierten Variantenmodellen einer Prozessfamilie dar. Ein solcher Wechsel zur Laufzeit könnte durch Abbruch der laufenden Prozessvariante und den Übergang der Ausführung auf die neue Prozessvariante abgebildet werden. Ein solcher Abbruch gefolgt von einem Neustart kann aber zu Verlusten und Doppelarbeit führen, was in der Praxis meist nicht tolerable ist (vgl. Beispiel 7.2).

Beispiel 7.2 (Kontextänderung zur Laufzeit und resultierender Variantenwechsel) *Abbildung 7.1 zeigt einen Basisprozess mit drei Optionen. Die Kontextbedingungen von zwei Optionen sind auf Basis der dynamischen Kontextvariable* `Phase` *definiert (vgl. Tabelle 7.2). Entsprechend ergeben sich sechs mögliche Prozessvarianten. In der gegebenen Kontextbeschreibung*

```
{<Phase, Anlauf>, <Marke, Fuso>}
```

wird Variante 2 ausgewählt, instanziiert und ausgeführt. Angenommen es tritt nun während der Ausführung eine Kontextänderung auf mit anschließender Kontextbeschreibung:

```
{<Phase, Produktion>, <Marke, Fuso>}
```

Diese Kontextbeschreibung erfordert die Ausführung von Variante 3. Um den Wechsel zwischen den Prozessvarianten zu ermöglichen, muss die Ausführung von Variante 2 gestoppt und Variante 3 gestartet werden. Dies führt dazu, dass Aktivitäten `A` *und B erneut ausgeführt werden müssen.*

Um solche Schwierigkeiten zu vermeiden, sollen in Provop zum Zeitpunkt der Konfiguration alle möglichen Prozessvarianten, die sich durch eine dynamische Kontextänderung ergeben können, in einem Variantenmodell abgebildet werden. Das heißt, eine Prozessvariante, die für eine spezifische Kontextbeschreibung konfiguriert ist, beinhaltet auch alle anderen Prozessvarianten, die sich nach Änderungen der Kontextvariablenwerte zur Laufzeit ergeben

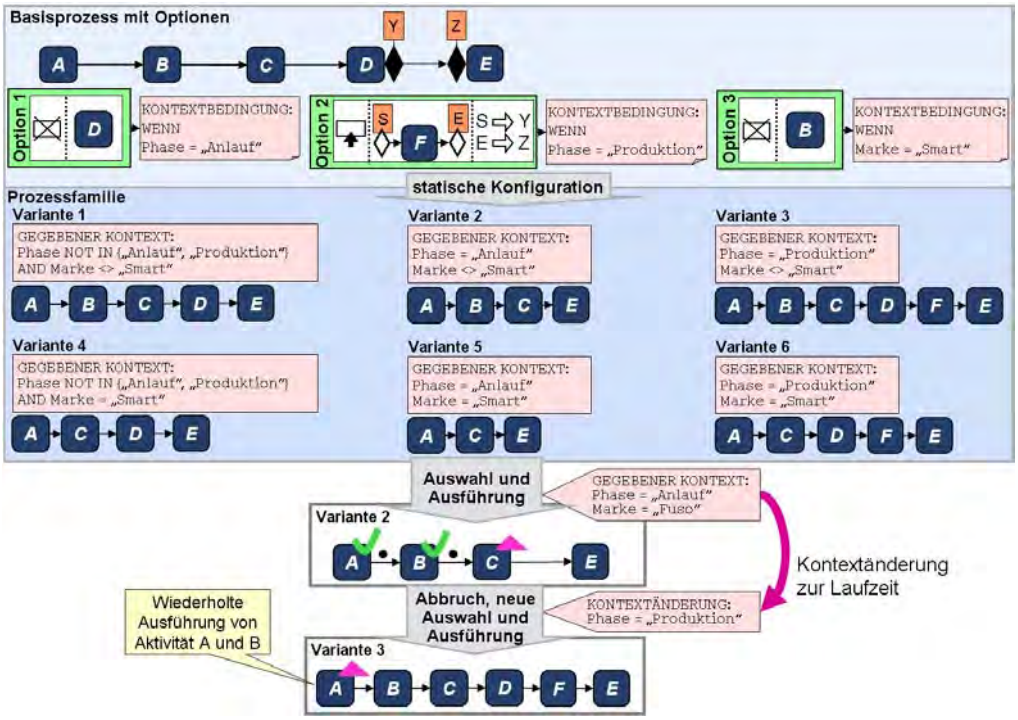


Abbildung 7.1: Kontextänderung zur Laufzeit und resultierender Variantenwechsel

können. In Abschnitt 4.2.2 haben wir den Ein-Modell-Ansatz zur Repräsentation mehrerer Prozessvarianten in einem Prozessmodell vorgestellt. Ziel der dynamischen Konfiguration ist es im Prinzip, einen solchen Ein-Modell-Ansatz für eine Teilmenge von Prozessvarianten einer Prozessfamilie zu realisieren. In Provop ist die Menge der gemeinsam in einem Prozessmodell abzubildenden Varianten auf die jeweils anzuwendenden Optionen zurückführbar. Indem wir mehrere Optionen, welche zur Konfiguration unterschiedlicher Prozessvarianten benötigt werden, gemeinsam und kontextabhängig (d.h. *dynamisch*) auf den Basisprozess anwenden, können wir zur Laufzeit letztlich die konkrete Prozessvariante determinieren. Wir sprechen von einer *dynamischen Konfiguration* der Prozessvarianten (vgl. Beispiel 7.3).

Beispiel 7.3 (Ein Prozessmodell zur Abbildung multipler Prozessvarianten) *Abbildung 7.2 zeigt, wie durch dynamische Konfiguration mehrere Varianten in einem Prozessmodell abbildbar sind. Variante 71 repräsentiert das integrierte Prozessmodell der Varianten 1, 2 und 3 aus Abbildung 7.1, während Variante 8 die Varianten 4, 5 und 6 aus Abbildung 7.1 zusammenfasst. Dabei werden alle Optionen, deren Kontextbedingungen von der dynamischen Kontextvariable* `Phase` *abhängen (d.h. Option 1 und 2), dynamisch auf den Basisprozess angewendet. An den Verzweigungsknoten kann zur Laufzeit jeweils entschieden werden, ob die Änderungen „*`DELETE` *Aktivität* `D`*" und „*`INSERT` *Aktivität* `F`*" tatsächlich vorgenommen werden sollen. Kontextänderungen führen nicht zum Abbruch der laufenden Prozessinstanz, sondern werden ab dem Zeitpunkt der Änderung im Kontextmodell für nachfolgende Verzweigungsknoten mit Kontextbedingungen berücksichtigt.*
Option 3 wird in Abbildung 7.2 nicht dynamisch auf den Basisprozess angewendet, da der Modus der Kontextvariable `Marke`*, auf der die Kontextbedingung der Option basiert, statisch definiert ist. Das heißt, der Wert dieser Variable wird sich zur Laufzeit nicht mehr ändern, und wir wissen somit bereits zum Zeitpunkt der Konfiguration, ob die Option angewendet werden soll oder nicht. Entsprechend resultiert eine Prozessfamilie mit zwei Varianten.*

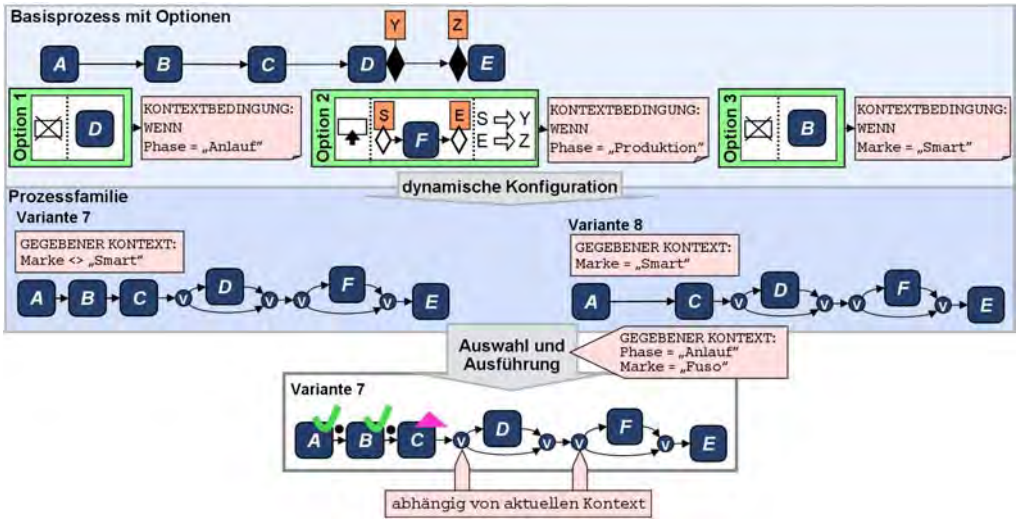


Abbildung 7.2: Ein Prozessmodell zur Abbildung multipler Prozessvarianten

Es ist nicht unser Ziel, alle Varianten eines Prozesstyps in einem Modell abzubilden. Stattdessen sollen nur solche Optionen integriert werden, die infolge dynamischer Kontextänderungen ggf. wirksam werden sollen (vgl. Beispiel 7.3). Wir unterscheiden entsprechend zwischen *statischen Optionen*, deren Anwendung auf den Basisprozess zum Zeitpunkt der Konfiguration bereits feststeht, sowie *dynamischen Optionen*, über deren Anwendung erst zur Laufzeit entschieden wird. Daraus resultieren folgende Fragestellungen:

- Wie lassen sich statische bzw. dynamische Optionen eindeutig identifizieren?
- Wie muss die statische Anwendung von Optionen auf den Basisprozess (vgl. Abschnitt 5.4) angepasst werden, so dass wir zur Laufzeit den aktuellen Kontext zur Bewertung einer Kontextbedingung berücksichtigen können und basierend auf diesem Ergebnis über die Anwendung von dynamischen Optionen entscheiden können?

7.2.2.1 Identifikation statischer und dynamischer Optionen

Zur Unterscheidung statischer und dynamischer Optionen müssen wir ihre Kontextbedingungen bewerten. Basiert die Kontextbedingung einer Option ausschließlich auf statischen Kontextvariablen, wird sie als statisch definiert. Der Umkehrschluss gilt hier jedoch nicht, d.h. Optionen, deren Kontextbedingung auf mindestens einer dynamischen Kontextvariable basiert, sind nicht notwendigerweise dynamisch (vgl. Beispiel 7.4).

Beispiel 7.4 (Identifikation statischer und dynamischer Optionen) *Tabelle 7.3 zeigt verschiedene Optionen und ihre Kontextbedingungen. Sie basieren auf dem Kontextmodell der Produktentwicklung (vgl. Tabelle 7.2). Unter der Annahme, dass eine Option statisch ist, wenn ihre Kontextbedingung ausschließlich auf statischen Kontextvariablen basiert, werden die Optionen 1 und 2 als statisch definiert. Da für alle anderen Optionen jeweils eine dynamische Kontextvariable in der Kontextbedingung verwendet wird, werden diese entsprechend als dynamisch definiert. Dieser Ansatz ist prinzipiell möglich und algorithmisch einfach zu realisieren, allerdings erhalten wir eine Vielzahl von dynamischen Optionen, die nicht wirklich dynamisch sind. Zum Beispiel können wir die Anwendung von Option 5 im vorliegenden Kontext* <`Fahrzeugtyp, PKW`> *bereits zum Zeitpunkt der Konfiguration bestimmen, da die Kontextbedingung erfüllt ist, unabhängig davon, welche Änderungen an der Kontextvariable* `Phase` *zur Laufzeit vorgenommen werden.*

Tabelle 7.3: Identifikation statischer und dynamischer Optionen

Nr.	Kontextbedingung	statische Variablen	dynamische Variablen	Modus
Option 1	`Marke ∈ {Fuso, Smart}`	ja	nein	statisch
Option 2	`Fahrzeugtyp = Bus`	ja	nein	statisch
Option 3	`Phase = Anlauf`	nein	ja	dynamisch
Option 4	`Fahrzeugtyp = PKW UND Phase = Anlauf`	ja	ja	dynamisch
Option 5	`Fahrzeugtyp = PKW ODER Phase = Anlauf`	ja	ja	dynamisch

In Provop klassifizieren wir Optionen als dynamisch, wenn ihre Kontextbedingung in der gegebenen Kontextbeschreibung von den Werten dynamischer Kontextvariablen abhängig ist. Um dies festzustellen, müssen wir für die dynamischen Kontextvariablen alle Werte ihres spezifischen Wertebereichs durchspielen und auf diese Weise alle Kontextbeschreibungen simulieren, welche sich durch Kontextänderungen zur Laufzeit ergeben können. Anschließend müssen wir für jede Option prüfen, ob ihre Kontextbedingung in allen simulierten Kontextbeschreibungen erfüllt ist (d.h., die Option ist statisch), in mindestens einer Kontextbeschreibung, aber nicht in allen simulierten Kontextbeschreibungen gültig ist (d.h. die Option ist dynamisch), oder ob sie nie erfüllt ist (d.h. die Option ist nicht relevant und kann auch nach dynamischen Kontextänderungen nicht relevant werden). Als Ergebnis erhalten wir dann die Menge aller statischen und dynamischen Optionen.

Beispiel 7.5 (Identifikation dynamischer Optionen) *Tabelle 7.4 zeigt ein Beispiel für die Bestimmung statischer und dynamischer Optionen. Wir verwenden die Funktion* `calculateDynamicOptions` *(siehe Anhang A), welche im vorliegenden Beispiel für die gegebene Kontextbeschreibung* `(K1)` *durch Fixieren der Werte statischer und Durchspielen der Werte dynamischer Kontextvariablen drei weitere Kontextbeschreibungen (K2, K3 und K4) erstellt. Diese simulieren alle möglichen Kontextbeschreibungen, die zur Laufzeit durch Kontextänderungen auftreten können. Für jede modellierte Option ist angegeben, ob ihre Kontextbedingung in der betrachteten Kontextbeschreibung erfüllt ist. Auf Basis der Variablen* `alwaysUsed` *und* `neverUsed` *(mit möglichen Werten* `(T)rue` *und* `(F)alse`*) können wir nun den Modus der Option bestimmen.*

Tabelle 7.4: Identifikation dynamischer Optionen

Gegebene Kontextbeschreibung: `K1: <Fahrzeugtyp, PKW>, <Marke, Smart>, <Phase, Anlauf>`								

Simulierte Kontextbeschreibungen:
`K2: <Fahrzeugtyp, PKW>, <Marke, Smart>, <Phase, Vorentwicklung>`
`K3: <Fahrzeugtyp, PKW>, <Marke, Smart>, <Phase, Entwicklung>`
`K4: <Fahrzeugtyp, PKW>, <Marke, Smart>, <Phase, Produktion>`								

Nr.	**Kontextbedingung**	**K1**	**K2**	**K3**	**K4**	**always-Used**	**never-Used**	**Modus**
Option 1	`Marke ∈ {Fuso, Smart}`	ja	ja	ja	ja	T	F	statisch
Option 2	`Fahrzeugtyp = Bus`	nein	nein	nein	nein	F	T	statisch [nicht relevant]
Option 3	`Phase = Anlauf`	ja	nein	nein	nein	F	F	dynamisch
Option 4	`Fahrzeugtyp = PKW UND Phase = Anlauf`	ja	nein	nein	nein	F	F	dynamisch
Option 5	`Fahrzeugtyp = PKW ODER Phase = Anlauf`	ja	ja	ja	ja	T	F	statisch

7.2.2.2 Anwendung dynamischer Optionen

Beispiel 7.3 erläutert, wie die Anwendung dynamischer Optionen mit Hilfe bedingter Verzweigungen realisiert werden kann (vgl. Anforderung 7.2). Bei Einführung der `INSERT`-Operation (vgl. Abschnitt 5.4.1) haben wir bereits motiviert, dass die Semantik normaler `ORJoin`-Knoten bei der Synchronisation eingefügter Prozessfragmente mit dem Basisprozess nicht geeignet ist. Dies gilt auch für den Fall der dynamischen Konfiguration. Wir verwenden entsprechend auch hier wieder `OPSplit`- und `OPJoin`-Knoten. Im Folgenden skizzieren wir die dynamische Anwendung der Provop-Änderungsoperationen.

Dynamische Anwendung der `INSERT`-Operation

Wie bei der statisch angewandten `INSERT`-Operation, realisieren wir auch im dynamischen Fall ein paralleles Einfügen. Dazu identifizieren wir zunächst die benötigten Ersetzungskonstrukte, die an den Aufsetzpunkten in den Basisprozess eingefügt werden sollen. Im Unterschied zur statischen `INSERT`-Operation, wird nun aber an die ausgehende Operationskante des `OPSplit`-Knotens die Kontextbedingung der Option annotiert. Die dynamisch eingefügten Prozessfragmente werden zusätzlich ausgeführt, wenn die Kontextbedingung der Option zur Laufzeit erfüllt ist. Beispiel 7.6 erläutert dies.

Beispiel 7.6 (Dynamische Anwendung der `INSERT`-Operation) *Abbildung 7.3a zeigt ein Beispiel für die dynamische Anwendung der `INSERT`-Operation. Nachdem wir das enstprechende Ersetzungskonstrukt identifiziert haben, werden die Kanten des Basisprozesses mit diesem Konstrukt neu verbunden. Die Kontextbedingung von Option 1 (`CtxtRule1`) wird danach als Kantenbedingung auf die Operationskante übertragen. Zur Laufzeit kann nun am `OPSplit`-Knoten über die Anwendung der Option und somit die Ausführung von Aktivität `D` auf Basis aktueller Werte der Kontextvariablen entschieden werden.*

Dynamische Anwendung der `DELETE`-Operation

Zum dynamischen Auslassen eines Fragments des Basisprozesses fügen wir einen leeren Pfad in den Basisprozess ein.[1] Zu diesem Zweck bestimmen wir zunächst die entsprechenden Ersetzungskonstrukte. Wird der zu löschende Bereich nicht durch Aufsetzpunkte sondern Prozesselement-IDs spezifiziert, werden jeweils Ein- und Ausgänge des auszulassenden Prozessfragments referenziert. Die Kontextbedingung der Option wird an der eingehenden Kante des dynamisch auszulassenden Fragments des Basisprozesses annotiert. Dieses Fragment wird entsprechend nur dann ausgeführt, wenn die Kontext- bzw. Kantenbedingung nicht zutrifft, d.h. kein Löschen vorgenommen werden soll. Beispiel 7.7 erläutert dies.

Beispiel 7.7 (Dynamische Anwendung der `DELETE`-Operation) *Abbildung 7.3b zeigt eine `DELETE`-Operation, die dynamisch auf den Basisprozess angewendet wird. Dazu werden ein leerer Pfad eingefügt und die Ausführung der auszulassenden Aktivität `B` abhängig von der negierten Kontextbedingung von Option 2 beschrieben (d.h. `NOT CtxtRule2`). Die Ein- und Ausgangskanten von `B` werden als Operationskanten typisiert.*

[1] In Abschnitt 5.4.2 haben wir zwischen Kontrollfluss-erhaltenden und -löschenden `DELETE`-Operationen unterschieden. Letztere sind bei einer dynamischen Anwendung nicht sinnvoll, da sie zur Laufzeit inkorrekte Prozessmodelle mit Lücken mit sich bringen würden. Wir betrachten diesen Fall daher nicht weiter.

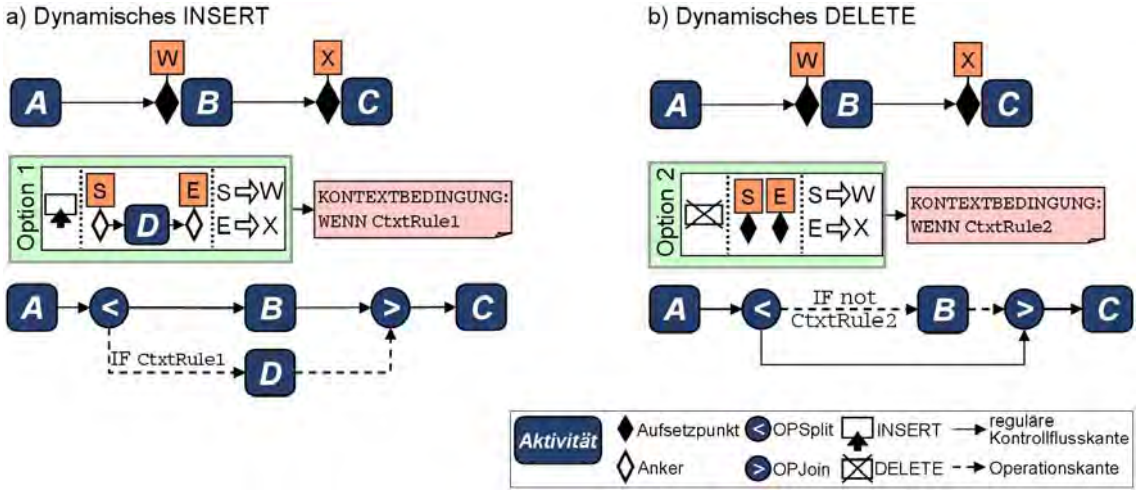


Abbildung 7.3: Dynamische Anwendung der Änderungsoperationen a) INSERT und b) DELETE

Dynamische Anwendung der `MOVE`-Operation

Bei dynamischer Anwendung der `MOVE`-Operation wird das zu verschiebende Prozessfragment durch eine dynamische `DELETE`-Operation entfernt und eine Kopie des Prozessfragments (d.h. ein Klon) an der neuen Position im Basisprozess mittels einer dynamischen `INSERT`-Operation eingefügt.[2] Dadurch können gleiche Prozessfragmente mehrfach in einem Prozessmodell vorkommen, nämlich an der alten und der neuen Position des Fragments im Basisprozess. Deshalb ist eine eindeutige Identifikation der Prozesselemente ggf. nicht mehr möglich. In solchen Fällen verwenden wir deshalb zusätzlich *Sub-IDs*, welche die eindeutige Unterscheidung zwischen Original und Kopie erlauben (vgl. Beispiel 7.8). Wir werden später zeigen, wie Änderungen, die sich auf die ID eines „geklonten" Prozesselements beziehen, korrekt umgesetzt werden können.

Beispiel 7.8 (Dynamische Anwendung der `MOVE`-Operation) *Abbildung 7.4a zeigt die dynamische Anwendung der `MOVE`-Operation. Aktivität `C` kann dann vor oder nach Aktivität `B` ausgeführt werden, abhängig von der Gültigkeit von `CtxtRule3` im gegebenen Kontext. Aktivität `C` ist mehrfach im Ergebnismodell vorhanden. Die Kopie der Aktivität an der neuen Position und das Original an der ursprünglichen können durch Sub-ID unterschieden werden.*

Dynamische Anwendung der `MODIFY`-Operation

Die dynamische Anwendung der `MODIFY`-Operation wird durch Einfügen einer entsprechend geänderten Kopie des betreffenden Prozesselements, alternativ zu seinem Original im Basisprozess, realisiert. Zur eindeutigen Identifikation wird, wie bei der statischen `MODIFY`-Operation, eine Sub-ID verwendet. Beispiel 7.9 erläutert dieses Vorgehen.

Beispiel 7.9 (Dynamische Anwendung der `MODIFY`-Operation) *Abbildung 7.4b zeigt ein Beispiel für die dynamische Anwendung der `MODIFY`-Operation. Die geänderte Aktivität `B` – mit Sub-ID 1 – wird parallel zu ihrem Original in den Basisprozess eingefügt. Zur Laufzeit wird die geänderte Aktivität nur dann ausgeführt, wenn die Kontextbedingung von Option 4 (d.h. `CtxtRule4`) im gegebenen Kontext gültig ist. Andernfalls wird die originale Aktivität ausgeführt. Die ein- und ausgehenden Kanten der (un-)veränderten Aktivität `B` werden als Operationskanten typisiert.*

[2] Wir werden später noch zeigen, wie wir die Atomarität einer dynamisch angewandten `MOVE`-Operation zur Laufzeit sicherstellen.

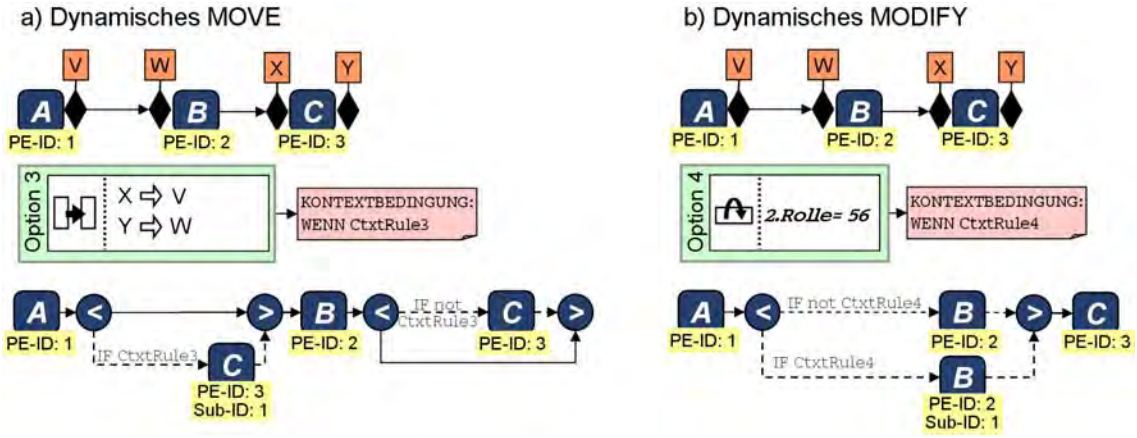


Abbildung 7.4: Dynamische Anwendung der Änderungsoperationen a) MOVE und b) MODIFY

Für eine korrekte Versorgung des `OPJoin`-Knotens mit regulären Kanten[3], muss bei der dynamischen Anwendung der `MODIFY`-Operation eine reguläre Kante zwischen `OPSplit`- und `OPJoin`-Knoten erzeugt werden. Dies ist nicht unbedingt wünschenswert, da wir aufgrund der Kantenbedingungen davon ausgehen können, dass immer genau ein Pfad durchlaufen wird (vgl. Abbildung 7.4b). Ursprünglich haben wir die Versorgung des `OPJoin`-Knotens mit einer regulären Kante gefordert, um sicherzustellen, dass die Ausführung des Basisprozesses nicht durch eine `INSERT`-Operation abgebrochen werden kann (vgl. Abschnitt 5.4.1). Da wir durch die Wahl der Ersetzungskonstrukte bei der statischen `INSERT`-Operation davon ausgehen können, dass eine korrekte Synchronisation am `OPJoin`-Knoten stattfindet, ist eine Änderung der Definition dieses Knotens möglich. Dies gilt auch für dynamische `DELETE`- und `MOVE`-Operationen. Wir ändern daher die Semantik des `OPJoin`-Knotens dahingehend ab, dass dessen Eingangs-Bedingung genau dann `true` ist, wenn alle eingehenden Kanten im Zustand `true_signaled` oder `false_signaled` sind und wenn sich alle (ggf. vorhandenen) regulären Kontrollfluss-Kanten, d.h. Kanten vom Typ `ControlFlow`, im Zustand `true_signaled` befinden. Ist keine reguläre Kontrollfluss-Kante gegeben, muss mindestens eine Operationskante im Zustand `true_signaled` sein.

Generell ist bei der Modellierung der Änderungsoperationen keine Unterscheidung zwischen dynamischen und statisch anzuwendenden Optionen erforderlich. Sie erfolgt stattdessen basierend auf der vorherigen Analyse der Kontextbedingung einer Option und der gegebenen Kontextbeschreibung.

Handhabung von Prozesselementen mit gleichem Bezeichner

Die dynamische Anwendung der `MOVE`- und `MODIFY`-Operation führt jeweils zu mehreren Prozesselementen mit gleichem Bezeichner bzw. Prozesselement-ID (vgl. Abbildung 7.4b). Dieses mehrfache Auftreten eines Elements im Prozessmodell hat Auswirkungen auf nachfolgend angewendete Änderungsoperationen. So müssen die ID-basierten Änderungsoperationen `DELETE` und `MODIFY` auf jedes Prozesselement mit der gegebenen Prozesselement-ID angewendet werden.

Eine weitere Konsequenz aus der dynamischen Anwendung von `MOVE`- und `MODIFY`-Operationen sind „redundante“ Aufsetzpunkte. Sie entstehen, wenn ein Prozessfragment mit Aufsetzpunkten (dynamisch) verschoben bzw. geändert wird. Das mehrfache Auftreten

[3]Siehe hierzu Abschnitt 5.4.1.

von Aufsetzpunkten mit gleichem Bezeichner wirkt sich auf die aufsetzpunktbasierten Änderungsoperationstypen `INSERT`, `DELETE` und `MOVE` aus. Beispiel 7.10 zeigt das Vorgehen in diesem Fall:

Beispiel 7.10 (Handhabung von Prozesselementen mit gleichem Bezeichner) *Abbildung 7.5 zeigt die dynamische Anwendung zweier Optionen: Option 1 verschiebt Aktivität* `C` *und ändert ein Attribut von Aktivität* `B`*. Anschließend fügt Option 2 die Aktivität* `F` *ein, wobei sowohl der Startanker* `S` *als auch der Endeanker* `E` *jeweils auf redundante Aufsetzpunkte abgebildet werden. Dies führt dazu, dass je zwei* `OPSplit`*- und* `OPJoin`*-Knoten an den Ein- und Ausgängen eingefügt werden müssen. Hieraus ergibt sich die Frage, wie die ein- bzw. ausgehenden Operationskanten der* `OPSplit`*- und* `OPJoin`*-Knoten korrekt zusammengeführt bzw. aufgesplittet werden können. Zur Zusammenführung verwenden wir einen zusätzlich eingefügten* `OPJoin`*-Knoten (vgl. Abbildung 7.5). Auf diese Weise wird das eingefügte Prozessfragment (d.h. Aktivität* `F`*) erst nach Beendigung von Aktivität* `A` *ausgeführt. Da die Ausführung von* `F` *von der Kontextbedingung der übergeordneten Option abhängig ist, wird die Kontextbedingung als Kantenbedingung der ausgehenden Operationskanten der* `OPSplit`*-Knoten übernommen. Die Aufsplittung der Operationskanten erfolgt durch einen* `OPSplit`*-Knoten. Dies führt dazu, dass Aktivität* `C` *erst dann ausgeführt werden kann, wenn Aktivität* `F` *beendet oder ausgelassen wurde. Dies ist unabhängig davon, wann Aktivität* `C` *im Ergebnismodell tatsächlich ausgeführt werden soll (d.h. vor oder nach Aktivität* `D`*). Desweiteren können alle nachfolgenden Aktivitäten (d.h.* `C`*,* `D` *und* `E`*) erst ausgeführt werden, nachdem Aktivität* `F` *beendet oder ausgelassen worden ist.*

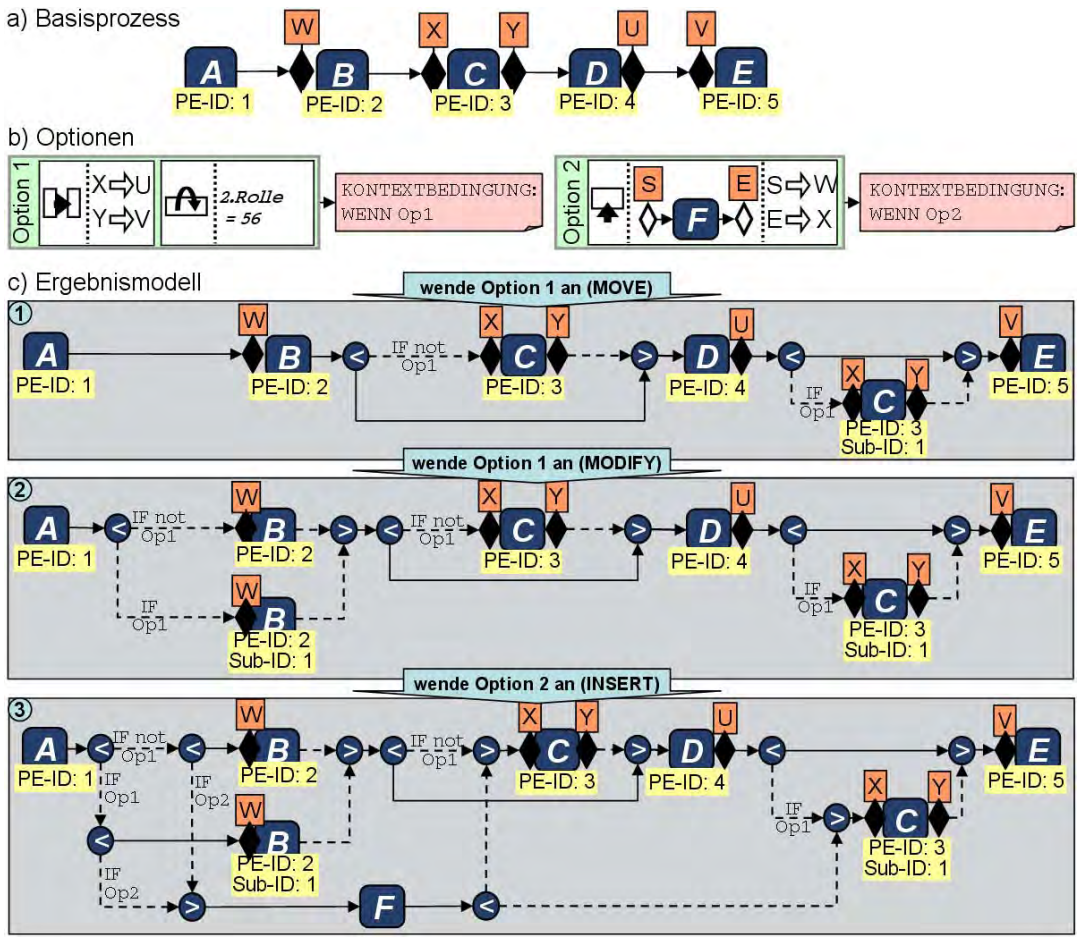


Abbildung 7.5: Dynamische Anwendung der Optionen aus (b) auf den Basisprozess aus (a) und schrittweise Ableitung des Ergebnismodells in (c)

7.2.2.3 Bewahrung der Atomarität bei Anwendung dynamischer Optionen

Bisher haben wir gezeigt, wie über die dynamische Anwendung einzelner Änderungsoperationen auf den Basisprozess zur Laufzeit entschieden wird. Zur dynamischen Anwendung einer ganzen Option, die mehrere Änderungsoperationen umfasst, sollte das von dieser Option veränderte Fragment des Basisprozesses durch ein entsprechendes `OP`-Konstrukt umschlossen werden. Problem dieser Lösung ist, dass „unschöne" Modellredundanzen resultieren können. Dies verdeutlicht Beispiel 7.11. Desweiteren wird ein Ergebnismodell bei vielen dynamisch und überlappend im Basisprozess wirkenden Optionen unübersichtlich.

Beispiel 7.11 (Dynamische Anwendung einer Option) *Abbildung 7.6 zeigt, wie die Änderungsoperationen einer Option mit Hilfe eines `OP`-Konstrukts dynamisch auf den Basisprozess angewendet werden können. Bei diesem Ansatz treten in Verbindung mit `INSERT`- und `DELETE`-Operationen dann Aktivitäten mehrfach im Ergebnismodell auf (vgl. Aktivitäten `C` und `D` in Abbildung 7.6c).*

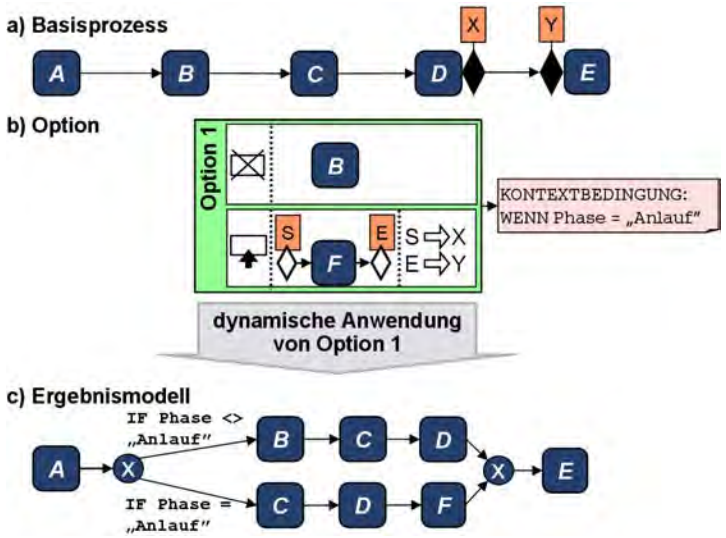


Abbildung 7.6: Dynamische Anwendung einer Option

Aus den oben genannten Gründen verfolgen wir in Provop den Ansatz, die Änderungsoperationen einer Option einzeln dynamisch einzubinden. Sollte sich der Kontext während der Ausführung des Fragments zwischen den `OPSplit`-Knoten zweier Änderungsoperationen einer Option allerdings so ändern, dass ihre Kontextbedingung nicht mehr erfüllt ist, führt dieses Vorgehen dazu, dass nicht alle Änderungsoperationen der Option angewendet werden. Das heißt, die Atomarität der Option würde verletzt werden. Dies ist jedoch in keinem Fall erwünscht, da der Variantenmodellierer eben genau solche Änderungsoperationen in einer Option bündelt, welche aufgrund semantischer und struktureller Abhängigkeiten zwingend gemeinsam wirksam werden sollen. Beispiel 7.12 illustriert dies:

Beispiel 7.12 (Unvollständige Anwendung von Änderungsoperationen einer Option) *Abbildung 7.7 zeigt ein Ergebnismodell, welches aus der dynamischen Anwendung zweier Optionen resultiert. Ist zur Laufzeit bei Erreichen des ersten `OPSplit`-Knotens der Option 1 die Kontextbedingung `Rule1` erfüllt, wird die eingefügte Aktivität `I` ausgeführt. Wir nehmen nun an, dass sich der Kontext im weiteren Verlauf derart ändert, dass bei Erreichen des zweiten `OPSplit`-Knotens von Option 1 die Kontextbedingung nicht mehr erfüllt ist. Berücksichtigen wir ausschließlich die Kontextbedingung der Option, wird die modifizierte Aktivität `C` (d.h. Aktivität `C'`) bzw. die ebenfalls*

durch Option 1 eingefügte Aktivität `H` *ausgelassen. Aufgrund semantischer Abhängigkeiten zwischen diesen Änderungen ist eine solche Verletzung der Atomarität von Option 1 aber unerwünscht.*
Ein zweites Beispiel bietet Option 2. Findet hier eine Kontextänderung zwischen den beiden `OPSplit`*-Knoten der dynamisch angewandten* `MOVE`*-Operation statt, wird die verschobene Aktivität* `F` *ggf. zweimal ausgeführt, nämlich an der alten und an der neuen Position im Basisprozess. Dies verletzt nicht nur die Atomarität von Option 2, sondern auch die Atomarität bzw. Semantik der dynamischen* `MOVE`*-Operation.*

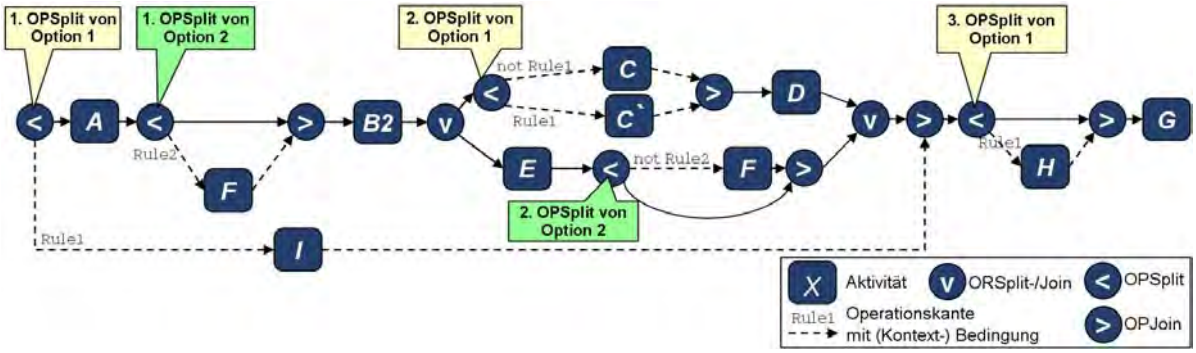


Abbildung 7.7: Prozessmodell mit dynamisch angewandten Änderungsoperationen

Um der Forderung nach atomarer Anwendung von Optionen nachzukommen (vgl. Anforderung 7.3), ist es erforderlich, die Anwendungsentscheidung für alle Änderungsoperationen einer Option gleich zu treffen. Wir führen dazu explizite Laufzeit-Zustände für Optionen ein (siehe Abbildung 7.8). Der initiale Zustand einer Option ist `undefined`. Er wird nach Auswertung ihrer Kontextbedingung und Optionsbeziehungen auf `skipped` gesetzt, wenn die Option nicht angewendet werden soll bzw. darf. Andernfalls wird ihr Zustand auf `applied` gesetzt. Ein Rücksetzen des Zustands zu `undefined` ist in keinem Fall möglich. Zur Dokumentation des Zustandes einer Option definieren wir die Variable `state`. Beispiel 7.13 erläutert diese Lösung.

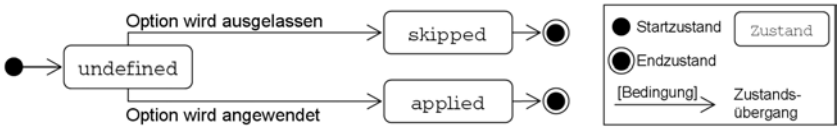


Abbildung 7.8: Zustandsdiagramm für Optionen

Beispiel 7.13 (Gewährleistung der Atomarität einer Option) *Abbildung 7.9 zeigt ein Beispiel für ein Ergebnismodell, welches aus der dynamischen Anwendung zweier Optionen resultiert. Die Kontextbedingungen an den ausgehenden Kanten werden durch entsprechende Bedingungen, basierend auf der Variable* `state` *der übergeordneten Option, formuliert. Nach Abfrage der Kontextbedingung am ersten* `OPSplit`*-Knoten von Option 1 legen wir ihren Zustand mit* `state = applied` *fest. Alle nachfolgenden* `OPSplit`*-Knoten dieser Option lesen nur noch diese Variable und führen keine Prüfung des Kontextes mehr durch. Für Option 2 ist die Kontextbedingung bisher noch nicht geprüft worden, d.h. es gilt* `state = undefined`*.*

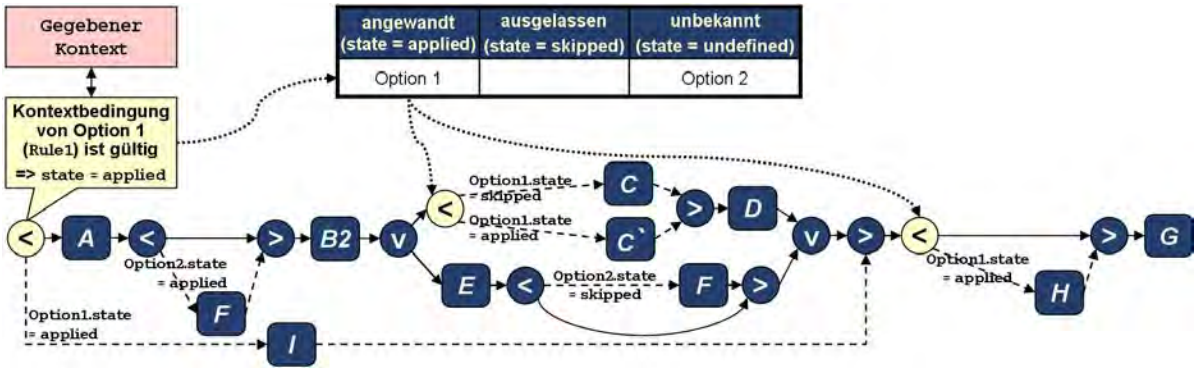


Abbildung 7.9: Gewährleistung der Atomarität von Optionen durch Variable `state`

7.2.3 Konflikte zwischen Optionsbeziehungen und Kontextabhängigkeit bei dynamischer Konfiguration

In Provop verwenden wir explizite Optionsbeziehungen, etwa Implikation und wechselseitigen Ausschluss, um semantische und strukturelle Abhängigkeiten zwischen Optionen abzubilden (vgl. Abschnitt 6.3). Bei der statischen Konfiguration berücksichtigen wir diese Optionsbeziehungen, indem wir die Kompatibilität einer ausgewählten Menge von Optionen mit allen Optionsbeziehungen prüfen. Nur wenn eine Optionsmenge hinsichtlich definierter Optionsbeziehungen kompatibel ist, wenden wir sie auf den Basisprozess an.

Durch dynamische Kontextvariablen ergibt sich nun eine neue Problemstellung: Die dynamische Anwendung von Optionen zur Laufzeit, abhängig von den aktuellen Werten der Kontextvariablen, verändert die Optionsmenge dynamisch. Dies kann dazu führen, dass die selektierten Optionen nicht kompatibel zueinander sind, d.h. mindestens eine Optionsbeziehung verletzt wird. Es resultiert somit ein Konflikt zwischen dem Kontextbezug der Optionen und den Einschränkungen für ihre Verwendung. Beispiel 7.14 zeigt diesen Fall.

Beispiel 7.14 (Konflikte zwischen Optionsbeziehungen und Kontextabhängigkeit)
Abbildung 7.10a zeigt, wie für den gegebenen Kontext eine Optionsmenge bestimmt wird. Diese ist hinsichtlich der definierten Optionsbeziehungen korrekt. Eine dynamische Änderung des Kontextes, d.h. des Wertes einer dynamischen Kontextvariable, führt im Beispiel dazu, dass weitere Optionen verwendet werden sollen. Die resultierende Optionsmenge ist weiterhin verträglich mit den definierten Beziehungen. Eine erneute Kontextänderung (vgl. Abbildung 7.10c) führt nun allerdings dazu, dass Option 1 nicht verwendet werden soll. Diese Änderung tritt erst ein, nachdem Optionen 2 und 3 bereits auf den Basisprozess angewendet worden sind. Option 3 impliziert jedoch die Anwendung von Option 1. Das heißt, die Optionsmenge ist nicht verträglich mit den definierten Optionsbeziehungen. Der gegebene Kontext und die definierten Optionsbeziehungen stehen somit im Widerspruch zueinander.

Derartige Konflikte können zur Laufzeit prinzipiell durch manuelle Eingriffe des Endanwenders aufgelöst werden. Allerdings sollten solche Eingriffe zur Laufzeit – wenn möglich – vermieden werden. Eine Abhilfe besteht darin, Optionsbeziehungen höher zu priorisieren als Kontextabhängigkeiten. In diesem Fall müssen die Optionen ggf. auch dann angewendet werden, wenn ihre Kontextbedingung nicht erfüllt ist oder aber der umgekehrte Fall tritt ein (z.B. bei wechselseitigem Ausschluss). Beispiel 7.15 beschreibt ein Szenario, in dem ein solches Vorgehen sinnvoll ist.

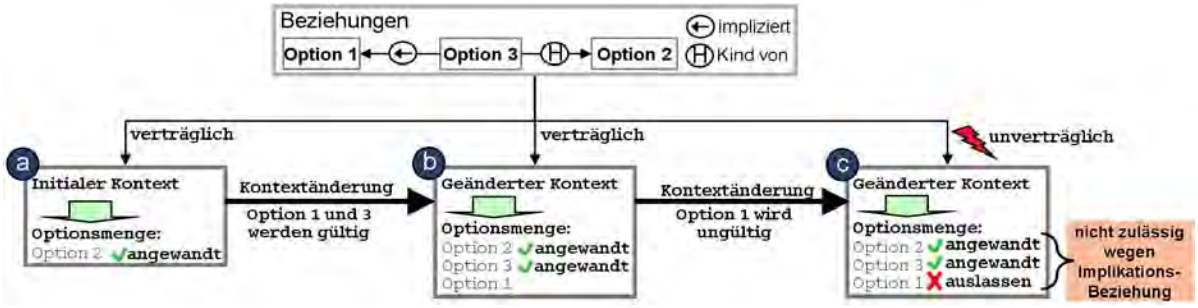


Abbildung 7.10: Verträglichkeit von Optionen nach Kontextänderungen

Beispiel 7.15 (Auflösung von Konflikten durch Priorisierung der Optionsbeziehungen) *Beim Änderungsmanagement (vgl. Abschnitt 2.1) wird eine Risikobewertung für die beantragte Produktänderung durchgeführt. Fällt diese initiale Bewertung während der Ausführung von Aktivität* `1` *hoch aus (d.h. für die dynamische Kontextvariable Risiko gilt:* `Risiko = „hoch“`*), soll der Projektleiter kontinuierlich über die Zwischenergebnisse der Stellungnahmen informiert werden. Fällt die Bewertung des Risikos niedrig aus (*`Risiko = „niedrig“`*), soll eine Zusammenfassung im Anschluss an die eingegangenen Stellungnahmen erfolgen. Abbildung 7.11 zeigt dieses Szenario: Optionen 1 und 2 fügen jeweils die Aktivität „Information an Projektleiter“ ein, allerdings an unterschiedlichen Positionen im Basisprozess. Durch semantische Überlappung der eingefügten Aktivitäten wird ein wechselseitiger Ausschluss zwischen den Optionen modelliert. War die initiale Risikobewertung z.B. „hoch“, findet zur Laufzeit während der Ausführung der Aktivitäten* `1a, 2a, 2b` *und* `2c` *eine Neubewertung des Risikos als „niedrig“ statt. Dies hätte zur Folge, dass auch Aktivität* `3a` *ausgeführt werden kann. Eine solche Doppelarbeit ist nicht erwünscht. Stattdessen sollte der wechselseitige Ausschluss beachtet werden und Aktivität* `3a` *trotz gültiger Kontextbedingung nicht ausgeführt werden. Dies kann durch höhere Priorisierung der Optionsbeziehungen gegenüber Kontextabhängigkeit erreicht werden.*

Die Forderung, dass immer alle Optionsbeziehungen zur Laufzeit erfüllt werden, ist allerdings sehr restriktiv. So kann es wünschenswert sein, dass solche semantischen Beziehungen zwar beim Prozessentwurf eingehalten werden, ihre strikte Einhaltung zur Laufzeit, insbesondere nach Kontextänderungen aber nicht erforderlich ist. Beispiel 7.16 beschreibt einen solchen Anwendungsfall.

Beispiel 7.16 (Ignorieren der Inkompatibilität einer Optionsmenge zur Laufzeit) *Im Änderungsmanagement werden, neben Stellungnahmen verschiedener Bereiche, auch Kostenschätzungen eingeholt. Abhängig von der Höhe dieser Schätzungen müssen andere Gremien ihre Zustimmung für eine beantragte Produktänderung geben. Abbildung 7.12 zeigt ein entsprechendes Szenario: Wenn die geschätzten Kosten einen bestimmten Betrag überschreiten, ist der sog. Steuerkreis (entspricht einem bestimmten Gremium) zu informieren. Dazu ist ein zusätzlicher Vorbereitungsschritt erforderlich. Entsprechend sind für den Basisprozess aus Abbildung 7.12a zwei Optionen modelliert, wobei die Option 1 (Ändern des Gremiums) die Option 2 (zusätzlicher Vorbereitungsschritt) impliziert. Die Kontextvariable* Kosten *ist dynamisch und wird zur Laufzeit auf einen Wert größer 100.000€ gesetzt. Entsprechend wird die Sitzung des Steuerkreises vorbereitet. Wir nehmen weiter an (d.h. im Verlauf der Ausführung von Aktivität* `4a`*), dass eine Korrektur der Kostenschätzung auf einen Betrag unter 100.000€ erfolgt. Die strikte Einhaltung der Implikationsbeziehung würde dazu führen, dass das Gremium unnötigerweise in den Genehmigungsprozess eingebunden wird. Dies ist i.d.R. aber nicht gewünscht.*

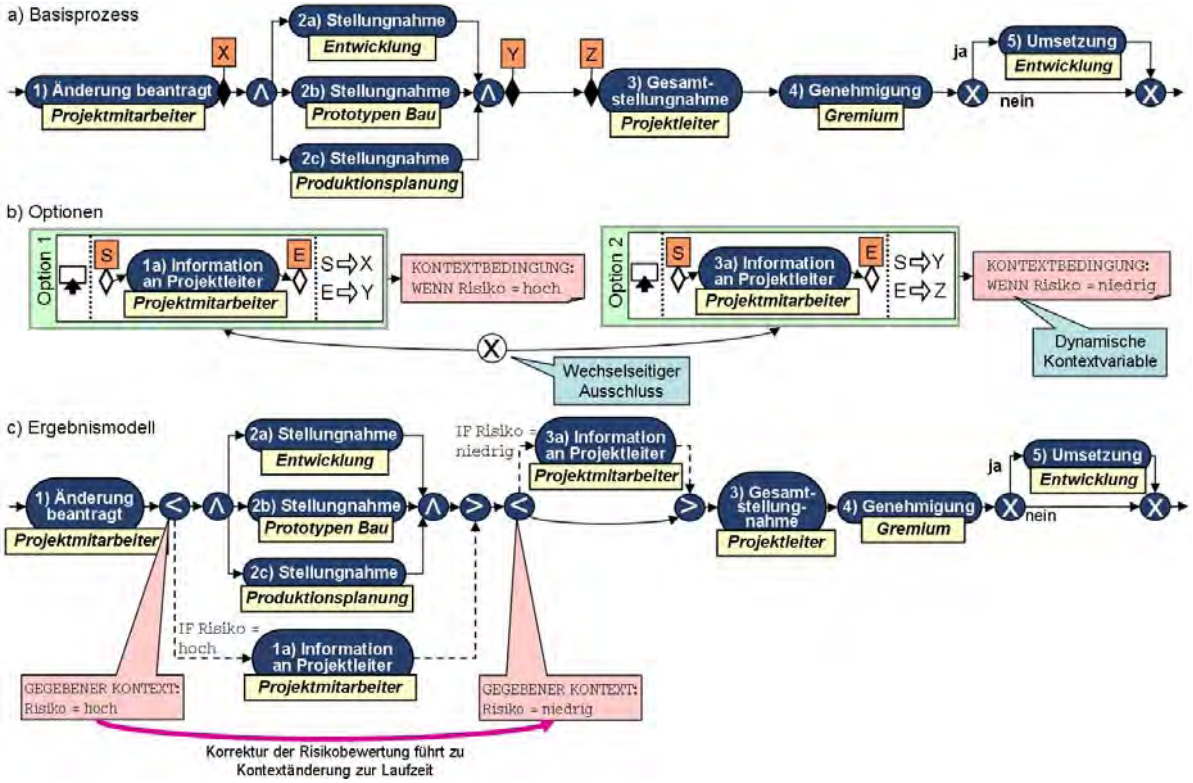


Abbildung 7.11: Auflösung von Konflikten durch Priorisierung der Optionsbeziehungen

Es ergeben sich somit zwei gegensätzliche Anwendungsfälle:

> **Anwendungsfall 1:** Zur Laufzeit muss die Kontextabhängigkeit von Optionen höher priorisiert sein als Optionsbeziehungen.
>
> **Anwendungsfall 2:** Zur Laufzeit muss die Einhaltung von Optionsbeziehungen höher priorisiert sein als die Kontextabhängigkeit von Optionen.

Um beide Anwendungsfälle abzudecken, geben wir dem Variantenmodellierer die Möglichkeit, wahlweise die Einhaltung der Optionsbeziehungen oder die Kontextabhängigkeit höher zu priorisieren (vgl. Anforderung 7.4). Dazu unterscheiden wir bei der Modellierung von Optionsbeziehungen zwischen *weichen* und *harten Optionsbeziehungen* (vgl. Anforderung 7.4). Diese definieren wir wie folgt: Eine weiche Optionsbeziehung bildet Anwendungsfall 1 ab. Dies bedeutet, dass während der Ausführung eines Prozessmodells weiche Optionsbeziehungen nicht mehr berücksichtigt werden. Ausschlaggebend für die Anwendung einer Option ist dann nur noch der Kontext bzw. die Gültigkeit der Kontextbedingung einer dynamisch angewandten Option. Harte Optionsbeziehungen dagegen bilden Anwendungsfall 2 ab. Sie werden, ebenso wie weiche Optionsbeziehungen, bei der statischen Konfiguration berücksichtigt. Hinzu kommt, dass sie auch zur Laufzeit von Prozessvarianten eingehalten werden müssen.[4]

Die Unterscheidung in weiche und harte Optionsbeziehungen kann für alle Beziehungstypen vorgenommen werden. Für die explizite Reihenfolgebeziehung von Optionen ergibt sich dadurch jedoch kein Unterschied. Unabhängig von ihrer Festlegung als weich oder hart, wird die Reihenfolgebeziehung nur zum Zeitpunkt der Konfiguration der Prozessvarianten

[4] Wir werden in Abschnitt 7.3.2 zeigen, wie wir die Kompatibilität von gemeinsam anzuwendenden Optionen unter den Aspekten weiche und harte Optionsbeziehungen sowie Kontextabhängigkeit zur Laufzeit sicherstellen.

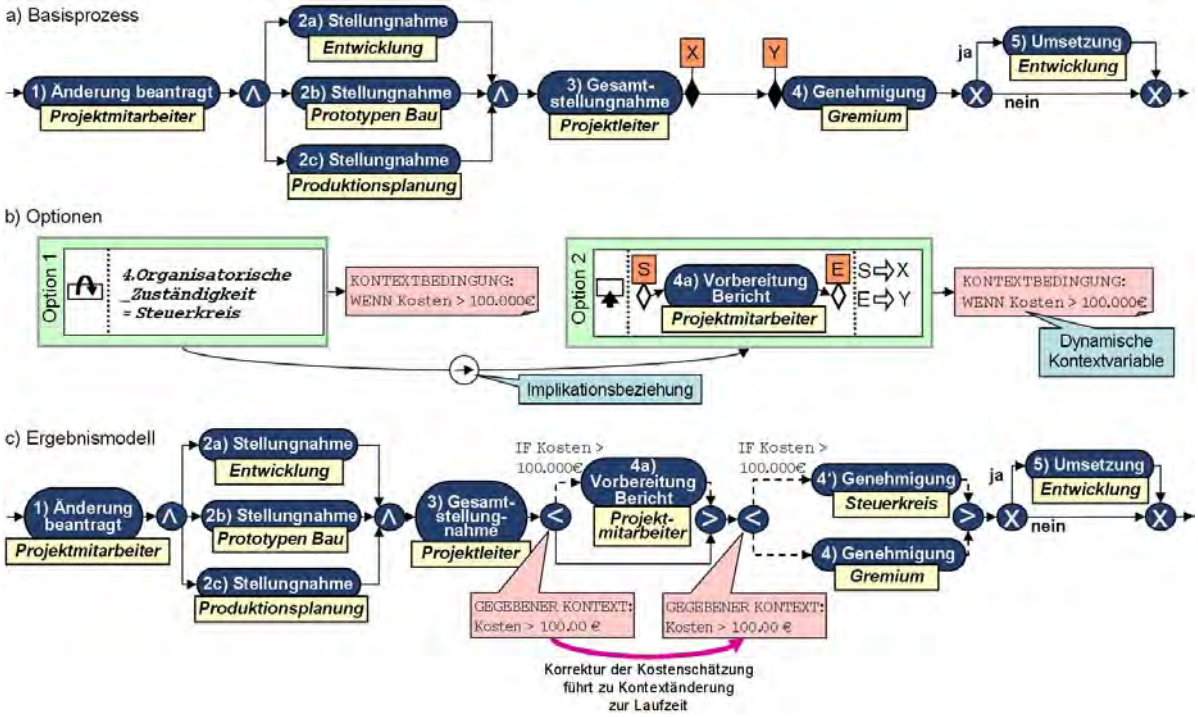
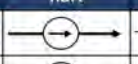
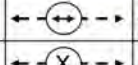
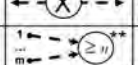
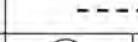
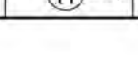
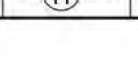


Abbildung 7.12: Ignorieren der Inkompatibilität einer Optionsmenge zur Laufzeit

berücksichtigt und spielt zur Laufzeit keine Rolle mehr. Die unterschiedliche Darstellung der weichen und harten Optionsbziehungen zeigt Tabelle 7.5.[5]

Tabelle 7.5: Darstellung weicher und harter Beziehungen

Beziehungstyp	Darstellung		Ausdruck
	weich	hart	
Implikation			$\neg(Option_x \wedge \neg Option_y)$
Wechselseitige Implikation			$\neg(Option_x \; xor \; Option_y)$
Wechselseitiger Ausschluss			$\neg(Option_x \wedge Option_y)$
Auswahl n-aus-m*			$\min_n(Option_1,...,Option_m)$
Reihenfolge			$Option_x \Rightarrow Option_y$
Hierarchie			$\neg(Option_x \wedge \neg Option_y) \wedge (Option_x \Rightarrow Option_y)$

* $n \leq m, n \in \mathbb{N}_0, m \in \mathbb{N}$
** bel. Relationssymbol

7.3 Gewährleistung der Korrektheit zur Laufzeit

Eine weitere Herausforderung dynamisch konfigurierter Prozessvarianten betrifft ihre korrekte Ausführung. Diese ist auch dann zu gewährleisten, wenn erst zur Laufzeit über die Anwendung von Optionen entschieden wird. Abschnitt 7.3.1 beschreibt zunächst, wie wir in Provop die Korrektheit einer Prozessfamilie im dynamischen Fall sicherstellen. Anschließend erläutert Abschnitt 7.3.2 welche zusätzlichen Prüfungen zur Laufzeit erforderlich werden, um die Einhaltung harter Optionsbeziehungen zu gewährleisten.

[5]Da die Reihenfolgebeziehung im Prinzip immer „weich“ ist, geben wir keine unterschiedlichen Darstellungen für diesen Beziehungstyp an.

7.3.1 Korrektheit einer Prozessfamilie im dynamischen Fall

Ein naheliegender Ansatz zur Gewährleistung der Korrektheit einer Prozessfamilie mit (teilweise) dynamisch konfigurierten Variantenmodellen, stellen Korrektheitsprüfungen zur Laufzeit dar. Dieser Ansatz ist jedoch nicht immer praktikabel, insbesondere wenn Endanwender zur Laufzeit nicht involviert werden sollen. In Provop garantieren wir Korrektheit daher bereits vor der Ausführung von Prozessvarianten (d.h. Vermeidung „böser Überraschungen“ zur Laufzeit). Dazu prüfen wir die Korrektheit der gesamten Prozessfamilie, einschließlich der Variantenmodelle, die durch dynamische Anwendung von Optionen, unter Berücksichtigung der Optionsbeziehungen, entstehen können. Auf diese Weise garantieren wir, dass Kontextänderungen zur Laufzeit niemals zu inkorrekten Prozessmodellen führen (vgl. Anforderung 7.5).

Die in Abschnitt 6.5.2 vorgestellte Korrektheitsprüfung berücksichtigt noch keine Kontextänderungen zur Laufzeit. Wir erweitern diese Prüfung deshalb entsprechend. Ein weiterer Schritt 7 prüft nun für jedes nach Schritt 6 der Korrektheitsprüfung ermittelte korrekte Prozessmodell, ob mindestens eine Option dynamisch angewendet wird, d.h. ob eine Kontextbedingung aufgrund von Änderungen dynamisch gültig oder ungültig werden kann. Abbildung 7.13 zeigt den Ablauf dieses Prüfschrittes: Für jede in Schritt 2 ermittelte Optionsmenge bestimmen wir zunächst die Menge der dynamischen Optionen (vgl. Anhang A). Ist diese nicht leer, müssen wir simulieren, welche Optionsmengen durch beliebige Kontextänderungen auftreten können. Zu diesem Zweck vereinen wir jede mögliche Teilmenge der dynamischen Optionen jeweils mit der Menge der statischen Optionen. Als Ergebnis erhalten wir eine Menge „simulierter“ Optionsmengen. Diese decken nun alle relevanten Fälle ab, d.h. alle Kontextbeschreibungen, die aufgrund der Dynamik der Kontextvariablen zur Laufzeit möglich sind.

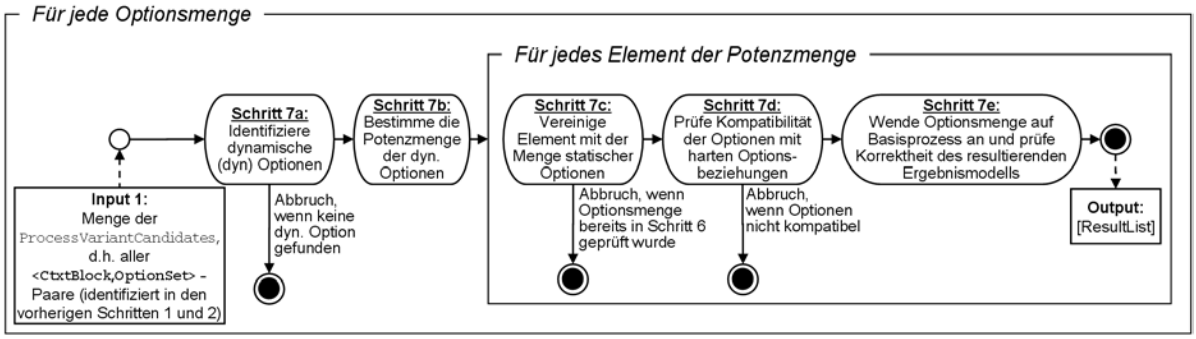


Abbildung 7.13: Ablauf Schritt 7 der Provop-Korrekheitsprüfung (UML-Aktivitätendiagramm)

Jede simulierte Optionsmenge wird auf ihre Verträglichkeit mit allen harten Optionsbeziehungen geprüft. Wird eine harte Optionsbeziehung verletzt, wird eine Warnung ausgegeben. Der Modellierer kann das Problem entweder beseitigen oder ignorieren. Letzteres ist nicht kritisch für die Ausführung, da wir zur Laufzeit verhindern, dass inkompatible Optionen angewendet werden und somit entsprechende Variantenmodelle nicht entstehen können. Auf diese Weise ist für die simulierte Optionsmenge keine Korrektheitsprüfung erforderlich. Die Korrektheitsprüfung wird daher mit der nächsten simulierten Optionsmenge fortgesetzt. Die Anwendung inkompatibler Optionen verhindern wir durch eine entsprechende Kompatibilitätsprüfung zur Laufzeit (vgl. Abschnitt 7.3.2).

Ist eine simulierte Optionsmenge mit den harten Beziehungen kompatibel, prüfen wir, ob die Optionsmenge bereits auf den Basisprozess angewendet und das dann erzeugte Ergebnismodell auf Korrektheit geprüft worden ist. Ist dies der Fall, kann mit der nächsten simulierten

Optionsmenge fortgefahren werden. Andernfalls werden an dieser Stelle die Schritte 4 bis 6 für die simulierte Optionsmenge wiederholt. Das heißt wir sortieren die Optionsmenge (Schritt 4), wenden sie auf den Basisprozess an (Schritt 5) und prüfen anschließend die Korrektheit des Ergebnismodells (Schritt 6). Beispiel 7.17 zeigt die Arbeitsweise von Schritt 7 exemplarisch.

Beispiel 7.17 (Arbeitsweise von Schritt 7 anhand des Szenarios aus Tabelle 7.4) *Für die gegebene Kontextbeschreibung*

{Fahrzeugtyp, PKW>, <Marke, Smart>, <Phase, Anlauf>}

können wir mit Hilfe der Funktion `calculateDynamicOptions` *(vgl. Anhang A) die in Tabelle 7.6 angegebenen, dynamisch und statisch anzuwendenden Optionen identifizieren. Option 2 ist in der betrachteten Kontextbeschreibung nicht relevant. Da die Menge der dynamischen Optionen nicht leer ist, kann die Prozessvariante zur Laufzeit rekonfiguriert werden. Um die Korrektheit der resultierenden Modelle zu gewährleisten, ist eine Korrektheitsprüfung durch Schritt 7 erforderlich. Wir erstellen daher zunächst die Potenzmenge der dynamischen Optionen wie in Tabelle 7.6 aufgeführt. Durch Vereinigung jeder möglichen Teilmenge der dynamischen Optionen mit der Menge der statischen Optionen, entstehen acht simulierte Optionsmengen (vgl. Tabelle 7.6). Für jede simulierte Optionsmenge ist die Kompatibilität mit harten Optionsbeziehungen zu prüfen. Wird eine solche Optionsbeziehung verletzt, ist der Variantenverantwortliche zu informieren. Andernfalls wird die Optionsmenge sortiert und auf den Basisprozess angewendet. Sind alle statischen und dynamischen Optionen erfolgreich anwendbar und sind die resultierenden Ergebnismodelle hinsichtlich des zugrunde liegenden Prozess-Metamodells korrekt, können wir die Korrektheit der Variantenmodelle zur Laufzeit garantieren.*

Tabelle 7.6: Simulierte Optionsmengen

staticOptions = {Option 1, Option 5}
dynamicOptions = {Option 3, Option 4}
getPowerSet(dynamicOptions) = {∅, {Option 3}, {Option 4}, {Option 3, Option 4}}
$simOptionSet_1$ = {Option 1, Option 5} $simOptionSet_2$ = {Option 1, Option 3, Option 5} $simOptionSet_3$ = {Option 1, Option 4, Option 5} $simOptionSet_4$ = {Option 1, Option 3, Option 4, Option 5}

Mit Hilfe von Schritt 7 können wir bereits zum Zeitpunkt der Konfiguration von Prozessvarianten die Korrektheit aller dynamisch konfigurierbaren Ergebnismodelle prüfen. Durch Berücksichtigung der Kontextabhängigkeiten und Optionsbeziehungen müssen wir nur diejenigen Ergebnismodelle prüfen, welche sich durch Kontextänderungen zur Laufzeit ergeben können. Durch Abfrage der Ergebnisliste können wir außerdem davon ausgehen, dass keine Optionsmenge wiederholt geprüft wird. Die Korrektheitsprüfung stellt sicher, dass alle Variantenmodelle, die unter Berücksichtigung des aktuellen Kontextes und der definierten Optionsbeziehungen abgeleitet werden können, konsistent sind. Um abzusichern, dass zur Laufzeit alle harten Optionsbeziehungen tatsächlich eingehalten werden, ist zusätzlich eine Überwachung und Steuerung der Auswahl dynamischer Optionen während der Ausführung der Variantenmodelle erforderlich (siehe Abschnitt 7.3.2).

7.3.2 Einhaltung von Optionsbeziehungen zur Laufzeit

Die Provop-Korrektheitsprüfung garantiert, dass bei Einhaltung der Optionsbeziehungen sowie unter Berücksichtigung des aktuellen Kontextes immer eine korrekte Prozessvariante resultiert. Wir müssen zur Laufzeit lediglich sicherstellen, dass alle harten Optionsbeziehungen tatsächlich eingehalten werden (vgl. Anforderung 7.6). Eine Kompatibilitätsprüfung ist

immer dann erforderlich, wenn eine Option dynamisch anzuwenden oder auszulassen ist, also an jedem `OPSplit`-Knoten, dessen übergeordnete Option im Zustand `undefined` ist. Eine Betrachtung des Zustandes einer Option an einem `OPSplit`-Knoten und ggf. eine entsprechende Zustandsänderung findet allerdings nur dann statt, wenn der Zustand der eingehenden Kante des `OPSplit`-Knotens `true_signaled` ist. Optionen, die in einem abgewählten Ausführungspfad „liegen“, verbleiben entsprechend im Zustand `undefined`. Für die Bewertung der Kompatibilität einer Optionsmenge zur Laufzeit ist dies unkritisch.

Der Entscheidungsbaum in Abbildung 7.14 zeigt die Abfragen an einem `OPSplit`-Knoten: Zunächst prüfen wir, ob die Kontextbedingung der Option aktuell erfüllt ist. Ist dies der Fall, prüfen wir zusätzlich, ob die Anwendung der Option auch hinsichtlich der vorliegenden harten Optionsbeziehungen korrekt ist. Dazu ermitteln wir, ob nach Anwendung der Option weiterhin eine kompatible Optionsmenge vorliegt.[6] Nur dann wird die Option angewendet, andernfalls darf aufgrund der höheren Priorisierung der Optionsbeziehungen die Option, trotz gültiger Kontextbedingung, nicht angewendet werden. Analog müssen wir auch im Falle einer ungültigen Kontextbedingung prüfen, ob bestehende harte Optionsbeziehungen das Auslassen der Option erlauben. Beispiel 7.18 erläutert ein Szenario zur Einhaltung harter Optionsbeziehungen zur Laufzeit.

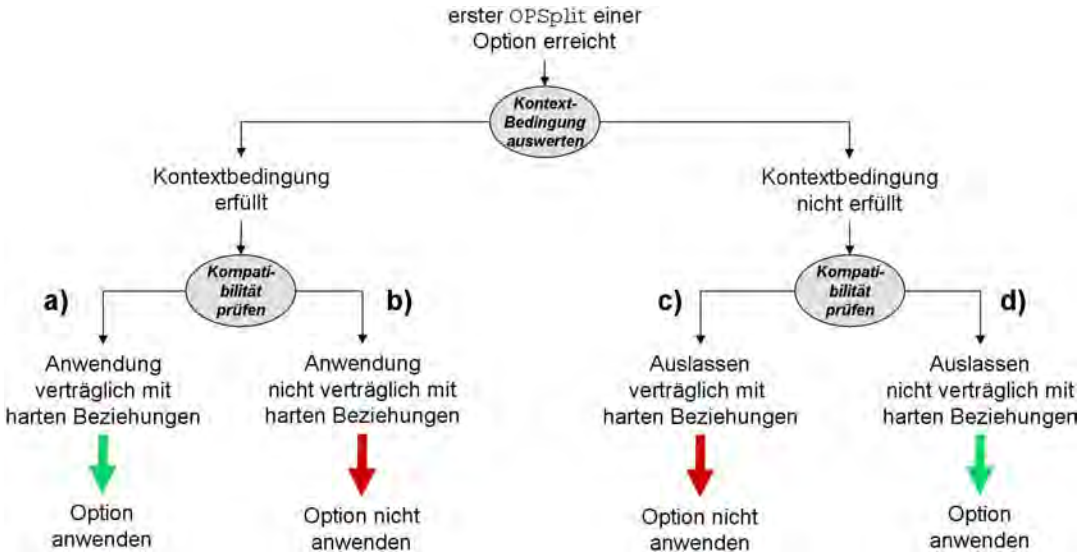


Abbildung 7.14: Abfragen an einem `OPSplit`-Knoten zur Laufzeit

Beispiel 7.18 (Einhaltung harter Optionsbeziehungen zur Laufzeit) *Abbildung 7.15 zeigt ein Szenario für die Einhaltung harter Optionsbeziehungen. Bei Ausführung des für den gegeben Kontext konfigurierten Ergebnismodells (vgl. Abbildung 7.15d) soll die harte Auswahlbeziehung „Es sind genau 2 Optionen aus der Menge {Option 1, Option 2, Option 3} anzuwenden“ eingehalten werden. Abbildung 7.16 zeigt, wie wir zur Laufzeit sicherstellen, dass neben der bereits statisch angewandten Option 3 noch genau eine weitere Option (d.h. Option 1 oder Option 2) angewendet wird: Bei Erreichen eines `OPSplit`-Knotens von Option 1 (vgl. Abbildung 7.16a) erkennen wir, dass die Kontextbedingung der Option nicht erfüllt ist. Da aufgrund der noch ausstehenden Option 2 weiterhin die Einhaltung der harten Optionsbeziehung möglich ist, lassen wir Option 1 aus (vgl. Abbildung 7.16a). Bei Auswertung des ersten zur Laufzeit erreichten `OPSplit`-Knotens von Option 2, an welchem Option 2 im Zustand `undefined` ist, stellen wir fest, dass wir diese Option nun aufgrund der Optionsbeziehung anwenden*

[6] Wir rufen dazu die Hilfsfunktion `checkCompatibilityAtRuntime` auf (vgl. Anhang A). Diese prüft, ob nach Anwenden bzw. Auslassen einer Option immer noch eine kompatible Optionsmenge vorliegt. Ist dies der Fall, gibt die Funktion den Wert `true` zurück. Andernfalls ist der Rückgabewert `false`.

müssen, obwohl auch hier die Kontextbedingung nicht erfüllt ist (vgl. Abbildung 7.16b). Wenn nachfolgende `OPSplit`-Knoten von bereits betrachteten Optionen erreicht werden, ist nur noch der Zustand der Option (d.h. der Wert der Variable `state`) relevant. Entsprechend entscheiden wir uns, am zweiten zur Laufzeit erreichten `OPSplit`-Knoten von Option 1, gegen die Ausführung von Aktivität `C'` (vgl. Abbildung 7.16c). Die gleiche Entscheidung treffen wir später auch am letzen `OPSplit`-Knoten von Option 1. Der zweite `OPSplit`-Knoten von Option 2 steht in einem bedingten Ausführungspfad, der zur Laufzeit abgewählt wird. Somit wird die zweite Änderungsoperation der Option nicht angewendet. Dies stellt allerdings keine Verletzung der Atomarität dieser Option dar.

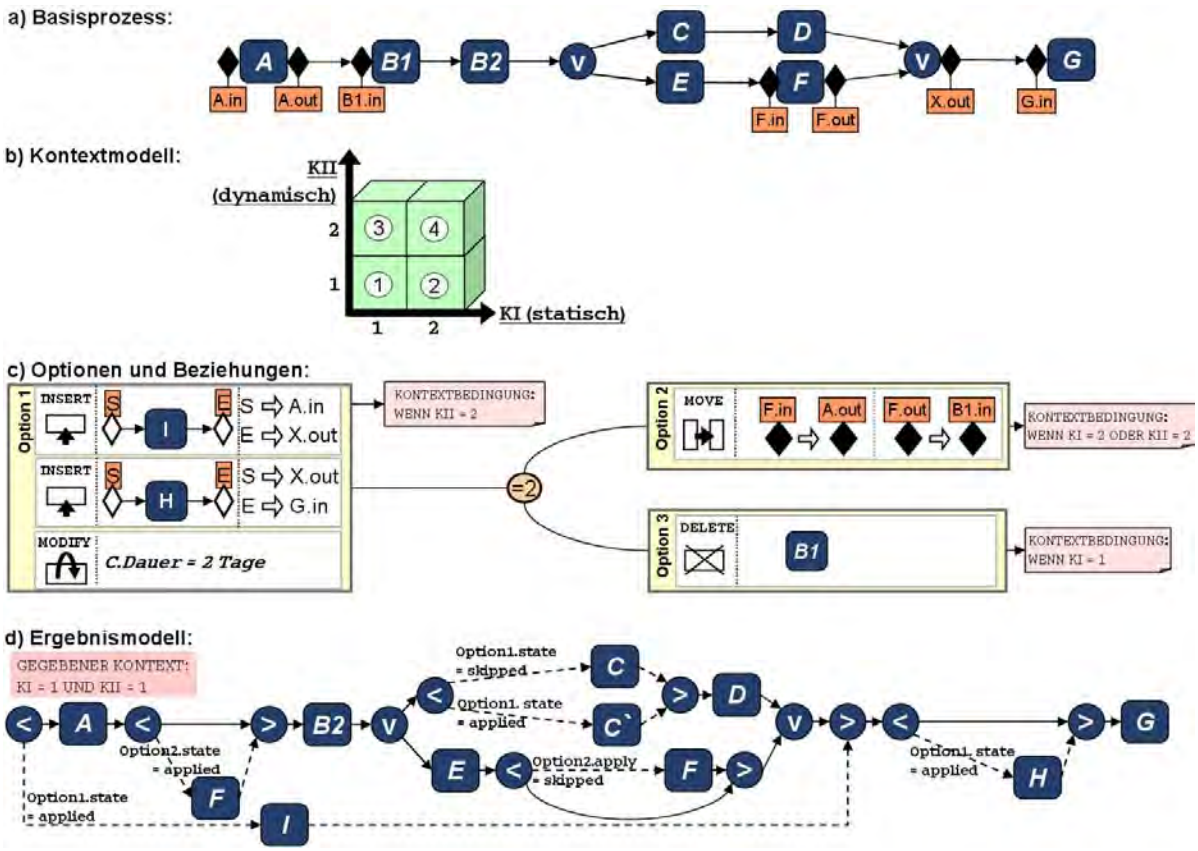


Abbildung 7.15: Szenario zur Einhaltung harter Optionsbeziehungen zur Laufzeit

7.4 Diskussion

Der Aspekt der Dynamik des Kontextes und der daraus resultierenden notwendigen Anpassung des Ergebnismodells, wird im Prozessmanagement auf verschiedene Weise adressiert. Die Ansätze können dabei wie folgt kategorisiert werden:

1. Vormodellierung aller möglichen Abweichungen
2. Nachträgliches Modellieren bzw. Einbinden von Abweichungen
3. Ausnahmebehandlung zur Laufzeit
4. Ad-hoc-Adaption von Prozessinstanzen zur Laufzeit

Im Folgenden diskutieren wir die wichtigsten Arbeiten zu diesen vier Kategorien.

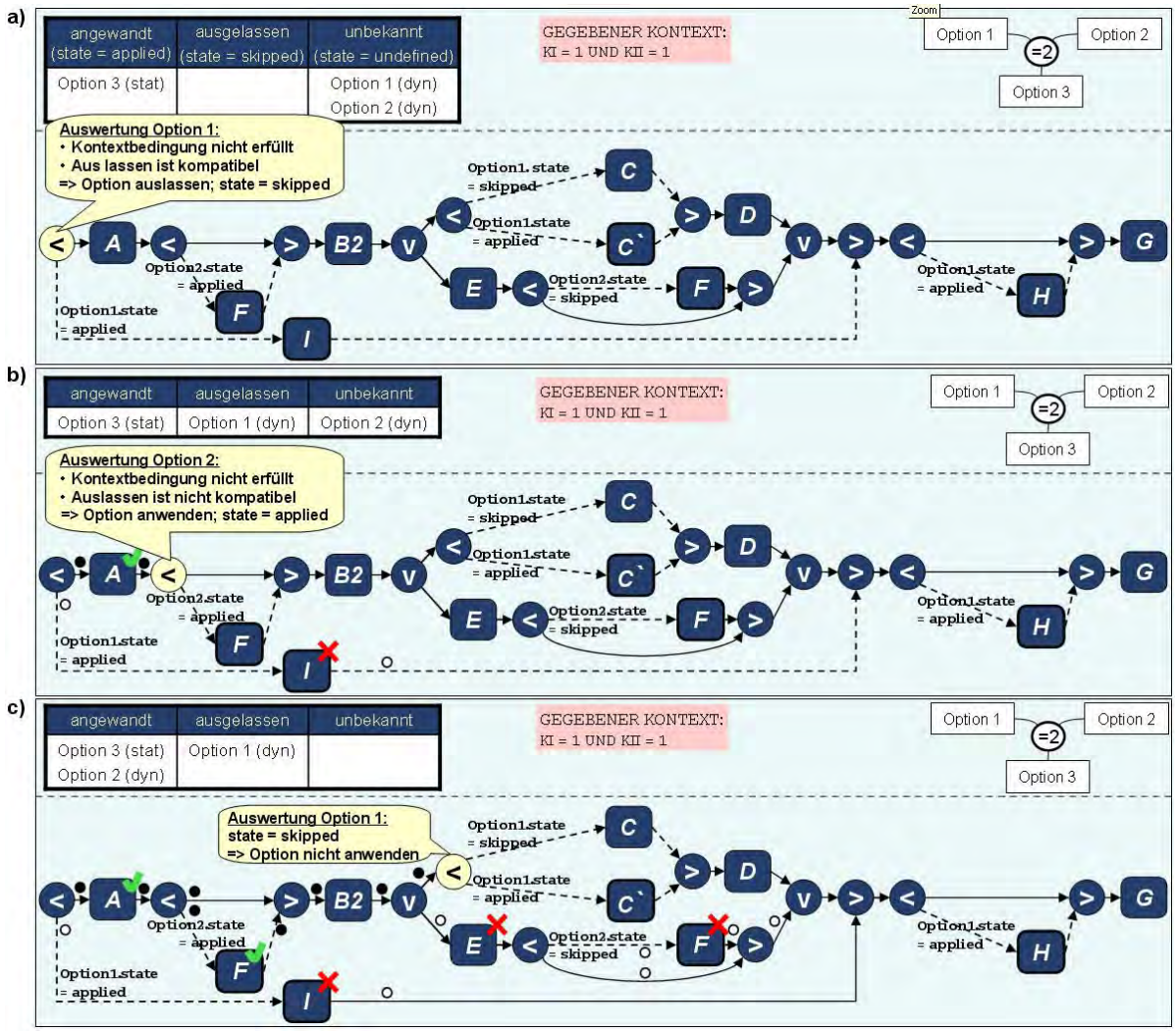


Abbildung 7.16: Abfrage der Kontextbedingungen und harten Optionsbeziehungen zur Laufzeit

Ansätze zur Vormodellierung aller möglichen Abweichungen

Provop implementiert den ersten Ansatz – die Vormodellierung aller möglichen Abweichungen. Durch Spezifikation der Änderungsoperationen sowie deren dynamische Anwendung bei der Konfiguration einer Prozessvariante, sind bereits vor Prozessstart alle möglichen Varianten berücksichtigt. Auch ein Ein-Modell-Ansatz, bei dem ein Modell alle Prozessvarianten beinhaltet, realisiert diesen Ansatz [WSR09]. Dabei bietet Provop eine Verbesserung im Vergleich zum konventionellen Ein-Modell-Ansatz, da wir nur solche Prozessvarianten in einem Modell integrieren, die zur Laufzeit auch tatsächlich relevant werden können, anstatt grundsätzlich alle Varianten eines Prozesstyps gemeinsam zu instanziieren. Dies bietet zum Beispiel verbesserte Monitoring-Möglichkeiten.

Ansätze zum nachträglichen Modellieren bzw. Einbinden von Abweichungen

Die Ansätze zum nachträglichen Modellieren bzw. Einbinden von Abweichungen lassen sich in drei Gruppen unterteilen [WSR09]. Die erste Gruppe realisiert sog. *Late Binding*, wie z.B. im *Worklets*-Konzept realisiert [AtHEvdA06]. Hier wird ein generisches Prozessmodell entwickelt, das den Prozess grobgranular beschreibt. Um an bestimmten Stellen Verfeinerungen des Prozesses zu erlauben, werden Platzhalter verwendet. Diese Platzhalter sind Aktivitäten,

welche zur Laufzeit, in Abhängigkeit vom aktuellen Kontext, dynamisch einen Sub-Prozess (d.h. Worklet) binden und ausführen können.

Die zweite Gruppe realisiert sog. *Late Modeling*. Hier werden das Prozessmodell oder einzelne Prozessfragmente erst zur Laufzeit modelliert. Eine Implementierung sind die *Pockets of Flexibility* [SOS05]. Hier wird innerhalb eines Prozessmodells ein Teilbereich markiert, der erst zur Laufzeit konkretisiert bzw. modelliert wird. Für dieses *Pocket of Flexibility* sind eine Menge von Aktivitäten und eine Menge von Abhängigkeiten (d.h. Constraints) gegeben. Letztere beschreiben, in welcher Reihenfolge die Aktivitäten auszuführen sind oder welche Implikationen und wechselseitigen Ausschlüsse zwischen ihnen existieren. Auf Basis der gegebenen Aktivitäten und Abhängigkeiten kann zur Laufzeit das Prozessfragment zusammengesetzt und ausgeführt werden.

Die dritte Gruppe realisiert sog. *Late Composition*. Ein Beispiel bietet *DECLARE* [PSSvdA07, PSvdA07, vdAPS09]. Statt vorgegebenem Kontrollfluss, wie bei imperativen Beschreibungssprachen, wird ein deklarativer Ansatz zur Beschreibung des Ausführungsverhaltens verfolgt. Zum Zusammensetzen von Prozessfragmenten wird eine graphische Modellierungssprache für Abhängigkeiten verwendet, die intern auf *Lineare Temporale Logik (LTL)* abgebildet wird. Ähnlich zu [SOS05] beschreiben hier Abhängigkeiten zwischen Aktivitäten deren mögliche Ausführung bzw. Ausführungsreihenfolge. Zur Laufzeit kann der Prozess zusammengefügt werden, wobei jedes Ausführungsverhalten, das nicht durch eine Abhängigkeit verboten wird, möglich ist. DECLARE kann auch gemeinsam mit imperativen Workflow-Beschreibungssprachen verwendet werden. So gibt es eine Erweiterung, welche jeweils die Beschreibung von Sub-Prozessen als DECLARE oder YAWL-Prozess unterstützt [vdAPS09].

Einen vergleichbaren aber performanteren Ansatz bietet *ALASKA* [WPZW09, WZPW09]. Der Alaska Simulater ermöglicht, auf Basis der Metapher Reiseplanung und -durchführung, die Analyse des Planungsverhalten. Dabei können die Ansätze der flexiblen Prozessgestaltung, wie Late Binding, Late Modeling und Late Composition, bei der Planung und Ausführung einer Reise angewendet werden.

Das System *FLOWer* [WG05] basiert auf den Konzepten des *Case Handling* und erlaubt ebenfalls das Zusammenfügen von Prozessfragmenten zur Laufzeit. Im Gegensatz zu den vorherigen Ansätzen ist FLOWer allerdings datenbasiert. Über die Ausführung von Aktivitäten wird daher in Abhängigkeit von der Verfügbarkeit bestimmter Prozessdaten entschieden.

Ein anderes Konzept zur dynamischen Konfiguration von Workflows sind *agenten-basierte Ansätze* [JFJ+96, MGR03, BACR08, RB07, BSBB06]. Agenten stellen autonome Einheiten dar, die dezentral gesteuert werden. In [BACR08] werden zu diesem Zweck Agenten mit der Information (d.h. Regel) ausgestattet, unter welchen Bedingungen (d.h. Kontext) welcher „Plan“ auszuführen ist und welche Ziele dadurch erreicht werden sollen. Ein einzelner Plan besteht aus Aktivitäten, die zur Erreichung eines spezifischen Teilziels des Agenten ausgeführten werden, und die wiederum zur Veränderung des Kontextes führen können. Sind alle Ziele erreicht, wird kein Agent mehr ausgeführt.

Bei agenten-basierten Ansätzen geht es weniger darum, flexibel vordefinierte Prozessvarianten auszuwählen und auszuführen, als um die dynamische Zusammenstellung des eigentlichen Prozessmodells aus gegebenen Fragmenten zur Laufzeit.

Ansätze zur Ausnahmebehandlung zur Laufzeit

Neben der dynamischen Anpassung von Prozessmodellen ist auch die Beschreibung von Abweichungen für Ausnahmesituationen ein in der Literatur behandelter Aspekt [CCPP99, EL96, HA00, RDB03]. Dabei können Ausnahmen (engl. *exceptions*) bereits bei

der Modellierung von Prozessen berücksichtigt werden. [RvdAtH06] bietet eine Übersicht existierender Ansätze und stellt verschiedene *Workflow Exception Patterns* vor, wie z.B. den Neustart einer Prozessinstanz und das erzwungene Beenden einer Aktivität.

Um zur Laufzeit flexibel auf verschiedene Ausnahmesituationen eingehen zu können präsentiert [AtHvdAE07] den *Exlet*-Ansatz. Dieser auf sehr niedriger Abstraktionsebene angesiedelte Vorschlag baut auf dem oben erwähnten Worklets-Konzept auf [AtHEvdA06]. Zur Laufzeit können verschiedene, in sich abgeschlossene Ausnahmeprozesse (sog. Exlets) aufgerufen werden. Dafür wird ein Web-Service verwendet, der dynamisch den aktuell benötigen Ausnahmeprozess auswählt und aufruft.

In [RR06] wird ein Ansatz für daten-basierte Ausnahmebehandlungen vorgestellt. Dabei können Ausnahmezustände durch Änderung von Datensätzen korrigiert werden. Darüber hinaus beschreiben die Autoren, wie Kontextinformationen auf einer höheren Abstraktionsebene genutzt werden können, um eine laufende Prozessinstanz dynamisch an eine gegebene Ausnahmesituation anzupassen. Eine Weiterentwicklung solch datengetriebener Ausnahmebehandlungen bietet Corepro [MRH07, MRH08]. Hierbei werden komplexe Prozessstrukturen dynamisch an geänderte Datenstrukturen angepasst.

Ansätze zur Ad-hoc-Adaption von Prozessinstanzen

Adaptive WfMS erlauben die dynamische Änderung einer Prozessinstanz zur Laufzeit. Im *ADEPT* Projekt wird z.B. ein Ansatz entwickelt, welcher die direkte Änderung einer betroffenen Prozessinstanz erlaubt [DR98, RD98, RRKD05, RKL+06, DR09]. Dabei können autorisierte Endanwender durch Hinzufügen, Löschen oder Verschieben von Aktivitäten die Prozessinstanz dynamisch adaptieren (vgl. [WRR08, WSR09]). Ein wichtiger Aspekt, den ADEPT ebenfalls adressiert, ist die Gewährleistung der syntaktischen Korrektheit von Prozessinstanzen bei solchen Änderungen [RRD03, RRD04a, RRD04b]. Weiter wird im *SeaFlows* Projekt ein Rahmenwerk entwickelt, das die Beschreibung von semantischen Abhängigkeiten erlaubt und deren Einhaltung auch in Verbindung mit dynamischen Änderungen sicherstellt [LRD06, LRD08b, LRMGD09].

Eine Weiterentwicklung dieses adaptiven Ansatzes präsentiert [WRW06, WWRD07, WRWRM09]. Die Autoren folgen hier dem Konzept des *Case-based Reasoning* (dt. fallbasiertes Schließen). Dabei werden zu allen durchgeführten dynamischen Anpassungen die Gründe in Form von Frage-Antwort-Paaren beschrieben und in einer Fallbasis gespeichert. Die auf diese Weise dokumentierten Fälle werden zum einen zur Prozessverbesserung verwendet und dienen zum anderen zur Laufzeit als Unterstützung für Prozessanpassungen in ähnlichen Situationen. Im zweitgenannten Anwendungsfall bedeutet dies, dass zur Laufzeit, wenn das Workflow-Modell adaptiert werden soll, in der Fallbasis nach Best Practices gesucht werden kann und die dokumentierten Anpassungen (ggf. nach Adjustierung) übernommen werden können. Ein manuelles Eingreifen des Endanwenders ist allerdings weiterhin erforderlich, um das Prozessmodell anzupassen. Eine automatische Anpassung, abhängig vom Kontext, ist nicht möglich und in der Regel hier auch nicht gewünscht.

Ein Beispiel für agenten-basiertes Prozessmanagement wird in [MGR03] vorgestellt. *AGENTWORK* unterstützt die automatische Adaptionen von Prozessinstanzen basierend auf ECR-Regeln und dem adaptiven WfMS ADEPT2 [RRKD05]. Die möglichen Abweichungen werden in Form von *Event Condition Action (ECA) Rules* während der Modellierung beschrieben und durch sog. *Aktive Temporal Frame Logic* abgebildet. Der temporale Aspekt wird als wesentlicher Faktor berücksichtigt und als Zeitabhängigkeit in den ECA-Regeln aufgeführt.

Die Flexibilität von Ad-hoc Adaptionen zur Laufzeit ist erforderlich, um auf ungeplante Änderungen der fachlichen Anforderungen oder nicht vorgeplante Ausnahmeszenarien reagieren zu können. Meist sind von solchen Adaptionen nur einzelne Prozessinstanzen betroffen. Somit ist der Zweck der dynamischen Adaption einer Prozessinstanz weniger das dauerhafte und langfristige Mangagement von Prozessvarianten, wie im Fall von Provop, sondern die einmalige und nicht vorher geplante Anpassung eines Modells an geänderte Rahmenbedingungen. In [Rei09] wird in diesem Zusammenhang von sog. „Fluid Processes“ gesprochen.

7.5 Zusammenfassung

Zur Abbildung der Kontextdynamik bieten wir dem Kontextmodellierer die Möglichkeit, explizit zwischen Kontextvariablen, deren Wert sich zur Laufzeit nicht mehr ändert und solchen, die während der Laufzeit eines Prozesses initialisiert und geändert werden können (d.h. dynamischen Kontextvariablen), zu unterscheiden. Optionen, deren Kontextbedingung sich auf dynamische Kontextvariablen beziehen, können zur Laufzeit dynamisch gültig oder ungültig werden. Um über die dynamische Anwendung solcher Optionen entscheiden zu können, werden diese dynamisch auf den Basisprozess angewendet. Dabei berücksichtigen wir, dass immer alle Änderungsoperationen einer Option gemeinsam wirksam sollen, d.h. die Atomarität der Optionen muss bewahrt werden.

Kontextänderungen und die damit verbundene dynamische Anwendung von Optionen wirken sich auf die Einhaltung von Optionsbeziehungen aus. Um Konflikte zwischen Kontextänderungen und Optionsbeziehungen aufzulösen, unterscheiden wir zwischen weichen Beziehungen, die nur bei der statischen Konfiguration von Prozessvarianten berücksichtigt werden, und harten Beziehungen, die zur Laufzeit selbst dann eingehalten werden müssen, wenn die Kontextbedingung einer Anwendung bzw. dem Auslassen einer Option widerspricht.

Durch dynamische Anwendung von Änderungsoperationen kann zum Zeitpunkt der Konfiguration ein vollständiges Prozessmodell erstellt werden. Dadurch kann bereits vor der Ausführung die strukturelle und semantische Korrektheit der Prozessmodelle, auch im Fall späterer Kontextänderungen, sichergestellt werden. Die in diesem Kapitel vorgestellte Korrektheitsprüfung erweitert dazu entsprechend die Prüfung aus Kapitel 6. Zur Laufzeit stellen wir dann, wenn eine Option dynamisch angewendet oder ausgelassen werden soll, ihre Atomarität sowie ihre Kompatibilität mit harten Optionsbeziehungen sicher.

8

Evolution und Refactoring von Prozessvarianten

Die vorangehenden Kapitel haben sich mit den Phasen der Modellierung, Konfiguration und Ausführung von Prozessvarianten befasst. Wir haben beschrieben, wie wir aus einem Basisprozess und seinen Optionen das korrekte Ergebnismodell einer Prozessvariante ableiten und ausführen können. Dieses Kapitel stellt vor, wie wir mit fachlichen Umgebungsänderungen umgehen und wie wir die Prozesslandschaft durch Evolution und Refactoring der Variantenmodelle besser handhabbar machen können. Kapitel 8 gliedert sich wie folgt: Abschnitt 8.1 motiviert die Notwendigkeit der Evolution von Prozessvarianten und erläutert die wesentlichen Anforderungen an diese Phase. Anschließend beschreibt Abschnitt 8.2 mögliche fachliche Änderungen zur Anpassung von Prozessvarianten. Abschnitt 8.3 präsentiert Ansätze für das Refactoring von Basisprozessen und Optionen. In Abschnitt 8.4 diskutieren wir verwandte Arbeiten und in Abschnitt 8.5 fassen wir das Kapitel zusammen.

8.1 Motivation und Anforderungen

Geänderte fachliche Anforderungen erfordern oftmals, Prozesse und ihre Varianten flexibel und schnell an die veränderten Gegebenheiten anzupassen. Basisprozesse, Optionen und Kontextmodelle sind daher nicht starr, sondern müssen durch den Modellierer einfach und kontinuierlich anpassbar sein. Eine wichtige Anforderung betrifft die Sicherstellung der Korrektheit der Prozessfamilie nach Änderungen. Das heißt, die Evolution von Prozessvarianten darf nicht zu einer inkorrekten Prozessfamilie bzw. zu inkorrekten Varianten führen. Um den Modellierer bestmöglich bei solchen Änderungen zu unterstützen, müssen ihm nach einer Anpassung alle ggf. resultierenden Inkonsistenzen angezeigt werden.

Die übersichtliche und einfache Handhabbarkeit von Prozessvarianten stellt eine weitere Anforderung dar. Es muss daher ein Refactoring von Basisprozessen und Optionen unterstützt werden. Das heißt es muss möglich sein, mit Hilfe geeigneter Maßnahmen, etwa dem Entfernen überflüssiger Aufsetzpunkte oder redundanter Änderungsoperationen, Basisprozesse und Optionen neu zu strukturieren und somit eine bessere Les- und Wartbarkeit zu erreichen. Durch ein solches Refactoring darf sich allerdings das Ausführungsverhalten der konfigurierbaren Prozessvarianten nicht ändern [WR08], ähnlich wie man dies von Refactoring aus dem Gebiet des Software Engineerings kennt [Opd92, Fow00, Bec00]. Außerdem muss die

Korrektheit der Prozessfamilie erhalten bleiben. Insgesamt ergeben sich die in Tabelle 8.1 aufgeführten Anforderungen.

Tabelle 8.1: Anforderungen an die Evolution und das Refactoring von Prozessvarianten

Evolution von Prozessvarianten
Anforderung 8.1 *Im Zuge geänderter fachlicher Anforderungen müssen entsprechende Anpassungen an den Prozessvarianten vorgenommen werden.*
Anforderung 8.2 *Bei der Evolution von Prozessvarianten muss die Korrektheit der Prozessfamilie sichergestellt werden.*
Anforderung 8.3 *Aus Änderungen resultierende Inkonsistenzen müssen dem Modellierer in einer transparenten Art und Weise angezeigt werden.*
Refactoring von Prozessvarianten
Anforderung 8.4 *Es muss ein (semi-)automatisches Refactoring des Basisprozesses und zugehöriger Optionen möglich sein, um die Variantenmodelle möglichst verständlich zu gestalten.*
Anforderung 8.5 *Ein Refactoring von Basisprozess und Optionen darf sich nicht auf die fachliche Ausführung der Prozessvarianten auswirken.*
Anforderung 8.6 *Bei einem Refactoring muss die Korrektheit einer Prozessfamilie bewahrt werden.*

8.2 Evolution von Prozessvarianten

Nachdem wir einen Basisprozess und zugehörige Optionen modelliert haben, können wir Prozessvarianten konfigurieren und anschließend in einem WfMS ausführen. Es kann allerdings der Fall eintreten, dass sich bisherige fachliche Anforderungen an ein Prozessmodell ändern. Um die Prozessfamilie an diese „neuen“ fachlichen Anforderungen anzupassen, können wir in Provop diverse Änderungen vornehmen. Das heißt, der Basisprozess und seine Optionen sowie das zugehörige Kontextmodell sind keine starren Objekte, die einmalig angelegt und nicht mehr verändert werden, sondern sind flexibel an geänderte Rahmenbedingungen anpassbar. Darüber hinaus stellen wir sicher, dass nach solchen Änderungen eine korrekte Prozessfamilie resultiert. Schließlich ermöglicht die Schemaevolution [Rin04] die Migration laufender Prozessinstanzen auf geänderte Versionen des zugrunde liegenden Variantenmodells. Der Ablauf der Evolution von Prozessvarianten kann, wie in Abbildung 8.1 dargestellt, skizziert werden.

Abbildung 8.1: Vorgehensweise der Übernahme von Änderungen

8.2.1 Schritt 1 (Nötige Änderungen identifizieren):

Eine Evolution von Prozessvarianten wird in der Praxis nötig, wenn fachliche Anforderungen eines Prozesstyps nicht adäquat abgedeckt werden oder sich Änderungen an bestehenden

fachlichen Anforderungen ergeben (vgl. Beispiel 8.1). Je nachdem, wie sich diese Anforderungen ändern, ist entweder die gesamte Prozessfamilie oder nur eine Teilmenge der Prozessvarianten anzupassen (vgl. Anforderung 8.1).

Beispiel 8.1 (Änderung der fachlichen Anforderungen an einen Prozesstyp) *In der Praxis ergeben sich oftmals Änderungen an fachlichen Anforderungen, etwa wenn für einen Unternehmensbereich neue Normen oder Verfahrensrichtlinien definiert werden, die eine Anpassung der Prozesstypen erforderlich machen. Im Bereich der Fahrzeugentwicklung sind dies z.B. die VDA-Empfehlung 4965 [VDA05] für das Änderungsmanagement (siehe auch Abschnitt 2.1) oder ISO 26262 für sicherheitsrelevante Systeme in Kraftfahrzeugen.*

8.2.2 Schritt 2 (Änderungen durchführen):

Zur Evolution von Prozessvarianten können in Provop verschiedene Objekte geändert werden. Eine Übersicht möglicher Anpassungen zeigt Abbildung 8.2.[1]

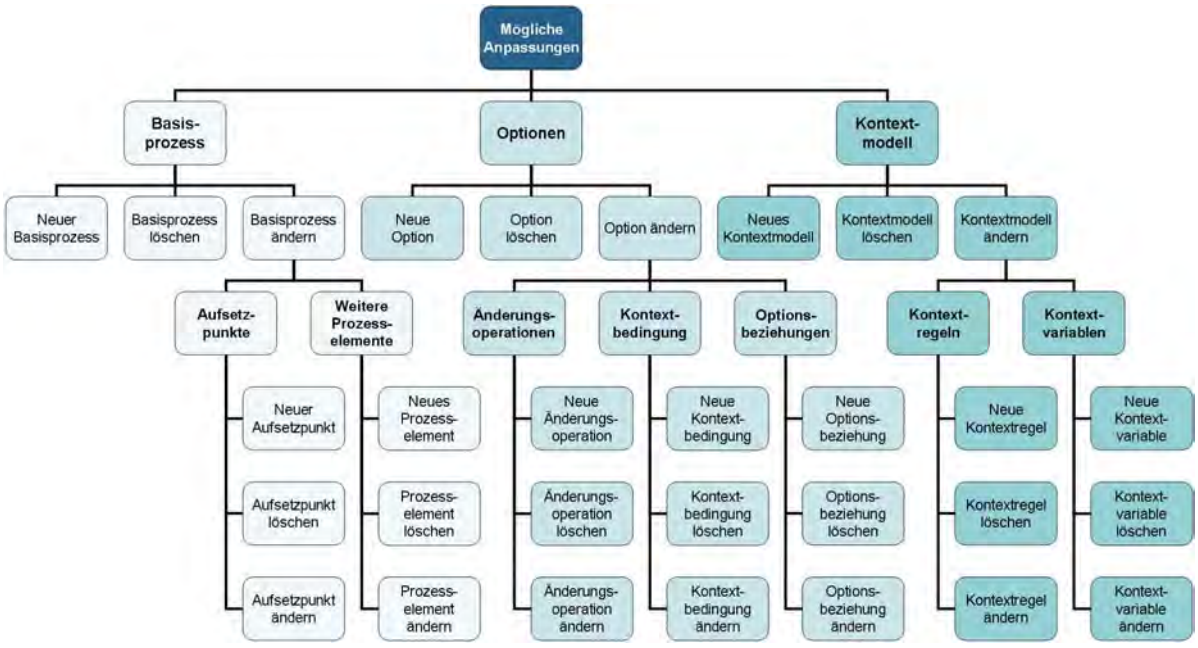


Abbildung 8.2: Mögliche Anpassungen einer Prozessfamilie

Da neben Basisprozess (inkl. Aufsetzpunkten) auch Optionen und Kontextmodell Änderungen unterliegen, hat der Modellierer verschiedene Möglichkeiten, um eine Prozessfamilie anzupassen. So kann er direkte Änderungen an einem Basisprozess vornehmen oder – im Falle lokaler Änderungen – auch nur einzelne Optionen anpassen (vgl. Beispiel 8.2).

Beispiel 8.2 (Anpassung einer Prozessvariante) *Für den zuvor vorgestellten Änderungsmanagementprozess sollen die Prozessvarianten an die VDA-Empfehlung angeglichen werden. Das heißt die Änderung ist für alle Variantenmodelle der Prozessfamilie relevant. Der Modellierer entscheidet sich in diesem Fall, den Basisprozess entsprechend abzuändern. Er stellt anschließend sicher, dass keine Änderungen redundant vorgenommen werden (etwa für den Fall, dass eine bestimmte Option bereits Teile dieser Änderungen abdeckt) und passt Optionen an, die sich auf geänderte Elemente beziehen.*

[1] Im Folgenden verstehen wir unter Anpassungen das Löschen, neu Hinzufügen und Ändern von Objekten.

Bezogen auf die Werkstattprozesse, möchte ein Händler seine Prozesse dahingehend anpassen, dass zukünftig vor Rückgabe des Fahrzeugs an den Kunden eine zusätzliche Kundenzufriedenheitsgefragung durchgeführt wird. Da er am Basisprozess, aufgrund dessen zentralen Vorgabe, keine Änderungen vornehmen darf, modelliert er eine Änderungsoperation, die das Einfügen dieser Umfrageaktivität vornimmt. Da die Kundenzufriedenheitsumfrage nur über einen begrenzten Zeitraum von zwei Monaten durchgeführt werden soll, entscheidet er sich, eine neue Option zu erstellen, anstatt eine existierende Option zu erweitern.

Welche konkrete Anpassung ein Modellierer durchführt, hängt von verschiedenen Aspekten ab. Nachfolgend seien beispielhaft verschiedene Leitfragen aufgeführt, die ihn bei seiner Entscheidung unterstützen können:

- Wie viele Varianten der Prozessfamilie müssen angepasst werden? Ist die Anpassung für alle oder nur einzelne Prozessvarianten, etwa in einem bestimmten Kontext, relevant?
- Kann der Anpassungsaufwand minimal gehalten werden? Das heißt kann eine geringe Anzahl an Objekten angepasst werden, um die fachlichen Anforderungen zu erfüllen?
- Welche Anpassungen dürfen überhaupt vorgenommen werden? Welche Restriktionen ergeben sich aufgrund der Rolle des Modellierers (z.B. Basisprozessmodellierer vs. Variantenmodellierer)?[2]
- Können existierende Objekte wiederverwendet werden oder sind neue Objekte erforderlich? Ist z.B. die Erweiterung einer bestehenden Option möglich oder muss eine neue Option angelegt werden?

8.2.3 Schritt 3 (Korrektheit nach Änderungen gewährleisten):

Führen wir Änderungen an einem Basisprozess, seinen Optionen oder seinem Kontextmodell durch, kann dies zu verschiedenen Fehlern führen. So können ungültige Objektreferenzen, strukturell oder semantisch inkorrekte Variantenmodelle sowie Syntaxfehler resultieren. Beispiel 8.3 und 8.4 unterstreichen dies.

Beispiel 8.3 (Fehler nach Änderung des Basisprozesses) *Abbildung 8.3a zeigt einen Basisprozess, der aufgrund geänderter fachlicher Anforderungen zu dem in Abbildung 8.3b dargestellten Modell angepasst wird. Dazu werden Aktivität* `D` *entfernt und Aktivität* `F` *in den Basisprozess eingefügt. Die in Abbildung 8.3c dargestellten Optionen beziehen sich noch auf den ursprünglichen Basisprozess. Ihre Anwendung auf den angepassten Basisprozess würde zu Fehlern führen: Option 1 kann nicht auf den angepassten Basisprozess angewendet werden, da die Referenz auf Aktivität* `D` *nicht auflösbar ist. Die Anwendung von Option 2 auf den Basisprozess ist zwar möglich, allerdings tritt Aktivität* `F` *anschließend mehrfach im Ergebnismodell auf. Das Ergebnismodell ist daher semantisch möglicherweise inkorrekt.*

Beispiel 8.4 (Fehler nach Änderung einer Option) *Abbildung 8.4 zeigt einen Basisprozess, für den zwei Optionen definiert sind. Diese beiden Optionen schließen sich wechselseitig aus. Aufgrund geänderter fachlicher Anforderungen wird nun Option 1 angepasst (vgl. Abbildung 8.4c). Konkret wird eine* `MODIFY`*-Operation hinzugefügt, welche die Dauer der Aktivität* `F`*, welche von Option 2 in den Basisprozess eingefügt wird, verändert. Nach dieser Anpassung besteht eine strukturelle Abhängigkeit*

[2] Diese Fragestellung ist vor allem relevant, wenn ein Basisprozess zentral vorgegeben ist und die Varianten dezentral modelliert werden (vgl. Beispiel 8.2).

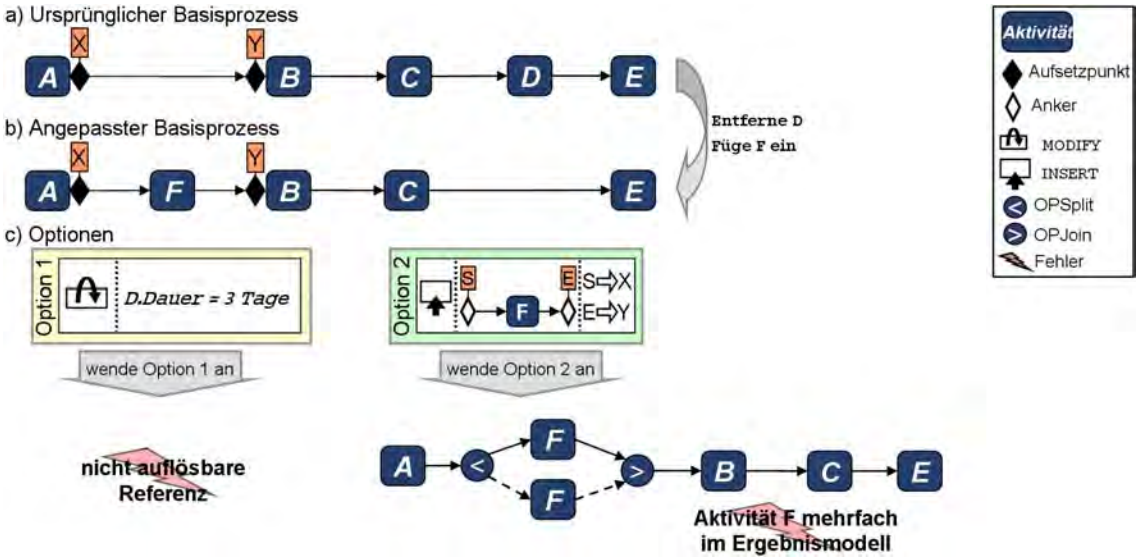


Abbildung 8.3: Fehler nach Änderung des Basisprozesses

zwischen den Optionen. Deshalb ist es nun nicht mehr möglich, durch alleinige Anwendung von Option 1, ein korrektes Ergebnismodell aus dem Basisprozess abzuleiten. Stattdessen ist die vorherige Anwendung von Option 2 erforderlich. Das heißt, der wechselseitige Ausschluss zwischen den beiden Optionen sollte nicht mehr gelten, oder aber die durchgeführte Änderung ist zu verwerfen. Entsprechend muss der Modellierer entweder seine Änderung zurücknehmen oder weitere Änderungen vornehmen, etwa das Entfernen des wechselseitigen Ausschlusses zwischen Option 1 und 2.

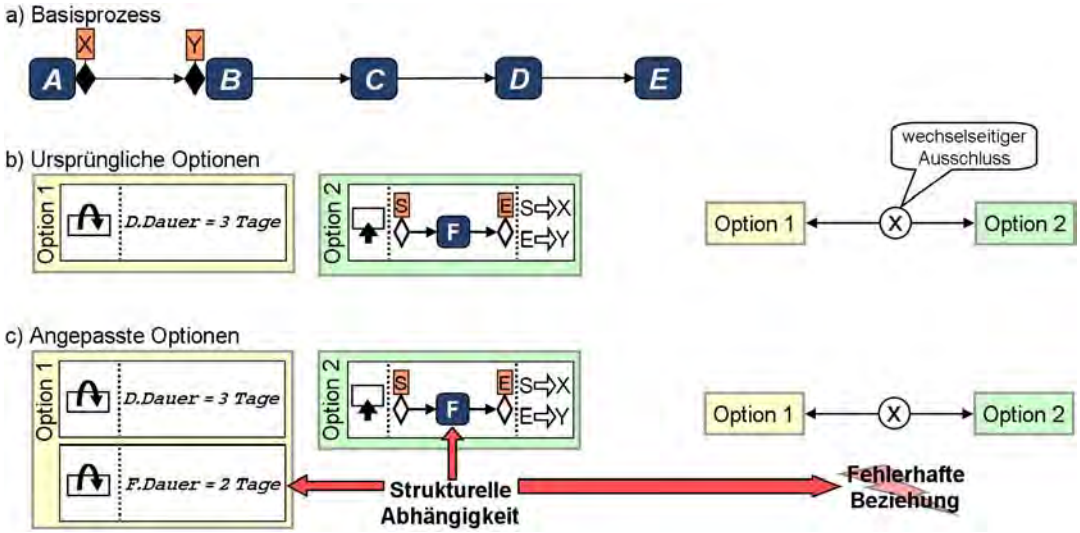


Abbildung 8.4: Fehler nach Änderung einer Option

Generell können wir nach Änderungen die Provop-Korrektheitsprüfung anstoßen, um die Korrektheit einer Prozessfamilie zu gewährleisten (vgl. Anforderung 8.2). Dadurch können wir ggf. vorhandene Fehler identifizieren. Nachteilig ist, dass bei inkorrektem Ergebnismodell nicht klar ist, welche konkrete Änderung dieses Problem verursacht hat. Zum Beispiel können sowohl Änderungen des Basisprozesses als auch einer Option dazu führen, dass andere Optionen nicht mehr korrekt anwendbar sind. Dies erschwert die Behebung von Fehlern. Um dieses Problem zu lösen, können wir prinzipiell nach jeder Änderung die Korrektheitsprüfung

erneut anstoßen und somit fehlerhafte Änderungen direkt identifizieren. Da meist erst nach mehreren Änderungen die Korrektheit der Prozessfamilie gewährleistet werden muss, führt dies aber zu unnötig hohen Aufwänden.

Ein alternativer Ansatz besteht darin, die Effekte einer Änderung zu prüfen und mögliche Inkonsistenzen als Warnungen an den Modellierer auszugeben. Die spezifischen Abhängigkeiten zwischen den Objekten in Provop zeigt Abbildung 8.5. Optionen können z.B. von Änderungen am Basisprozess, am Kontextmodell und an anderen Optionen beeinflusst werden. Da Änderungsoperationen auch auf Basis anderer Optionen beschrieben werden können (z.B. wenn eine zuvor eingefügte Aktivität durch eine nachfolgend angewandte Option geändert wird), können sich Anpassungen von Änderungsoperationen auf mehrere Optionen auswirken. Beispiel 8.5 erläutert den Zusammenhang zwischen den vorangehenden Beispielen 8.3 und 8.4 sowie den existierenden Abhängigkeiten zwischen den anpassbaren Objekten in Provop.

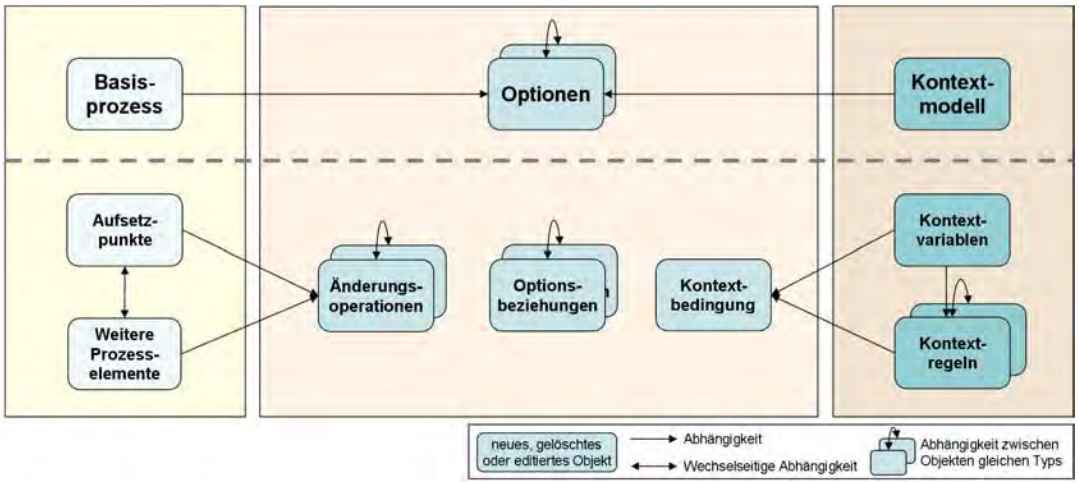


Abbildung 8.5: Abhängigkeiten zwischen Basisprozess, Optionen und Kontextmodell

Beispiel 8.5 (Abhängigkeiten zwischen Basisprozess und Optionen) *In den Beispielen 8.3 und 8.4 haben wir gezeigt, welche Probleme infolge von Änderungen am Basisprozess bzw. an zugehörigen Optionen entstehen können. Diese Probleme sind auf bestimmte Abhängigkeiten zurückzuführen. Die nicht auflösbare Referenz nach Löschen einer Aktivität des Basisprozesses in Beispiel 8.3 resultiert aufgrund der Abhängigkeit zwischen Prozesselementen und Änderungsoperationen, die diese Elemente referenzieren. Das semantisch inkorrekte Ergebnismodell in Beispiel 8.3 resultiert, da Basisprozess und Optionen voneinander abhängig sind. Die strukturelle Abhängigkeit der beiden Optionen in Beispiel 8.4 schließlich resultiert aus der Abhängigkeit zwischen zwei Optionen bzw. deren Änderungsoperationen.*

Der Abhängigkeitsgraph in Abbildung 8.5 gibt einen Überblick über alle möglichen Abhängigkeiten. Änderungen an Prozesselementen wirken sich allerdings nicht generell auf alle definierten Änderungsoperationen aus. Vielmehr können die konkreten Änderungsoperationen bzw. Optionen anhand der Prozesselement-ID einer geänderten Aktivität und die Analyse aller ID-basierter Änderungsoperationen genau identifiziert und dem Modellierer entsprechend gemeldet oder für ihn visualisiert werden (vgl. Anforderung 8.3). Zum Beispiel können inkonsistente Objekte hervorgehoben oder über ein zusätzliches Protokollfenster mit Textausgabe an den Modellierer gemeldet werden. Beispiel 8.6 zeigt eine mögliche Visualisierung von Inkonsistenzen nach einer Änderung durch entsprechende Symbole in der Modellierungsoberfläche.

Beispiel 8.6 (Visualisierung von Inkonsistenzen nach Änderungen) *Abbildung 8.6 zeigt einen Basisprozess und sein Kontextmodell. Diese werden durch Löschen von Prozesselementen bzw. einer Kontextvariable angepasst, wodurch sich Inkonsistenzen ergeben: Eine der beiden Kontextregeln ist nach den Änderungen nicht mehr aktuell, da sie auf Basis der gelöschten Kontextvariable formuliert wurde. Dies wird dem Kontextmodellierer anhand einer Markierung (Ausrufezeichen) angezeigt (vgl. Abbildung 8.6). Eine weitere Inkonsistenz, die aus dem Löschen der Kontextvariable resultiert, wird für die Kontextbedingung von Option 2 identifiziert. Option 3 ist ebenfalls inkonsistent, allerdings aufgrund des Löschens von Aktivität* `D`*. Option 1 ist von den Änderungen nicht betroffen.*

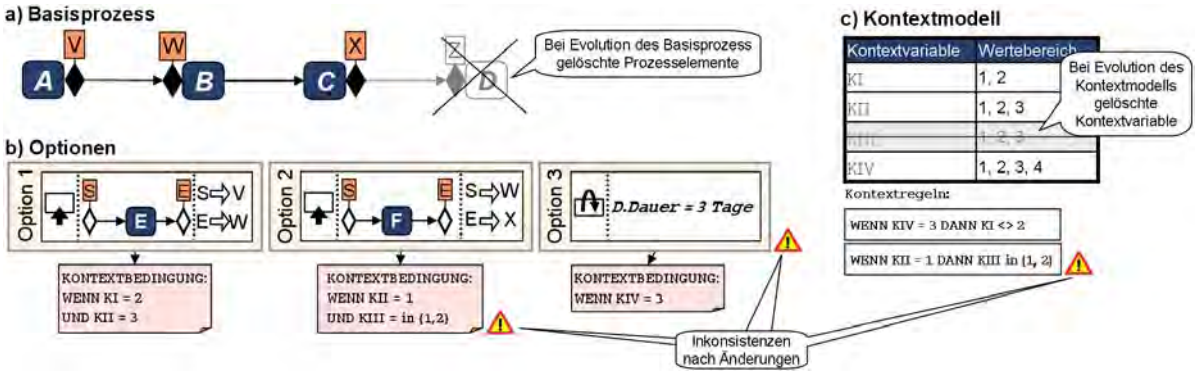


Abbildung 8.6: Visualisierung von Inkonsistenzen nach Änderungen

8.2.4 Schritt 4 (Änderungen propagieren):

Änderungen am Basisprozess und zugehörigen Optionen können für eine bestimmte Kontextbeschreibung zu einem anderen Ergebnismodell führen, als zu jenem, das für diese Kontextbeschreibung vor der Änderung konfiguriert wird. Das heißt, für ein Ergebnismodell liegt eine neu Version vor. Dieser Aspekt wird in der wissenschaftlichen Literatur als Schemaevolution bezeichnet [Rin04, RRD04a, RRD04c, RRD04b].[3] Aus einer Schemaevolution resultieren diverse Fragestellungen. Da von einer alten Version eines Ergebnismodells ggf. noch Prozessinstanzen ausgeführt werden, ist z.B. zu klären, ob die Änderung des Ergebnismodells erst für Neuinstanziierungen übernommen werden soll oder ob die Änderungen auch für laufende Prozessinstanzen der alten Version relevant sind. Beispiel 8.7 zeigt dies.

Beispiel 8.7 (Schemaevolution) *Abbildung 8.7a zeigt das Ergebnismodell von Variante 1. Zu diesem Modell existieren drei Prozessinstanzen (vgl. Abbildung 8.7b). Diese befinden sich in unterschiedlichen Ausführungszuständen. Durch eine Schemaevolution wird Variante 1 zu Variante 1' weiterentwickelt (vgl. Abbildung 8.7c). Dazu wird Aktivität* `D` *in den echt alternativen Pfad hinter Aktivität* `C` *verschoben. Aktivitäten* `B` *und* `D` *können somit nicht mehr gemeinsam ausgeführt werden. Abbildung 8.7d zeigt nun zwei Instanzen von Variante 1', für welche die Änderungen bereits gelten. Die Prozessinstanzen 1, 2 und 3 jedoch laufen noch nach dem alten Schema von Variante 1.*

Entscheidet sich der Implementierer (vgl. hierzu das Rollenmodell in Abschnitt 4.3.2.6), die Änderungen nur für zukünftige Prozessinstanzen einer Variante zu übernehmen, so entsteht bei langlaufenden Prozessen der Nachteil, dass diese ggf. sehr lange nach einem veralteten Schema ablaufen. Wie in der Literatur berichtet ist dies meist nicht gewünscht [RRD04a]. Stattdessen müssen Prozessinstanzen an das neue Schema angepasst werden können, d.h. die

[3] Ein Schema entspricht einem Prozessmodell.

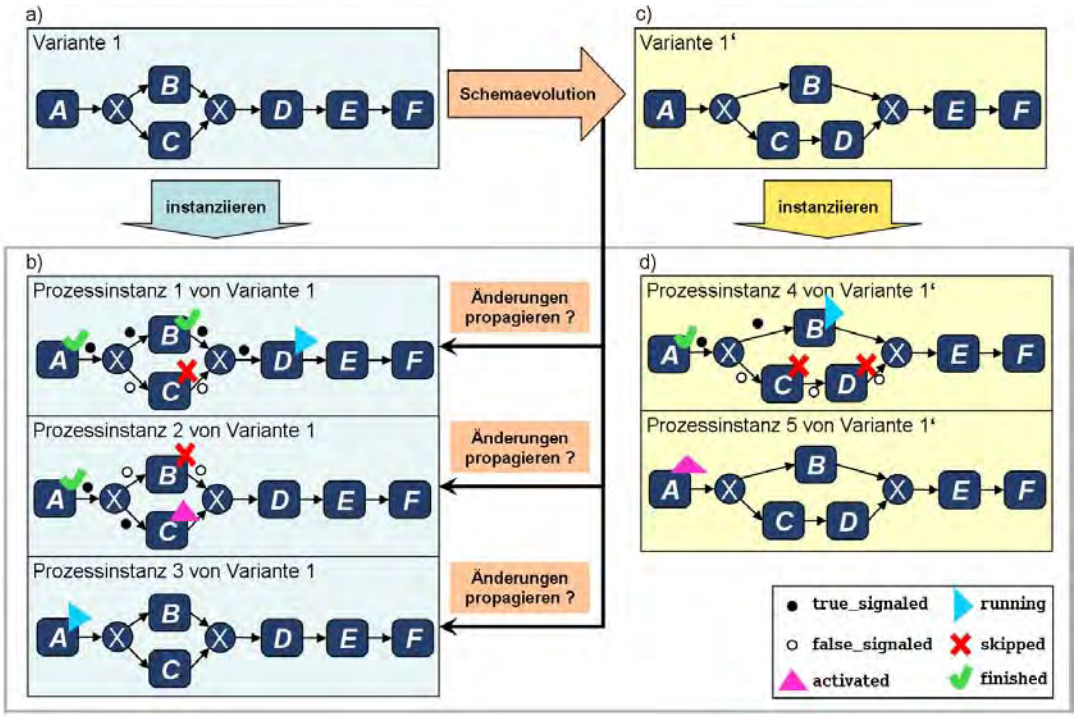


Abbildung 8.7: Propagieren von Änderungen nach einer Schemaevolution

Änderungen sind auf laufende Prozessinstanzen zu *propagieren*. Dies ist jedoch nicht trivial, wie Beispiel 8.8 zeigt.

Beispiel 8.8 (Propagieren von Änderungen nach einer Schemaevolution) *Nachdem Variante 1 zu Variante 1' abgeändert wird sollen die laufenden Prozessinstanzen 1, 2 und 3 von Variante 1 ebenfalls angepasst werden. Dazu muss die Änderung (d.h. Verschieben von Aktivität* `D`*) auf die Prozessinstanzen „propagiert" werden. Für die Prozessinstanzen 2 und 3 kann die Änderung problemlos übernommen werden, da der Ausführungszustand dieser Instanzen eine solche Änderung zulässt, d.h. es kann kein struktureller oder semantischer Fehler zur Laufzeit resultieren. Im Falle von Prozessinstanz 1 gilt dies nicht. Hier wird Aktivität* `D` *bereits ausgeführt, ohne dass zuvor Aktivität* `C` *ausgeführt worden ist. Ein Verschieben und Anpassen von* `D` *hätte daher einerseits zur Folge, dass Abhängigkeiten zwischen Aktivitäten* `C` *und* `D` *nicht erfüllt sind und andererseits, aufgrund der vorangegangenen Ausführung von* `B`*, keine korrekte Synchronisation am* `XORJoin`*-Knoten erfolgen kann. Das heißt, das Propagieren der Änderungen auf Prozessinstanz 1 ist nicht möglich.*

Beim Propagieren von Änderungen muss der Ausführungszustand der betreffenden Prozessinstanz berücksichtigt werden. Insbesondere können Änderungen nur dann übernommen werden, wenn daraus kein strukturell oder semantisch inkorrekte Prozessinstanz resultiert (vgl. Beispiel 8.8). Das heißt, es sind zusätzliche Korrektheitsprüfungen durchzuführen, bevor Änderungen propagiert werden können. Solche Korrektheitsprüfungen im Zusammenhang mit Schemaevolution und Propagieren von Änderungen, wurden im Rahmen des Forschungsprojekts ADEPT an der Universität Ulm im Detail untersucht und in diversen Arbeiten beschrieben. Wir vertiefen diese Themen, die zudem unabhängig vom Aspekt des Managements von Prozessvarianten betrachtet werden können, in dieser Arbeit nicht weiter, sondern verweisen auf einschlägige Arbeiten hierzu [Rin04, RRD03, RRD04c, RRD03, RWR06, DR09].

8.3 Refactoring von Prozessvarianten

Um die Handhabung einer Prozessfamilie zu verbessern, ist es sinnvoll, die Menge der modellierten Optionen und Aufsetzpunkte eines Basisprozesses erforderlichenfalls zu reduzieren bzw. neu zu strukturieren. In Provop führen wir daher sog. Refactorings durch, um die Struktur eines Basisprozesses und zugehörigen Optionen ggf. anzupassen, ohne dabei das Verhalten der konfigurierbaren Varianten zu beeinflussen (vgl. Anforderung 8.4 und 8.5). Ziel ist eine kompaktere und einfach pflegbare Beschreibung der verschiedenen Objekte. Wir betrachten die in Tabelle 8.2 aufgeführten Refactorings für Basisprozesse und Optionen.

Tabelle 8.2: Refactorings in Provop

Basisprozess
Refactoring 1: *Entfernen überflüssiger Aufsetzpunkte.*
Refactoring 2: *Vereinfachung des Basisprozesses mittels Vereinfachungsoperationen.*
Refactoring 3: *Neukonstruktion des Basisprozesses, so dass dieser anschließend eine minimale durchschnittliche Modelldifferenz zu all seinen Prozessvarianten aufweist.*
Refactoring 4: *Best Practices aus Optionen auf Basisprozess übertragen.*
Optionen
Refactoring 5: *Entfernen überflüssiger Optionen.*
Refactoring 6: *Neugruppieren von Änderungsoperationen zu Optionen.*
Refactoring 7: *Best Practices aus Änderungshistorien in Optionen erfassen.*

Bei Durchführung von Refactorings ist es wichtig, den Modellierer entsprechend einzubinden, um z.B. nicht Objekte automatisch zu löschen, die ggf. zukünftig relevant werden können oder die gezielt redundant modelliert worden sind. Wir generieren daher nur Vorschläge für mögliche Anpassungen und führen ein Refactoring nur bei expliziter Zustimmung des verantwortlichen Basisprozess-, Kontext- oder Optionsmodellierers durch.

Refactorings sind nicht immer trivial, da wir auch Objekte wie Aufsetzpunkte und Optionsbeziehungen berücksichtigen müssen. Sie bedürfen außerdem vor ihrer Umsetzung der Zustimmung des Modellierers. Dies ist erforderlich, da resultierende Änderungen ggf. umfangreich sind oder zu einem ungewollten Ergebnis führen könnten (z.B. wenn aktuell nicht verwendete, aber zukünftig relevante Aufsetzpunkte entfernt werden). Um bewerten zu können, ob ein Refactoring tatsächlich sinnvoll ist, existieren verschiedene Metriken, wie z.B. die Anzahl an Optionen, Änderungsoperationen und Optionsbeziehungen.

Im Folgenden stellen wir ausgewählte Techniken für das Refactoring von Prozessen vor. Wir beschreiben, wie das jeweilige Refactoring durchgeführt wird und welche korrigierenden Maßnahmen ggf. vorgenommen werden, um das Ausführungsverhalten der Prozesse und die Korrektheit der Prozessfamilie zu bewahren (vgl. Anforderung 8.6).

8.3.1 Refactoring 1: Entfernen überflüssiger Aufsetzpunkte

Die Menge der modellierten Aufsetzpunkte kann reduziert werden, indem nicht verwendete bzw. redundante Aufsetzpunkte aus dem Basisprozess entfernt werden. Das Erkennen redundanter Aufsetzpunkte ist nicht trivial. Für jeden Aufsetzpunkt muss geprüft werden, ob es einen anderen Aufsetzpunkt gibt, dessen Verwendung in den Änderungsoperationen zu Verhaltens-identischen Ergebnismodellen führt. Das heißt wir müssen prüfen, ob Aufsetzpunkte wechselseitig austauschbar sind. Ein wichtiger Aspekt betrifft die Parametrisierung der Aufsetzpunkte. So kann es ggf. gewünscht sein, redundante Aufsetzpunkte zu definieren,

die unterschiedlich parametrisiert sind und die sich daher bei `DELETE`- und `MOVE`-Operationen unterschiedlich verhalten (vgl. Abschnitt 5.4.5).

Im Gegensatz zu redundanten Aufsetzpunkten lassen sich nicht verwendete Aufsetzpunkte leicht durch Analyse der Änderungsoperationen identifizieren. Wird ein Aufsetzpunkt in keiner Änderungsoperation verwendet, wird er zur Konfiguration der Prozessfamilie auch nicht benötigt und kann prinzipiell entfernt werden. Beispiel 8.9 zeigt verschiedene Szenarien für nicht verwendete und redundante Aufsetzpunkte in einem Basisprozess.

Beispiel 8.9 (Überflüssige Aufsetzpunkte) *Abbildung 8.8a zeigt einen Basisprozess mit verschiedenen Aufsetzpunkten: Aufsetzpunkte* `Z` *und* `Y2` *werden von keiner der in Abbildung 8.8b angegebenen Optionen verwendet. Aufsetzpunkte* `Y1` *und* `Y2` *sind am gleichen Knoteneingang platziert und somit redundant. Darüber hinaus führt die Verwendung der Aufsetzpunkte* `V` *bzw.* `W` *und* `Y1` *bzw.* `X` *bei den gegebenen Optionen in Abbildung 8.8b, zu strukturell identischen Ergebnismodellen (vgl. Abbildung 8.8c und 8.8d). Beim Mapping der Anker auf Aufsetzpunkte in den* `INSERT`*-Operationen sind die referenzierten Aufsetzpunkte daher paarweise austauschbar. Den bereinigten Basisprozess zeigt Abbildung 8.8e.*

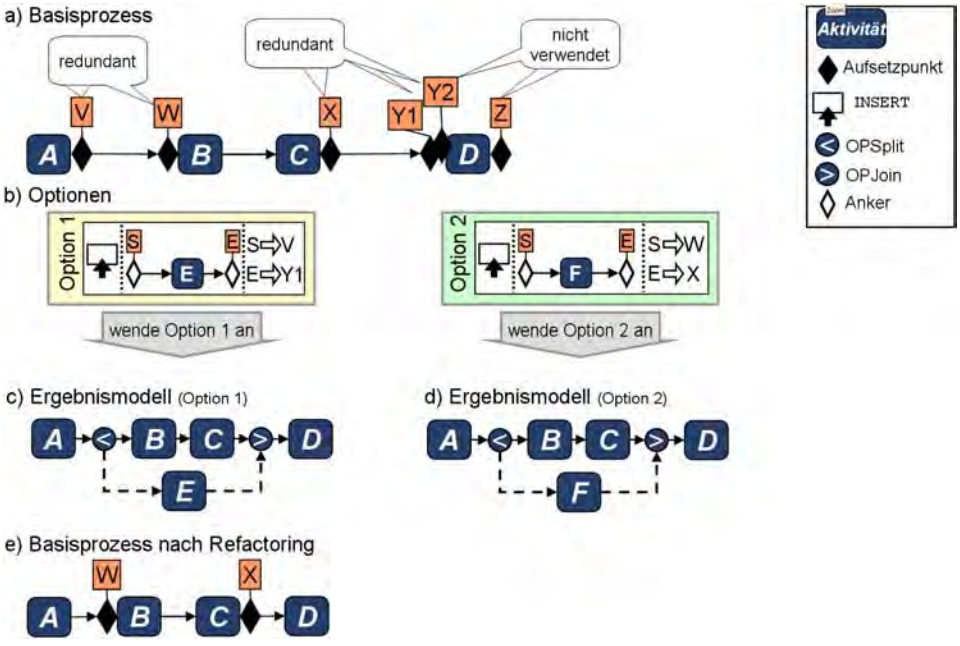


Abbildung 8.8: Nicht verwendete und redundante Aufsetzpunkte in einem Basisprozess

Refactoring durchführen: Nachdem wir redundante bzw. nicht verwendete Aufsetzpunkte identifiziert haben, kann das konkrete Löschen vom Modellierer angestoßen werden. Die Aufsetzpunkte werden automatisch aus dem Basisprozess entfernt.

Korrigierende Maßnahmen: Haben wir einen redundanten Aufsetzpunkt entfernt, passen wir die betroffenen Änderungsoperationen entsprechend an.

Bewertung: Die Reduktion modellierter Aufsetzpunkte ist ein guter Weg, um die Übersichtlichkeit und Handhabbarkeit des Provop-Ansatzes auch für sehr viele Prozessvarianten bzw. komplexe Basisprozesse zu gewährleisten.

8.3.2 Refactoring 2: Vereinfachung eines Basisprozesses

In Abschnitt 3.4 haben wir Vereinfachungsoperationen für Prozessmodelle vorgestellt. Diese gelten prinzipiell auch für Basisprozesse. Ein zu berücksichtigender Aspekt betrifft allerdings die Aufsetzpunkte an überflüssigen Strukturknoten. Hier gilt, dass diese Knoten nicht entfernt werden dürfen (vgl. Beispiel 8.10). Da bei einer Vereinfachung des Basisprozesses einzelne Knoten, an denen Aufsetzpunkte platziert sind, nicht entfernt werden dürfen, ist es sinnvoll, dieses Refactoring nach Entfernen überflüssiger Aufsetzpunkte durchzuführen. Auf diese Weise sind ggf. umfangreichere Vereinfachungen des Basisprozesses möglich.

Beispiel 8.10 (Vereinfachung eines Basisprozesses) *In Anlehnung an die bisher vorgestellten Vereinfachungsoperationen für normale Prozessmodelle und Ergebnismodelle (vgl. Abschnitt 3.4) zeigt Abbildung 8.9 exemplarisch, welche Vereinfachungen für einen Basisprozess durchgeführt werden können.*

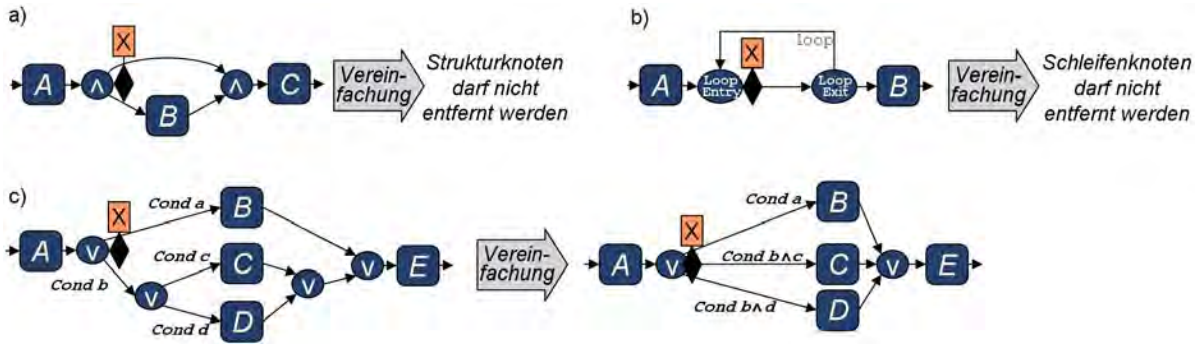


Abbildung 8.9: Vereinfachung eines Basisprozesses

Refactoring durchführen: Die (verhaltensbewahrenden) Vereinfachungsoperationen werden solange sukzessive auf den Basisprozess angewendet, bis keine weitere Vereinfachung möglich ist.

Bewertung: Die Vereinfachung von Basisprozessen reduziert die Anzahl der Knoten und Kanten eines Prozessmodells. Damit trägt dieses Refactoring dazu bei, die Komplexität der Prozessmodelle zu reduzieren und diese besser handhabbar zu machen.

8.3.3 Refactoring 3: Neukonstruktion des Basisprozess

In Abschnitt 5.2 haben wir fünf verschiedene Typen bzw. Vorgehensweisen zur Beschreibung des Basisprozesses vorgestellt: Standardprozess, „häufigste“ Prozessvariante, minimale durchschnittliche Distanz, Schnittmenge und Obermenge. Diese Typen lassen sich in zwei Gruppen einteilen: Basisprozesse, die für einen realen Anwendungsfall modelliert worden sind und solche, die ein konstruiertes Prozessmodell darstellen. Letztere sind für Optimierungen meist geeigneter, da keine Rücksicht auf einen konkreten Anwendungsfall, den der Basisprozess abdecken soll, genommen wird. Durch Wechsel von einem Basisprozess, der zur Abbildung eines spezifischen Anwendungsfall erstellt worden ist, zu einem konstruierten Basisprozess zur Bildung von Prozessvarianten, kann die Anzahl benötigter Änderungsoperationen bzw. Optionen ggf. reduziert werden.

Refactoring durchführen: Der Basisprozess wird auf Grundlage der bekannten Prozessvarianten so neu konstruiert, dass er die minimale durchschnittliche Modelldifferenz zu allen Prozessvarianten aufweist [LRWc08, LRWb08, LRWf09]. Über die Information hinaus, welche Prozessvarianten ausgeführt worden sind, gewinnen wir aus den Ausführungshistorien weitere

interessante Daten, wie z.B. die Ausführungshäufigkeit eines Variantenmodells. Diese Information dient als Gewichtung bei der Berechnung des Basisprozesses. Das heißt Prozessvarianten, die sehr häufig ausgeführt werden, haben einen stärkeren Einfluss auf die Gestaltung des Basisprozesses als solche, die eher selten ausgeführt werden [LRWc08, LRWb08, LRWf09].

Korrigierende Maßnahmen: Durch diesen Ansatz wird der Basisprozess ggf. stark verändert, d.h. es müssen viele Optionen angepasst bzw. neue Optionen erstellt werden.

Bewertung: Der Vorteil eines auf diese Weise konstruierten Basisprozesses besteht darin, dass zur Abbildung aller Prozessvarianten eine minimale Anzahl an Änderungsoperationen erforderlich ist. Nachteilig ist, dass aufwendige Analysen und Berechnungen zur Ermittlung der minimalen durchschnittlichen Modelldifferenz durchgeführt werden müssen. Darüber hinaus sind ggf. umfangreiche Transformationen des Basisprozesses erforderlich, die dann zu ebenfalls sehr umfangreichen Änderungen aller Optionen führen können.

8.3.4 Refactoring 4: Best Practices aus Optionen auf den Basisprozess übertragen

Eine Quelle zur Identifikation von „Best Practices“ bilden modellierte Optionen des Basisprozesses eines bestimmten Prozesstyps. So können Optionen, die zur Bildung (fast) aller Prozessvarianten verwendet werden, direkt auf den Basisprozess übertragen werden. Das heißt die Änderungsoperationen werden als „Best Practices“ in den Basisprozess eingearbeitet.

Refactoring durchführen: Die in der Option beschriebenen Änderungen werden direkt auf den Basisprozess übertragen.

Korrigierende Maßnahmen: Die übertragene Option wird aus der Optionslandschaft entfernt. Existieren Prozessvarianten, für die dieses Best Practice nicht durchgeführt werden soll, muss eine inverse Option definiert werden. Die Kontextbedingung dieser Option ist dann entsprechend der Kontexte der Ausnahmefälle zu definieren.

Bewertung: Die Anzahl der Optionen kann reduziert werden. Darüber hinaus wird die Modelldifferenz des Basisprozesses zu allen Prozessvarianten verkleinert.

8.3.5 Refactoring 5: Entfernen überflüssiger Optionen

Bei Konfiguration einer Prozessfamilie können wir prüfen und dokumentieren, welche Optionen bisher nicht zur Bildung eines Ergebnismodells auf den Basisprozess angewendet werden. Diese sind ggf. nicht relevant und können bei Zustimmung des Modellierers entfernt werden. Das Erkennen redundanter Optionen ist nicht trivial. So müssen wir für Aufsetzpunkt-basierte Änderungsoperationen paarweise prüfen, ob nach ihrer Anwendung auf den Basisprozess identische Ergebnismodelle resultieren. Id-basierte Änderungsoperationen werden direkt verglichen. Darüber hinaus müssen wir Optionen nicht nur paarweise vergleichen (bzw. die resultierenden Ergebnismodelle), sondern auch alle Kombinationen von Optionen.

Refactoring durchführen: Nachdem wir redundante bzw. nicht verwendete Optionen identifiziert haben, sind diese dem Variantenmodellierer gegenüber anzuzeigen. Dieser kann das konkrete Löschen anstoßen. Daraufhin werden die Optionen (automatisch) aus der Menge der modellierten Optionen entfernt. Dies inkludiert auch das Entfernen der zugehörigen Optionsbeziehungen.

Korrigierende Maßnahmen: Die Kontextbedingungen und Optionsbeziehungen der verbleibenden Optionen werden entsprechend angepasst.

Bewertung: Dieses Refactoring reduziert die Anzahl an Optionen, Änderungsoperationen und Optionsbeziehungen und ist nötig, um die Komplexität der Optionslandschaft zu reduzieren. Dies wiederum verbessert die Handhabbarkeit der Prozessvarianten.

8.3.6 Refactoring 6: Neugruppierung von Änderungsoperationen zu Optionen

Das „Neugruppieren" von Änderungsoperationen zu anderen Optionen stellt ein weiteres Refactoring dar. In Abschnitt 5.5 haben wir erläutert, dass Änderungsoperationen sowohl feingranular als auch grobgranular zu Optionen gruppiert werden können. Dabei können wir hinsichtlich minimaler Redundanz der Änderungsoperationen und Anzahl an Optionen, eine optimale Granularität finden. Eine solche Optimierung kann entweder durch Aufteilen der Änderungsoperationen auf mehrere Optionen (*Verfeinerung der Granularität*) oder durch Zusammenfassen zu wenigeren Optionen (*Vergröberung der Granularität*) erreicht werden.

Verfeinerung der Granularität

Eine feinere Granularität bei der Festlegung der Optionen ist sinnvoll, wenn Änderungsoperationen (mit gleicher Parametrisierung) redundant über mehrere Optionen verteilt vorkommen (vgl. Beispiel 8.11). Dazu müssen wir herausfinden, ob identische Änderungsoperationen vorliegen, indem wir Änderungsoperationen gleichen Typs miteinander vergleichen. Zudem müssen wir prüfen, ob mehrere Änderungsoperationen kombiniert das gleiche tun wie eine dritte Änderungsoperation bzw. die Kombination mehrerer anderer Änderungsoperationen (z.B. ist eine Option X mit `INSERT A` und `MOVE A` identisch zu einer Option Y mit `INSERT A` an der Zielpositition der `MOVE`-Operation von Option X).

Refactoring durchführen: Ist eine feinere Granularität erforderlich, entfernen wir die zuvor identifizierten, redundanten Änderungsoperationen und bilden aus ihnen eine neue Option. Diesen weisen wir dann eine Kontextbedingung zu, welche sich aus den Kontextbedingungen der ursprünglichen Optionen ergibt. Des weiteren definieren wir entsprechende Optionsbeziehungen, um die korrekte Anwendung der Optionen nach der Verfeinerung sicherzustellen. Diese Vorgehensweise erläutern wir in Beispiel 8.11.

Beispiel 8.11 (Verfeinerung der Granularität) *Abbildung 8.10a zeigt Änderungsoperationen, die redundant in verschiedenen Optionen auftreten. Durch Neugruppieren dieser Änderungsoperationen zu neuen Optionen, kann diese Redundanz vermieden werden (vgl. Abbildung 8.10b): Die immer gemeinsam angewendeten Änderungsoperationen* `DELETE A` *und* `MODIFY B` *werden in einer neuen Option 3 gruppiert. Deren Kontextbedingung ergibt sich aus der logischen OR-Verknüpfung der originalen Kontextbedingungen. Da wir Option 3 immer in Kombination mit Option 1 bzw. 2 verwenden müssen, beschreiben wir entsprechende Implikationsbeziehungen. Diese Beziehungen sind jedoch nicht ausreichend, um die ursprünglichen Kombinationsmöglichkeiten der Optionen abzubilden. Zusätzlich muss eine Auswahl-Beziehung definiert werden, die besagt, dass mindestens zwei Optionen aus der Menge der verfeinerten Optionen und der neu erzeugten Option angewendet werden müssen.*

Weitere Aspekte: Bei der Verfeinerung von Optionen ist die Reihenfolge der Änderungsoperationen zu beachten (vgl. Beispiel 8.12). So kann das Ausgliedern bestimmter Änderungsoperationen zu einer neuen Option dazu führen, dass wir die Änderungsoperationen in einer geänderten Reihenfolge anwenden müssen. Dies ist unkritisch, wenn diese in der vorliegenden Kontextbeschreibung kommutativ sind. Andernfalls kann dies zu semantischen und strukturellen Fehlern führen. Wir müssen daher vor Verfeinerung prüfen, ob diese korrekt durchführbar ist, d.h. ob die Änderungsoperationen kommutativ sind. Die semantische

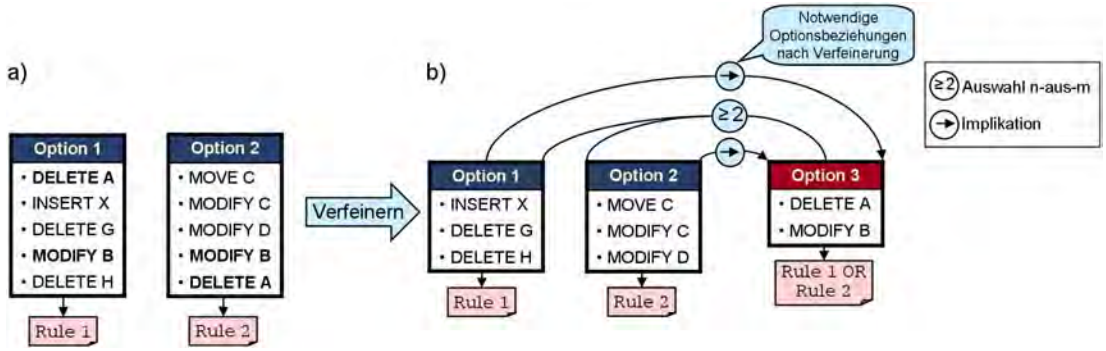


Abbildung 8.10: Redundanzen vermeiden durch feinere Granularität

Korrektheit wird durch den Variantenmodellierer geprüft, der die Verfeinerung freigeben muss.

Beispiel 8.12 (Reihenfolge der Änderungsoperationen nach einer Verfeinerung) *In Abbildung 8.10a verfeinern wir Option 1 und 2, indem wir die Änderungsoperationen* `DELETE A` *und* `MODFIY B` *in eine neue Option auslagern. Dabei müssen wir für die Änderungsoperationen in Option 3 eine bestimmte Reihenfolge angeben. Des Weiteren führt die Verfeinerung dazu, dass bei gemeinsamer Anwendung der Optionen 1 und 3 die ausgelagerten Änderungsoperationen erst nach Option 1 angewendet werden. Diese beiden Aspekte führen ggf. zur Verletzung struktureller und semantischer (Reihenfolge-)Abhängigkeiten.*

Eine fundamentale Fragestellung bei der Verfeinerung von Optionen ist ferner, ob bisherige Optionsbeziehungen auf die neue Option übertragen werden müssen oder nicht. Generell übertragen wir nur Reihenfolgebeziehungen auf die neue Option.[4] Alle anderen Abhängigkeiten sind bereits durch die bei der Verfeinerung erzeugten Implikations- und Auswahl-Beziehungen abgedeckt (vgl. Beispiel 8.13).

Beispiel 8.13 (Übertragen von Optionsbeziehungen nach Verfeinerung) *Abbildung 8.11 zeigt Optionen und zugehörige Optionsbeziehungen. Nach Verfeinerung sind die Änderungsoperationen* `DELETE A` *und* `MODFIY B` *in der neuen Option 4 gruppiert, und es existieren entsprechende Implikationsbeziehungen. Übertragen wir nun den wechselseitigen Ausschluss der ursprünglichen Optionen 1 und 3 auf Option 4, erhalten wir eine inkorrekte Beziehungsstruktur. Option 4 wäre in diesem Fall gar nicht gemeinsam mit den Optionen 1 und 3 anwendbar. Es ist daher nicht sinnvoll diese Optionsbeziehung zu übertragen. Anders verhält es sich jedoch mit den Reihenfolgebeziehungen. Diese müssen übertragen werden, um das Einfügen der Aktivität* `B` *durch Option 4 vor dem Ändern dieser Aktivität durch Option 2 zu gewährleisten.*

Bewertung: Die Verfeinerung der Granularität führt zu mehr Optionen, allerdings wird die Anzahl der Änderungsoperationen reduziert und deren Wiederverwendung erhöht.

Vergröberung der Granularität

Eine gröbere Granularität ist sinnvoll, wenn Änderungsoperationen, die immer gemeinsam angewendet werden, in getrennten Optionen vorliegen. Um solche Fälle zu erkennen, analysieren wir die verknüpften Kontextbedingungen der Optionen (vgl. Beispiel 8.14).

[4] Hierarchiebeziehungen, die eine Kombination aus Reihenfolge- und Implikationsbeziehung darstellen, werden ebenfalls auf die neue Option übertragen, allerdings nur in Form einer Reihenfolgebeziehung; das Übertragen von Implikationsbeziehung kann zu Fehlern führen.

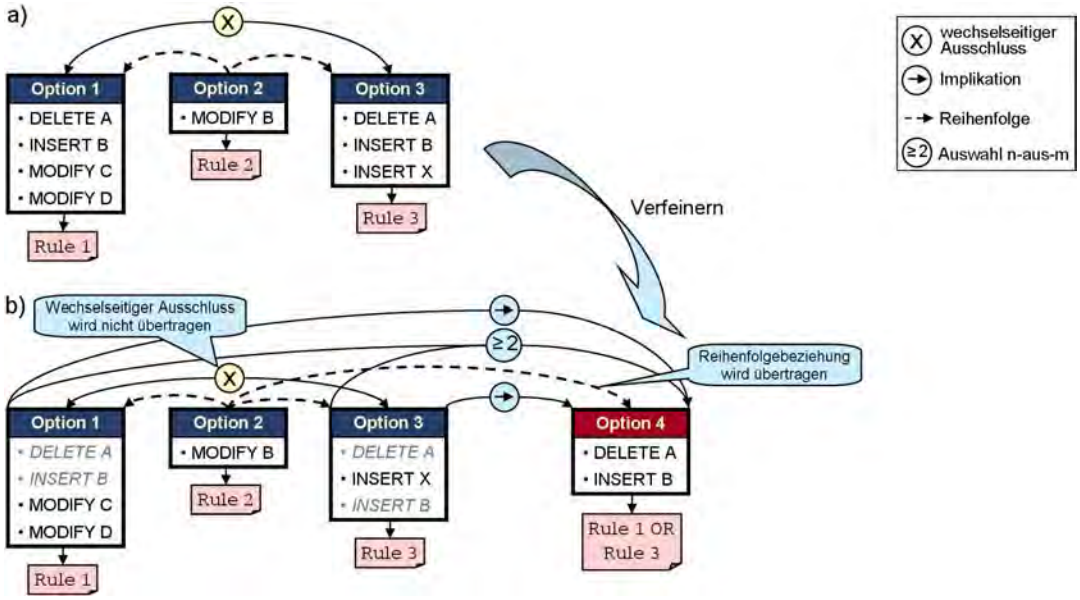


Abbildung 8.11: Übertragen von Beziehungen nach Verfeinerung

Refactoring durchführen: Soll eine gröbere Granularität realisiert werden, fassen wir die gemeinsam angewandten Optionen zusammen und übertragen ggf. ihre Kontextbedingungen auf die neue Option. Alte Optionsbeziehungen werden ebenfalls auf die neue Option übertragen, da durch die gemeinsame Gültigkeit im gleichen Kontext kein Widerspruch der Optionsbeziehungen entstehen kann. Beispiel 8.14 zeigt diese Vorgehensweise.

Beispiel 8.14 (Vergröberung der Granularität) *Bei Untersuchung der Optionen in Abbildung 8.12a fällt auf, dass manche von ihnen immer im gleichen Kontext ihre Anwendung erfahren (z.B. Option 1 und 2). Durch Zusammenfassen dieser Optionen kann die Gesamtzahl modellierter Optionen reduziert werden. Konkret erstellen wir für die zusammengefassten Optionen jeweils eine neue Option, welche die Kontextbedingung und Optionsbeziehungen der ursprünglichen Optionen übernimmt. Das Ergebnis dieses Refactorings, mit entsprechenden Anpassungen der Kontextbedingungen und Optionsbeziehungen, zeigt Abbildung 8.12b.*
Wir haben an dieser Stelle ein einfaches Beispiel mit identischen Regeln gewählt. Es ist natürlich auch möglich, Optionen mit unterschiedlichen Kontextbedingungen, welche semantisch identisch sind, zusammenzufassen. Solche Optionen bzw. Kontextbedingungen sind durch Analyse von Ausführungshistorien erkennbar.

Bewertung: Die Vergröberung der Granularität reduziert die Anzahl an Optionen ohne redundante Änderungsoperationen zu erzeugen. Dieses Refactoring kann dazu beitragen, die Komplexität der Optionslandschaft zu reduzieren.

8.3.7 Refactoring 7: Best Practices aus Änderungshistorien in Optionen erfassen

Änderungshistorien (engl. *Change Log*) protokollieren zur Laufzeit jede (ungeplante) Ad-hoc Änderung an einer Prozessinstanz [RRJK06]. Liegt eine gewisse Anzahl an solchen Änderungshistorien vor, können wir mit Hilfe von *Process Mining*-Techniken untersuchen, welche Änderungen z.B. sehr häufig vorgenommen wurden und daher „Best Practices“ darstellen. Diese Best Practices können dann in Optionen erfasst werden. Vorteilig ist, dass wir Variantenmodelle so anpassen können, dass Anpassungen zur Laufzeit zukünftig reduziert werden

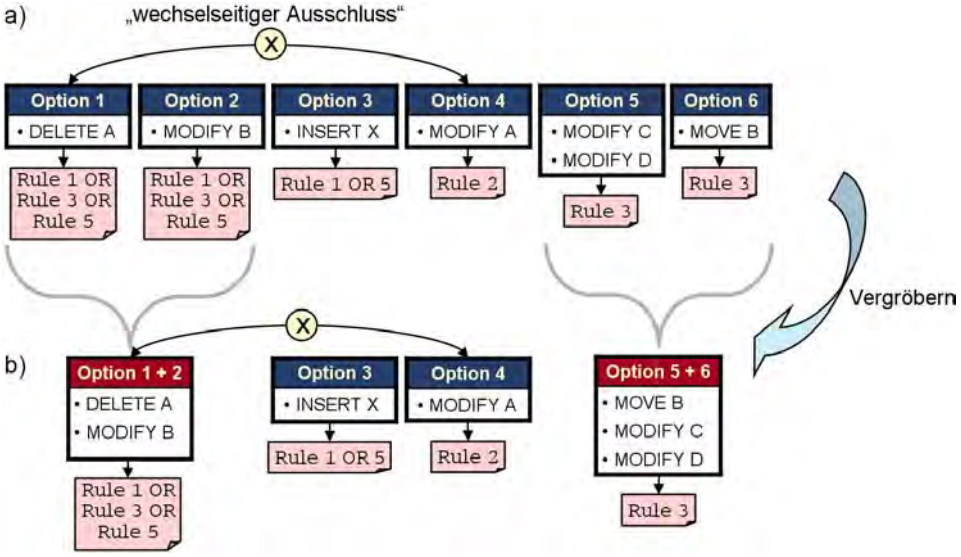


Abbildung 8.12: Anzahl Änderungsoperationen reduzieren durch gröbere Granularität

können. Nachteilig ist, dass die Änderungshistorien ggf. nur bestimmte Änderungen protokollieren und somit nicht immer vollständig und aussagekräftig sind. Außerdem muss zunächst eine gewisse Anzahl an Ausführungen protokolliert werden, um zulässige und fundierte Aussagen zu gewinnen.

Refactoring durchführen: Die identifizierten Best Pracices werden als Änderungsoperationen in einer Option gruppiert und erhalten entsprechende Kontextbedingungen (vgl. Beispiel 8.15). Alle bisherigen Varianten bleiben weiter konfigurierbar.

Beispiel 8.15 (Auswertung von Änderungshistorien) *Abbildung 8.13 zeigt fünf verschiedene Änderungshistorien für Prozessinstanzen eines spezifischen Variantenmodells. Bei Untersuchung dieser Änderungshistorien fällt auf, dass vier der dargestellten Prozessinstanzen zur Laufzeit in der gleichen Art und Weise angepasst worden sind; es wurden jeweils die Änderungen* `INSERT Aktivität E` *und* `DELETE Aktivität G` *gemeinsam angewendet. Aufgrund der Häufigkeit dieser Änderungen können diese als „Best Practices“ bezeichnet werden. Des Weiteren kann festgestellt werden, dass diese Best Practices für einen bestimmten Kontext (*`Wartung = ja UND kritisch = ja`*) erforderlich werden. Wir können nun ein Refactoring durchführen und die identifizierten Best Practices als Änderungsoperationen in einer neuen Option definieren (vgl. Abbildung 8.13). Die Kontextbedingung dieser Option ergibt sich dann aus der zuvor identifizierten Kontextbeschreibung.*

Bewertung: Durch Erfassung und Modellierung neuer Optionen, auf Basis der Analyse von Änderungshistorien, können neue bzw. geänderte Prozessvarianten identifiziert und schnell abgebildet werden. Problematisch ist, dass andere Zusammenhänge, welche die Änderungen erforderlich machen, ggf. nicht aus den Änderungshistorien abgelesen werden können. Somit können fälschlicherweise neue Optionen für einen gegeben Kontext definiert werden. Da die Übernahme fachlicher Änderungen mit dem Variantenmodellierer abgestimmt wird, kann an dieser Stelle eine semantische Prüfung durch ihn vorgenommen werden. Er kann entscheiden, ob er Best Practices übernehmen möchte. Ein weitergehender Ansatz, der Techniken des Fallbasierten Schließens sowie domänenspezifischer Kontextmodelle nutzt, bietet ProCycle [WWRD07, WRWRM09].

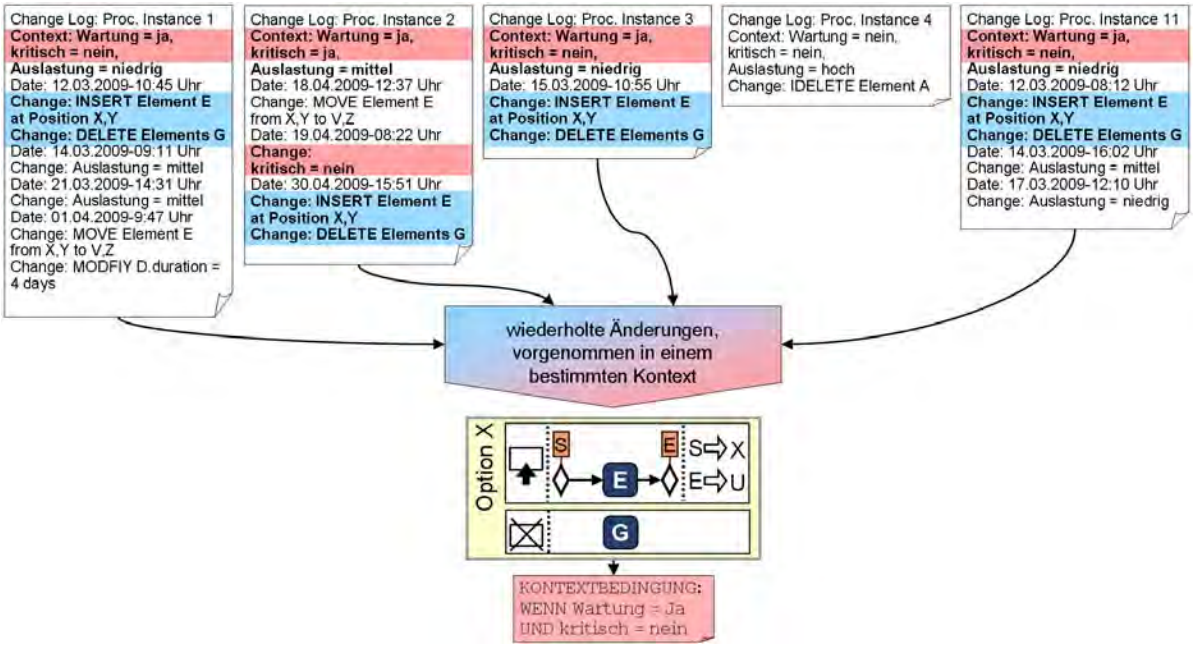


Abbildung 8.13: Auswertung von Änderungshistorien

8.4 Diskussion

In der Literatur wird die Weiterentwicklung und Optimierung von Prozessmodellen bereits umfassend diskutiert. Die im Kontext von Provop als relevant erachteten Ansätze werden im Folgenden vorgestellt und diskutiert.

Ansätze zur Evolution und Versionierung von Prozessmodellen

In *ADEPT* [Dad05, RRD03, Rin04] können Prozessmodellierer das Prozessschema auf Typebene durch Hinzufügen, Löschen oder Verschieben von Aktivitäten sowohl statisch als auch dynamisch adaptieren. Insbesondere können die vorgenommenen Änderungen (partiell) auf laufende Prozessinstanzen propagiert werden. Beim Propagieren werden auch Korrektheitsaspekte berücksichtigt. So werden z.B. nur solche Änderungen propagiert, die mit dem aktuellen Ausführungszustand einer Prozessinstanz verträglich sind [RRD03, RRD04a, RRD04c, RRD04b].

Ein anderer Ansatz ist im WfMS *ULTRAflow* implementiert [FRF01]. Hier werden Änderungen am Prozessschema durch ein sog. *Multi-Version Concurrency Control Protocol* gehandhabt. Die spezielle Modellierungssprache erlaubt dazu eine Unterteilung in mehrere Sub-Workflows. Für jede Workflow-Instanz kann für einzelne Sub-Workflows auf eine neue Version umgestiegen werden. Des Weiteren beschreibt [FRF01], wie dieser Ansatz in einem verteilten System eingesetzt werden kann. Ähnliche Ansätze für die Prozessschemaevolution bieten *WIDE* [CCPP96], *WASA2* [VW99, Wes01] und *Mokassin* [JWH00].

Ansätze zum Refactoring von Prozessvarianten

Im Bereich der Softwareentwicklung wird das Refactoring eingesetzt, um die Code-Qualität von Programmen zu verbessern [Opd92, Fow00, Bec00]. Dabei werden Redundanzen im Code entfernt und allgemein die Lesbarkeit verbessert sowie das Design der Software vereinfacht. Die hier entwickelten Ansätze sind (teilweise) auf Prozessmodelle übertragbar. In [WR08] werden dazu verschiedene Refactorings vorgestellt, die sich auf Prozessbäume (d.h. Hierarchien

von Prozessmodellen), Prozessvarianten und die Evolution von Prozessmodellen beziehen. Diese Maßnahmen werden in ähnlicher Form auch in Provop adressiert.

Ein wichtiger Aspekt des Refactoring von Prozessmodellen ist die Wahrung des Verhaltens eines Prozessmodells. Das heißt, die strukturellen Anpassungen dürfen nicht zu einem anderen Ausführungsverhalten führen [WR08]. Der wesentliche Vorteil eines Refactoring ist in der Komplexitätsreduktion der Prozessmodelle zu sehen. [WR08] bewertet daher Refactoring anhand spezifischer Metriken, etwa die Anzahl der Knoten, Verschachtelungstiefe bei hierarchischen Prozessmodellen, Kontrollfluss-Komplexität und Modelldifferenz.[5]

Ansätze für das Monitoring und Mining von Varianten

Das Prozess-Monitoring erlaubt es, die Ausführung aller Prozessinstanzen zu überblicken und zu überwachen [WSR09]. Zur besseren Nachvollziehbarkeit werden die Daten in entsprechenden Ausführungshistorien gespeichert. Diese dokumentieren den aktuellen Status und das Ausführungsverhalten eines Prozesses. Eine Weiterentwicklung dieser klassischen Ausführungshistorien stellen Änderungshistorien dar. Diese protokollieren auch alle zur Laufzeit vorgenommenen Änderungen an einer Prozessinstanz [RRJK06, GRRvdA06, GRMR+08]. Durch Auswertung von Ausführungs- und Änderungshistorien können Anpassungs- und Optimierungsempfehlungen für die Prozessmodelle abgleitet werden [LRWc08, RJR07]. Dabei können verschiedene Prozess-Mining-Techniken eingesetzt werden wie Multi-Phase Mining, genetisches Mining oder Cluster-Techniken [vdAGRR06, GRRvdA06, GvdAJV08].

Im *MinAdept*-Projekt werden verschiedene Prozess-Mining-Techniken vorgestellt, um einen Referenzprozess zu identifizieren. Dazu dient eine Menge von Prozessgraphen (= Prozessvarianten) mit Gewichtung als Input. Hieraus wird ein Referenzprozessmodell algorithmisch bestimmt, das eine minimale durchschnittliche Distanz zu den Varianten aufweist [LRWc08, LRWb08, LRWa08]. Die Distanz wird gemessen durch die Anzahl höherwertiger Änderungsoperationen (z.B. Einfügen, Verschieben), die nötig sind, um das Quell- ins Zielmodell zu transformieren [LRWe08]. Die Algorithmen betrachten sowohl die Situation, dass bereits ein Referenzprozessmodell vorliegt [LRWa08] als auch den Fall, dass kein solches Modell bekannt ist. Ein Szenario des Ansatzes sind Varianten, die als Ad-hoc-Abweichungen auf Instanzebene hervorgegangen sind. Ein weiteres Szenario ist die Neukonfiguration eines Basisprozesses in Provop, so dass eine minimale Anzahl Optionen bzw. Änderungsoperationen erforderlich sind, um die Prozessfamilie erzeugen zu können.

8.5 Zusammenfassung

In Kapitel 8 haben wir beschrieben, wie zur Anpassung von Prozessvarianten aufgrund geänderter fachlicher Anforderungen der Basisprozess, die Optionen oder das Kontextmodell einer Prozessfamilie geändert werden können. Zwischen diesen Objekten existieren Abhängigkeiten, die berücksichtigt werden müssen, um die Korrektheit der Prozessfamilie zu erhalten. Gegebenenfalls auftretende Inkonsistenzen werden dem Modellierer angezeigt.

Kapitel 8 hat weiter Refactorings für die Optimierung von Basisprozess und zugehörigen Optionen diskutiert. Unter Beachtung der spezifischen Eigenschaften dieser Objekte stellen wir verschiedene Refactorings vor, wie z.B. die Vereinfachung des Basisprozesses oder das Entfernen redundanter Aufsetzpunkte und Optionen. Auf diese Weise können wir Redundanzen minimieren und eine einfachere Handhabung der Prozessfamilie erzielen.

[5] Für weitere Details zum Thema Metriken von Prozessvarianten verweisen wir auf [RV04, GL06, Car08, Men08, VRM+08].

9

Darstellung von Prozessvarianten und Interaktionsformen

In den vorangehenden Kapiteln haben wir den Provop-Lösungsansatz entlang des Prozess-Lebenszyklus vorgestellt. Im Folgenden gehen wir auf phasenübergreifende Anforderungen ein, insbesondere die Darstellung von Optionen und Ergebnismodelle betreffend (vgl. Kapitel 2). Abschnitt 9.1 motiviert phasenübergreifende Aspekte, wie Darstellung und Benutzerfreundlichkeit. Abschnitt 9.2 beschreibt verschiedene Darstellungskonzepte für Optionen und Ergebnismodelle, während Abschnitt 9.3 ausgewählte Aspekte zur einfachen Handhabung des Provop-Ansatzes vorstellt. Abschnitt 9.4 diskutiert verwandte Ansätze. Wir schließen mit einer Zusammenfassung in Abschnitt 9.5.

9.1 Motivation und Anforderungen

Zur adäquaten Visualisierung von Prozessvarianten reicht eine Darstellung der Variantenmodelle, die für einen spezifischen Kontext konfiguriert werden, allein nicht aus. Darüber hinaus bedarf es geeigneter Konzepte für die Darstellung von Basisprozessen und zugehörigen Optionen. Diese Darstellungskonzepte müssen Aspekte wie Informationsfülle und Positionierung von Optionen in Relation zum Basisprozess berücksichtigen.

Bei der Darstellung von Ergebnismodellen (d.h. konfigurierten Variantenmodellen) sind Ansichten hilfreich, die einen Vergleich verschiedener Variantenmodelle erlauben. Beispielsweise sollte es möglich sein, die zur Ableitung eines bestimmten Ergebnismodells angewandten Optionen oder die Differenz (d.h. Distanz) zwischen Basisprozessmodell und Variantenmodell hervorzuheben [LRWe08].

Weiter sollte der Umgang mit dem Provop-Ansatz möglichst einfach und intuitiv gestaltet werden. Für Einzelaspekte haben wir dies bereits diskutiert (z.B. Intuitivität des Kontextmodells). Noch nicht betrachtet haben wir die Handhabung komplexer Änderungsoperationen, die Modellierungsmethodik für Optionen sowie die benutzerfreundliche Modellierung von Optionsbeziehungen.

Daraus ergeben sich im Wesentlichen die in Tabelle 9.1 aufgeführten Anforderungen.

Tabelle 9.1: Anforderungen an die Darstellung von und Interaktion mit Prozessvarianten

Darstellung von Prozessvarianten
Anforderung 9.1 *Informationen (z.B. Parameter von Änderungsoperationen) müssen flexibel ein- und ausgeblendet werden können.*
Anforderung 9.2 *Optionen müssen in Relation zum Basisprozess positioniert werden können.*
Anforderung 9.3 *Die verschiedenen Optionen bzw. Änderungsoperationen, die zur Ableitung auf den Basisprozess angewendet werden, müssen innerhalb des Ergebnismodells hervorgehoben werden können.*
Anforderung 9.4 *Die Differenz zwischen Basisprozess und Ergebnismodell muss angezeigt werden können.*
Anforderung 9.5 *Variantenmodelle der selben Prozessfamilie müssen miteinander verglichen werden können.*
Endbenutzerinteraktion
Anforderung 9.6 *Komplexe Änderungen und Änderungsmuster (d.h. wiederkehrende Kombinationen von Änderungsoperationen) müssen einfach und intuitiv modelliert werden können.*
Anforderung 9.7 *Optionsbeziehungen müssen übersichtlich und einfach modelliert werden können.*

9.2 Darstellungsaspekte

Die Darstellung von Basisprozess, zugehörigen Optionen und konfigurierten Ergebnismodellen sollte übersichtlich und möglichst intuitiv sein, um das Modellieren und die Analyse von Prozessvarianten zu erleichtern. Dies ist insbesondere bei Vorliegen einer großen Anzahl von Optionen essentiell. In diesem Fall muss der dargestellte Informationsgehalt „reguliert“ werden können, indem Informationen flexibel ein- bzw. ausgeblendet werden. Des Weiteren müssen die Ergebnismodelle der Prozessvarianten leicht vergleichbar sein, so dass Unterschiede und Gemeinsamkeiten der Varianten einer Prozessfamilie leicht identifiziert werden können. Dazu sind spezielle Sichten auf ausgewählte Varianten zu generieren. Generell ist zu fordern, dass zu jeder Zeit ein Wechsel der Darstellungsform von Basisprozess, Optionen und Ergebnismodellen möglich sein sollte.

9.2.1 Darstellung von Optionen

Eine wichtiges Objekt des Provop-Ansatzes ist die *Option*. Für diesen Objekttyp müssen eigenständige Visualisierungskonzepte entwickelt werden, die sich weitestgehend an der Darstellung bekannter Prozesselemente orientieren sollen (vgl. Anforderung 9.2). Die wesentlichen Inhalte einer Option und die Darstellung von Änderungsoperationen mit Symbolen, Ankern und Aufsetzpunkten haben wir in den vorangehenden Kapiteln bereits eingeführt. Im Folgenden behandeln wir nun wichtige Darstellungsaspekte, etwa den Informationsgrad der Anzeige und die Positionierung von Optionen in Relation zum Basisprozess betreffend. Abbildung 9.1 gibt eine Übersicht der Darstellungsformen.

9.2.1.1 Ein- und Ausblenden von Informationen

Ein Aspekt der Darstellung betrifft die Fülle der angezeigten Informationen, d.h. dem Variantenmodellierer soll eine flexible Anzeige von Inhalten ermöglicht werden. Konzepte zur Einschränkung angezeigter Informationen sollten intuitiv und einfach handhabbar sein (vgl.

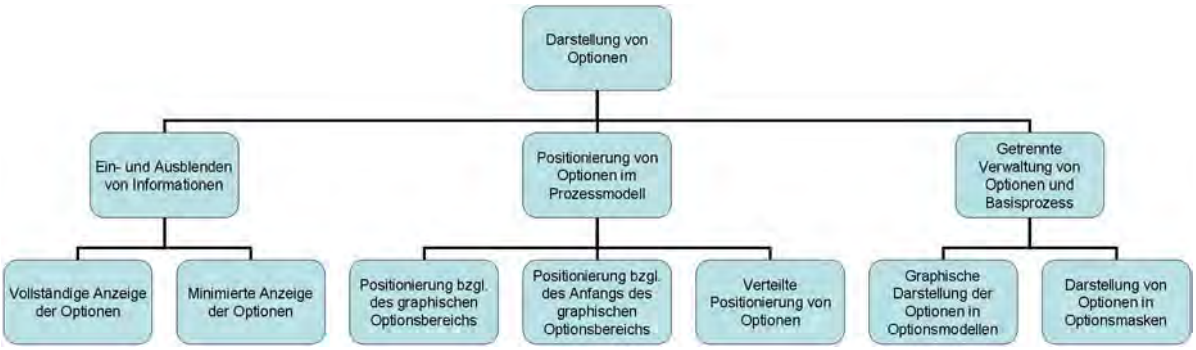


Abbildung 9.1: Darstellungsformen von Optionen

Anforderung 9.1). In Provop kann der Modellierer zwischen verschiedenen Anzeigemodi wählen, die jeweils einen unterschiedlichen Informationsgehalt besitzen:

Anzeigemodus 1 (Vollständige Anzeige der Optionen): Generell sollte es möglich sein, alle Optionen bzw. deren Änderungsoperationen vollständig anzuzeigen. Dadurch erhält der Variantenmodellierer eine detaillierte Sicht. Jedoch wird die Darstellung durch die Vielzahl an Informationen ggf. überladen, da irrelevante Informationen nicht ausgeblendet werden. Dies ist problematisch, wenn viele Optionen vorliegen.

Anzeigemodus 2 (Minimierte Anzeige der Optionen): Ein anderer Ansatz ist die minimierte Anzeige von Optionen, bei der nur bestimmte Informationen dargestellt werden. Zu einer solchen Minimierung kann entweder eine Grundeinstellung vorgegeben werden oder der Variantenmodellierer definiert selbst, welche Information angezeigt werden soll. Dadurch lassen sich irrelevante Daten ausblenden. Das Prozessmodell ist somit übersichtlicher und verständlicher.

Generell sollte ein Umschalten zwischen den beschriebenen Alternativen (vollständige bzw. minimierte Anzeige) möglich sein. In Provop kann zwischen *globalem* und *lokalem* Umschalten gewählt werden (vgl. Beispiel 9.1). Ersteres erlaubt es, alle Optionen gleichzeitig von einer vollständigen Anzeige in eine minimierte Anzeige zu überführen und umgekehrt. Beim lokalen Umschalten kann der Variantenmodellierer für jede Option die gewünschte Informationstiefe separat vorgeben. Die Darstellung nach globalem oder lokalem Umschalten wird in Provop automatisch und korrekt erzeugt. Dabei werden Usability-Aspekte berücksichtigt, etwa das Beibehalten der Position der Elemente auf dem Bildschirm (vgl. „Mental Map“ [MELS95]).

Beispiel 9.1 (Anzeigemodi von Optionen) *Abbildung 9.2a zeigt ein Beispiel für die vollständige Anzeige von Optionen, während in Abbildung 9.2b bestimmte (benutzerdefinierte) Informationen der Optionen 1 und 2 ausgeblendet sind; es werden nur noch Optionsnamen und Änderungsoperationstypen angezeigt. Ein Wechsel der Anzeige von Abbildung 9.2a zu 9.2b bezeichnen wir als* globales Umschalten*, da alle Optionen betroffen sind. In Abbildung 9.2c sind Optionen 1 und 2 unterschiedlich detailliert angezeigt. Mit Hilfe der annotierten Pfeile an den Optionen können diese unabhängig voneinander minimiert (Pfeilspitze nach links) oder maximiert (Pfeilspitze nach rechts) werden. Diesen Wechsel der Anzeige bezeichnen wir als* lokales Umschalten*, da nur einzelne Optionen betroffen sind.*

9.2.1.2 Positionierung von Optionen im Prozessmodell

Für die modellierten Optionen muss festgelegt werden, an welcher Position im Basisprozess sie dargestellt werden sollen. Der inhaltliche Bezug einer Option zum Basisprozess kann als

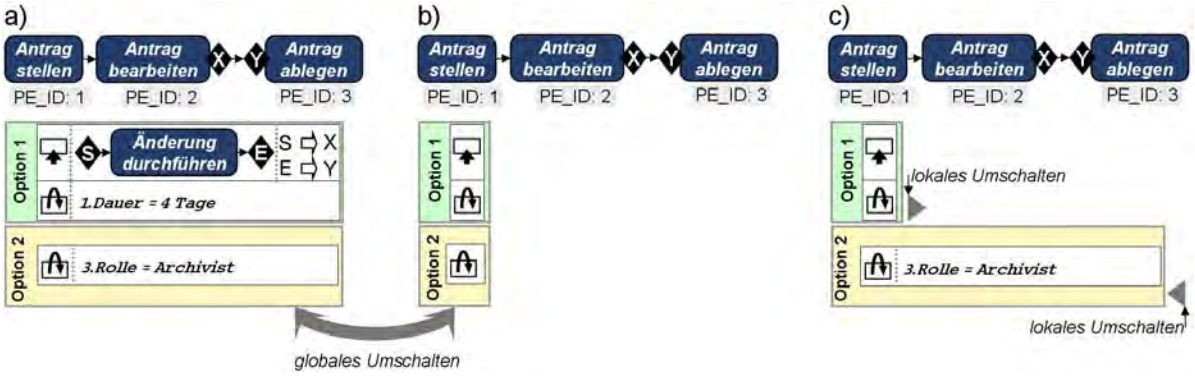


Abbildung 9.2: a) Vollständige, b) minimierte und c) benutzerdefinierte Anzeige von Optionen

Positionierungsinformation genutzt werden: Die Oberfläche eines Modellierungswerkzeugs kann als Koordinatensystem betrachtet werden, dessen Ursprung in der linken oberen Ecke der Modellierungsoberfläche liegt (vgl. Abbildung 9.3). Wird ein Modell erstellt, können für jedes Prozesselement je zwei x- und y-Koordinaten bestimmt werden.[1] Der jeweils kleinere Wert beschreibt den Anfang und der größere Wert das Ende des Prozesselements. Auf Basis dieser Koordinaten kann für Optionen ein *graphischer Optionsbereich* bestimmt werden, indem für jede Option eine minimale und eine maximale x-Koordinate ermittelt wird. Dazu werden die Änderungsoperationen einer Option betrachtet. Das heißt wir untersuchen, welche Aufsetzpunkte bzw. Prozesselement-IDs im Basisprozess referenziert werden.[2] Der kleinste Wert (d.h. die kleinste x-Koordinate) beschreibt den Anfang des graphischen Optionsbereichs. Analog beschreibt der größte Wert (d.h. die größte x-Koordinate) das Ende des graphischen Optionsbereichs einer Option (vgl. Beispiel 9.2). Beziehen sich Änderungsoperationen auf Prozesselemente, die durch eine andere Option erst in den Basisprozess eingefügt werden, werden die ermittelten x-Koordinaten der referenzierten Prozesselemente zur Bestimmung des graphischen Optionsbereichs verwendet.

Beispiel 9.2 (Bestimmung des graphischen Optionsbereichs) *Abbildung 9.3a zeigt einen Basisprozess entlang einer x-Achse. Für jedes seiner Elemente können je zwei x-Koordinaten bestimmt werden. Dies ist durch gepunktete Pfeile visualisiert. In Abbildung 9.3b sind zwei Optionen dargestellt: Option 1 fügt eine Aktivität an den Aufsetzpunkten* `X` *und* `Y` *ein und ändert die Dauer der Aktivität mit Prozesselement-ID 1 (d.h. Aktivität* Antrag stellen*). Der graphische Optionsbereich der Option beginnt daher am Eingang von Aktivität* Antrag stellen *und endet am Eingang von Aktivität* `Antrag ablegen`*. Option 2 führt eine* `MODIFY`*-Operation der Aktivität* `Antrag ablegen` *durch. Ihr graphischer Optionsbereich beginnt daher am Eingang dieser Aktivität und endet an deren Ausgang. In Abbildung 9.3a sind diese Optionbereiche hervorgehoben.*

Mit Hilfe des graphischen Optionsbereichs kann die Information, welches Prozesselement bzw. -fragment im Basisprozess von welcher Option verändert wird, auf eine Position in einem Koordinatensystem abgebildet werden. Basierend auf der Definition des graphischen Optionsbereichs, skizzieren wir verschiedene Positionierungskonzepte für Optionen:

Ansatz 1 (Positionierung bzgl. des graphischen Optionsbereichs): Bei dieser Methode wird eine Option, über die gesamte Breite ihres graphischen Optionsbereichs, im Modell des Ba-

[1] In manchen Modellierungswerkzeugen (z.B. ARIS [IDS08]) wird außerdem die z-Koordinate der Prozesselemente betrachtet. Diese ist z.B. relevant, wenn Prozesselemente auch übereinander angeordnet sein können.

[2] Bei Verwendung von Prozesselement-IDs wird als x-Koordinate der kleinere Wert verwendet, d.h. wir gehen vom Anfang der Darstellung des Prozesselements auf der Modellierungsoberfläche aus.

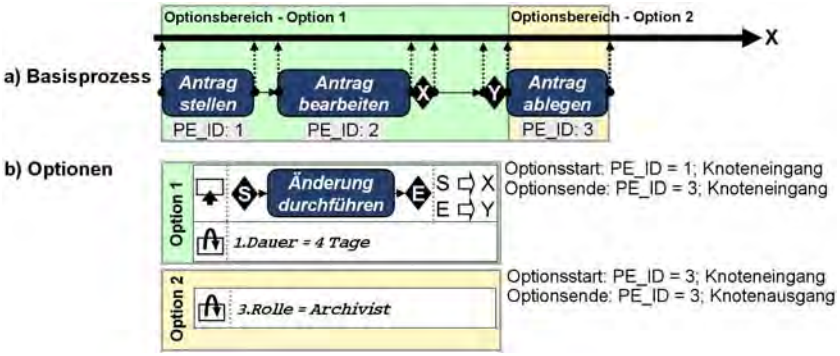


Abbildung 9.3: Bestimmung des graphischen Optionsbereichs

sisprozesses angezeigt. Das heißt, dieser Ansatz zeigt den von einer Änderung betroffenen Bereich vollständig an. Nachteilig ist, dass bei einer Vielzahl von Optionen mit großen Optionsbereichen ein unübersichtliches Prozessmodell resultieren kann. Auch bei Optionen mit sehr kleinem Optionsbereich ist die Darstellung ungeeignet, da diese Optionen dann nur ausschnittsweise dargestellt werden, so dass z.B. mit Scrollbars gearbeitet werden muss (vgl. Beispiel 9.3).

Beispiel 9.3 (Positionierung bzgl. des graphischen Optionsbereichs) *Abbildung 9.4 zeigt ein Beispiel für die Positionierung von Optionen bzgl. ihres graphischen Optionsbereichs. Optionen 1 und 2 werden jeweils parallel zu ihrem graphischen Optionsbereich (vgl. Abbildung 9.3) angezeigt. Dabei wird Option 2 abgeschnitten und ein Scrollbalken hinzugefügt, da der Optionsbereich zu klein ist, um die Option vollständig anzeigen zu können.*

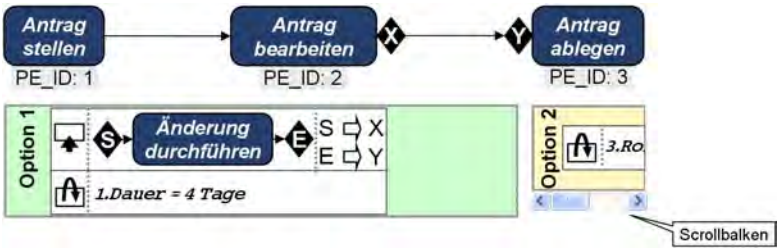


Abbildung 9.4: Positionierung von Optionen bzgl. des graphischen Optionsbereichs

Ansatz 2 (Positionierung bzgl. des Anfangs des graphischen Optionsbereichs): Bei dieser Form der statischen Positionierung wird die Option am Anfang ihres Optionsbereichs positioniert (vgl. Beispiel 9.4). Das Ende des graphischen Optionsbereichs ist für die Positionierung nicht relevant. Bei diesem Ansatz sieht der Modellierer gut, wann über die Verwendung einer Option entschieden wird.[3]

Beispiel 9.4 (Positionierung bzgl. des Anfangs des graphischen Optionsbereichs) *Abbildung 9.5 zeigt eine Anordnung der Optionen am Anfang ihres graphischen Optionsbereichs. Im Unterschied zu Abbildung 9.4 werden die Optionen nicht über die gesamte Länge des graphischen Optionsbereichs gestreckt bzw. gestaucht.*

Ansatz 3 (Verteilte Positionierung von Optionen): Es ist auch möglich, die verschiedenen Änderungsoperationen einer Option verteilt im Prozessmodell anzuzeigen und zwar jeweils

[3]Diese Information ist vor allem bei dynamisch anzuwendenden Optionen interessant.

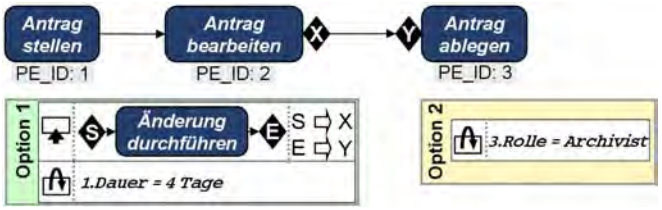


Abbildung 9.5: Positionierung von Optionen bzgl. des Anfangs des graphischen Optionsbereichs

ausgerichtet nach dem Anfang ihres Wirkungsbereichs. Der Modellierer sieht dann die einzelnen Änderungsoperationen dort, wo sie auf den Basisprozess wirken. Die Anzahl der dargestellten Objekte erhöht sich bei Verwendung dieses Ansatzes allerdings stark. Der Zusammenhang bzw. die Gruppierung der einzelnen Änderungsoperationen zu Optionen geht in der Visualisierung verloren (vgl. Beispiel 9.5).[4]

Beispiel 9.5 (Verteilte Positionierung von Optionen) *Abbildung 9.6 zeigt die verteilte Positionierung von Optionen. So wird die aus zwei Änderungsoperationen bestehende Option 1 in zwei Visualisierungsobjekte aufgesplittet. Beide werden am Anfang ihres graphischen Optionsbereichs positioniert. Eine Kombination von Ansatz 3 mit Ansatz 1, d.h. die Positionierung der Änderungsoperationen bzgl. des gesamten graphischen Optionsbereichs, ist ebenfalls denkbar.*

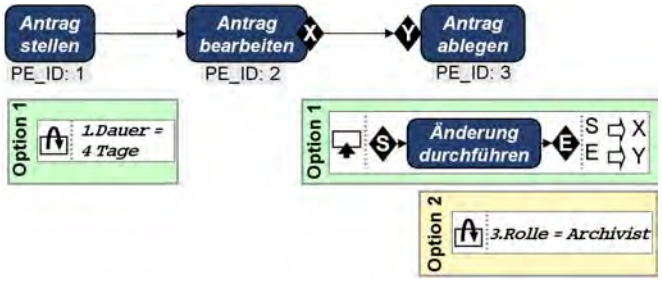


Abbildung 9.6: Verteilte Positionierung von Optionen

In einem Modellierungswerkzeug sollten alle vorgestellten Konzepte zur Positionierung von Optionen angeboten werden (vgl. Tabelle 9.2). Der Modellierer kann dann entsprechend seiner Präferenz und dem jeweiligen Anwendungsfall zwischen den alternativen Positionierungskonzepten wählen.

9.2.1.3 Getrennte Visualisierung von Optionen und Basisprozess

Eine alternativer Visualisierungsansatz besteht darin, Optionen und Basisprozess getrennt zu verwalten. Dazu sind grundsätzlich zwei Techniken denkbar:

Ansatz 1 (Graphische Darstellung der Optionen in Optionsmodellen): Zur graphischen Darstellung der Optionen außerhalb des Basisprozesses werden diese in eigenständigen *Optionsmodellen* verwaltet. Dabei wird entweder für jede Option ein separates Optionsmodell erstellt oder es werden mehrere (alle) Optionen in einem Modell gebündelt dargestellt. Bei der Realisierung unseres Provop-Prototypen haben wir Ansatz 1 zur Darstellung von Optionen gewählt (siehe Kapitel 10).

[4] Verbesserungen dieses Ansatzes lassen sich z.B. durch Einfärben der Visualisierungsobjekte entsprechend ihrer Zugehörigkeit zu einer Option erzielen.

Tabelle 9.2: Vergleich der Ansätze zur Positionierung von Optionen

Ansatz Aspekt	Positionierung bzgl. des graph. Opt.-Bereichs	Positionierung bzgl. des Anfang des Opt.-Bereichs	Verteilte Positionierung
Vorteile	Der Wirkungsbereich einer Änderungsoperation ist gut erkennbar.	Optionen werden dort angezeigt, wo ihr Wirkungsbereich beginnt. Sie werden mit minimaler Breite dargestellt.	Die Änderungsoperationen werden dort angezeigt, wo sie auf den Basisprozess angewendet werden. Dadurch wird ein Vergleich mit dem Basisprozess oder anderen Änderungsoperationen an diesen Positionen möglich.
Nachteile	Eine Option, deren Änderungsoperationen sehr weit verteilt über den Basisprozess wirken, nehmen viel Platz in der Darstellung ein.	Der Bezug zum gesamten Optionsbereich geht verloren, wenn der Anfang des Optionsbereichs beim Scrollen durch das Prozessmodell aus dem Darstellungsfenster „verschwindet“.	Der Bezug zu einer Option geht verloren. Dadurch ist der Zusammenhang von Änderungen nicht mehr optimal nachvollziehbar.
Einsatzzweck	Bei vielen Optionen mit kleinem Wirkungsbereich oder wenigen Optionen mit großem Wirkungsbereich für jede Prozessmodellgröße gut einsetzbar.	Sehr gute Platzausnutzung bei vielen Optionen. Allerdings weniger geeignet für ein großes Prozessmodell mit vielen, verteilt wirkenden Optionen.	Sehr gute Platzausnutzung. Auch für große Prozessmodelle gut geeignet. Verteilung der Änderungsoperationen (d.h. Ausdehnung des Wirkungsbereichs) einer Option irrelevant.

Ansatz 2 (Darstellung von Optionen in Optionsmasken): Eine Alternative zur graphischen Darstellung von Optionen ist deren Beschreibung und Verwaltung mittels *Optionsmasken*. Hier können alle Daten einer Option (z.B. Änderungsoperationen, Optionbeziehungen und Kontextabhängigkeit) direkt geändert werden. Zur Modellierung der Änderungsoperationen kann in eine spezielle Modellierungsumgebung gewechselt werden.[5]

Beispiel 9.6 (Darstellung von Optionen in Optionsmasken) *Abbildung 9.7 zeigt beispielhaft die Darstellung von Optionen mittels Optionsmasken: Zunächst wird Option 5 als anzuzeigende Option ausgewählt (vgl. Abbildung 9.7). Die spezifischen Daten dieser Option werden dann, wie in Abbildung 9.7b bzw. Abbildung 9.7c gezeigt, in einer Maske dargestellt. Optionsbeziehungen und Änderungsoperationen müssen außerhalb der Optionsmaske angepasst werden. Dazu sind weitere Masken bzw. Modellierungsoberflächen erforderlich.*

Die Separierung von Basisprozess und Optionen hat einerseits den Vorteil, dass diese verteilt weiterentwickelt und dargestellt werden können. Andererseits sind dadurch Änderungen schwerer nachvollziehbar und der optische Bezug zwischen Basisprozess und Optionen geht verloren. Unabhängig davon bieten sowohl Optionsmodelle als auch Optionsmasken gute Ansätze, um alle Informationen einer Option übersichtlich zu verwalten. Optionsmasken sind darüber hinaus mit anderen Darstellungsformen von Optionen gut einsetzbar.

9.2.2 Darstellung von Ergebnismodellen

Ein weiterer Visualisierungsaspekt betrifft die adäquate Darstellung von Prozessvarianten (vgl. Anforderung 9.3). Dass heißt, ein Ergebnismodell, das aus der Anwendung einer Menge

[5] Wir werden in Abschnitt 9.3.1.2 zeigen, welche weiteren Modellierungsmethoden für Änderungsoperationen und Optionen in Frage kommen.

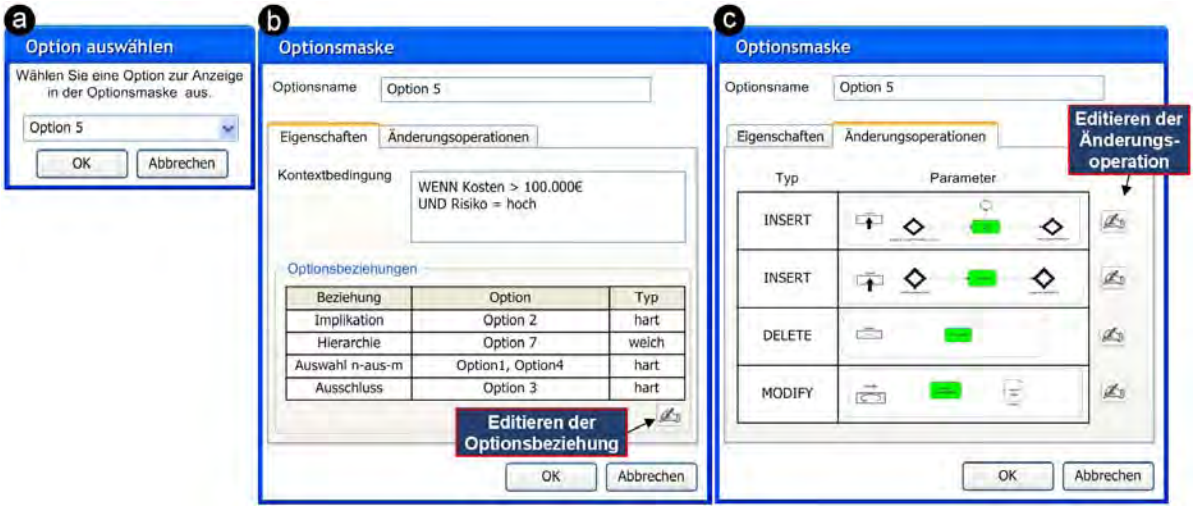


Abbildung 9.7: Darstellung von Optionen in Optionsmasken

von Optionen auf den Basisprozess entsteht, muss im verwendeten Modellierungswerkzeug anzeigbar sein. Je nach Anwendungsfall sind unterschiedliche Schwerpunkte bei der Darstellung relevant. Neben der Untersuchung, der von einer spezifischen Option bewirkten Änderung, kann es relevant sein, die Modelldifferenz zwischen Basisprozess und Ergebnismodell zu betrachten. Entsprechend sind verschiedene Darstellungsformen zu unterstützen, zwischen denen der Variantenverantwortliche frei wählen können sollte (vgl. Abbildung 9.8). Im Folgenden diskutieren wir einige Ansätze entlang des in Beispiel 9.7 beschriebenen Szenarios.

Abbildung 9.8: Darstellungsformen von Ergebnismodellen

Beispiel 9.7 (Darstellung von Ergebnismodellen) *Abbildung 9.9 zeigt einen Basisprozess und zwei modellierte Optionen. Wir gehen im Folgenden davon aus, dass zur Erstellung der Ergebnismodelle aus den Abbildungen 9.10 bis 9.12 immer beide Optionen auf den Basisprozess angewendet werden. Nach entsprechenden Vereinfachungen resultiert daraus das Ergebnismodell aus Abbildung 9.9c.*

Standard-Prozessmodell: Das aus der Anwendung der Optionen 1 und 2 auf den Basisprozess resultierende Ergebnismodell kann wie ein „normales“ Prozessmodell dargestellt werden. Dabei werden die veränderten Prozesselemente in der Darstellung nicht besonders hervorgehoben oder zusätzliche Informationen (z.B. bzgl. der angewandten Optionen) angezeigt. Durch Beschränkung der dargestellten Informationen bzw. Prozesselemente auf bekannte Formen, erhalten wir die „gewohnte“ Prozessmodelldarstellung. Beispiel 9.7 zeigt dies exemplarisch.

Hervorheben geänderter Prozesselemente: Es werden diejenigen Prozesselemente des Ergebnismodells hervorgehoben, die durch eine Änderungsoperation angepasst worden sind. Dazu können die Hintergrundfarbe der Aktivität, ihr Rahmen oder der Text ihres Bezeichners angepassst werden. Auf diese Weise sind die geänderten Prozesselemente gut erkennbar, ohne dass das Ergebnismodell mit Informationen überladen oder gar unleserlich wird. Es

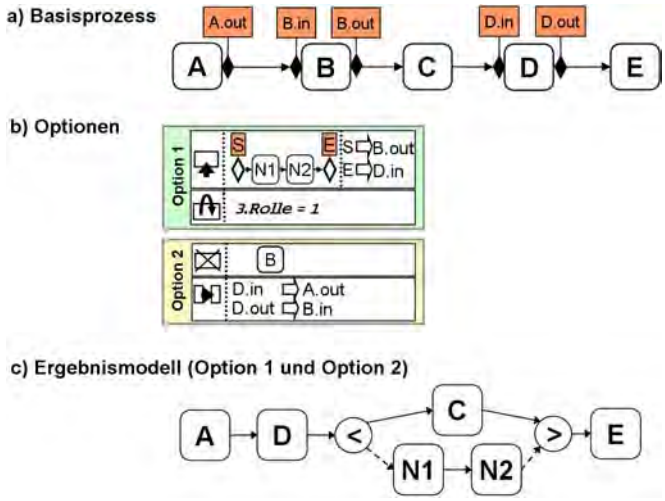


Abbildung 9.9: Szenario zur Diskussion unterschiedlicher Darstellungsformen von Ergebnismodellen

ist allerdings nicht nachvollziehbar, welches Prozesselement durch welche Option geändert worden ist.

Beispiel 9.8 (Hervorheben geänderter Prozesselemente) *Abbildung 9.10a zeigt ein Ergebnismodell mit hervorgehobenen Prozesselementen. Hier bleibt die Darstellung auf geänderte Aktivitäten beschränkt, die sich durch einen dunkleren Hintergrund und eine stärkeren Rahmen von den nicht geänderten Aktivitäten abheben.*

Hervorheben der Optionen: Es werden die zur Ableitung der Prozessvariante durch eine bestimmte Option veränderten Prozesselemente hervorgehoben. Zur Unterscheidung der Optionen wird jeweils ein anderes Hervorhebungsmerkmal (z.B. Hintergrundfarbe, Rahmen) gewählt und in der Legende angezeigt. Dadurch sind die von Optionen bewirkten Änderungen gut erkennbar und miteinander vergleichbar. Nachteilig ist die Beschränkung auf eine geringe Anzahl von Optionen. Diese ergibt sich aus den wenigen zur Verfügung stehenden Hervorhebungsmerkmalen, die zur Unterscheidung von Optionen verwendet werden können. Außerdem können Ergebnismodelle durch Verwendung verschiedener Rahmentypen, Farben und Formen unübersichtlich werden.

Beispiel 9.9 (Hervorheben der Optionen) *Abbildung 9.10b zeigt ein Ergebnismodell, bei dem diejenigen Aktivitäten hervorgehoben sind, die durch Optionen angepasst worden sind. Zwecks Hervorhebung werden unterschiedliche Merkmale adaptiert: Alle von Option 1 angepassten Aktivitäten (d.h. C, N1 und N2) werden mit verstärktem Rahmen und mit kursiver Beschriftung dargestellt; die von Option 2 angepasste Aktivität* `D` *ist farbig hinterlegt.*

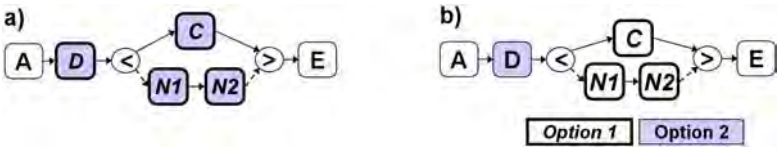


Abbildung 9.10: Hervorheben der Optionen

Auswahl hervorzuhebender Optionen: Das Ergebnismodell wird zunächst als normales Prozessmodell angezeigt. Allerdings können Optionen flexibel an- bzw. abgewählt werden. Da-

durch lassen sich die von einer Option geänderten Prozesselemente im Ergebnismodell hervorheben (vgl. Beispiel 9.10). Diese Darstellung erhöht die Übersichtlichkeit des Prozessmodells und kann durch Hervorheben verschiedener Merkmale weiter verbessert werden.

Beispiel 9.10 (Auswahl hervorzuhebender Optionen) *Bei diesem Ansatz wird das Ergebnismodell nach Anwendung der Optionen 1 und 2 als normales Prozessmodell angezeigt (vgl. Abbildung 9.11a). Wird eine Option zur Anzeige ausgewählt, werden die von ihr angepassten Aktivitäten im Prozessmodell hervorgehoben (vgl. Abbildungen 9.11b bis 9.11d).*

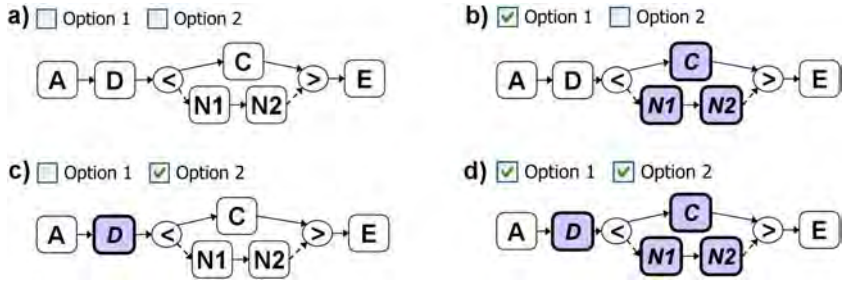


Abbildung 9.11: Auswahl hervorzuhebender Optionen

Delta zu Basisprozess: Bei den bisher diskutierten Darstellungsformen für Ergebnismodelle werden nur die „sichtbaren" Änderungen an Prozesselementen berücksichtigt. Das Entfernen von Elementen aus dem Basisprozess hingegen ist im Ergebnismodell nicht erkennbar. Um solche subtraktiven Veränderungen ebenfalls zu kennzeichnen, können entfernte Prozesselemente in der Darstellung verbleiben und ausgegraut oder durchgestrichen dargestellt werden. Das Ergebnismodell bildet dann das Delta der Prozessvariante im Vergleich zum Basisprozess ab (vgl. Anforderung 9.4).

Beispiel 9.11 (Delta zu Basisprozess) *Im Ergebnismodell aus Abbildung 9.12a wird Aktivität* `B` *durchgestrichen dargestellt. Sie ist durch Option 2 aus dem Basisprozess entfernt worden und wird nun – zur Hervorhebung des Unterschieds – entsprechend im Ergebnismodell markiert. Aktivität* `D` *ist doppelt im Ergebnismodell vorhanden: durchgestrichen angezeigt an der ursprünglichen Position im Basisprozess und „normal" an ihrer neuen Position.*

Markierung der Änderungsart: Zusätzlich zur Anzeige entfernter bzw. verschobener Prozesselemente können verschiedene Hervorhebungsmerkmale (z.B. Hintergrundfarbe und Rahmen) verwendet werden, um angewandte `INSERT`- und `MODIFY`-Operationen im Ergebnismodell kenntlich zu machen. Diese Darstellung erlaubt das Nachvollziehen der vorgenommenen Änderungen, unterscheidet aber nicht mehr nach Optionen. Die Konzepte des Hervorhebens ausgewählter Optionen und die Unterscheidung der Änderungsoperationen sind jedoch kombinierbar.

Beispiel 9.12 (Markierung der Änderungsart) *Das Ergebnismodell aus Abbildung 9.12 zeigt, wie sich die Art der angewandten Änderungsoperationen durch entsprechende Darstellung der betroffenen Prozesselemente hervorheben lässt. So werden Aktivitäten* `N1` *und* `N2` *durch eine* `INSERT`*-Operation in das Ergebnismodell eingefügt und deshalb als farbig hinterlegte Aktivitäten dargestellt. Aktivität* `B` *wird aus dem Basisprozess entfernt und erscheint im Ergebnismodell als durchgestrichener Knoten mit gestricheltem Rahmen. Aktivität* `D` *wird im Ergebnismodell einmal als gelöscht und einmal als neu eingefügt dargestellt. Der Pfeil zwischen gelöschter und eingefügter Aktivität zeigt an, dass Aktivität* `D` *verschoben wurde. Aktivitäten A, C und E werden unverändert aus dem Basisprozess übernommen.*

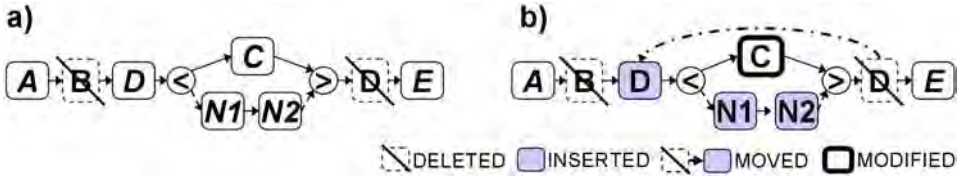


Abbildung 9.12: Markierung der Änderungsart

Vergleichende Darstellung von Ergebnismodellen: Zur Analyse von Prozessvarianten ist es notwendig, diese miteinander vergleichen zu können. Dabei sollen die einzelnen Modelle mit den oben beschriebenen Darstellungsformen repräsentiert und angezeigt werden können (vgl. Anforderung 9.5). Ein naiver Ansatz wäre es, die Prozessmodelle direkt nebeneinander bzw. untereinander darzustellen, ohne sie miteinander in Bezug zu bringen. Eine Verbesserung wird erreicht, wenn durch die graphische Repräsentation ein struktureller oder semantischer Bezug zwischen den Modellen vermittelt wird. So kann durch das identische Anordnen von strukturell identischen Prozessfragmenten eine Verbesserung hinsichtlich der Les- und Vergleichbarkeit erreicht werden. Darüber hinaus können identische Fragmente in den zu vergleichenden Prozessmodellen hervorgehoben werden.

Beispiel 9.13 (Vergleichende Darstellung von Ergebnismodellen) *Abbildung 9.13a zeigt strukturell identische Prozessmodelle, die untereinander dargestellt werden. Identische Aktivitäten* `A` *und* `B` *in den beiden Prozessmodellen werden durch dieselbe Hintergrundfarbe und verstärkten Rahmen hervorgehoben. Auf diese Weise sind sie im Prozessmodell gut erkennbar. Abbildung 9.13b zeigt, wie identische Elemente durch Anpassung der räumlichen Anordnung, in diesem Fall durch Anordnen der identischen Aktivitäten an den selben x-Koordinaten, hervorgehoben werden können. In Kombination mit dem Hervorheben identischer Prozesselemente kann so zusätzlich eine gute Vergleichbarkeit geschaffen werden. Das Anpassen der räumlichen Anordnung ist jedoch nicht trivial, da identische Elemente nicht in der gleichen Reihenfolge auftreten müssen. In diesem Fall sind weitergehende Layoutalgorithmen erforderlich.*

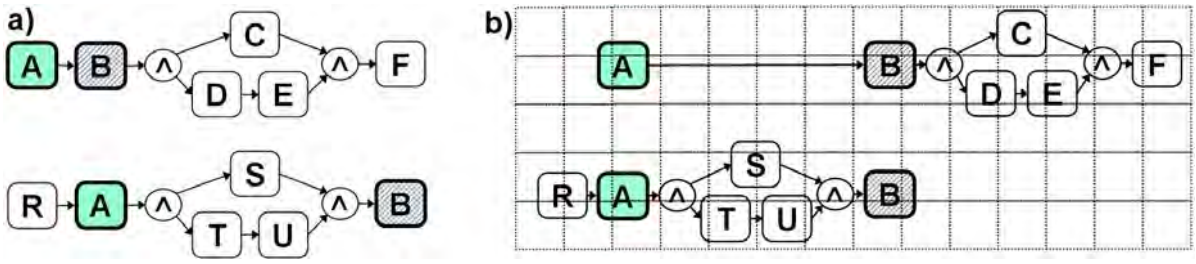


Abbildung 9.13: Vergleichende Darstellungen von Ergebnismodellen

Die Auswahl der vergleichend darzustellenden Ergebnismodelle wird bei der Konfiguration getroffen. So kann der Variantenverantwortliche auswählen, welche Varianten er vergleichen möchte. Anschließend kann eine entsprechende Ansicht erzeugt werden.

Ebenso wie für Optionen, sollten auch für Ergebnismodelle nach Möglichkeit alle Darstellungsformen unterstützt werden, da jede dieser Formen für einen bestimmten Anwendungsfall relevant ist (vgl. Tabelle 9.3). Um die verschiedenen Darstellungsformen zu realisieren, sind entsprechende Informationen relevant. So muss z.B. in den Meta-Daten des Ergebnismodells enthalten sein, welche Optionen auf den Basisprozess angewendet worden sind. Es muss weiter bekannt sein, welche Änderungen durch welche Option verursacht worden sind, um die geänderten Prozesselemente entsprechend identifizieren und hervorheben zu können.

Tabelle 9.3: Vergleich der Darstellungsformen von Ergebnismodellen

Ansatz / Aspekt	**Standard-prozessmodell**	**Hervorheben geänderter Prozesselemente**	**Hervorheben der Optionen**	**Auswahl hervorzuhebender Optionen**	**Delta zu Basisprozess**	**Markierung der Änderungsart**	**Vergleichende Darstellung von Ergebnismodellen**
Vorteile	Leicht verständliches Prozessmodell in bekannter Darstellung.	Geänderte Elemente sind im Prozessmodell gut erkennbar.	Geänderte Elemente werden für ausgewählte Optionen hervorgehoben. Gute Vergleichbarkeit einzelner Optionen.	Geänderte Elemente von ausgewählten Optionen werden hervorgehoben.	Gelöschte Elemente werden hervorgehoben dargestellt.	Änderungsoperation sind direkt erkennbar.	Direkter Vergleich verschiedener Prozessmodelle.
Nachteile	Keine Informationen über die Art der vorgenommenen Änderungen. Gelöschte Elemente werden nicht angezeigt.	Ggf. unübersichtliche Darstellung. Keine Informationen über die Art der vorgenommenen Änderungen. Gelöschte Elemente werden nicht angezeigt.	Ggf. unübersichtliche Darstellung durch viele unterschiedliche Hervorhebungsmerkmale. Anzahl der unterschiedlich darstellbaren Optionen beschränkt. Keine Informationen über die Art der vorgenommenen Änderungen. Gelöschte Elemente werden nicht angezeigt.	Keine Informationen über die Art der vorgenommenen Änderungen. Gelöschte Elemente werden nicht angezeigt.	Keine Information über Optionen. `INSERT` und `MODIFY`-Operationen sind nicht mehr nachvollziehbar.	Keine Information über Optionen (ggf. Kombinieren mit anderen Ansätzen). Ggf. unübersichtliche Darstellung.	Komplexe Darstellung der Prozessmodelle (ggf. durch mächtige Layoutalgorithmen kompensierbar).
Einsatzzweck	Sehr gut geeignet, wenn eine einfache Visualisierung des Ergebnismodells ohne Betonung des Variantenaspekt gewünscht ist.	Sehr gut geeignet für einfache Darstellung. Weniger geeignet für Vergleiche zwischen Optionen.	Gut geeignet für den Vergleich von einzelnen, wenigen Optionen.	Sehr gut geeignet für den Vergleich von Optionen.	Gut geeignet für einfache Visualisierung. Nicht geeignet für den Vergleich von Optionen.	Sehr gute zur Analyse der angewandten Änderungsoperationen geeignet. Nicht geeignet für den Vergleich von Optionen.	Gut geeignet zum Vergleich weniger Ergebnismodelle.

9.3 Interaktionsformen

Generell sollten vor der Realisierung einer Software auch Usability-Aspekte berücksichtigt werden, um eine gute Akzeptanz des Systems bei Endanwendern zu erzielen [Dav93, VRM+08]. Um den Umgang mit Provop möglichst intuitiv zu gestalten, ist es notwendig, die Interaktion zwischen Anwender und Werkzeug im Detail zu betrachten. An dieser Stelle möchten wir keine ausführlichen Designvorschläge für eine Benutzeroberfläche vorstellen. Stattdessen behandeln wir exemplarisch wichtige Aspekte der Handhabung von Änderungsoperationen und der Erstellung von Optionsbeziehungen.

9.3.1 Umgang mit Änderungsoperationen

Zur Beschreibung von Variantenmodellen können komplexe Änderungen erforderlich werden. So muss der Variantenmodellierer z.B. Fragmente des Basisprozesses in Schleifen einbetten können. In Abschnitt 9.3.1.1 zeigen wir, wie durch eine geeignete Benutzerinteraktion die Modellierung solcher komplexen Änderungen auf intuitive Weise möglich wird. Darüber hinaus diskutieren wir in Abschnitt 9.3.1.2 grundlegende Modellierungsmethoden für Optionen und Änderungsoperationen.

9.3.1.1 Einbetten von Prozessfragmenten in komplexe Kontrollfluss-Strukturen

In der Praxis kann es erforderlich sein, ausgewählte Fragmente des Basisprozesses in einer Prozessvariante nur dann auszuführen, wenn eine bestimmte Bedingung zutrifft. In diesem Fall müssen wir das Fragment in eine bedingte Verzweigung einbetten. Die spezielle Semantik des `OPJoin`-Knotens verhindert die Lösung des direkten Einbettens eines Fragments durch das Einfügen des entsprechenden `OR`- bzw. `XOR`-Konstruktes. Stattdessen ist eine Kombination von Änderungsoperationen erforderlich.

Beispiel 9.14 (Einbetten eines Fragments in eine bedingte Verzweigung) *Aktivität `C` aus dem in Abbildung 9.14a dargestellten Basisprozess soll zwecks Bildung einer Prozessvariante in eine bedingte Verzweigung eingebettet werden. Das heißt wir möchten die Aktivität nur dann ausführen, wenn die (Kanten-)Bedingung `CondC` erfüllt ist. Aktivität `B` soll alternativ zur Aktivität `C` ausgeführt werden, wenn die Bedingung `CondB` erfüllt ist. Es gilt `CondB` ≡ `not CondC`. Um diese Änderung umzusetzen, müssen wir zwei Änderungsoperationen beschreiben: Operation 1 fügt je einen `XORSplit`- und `XORJoin`-Knoten sowie alternative Aktivitäten ein (vgl. Abbildung 9.14b und 9.14c1). Außerdem fügen wir einen leeren Knoten mit Aufsetzpunkten als Platzhalter ein. Operation 2 verschiebt nun Aktivität C des Basisprozesses an die Aufsetzpunkte des leeren Knotens (vgl. Abbildung 9.14b bzw. Abbildung 9.14c②). Anschließend werden leere Knoten entfernt (vgl. Abbildung 9.14c③) und das Prozessmodell vereinfacht (vgl. Abbildung 9.14c④).*

Ein weiterer Anwendungsfall für das Einbetten eines Prozessfragments stellt dessen wiederholte Ausführung im Basisprozess dar. Dies kann z.B. erforderlich werden, wenn für hohe Qualtitätsansprüche wiederholte Stichproben-Prüfungen notwendig sind.

Beispiel 9.15 (Einbetten eines Prozessfragments in eine Schleife) *Abbildung 9.15 zeigt, wie eine Aktivität des Basisprozesses in eine Schleife eingebettet werden kann. Dazu wird ein Schleifen-Startknoten (d.h. `Loop Entry`), ein Schleifen-Endknoten (d.h. `Loop Exit`) sowie eine zugehörige Schleifen-Kante (d.h. `Loop`-Kante) eingefügt. Im Gegensatz zum Einbetten in eine bedingte Verzweigung ist hier kein Platzhalterkonstrukt (in Form eines leeren Knotens) erforderlich. Stattdessen können wir die benötigten Aufsetzpunkte an den Knoten innerhalb der Schleife bzw. am Eingang des `Loop`*

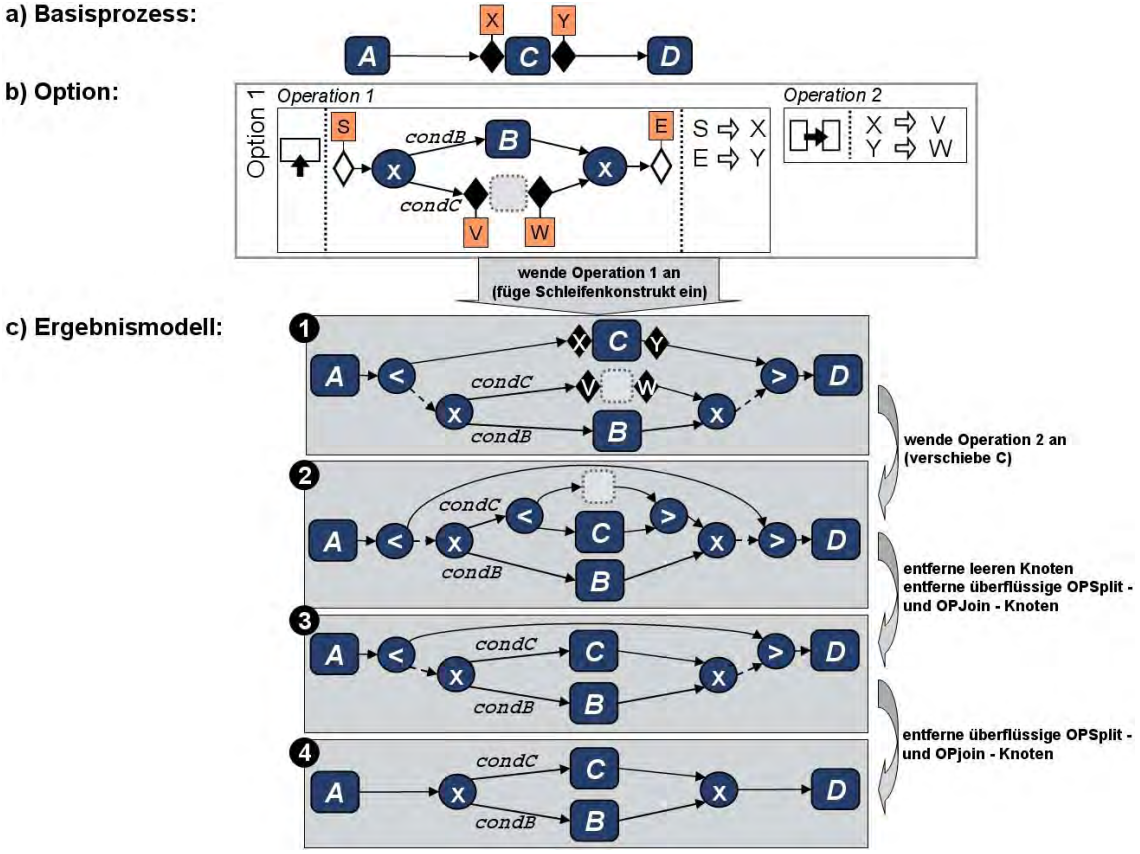


Abbildung 9.14: Einbetten einer Aktivität in eine XOR-Struktur

`Exit`-Knotens platzieren. Diese Aufsetzpunkte werden von der zweiten Änderungsoperation referenziert, welche Aktivität `C` in die Schleife verschiebt (vgl. Abbildung 9.15c②). Anschließend wird das Prozessmodell vereinfacht, indem wir jeweils die überflüssigen `OPSplit`- und `OPJoin`-Knoten entfernen (vgl. Abbildung 9.15c③ und 9.15c④).

Um solche komplexe Änderungen für den Variantenmodellierer möglichst einfach zu gestalten, müssen wir von den tatsächlich durchgeführten Graphmodifikationen abstrahieren. Das heißt, wir verbergen die Änderungsoperationen zum Einbetten eines Prozessfragments vor dem Variantenmodellierer und bieten ihm stattdessen einen konkreten Änderungsoperationstyp, wie z.B. `NEST-in-OR`, `NEST-in-XOR` oder `NEST-in-LOOP`, an (vgl. Anforderung 9.6). Wir visualisieren diese Änderungsoperationen im Folgenden in Form spezieller `INSERT`-Operationen (vgl. Abbildung 9.16b). Die Änderungsoperation ist auf diese Weise intuitiv verständlich.

9.3.1.2 Modellierungsmethodik für Optionen und Änderungsoperationen

Variantenmodellierer sollten Optionen (und deren Änderungsoperationen) für einen Basisprozess intuitiv erstellen können. Dies erfordert eine geeignete Modellierungsmethodik und -umgebung. Grundsätzlich gibt es zur Modellierung von Optionen drei Ansätze:

- **Ansatz 1 (Zeichnen):** Heutige Prozessmodellierungswerkzeuge, wie ARIS Business Architect [IDS08] oder Innovator [MID08], erlauben das „Zeichnen“ eines Prozessmodells. Dazu werden Knoten und Kanten unterschiedlichen Typs aus einer Palette gewählt und

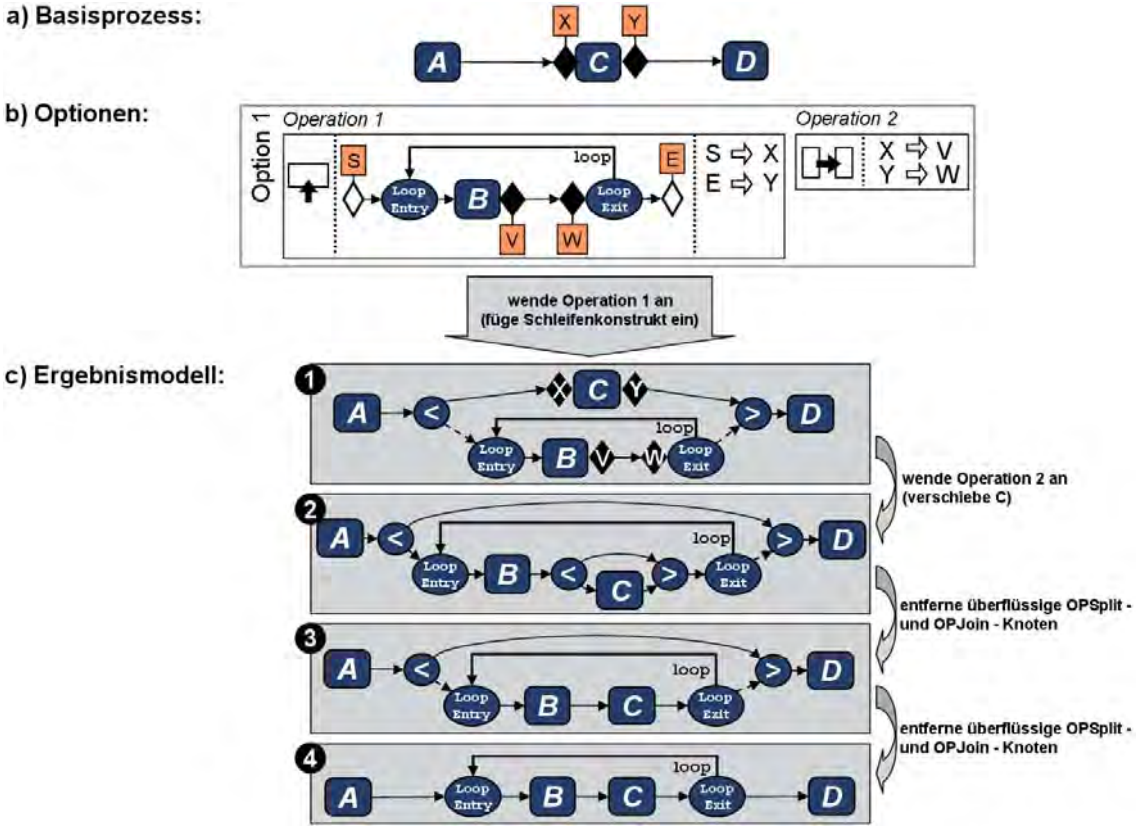


Abbildung 9.15: Einbetten einer Aktivität in eine Schleife

zu einem Prozessmodell zusammengesetzt. Dieser Ansatz kann auch auf Optionen und Änderungsoperationen übertragen werden.

- **Ansatz 2 (Modelldifferenz):** Nachdem ein Basisprozess erstellt worden ist, können Optionen und Änderungsoperationen auch durch Modelldifferenzen erhoben werden. So könnte man im Prinzip Änderungen am Basisprozess vornehmen und diese Änderungen anschließend, durch Vergleichen mit dem ursprünglichen Basisprozess, als Optionen bzw. Änderungsoperation erfassen.
- **Ansatz 3 (Modellierungsoperationen):** Einen weiteren Ansatz bieten syntax-gesteuerte Modellierungsoperationen, wie z.B. „neue Option anlegen“ oder „Änderungsoperation

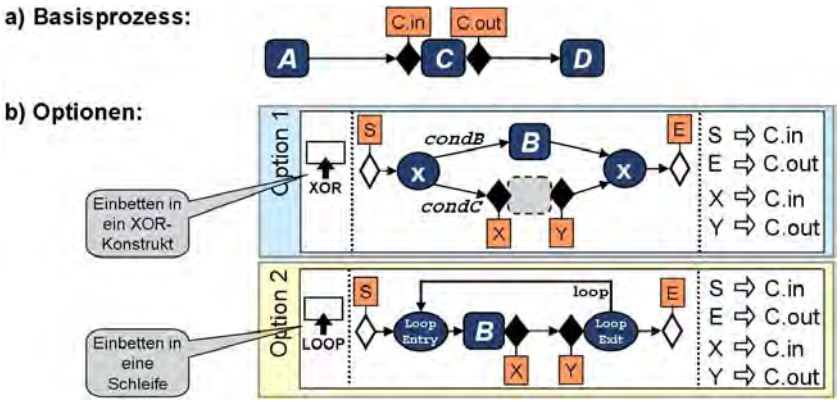


Abbildung 9.16: Visualisierung der komplexen Änderungen in jeweils einer Änderungsoperationen

hinzufügen“, die über ein (Kontext-) Menü abrufbar sind. Ein vergleichbarer Ansatz wird z.B. in der AristaFlow BPM Suite verfolgt [DRRM+09].

Tabelle 9.4 zeigt die Vor- und Nachteile dieser drei Ansätze im Überblick. Die Umsetzung einer Modellierungsmethode ist abhängig davon, ob die Konzepte von Provop in einem existierenden Modellierungswerkzeug oder in einer Neuentwicklung umgesetzt werden sollen.[6] Gibt es keine Restriktionen, ist Ansatz 3 (d.h. explizite Modellierungsoperationen) zu bevorzugen. Diese Modellierungsoperationen können dann mit anderen Funktionen der Modellierungsoberfläche verknüpft werden. Zum Beispiel kann der Variantenmodellierer zunächst mit einem Auswahlwerkzeug einen Bereich im Basisprozess markieren. Anschließend kann er über das Kontextmenü eine entsprechende Modellierungsoperation auswählen und z.B. eine `DELETE`-Operation, welche den ausgewählten Bereich löscht, in einer vorhandenen oder neuen Option erstellen. Auf diese Weise können Änderungsoperationen intuitiv erstellt werden.

Tabelle 9.4: Vergleich der Ansätze zur Modellierung von Optionen

Ansatz / Aspekt	Zeichnen	Modelldifferenz	Modellierungsoperationen
Vorteile	Intuitiver Ansatz der von den meisten existierenden Modellierungswerkzeugen unterstützt wird.	Die syntaktische Korrektheit der Optionen kann leicht gewährleistet werden. Änderungsoperationen werden automatisch erkannt. Die Auswirkungen von Änderungsoperationen sind direkt definier- und sichtbar.	Die syntaktische Korrektheit der Optionen ist gewährleistet.
Nachteile	Fehleranfälliger, da mehr Freiheiten bei der Modellierung bestehen. Zusätzliche Syntax-Prüfung für die Optionen erforderlich.	Es entstehen zusätzliche Aufwände zur Erkennung von Änderungsoperationen aus Modellvergleichen bzw. Änderungsprotokollen.	Die Modellierungsfreiheit ist eingeschränkt.

9.3.2 Erstellung und Beschreibung von Optionsbeziehungen

Die Erstellung und Beschreibung von Optionsbeziehungen ist eine wichtige Aufgabe des Variantenmodellierers, um die semantische und strukturelle Kompatibilität wählbarer Optionen zu gewährleisten. Im Folgenden diskutieren wir Ansätze zur Unterstützung dieser Aufgabe in einem Modellierungswerkzeug (vgl. Anforderung 9.7).

9.3.2.1 Graphische Beschreibung von Optionsbeziehungen

Eine Möglichkeit, um Beziehungen zwischen Optionen sowie Beschränkungen für ihre Verwendung zu beschreiben, ist die graphische Modellierung dieser Abhängigkeiten in einer entsprechenden Modellierungsoberfläche.[7] Dazu kann der Variantenmodellierer die Beziehungen zwischen den modellierten Optionen „zeichnen“ (vgl. Beispiel 9.16). Die graphische Beschreibung gibt einen guten Überblick über modellierte Optionsbeziehungen.

Beispiel 9.16 (Graphische Beschreibung von Optionsbeziehungen) *Abbildung 9.17a zeigt eine Maske zur Auswahl von Optionen. Links werden die modellierten Optionen angezeigt, rechts sind die Optionen gelistet, die bereits zur Beschreibung von Beziehungen ausgewählt worden sind. Die*

[6] Siehe hierzu auch die Realisierung der Modellierungskonzepte in einem Prototypen, basierend auf dem ARIS Business Architect in Abschnitt 10.3.

[7] Dies ist ähnlich zur graphbasierten Darstellung von deklarativen Workflow-Modellen [PSvdA07, WPZW09].

ausgewählten Optionen erscheinen zudem im Fenster aus Abbildung 9.17b. Hier werden außerdem die modellierten Beziehungen angezeigt, und es können weitere Beziehungen definiert werden. Dazu kann der Variantenmodellierer aus einer Palette von Beziehungstypen wählen (vgl. Abbildung 9.17c). Anschließend können mit Hilfe von Drag'n'Drop die entsprechenden Verbindungen zwischen den Optionen beschrieben werden. Durch das Aus- bzw. Abwählen von Optionen (vgl. Abbildung 9.17a), kann die Zahl dargestellter Optionen und Beziehungen eingeschränkt bzw. erweitert werden.

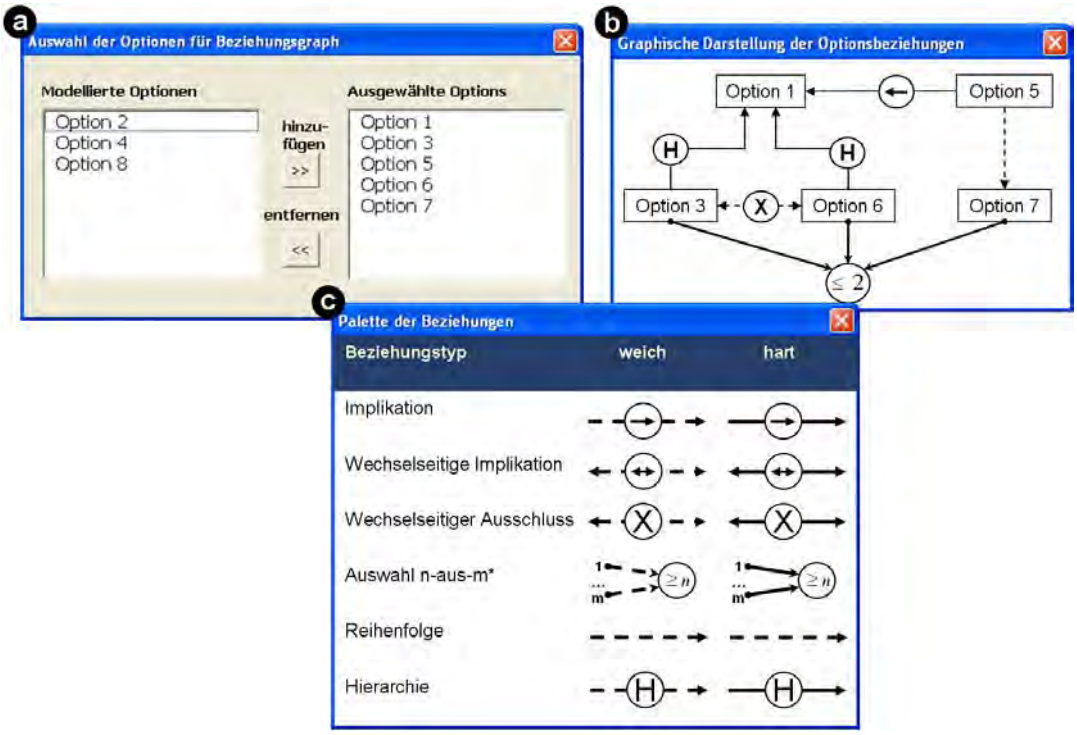


Abbildung 9.17: Graphische Beschreibung von Optionsbeziehungen

9.3.2.2 Regeleditoren zur Beschreibung von Optionsbeziehungen

Eine andere Art der Modellierung von Optionsbeziehungen bieten Regeleditoren. Diese erlauben die Angabe von Regeln, z.B. durch entsprechende Auswahlfelder in einer Maske (vgl. Beispiel 9.17). Regeleditoren sind gut geeignet um Optionsbeziehungen schnell beschreiben zu können. Ohne eine entsprechende Visualisierung wird die anschließende Pflege zahlreicher Optionsbeziehungen jedoch sehr unübersichtlich und komplex.

Beispiel 9.17 (Regeleditoren zur Beschreibung von Optionsbeziehungen) *Abbildung 9.17b zeigt die Beschreibung von Optionsbeziehungen mit Hilfe eines Regeleditors. Dieser ist in zwei Felder aufgeteilt. Abbildung 9.18a beschreibt alle Beziehungen, die zwischen genau zwei Optionen definiert werden können (z.B. Implikation, Ausschluss, Hierarchie); Abbildung 9.18b zeigt Beziehungen zwischen mehreren Optionen (d.h. Auswahl n-aus-m). Mit Hilfe entsprechenden Schaltflächen können Beziehungen hinzugefügt oder entfernt werden.*

9.3.2.3 Automatische Generierung von Optionsbeziehungen

Ein wichtiger Aspekt der Modellierung von Optionsbeziehungen ist das automatische Erkennen und Generieren expliziter Beziehungen. Zum Beispiel kann automatisch erkannt

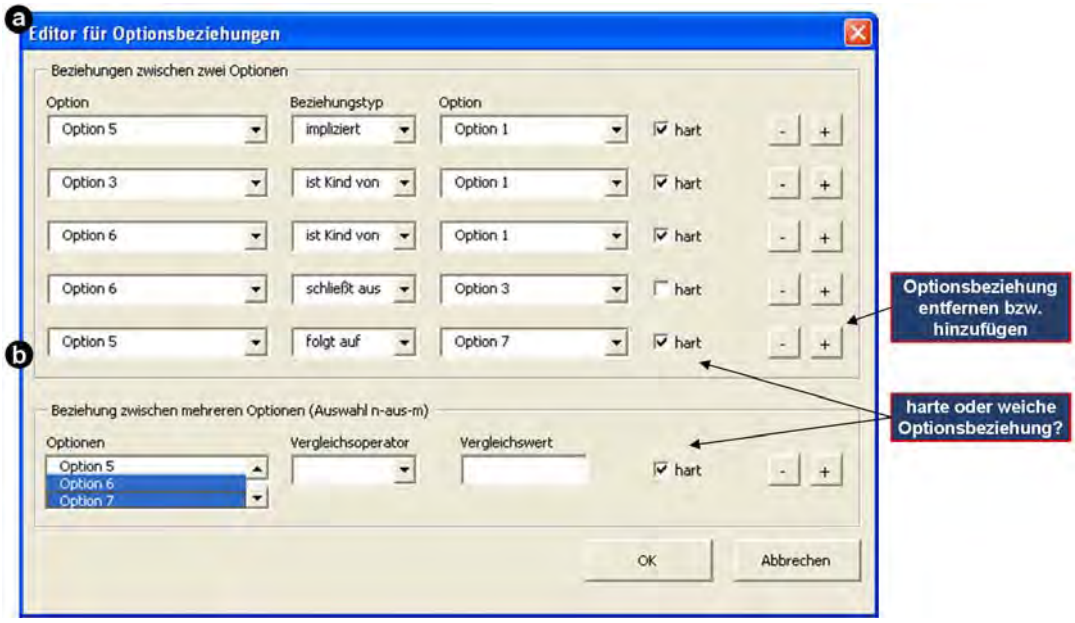


Abbildung 9.18: Regeleditor zur Beschreibung von Optionsbeziehungen

werden, ob zwei modellierte Optionen einander strukturell ausschließen oder implizieren (vgl. Beispiel 9.18). Die automatische Generierung von Beziehungen unterstützt den Variantenmodellierer zwar, jedoch werden Optionsbeziehungen oft auch aufgrund semantischer Abhängigkeiten definiert. Solche Abhängigkeiten sind für das System nicht direkt erkennbar. Des Weiteren verliert der Modellierer die Kontrolle über die explizit zu setzenden Beziehungen und kann Modellierungsfehler ggf. nicht mehr erkennen. Das Vorschlagen strukturell erforderlicher Optionsbeziehungen ist allerdings sinnvoll.

Beispiel 9.18 (Automatische Generierung von Optionsbeziehungen) *Setzen zwei Optionen z.B. ein bestimmtes Attribut der gleichen Aktivität auf jeweils unterschiedliche Werte, kann eine Beziehung zum wechselseitigen Ausschluss der Optionen automatisch generiert werden. Analog können bei strukturell voneinander abhängigen Optionen (etwa wenn eine Option ein Element einfügt, dass von einer anderen Option geändert wird) Reihenfolge- oder Implikationbeziehungen automatisch generiert werden.*

Tabelle 9.5 gibt einen Überblick der Ansätze zur Beschreibung von Optionsbeziehungen. Die Nachteile der graphischen Repräsentation von Optionsbeziehungen können durch geeignete Mechanismen (z.B. Ein-/ Ausblenden von Optionen oder bestimmter Optionsbeziehungen) vermieden werden. Aufgrund der Vorteile graphischer Repräsentation sollten Optionsbeziehungen auch graphisch repräsentiert werden, selbst wenn sie textuell oder automatisch generiert werden. Einige der hier getroffenen Aussagen über die Erstellung und Beschreibung von Optionsbeziehungen sind direkt übertragbar auf Kontextregeln (d.h. Regeln zur Bewertung der Gültigkeit von Kontextbeschreibungen) und Kontextbedingungen (d.h. Bedingungen zur Bewertung der Relevanz einer Option in einem gegebenen Kontext). Wir verzichten daher auf eine Betrachtung dieser „Regeltypen“.

9.4 Diskussion

In der Literatur finden sich zahlreiche Ansätze zur Darstellung von Prozessmodellen. Zum Beispiel existieren diverse Prozess-Metamodelle zur graphischen Repräsentation von Pro-

Tabelle 9.5: Vergleich der Ansätze zur Modellierung von Optionsbeziehungen

Ansatz / Aspekt	Graphische Beschreibung	Regeleditoren	Autom. Generierung
Vorteile	Optionsbeziehungen können direkt definiert und visualisiert werden. Widersprüchliche und redundante Regeln sind damit sehr gut im Graphen erkennbar.	Optionsbeziehungen können einzeln erstellt werden. Kein Informationsüberfluss durch Darstellung aller Optionen.	Der Variantenmodellierer wird bei der Beschreibung von Optionsbeziehungen unterstützt, d.h. der Modellierungsaufwand wird reduziert.
Nachteile	Visualisierung von sehr vielen Optionen bzw. Optionsbeziehungen führt zu einer unübersichtlichen Darstellung.	Ohne eine Visualisierung der Optionsbeziehungen wird diese Vorgehensweise undurchschaubar und komplex. Redundante und widersprüchliche Optionsbeziehungen sind nicht direkt erkennbar.	Optionsbeziehungen, die aufgrund semantischer Abhängigkeiten erforderlich sind, sind nicht automatisch erkennbar.

zessen (z.B. EPK [Sch98] und BPMN [BPM09]). Neben der Darstellung eines einzelnen Prozessmodells, ist es oft auch erforderlich, mehrere Modelle gleichzeitig zu betrachten. Es gibt einige Ansätze bzw. Algorithmen zur Beschreibung und Berechnung der Ähnlichkeit von Prozessmodellen [MGMR02, Wom06, AAAN+05, vdAdMW06, LRWe08]. Die vergleichende Darstellung in einer Modellierungsoberfläche wird jedoch nicht thematisiert.

In [BRB05, BRB07, BBb07, Bob08] wird Proviado – ein Ansatz zur konfigurierbaren Darstellung von Prozessmodellen – präsentiert. Um unterschiedliche Sichten auf ein Prozessmodell zu erlauben, wird ein View-Mechanismus vorgestellt, welcher das Ein- und Ausblenden von Informationen unterstützt. Dabei werden verschiedene Operationen, wie Aggregation und Reduktion, auf das Prozessmodell angewendet. Eine Aggregation ermöglicht es, mehrere Aktivitäten zu einem abstrakten Knoten zu aggregieren; eine Reduktion dagegen entfernt Prozesselemente und vereinfacht den Prozessgraph entsprechend. Bei Anwendung dieser Operationen wird neben dem Kontroll- auch der Datenfluss der Aktivitäten berücksichtigt. Zusätzlich zu diesem View-Mechanismus beschreibt [BRB05, Bob08] Ansätze zur Konfiguration der graphischen Darstellung. Diese erlauben die Definition von Templates für Symbole und die kontextabhängige Festlegung der Symbole für verschiedene Knoten- und Kantentypen. Weiter können personalisierte Visualisierungen realisiert werden. Schließlich wird das dynamische Ein-/ Ausblenden von Detailinformationen über Tooltipps ermöglicht.[8]

Grundlegende Konzepte zum Ein-/ Ausblenden von Informationen in Prozessmodellen präsentiert [Her99]. Hier werden neben dem stufenweisen Ein-/ Ausblenden auch komplexe Gruppierungsmechanismen vorgestellt; ebenso diskutiert werden Ansätze, welche eine Ebenen-spezifische Betrachtung von Informationen erlauben. Ansätze zum Ein-/ Ausblenden von Informationen sind in Provop für die Darstellung von Optionen und Änderungsoperationen ebenfalls relevant. Darüber hinaus finden sich Analogien zur Referenzprozessmodellierung. So sind die Aggregation und Reduktion von Prozessfragmenten durchaus verwandt mit der Konfiguration von Prozessmodellen, basierend auf einem Referenzprozess. Das heißt, die Konfiguration eines Modells ist vergleichbar mit einer Sichtenbildung. Damit sind solche Darstellungskonzepte ebenfalls Optimierungen des Ein-Modell-Ansatzes (vgl. Abschnitt 4.2.2).

Adaptierbare Ansätze zur Darstellung von Prozessvarianten, stammen aus dem Versionsmanagement von Software-Produkten [CW97, CW98]. Hier gibt es verschiedene Ansätze zur

[8] Tooltipps erscheinen auf dem Bildschirm, wenn der Modellierer mit dem Mauszeiger über ein Symbol navigiert. Sie verschwinden wieder sobald der Mauszeiger nicht mehr über dem Symbol steht.

Repräsentation von Produkten (z.B. Bäume) und deren Versionen (z.B. Sequenzen, Bäumen oder gerichteten azyklischen Graphen). Auch eine gemeinsame Darstellung dieser Aspekte in einem integrierten Modell (sog. UND/ODER-Graph) wird diskutiert.

Einen interessanten Visualisierungsansatz aus dem Bereich des änderungsbasierten Versionierens stellt die Matrix-Repräsentation von Versionen und Änderungen dar (vgl. Abschnitt 5.7). Übertragen auf unseren Ansatz würde diese Visualisierung sehr gut zeigen, welche Prozessvariante durch Anwendung welcher Option erstellt worden ist. Auf diese Weise ist ein Vergleich von Prozessvarianten auf höherer Ebene möglich.

9.5 Zusammenfassung

Um die Handhabung von Provop intuitiv zu gestalten, kann der Modellierer zwischen unterschiedlichen Darstellungsformen wählen. So können die Fülle angezeigter Optionsinformationen angepasst sowie Optionen nach unterschiedlichen Kriterien in Relation zum Basisprozess in der Modellierungsoberfläche angeordnet werden. Zusätzlich kann das Ergebnismodell, das aus der Anwendung von Optionen auf den Basisprozess entsteht, in unterschiedliche Darstellungsformen überführt werden. Diese unterscheiden sich in der Verwendung von Hervorhebungsmerkmalen für geänderte Prozesselemente sowie in der Vergleichbarkeit einzelner Prozessvarianten. Komplexe Änderungsoperationen, wie das Einbetten eines Prozessfragments in eine Schleife, sind mit Hilfe höherwertiger Änderungsoperationen modellierbar. Operationsbasierte Modellierungstechniken unterstützen den Variantenmodellierer bei der Erstellung von Optionen und ihren Änderungsoperationen. Des Weiteren kann er für die Erstellung von Regeln und Abhängigkeiten einen Regeleditor verwenden.

Teil III:
Validation des Provop-Ansatzes

10

Prototypische Umsetzung

Im dritten Teil dieser Arbeit befassen wir uns mit der Validation der Provop-Konzepte. Kapitel 10 stellt zunächst den Prototypen vor. Ausgewählte Fallstudien werden in Kapitel 11 behandelt, bevor wir weitere verwandte Arbeiten in Kapitel 12 diskutieren.

Kapitel 10 gliedert sich wie folgt: Abschnitt 10.1 fasst wichtige Anforderungen an den zu entwickelnden Prototyp zusammen. Abschnitt 10.2 gibt einen Überblick zu dessen Architektur. In den Abschnitten 10.3 und 10.4 stellen wir die realisierte Funktionalität für die Modellierung, Konfiguration und Ausführung von Prozessvarianten vor. Wir schließen mit einer Zusammenfassung in Abschnitt 10.5.

10.1 Motivation

Durch prototypische Implementierung der Provop-Konzepte soll deren Umsetzbarkeit in einem Prozessmanagement-Werkzeug nachgewiesen werden. Darüber hinaus wollen wir erste Eindrücke zum Umgang mit dem Provop-Ansatz vermitteln. Zu diesem Zweck wird der entwickelte Provop-Prototyp auch im Rahmen unserer Fallstudien eingesetzt (siehe Kapitel 11).

Im Prototyp sind die Modellierung, Konfiguration und Ausführung von Prozessvarianten abzubilden. Zu diesem Zweck gehen wir in zwei Stufen vor. Zunächst findet die fachliche Modellierung und Konfiguration der Prozessvarianten statt. Anschließend erfolgt die Überführung in ein ausführbares Workflow-Modell.

Der Prototyp soll die bereits diskutierten funktionalen Anforderungen erfüllen und die verschiedenen Lebenszyklusphasen unterstützen. Darüber hinaus soll er erweiterbar sein, um zukünftige Anforderungen noch abbilden zu können.

Tabelle 10.1 zeigt die wesentlichen Anforderungen an den Provop-Prototypen im Überblick.

Tabelle 10.1: Anforderungen an die prototypische Realisierung des Provop-Ansatzes

Funktionale Anforderungen
Anforderung 10.1 *Es muss ein Basisprozess zur Ableitung der Ergebnismodelle modelliert werden können. Die Aufsetzpunkte müssen eindeutig an einem Knotenein- bzw. Knotenausgang platziert werden können.*
Anforderung 10.2 *Die Änderungsoperationstypen* `INSERT`, `DELETE`, `MOVE` *und* `MODIFY` *müssen modelliert werden können. Dabei sollen Aufsetzpunkte des Basisprozesses referenziert werden können.*
Anforderung 10.3 *Optionen müssen sowohl manuell als auch kontextabhängig ausgewählt und auf den Basisprozess angewendet werden können.*
Anforderung 10.4 *Die graphische Darstellung der Ergebnismodelle sollte anpassbar sein. Aufsetzpunkte müssen aus dem Ergebnismodell wahlweise entfernt und Vereinfachungsoperationen angewendet werden können.*
Technische Anforderungen
Anforderung 10.5 *Der Prototyp soll alle Phasen des Prozesslebenszyklus abbilden.*
Anforderung 10.6 *Der Prototyp muss erweiterbar sein.*

10.2 Architektur des Prototyps

Vor Entwicklung der Provop-Modellierungskomponente ist zu entscheiden, ob wir ein komplett neues Werkzeug realisieren oder auf eine bestehende Modellierungssoftware aufsetzen. Mit der erstgenannten Alternative resultieren hohe Zeit- und Kostenaufwände einher, da dann die Basisfunktionalität einer Prozessmodellierungssoftware (z.B. Prozessmodelle erstellen) zusätzlich implementiert werden muss. Desweiteren ist eine produktive Nutzung schwierig, da für die Prozessmodellierung und -ausführung oftmals bereits existierende Werkzeuge eingesetzt werden. Vorteilig ist jedoch, dass eine vollständige Umsetzung der Provop-Funktionalität möglich wird. Bei der Weiterentwicklung eines existierenden Werkzeugs sind die Basisfunktionen bereits gegeben. Allerdings ist hier ggf. nur eine eingeschränkte Umsetzung der Konzepte möglich. Ein wesentlicher Vorteil ist, dass die Weiterentwicklung eines in der Praxis etablierten und in den Fachbereichen bereits eingesetzten Werkzeugs bessere Akzeptanz erfährt.

Wir haben uns für den zweitgenannten Fall entschieden, auch weil wir die Provop-Konzepte in der Praxis validieren wollen. Unser Prototyp basiert auf dem ARIS Business Architect [IDS08], dem WebSphere Integration Developer und dem WebSphere Process Server [IBM09]. Abbildung 10.1 gibt dazu einen Überblick: Die erste Komponente unseres Prototypen basiert auf ARIS und erlaubt die Modellierung von Basisprozess und zugehörigen Optionen. Davon ausgehend erfolgt die Auswahl und Anwendung der Optionen mit Hilfe von ARIS-Reports. Änderungsoperationen können entsprechend der in Abschnitt 5.4 beschriebenen Vorgehensweise auf den Basisprozess angewendet werden. Nach Modellierung und Konfiguration der Prozessvarianten in ARIS findet ein Systemwechsel statt. In der zweiten Komponente des Prototypen werden Variantenmodelle in ausführbare Workflow-Modelle überführt und im WebSphere Integration Developer um technische Attribute ergänzt (z.B. Maskeninformationen, Applikationsaufrufe, Rollenzuweisungen). Anschließend erfolgt die Ausführung der Prozessvarianten im WebSphere Process Server. Unser Prototyp deckt somit alle Prozesslebenszyklusphasen ab (vgl. Anforderung 10.5).

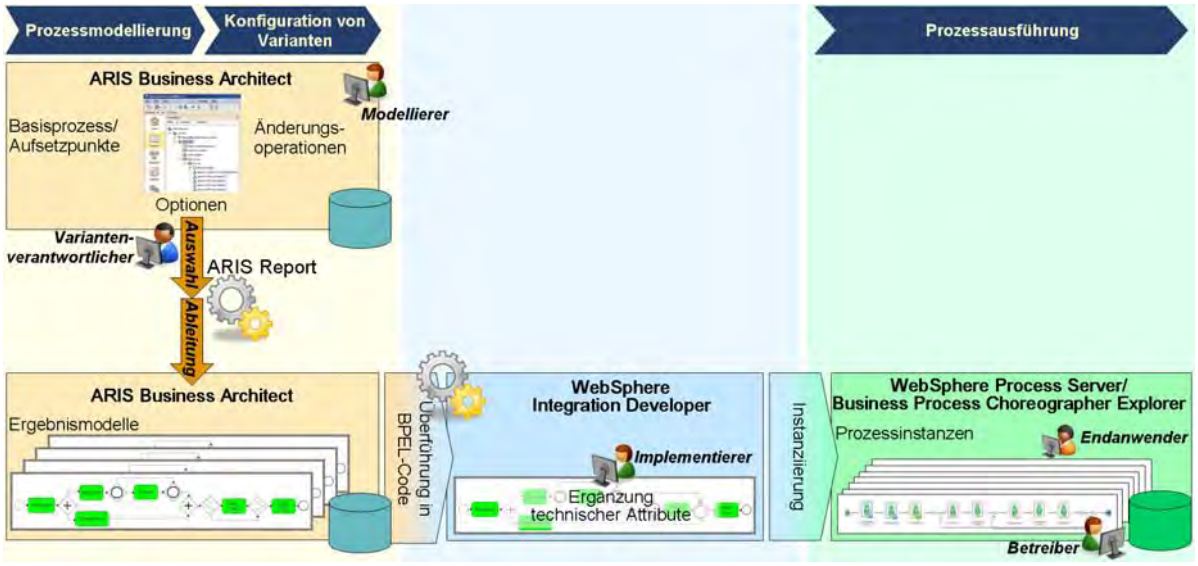


Abbildung 10.1: Komponenten des Provop-Prototyp im Überblick

10.3 Modellierung und Konfiguration von Prozessvarianten

Die erste Komponente des Prototypen deckt die Modellierung und Konfiguration von Prozessvarianten entsprechend dem Provop-Ansatz ab. Im Folgenden zeigen wir, wie ein Basisprozess mit Aufsetzpunkten und Optionen modelliert werden kann. Darüber hinaus demonstrieren wir, wie die modellierten Optionen und deren Änderungsoperationen auf den Basisprozess angewendet werden können, um eine konkrete Variante zu konfigurieren.

10.3.1 Entwurfsentscheidungen

Vor Entwicklung des Prototypen sind gewisse Entwurfsentscheidungen zu treffen. Diese betreffen die Art der Weiterentwicklung des verwendeten Modellierungs-Werkzeugs, die Wahl des zugrunde liegende Prozess-Metamodells sowie die Definition der Modellierungsmethodik für Optionen und Änderungsoperationen.

10.3.1.1 Weiterentwicklung des ARIS Business Architects

Für die Komponente zur Modellierung und Konfiguration von Prozessvarianten, entwickeln wir den ARIS Business Architect [IDS08] weiter. Vorteilig ist, dass ARIS eine breite Auswahl von Prozess-Metamodellen unterstützt und bereits ein rudimentäres Variantenmanagementkonzept beinhaltet (vgl. Abschnitt 2.4).

10.3.1.2 Prozess-Metamodell

Für unseren auf ARIS aufgesetzten Prototyp ist ein geeignetes Prozess-Metamodell zu wählen. ARIS unterstützt u.a. Ereigniss-gesteuerte-Prozessketten (EPK) [Sch98] und die Business Process Modelling Notation (BPMN) [BPM09]; beide Modellierungssprachen sind in der Praxis verbreitet. Wir haben uns für BPMN entschieden, da diese Sprache, im Vergleich zu EPKs, bereits Elemente zur visuellen und logischen Gruppierung von Elementen anbietet. Dies ist einerseits erforderlich, um mehrere Änderungsoperationen in einer Option zu gruppieren und andererseits, um einer Änderungsoperationen ihre Parameter (z.B. Aufsetzpunkte, Änderungsoperationstyp) zuordnen zu können. Ein weiterer Vorteil von BPMN ist, dass der

Übergang zu technischen Beschreibungssprachen (z.B. Business Process Execution Language (BPEL)) einfacher zu realisieren ist.

Die Modellierung und Konfiguration der Prozessvarianten ist in der ersten Komponente des Prototyp durchgängig mit BPMN realisiert, d.h. wir modellieren den Basisprozess und zugehörige Optionen als BPMN-Modelle und konfigurieren aus diesen wieder BPMN-(Ergebnis)-Modelle. Für eine Angleichung der Darstellung von Provop-Komponenten in BPMN, an ihre bisherige Visualisierung in dieser Arbeit, erweitern wir die Basiselemente der BPMN um Aufsetzpunkte, Anker und Symbole zur Repräsentation des Änderungsoperationstyps.

10.3.2 Realisierung der Modellierungskonzepte

Im Folgenden beschreiben wir Details zur Realisierung der Provop-Modellierungskonzepte. Wir gehen in Abschnitt 10.3.2.1 zunächst auf technische Aspekte des Prototypen ein und erläutern in Abschnitt 10.3.2.2 die Benutzersicht.

10.3.2.1 Technische Aspekte

Basisprozess und zugehörige Optionen werden in getrennten Modellen repräsentiert. Dies ist aufgrund der gewählten Lösung zwar nicht erforderlich, erhöht aber die Übersichtlichkeit. Ein Basisprozess stellt in unserem Ansatz ein BPMN-Modell dar. Wir haben das BPMN-Metamodell in ARIS kopiert und zum Metamodell `Provop-Basemodel` umbenannt. Die gleiche Vorgehensweise haben wir für Optionen verfolgt, die als Optionsmodelle (Typ `Provop-Option`) angelegt werden. Der Vorteil dieser Typisierung ist, dass wir später die Möglichkeit haben, spezielle (Methoden-) Filter zu erzeugen. Diese erlauben es in ARIS, unwichtige Modellelemente auszublenden. Abhängig davon, ob der Modellierer einen Basisprozess oder Optionen modelliert, bieten wir ihm unterschiedliche Elemente in der Symbolpalette an. Die konfigurierten Ergebnismodelle werden in einem separaten Ordner gespeichert. Sie sind ebenfalls vom Typ `Provop-Basemodel`.

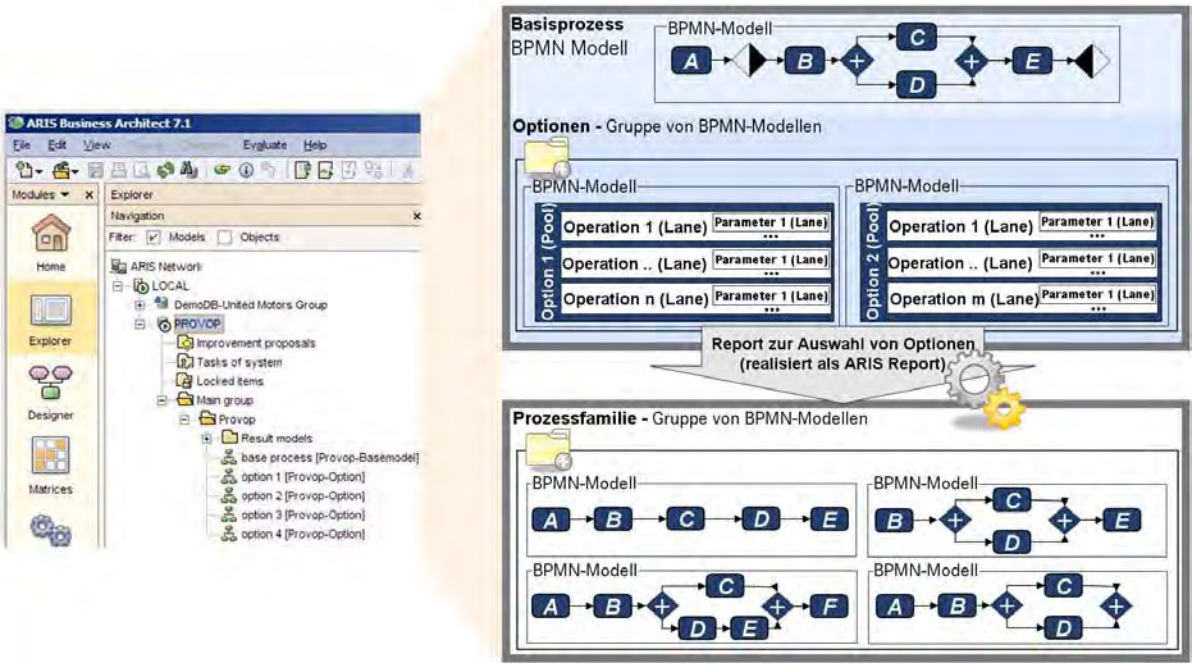


Abbildung 10.2: Architektur des ARIS-Prototypen

10.3.2.2 Benutzersicht

Basisprozess mit Aufsetzpunkten. Da mit den gegebenen Symbolen des BPMN Metamodells keine Repräsentation von Aufsetzpunkten möglich ist, werden diese von uns im ARIS Symbol Designer nachgezeichnet und als neuer Symboltyp in das Metamodell des Provop-Basisprozesses importiert. Aufsetzpunkte stellen nun einen neuen „Funktionstyp" dar. Nicht realisierbar ist damit die Platzierung der Aufsetzpunkte direkt an einem Knoten (d.h. Funktionen oder Strukturknoten), da Knoten immer über Kanten miteinander verbunden sein müssen. Dadurch ist es ebenfalls nicht möglich, zu unterscheiden, ob ein Aufsetzpunkt am Ausgang des vorherigen oder Eingang des nachfolgenden Knotens liegt. Dies ist zur Umsetzung der Änderungsoperationen allerdings unverzichtbar (vgl. Anforderung 10.1). Um zwischen Aufsetzpunkten an Knotenein- bzw. Knotenausgängen zu unterscheiden, führen wir zwei verschiedene Symbole ein: Aufsetzpunkte werden als Rauten dargestellt. Alle Aufsetzpunkte, deren rechte Hälfte schwarz gefüllt ist, liegen am Eingang des nachfolgenden Knotens (vgl. Aufsetzpunkt `Maintenance.in` in Abbildung 10.3). Analog liegen alle Aufsetzpunkte, deren linke Hälfte schwarz gefüllt ist, am Ausgang des vorherigen Knotens (vgl. Aufsetzpunkt `Maintenance.out` in Abbildung 10.3).

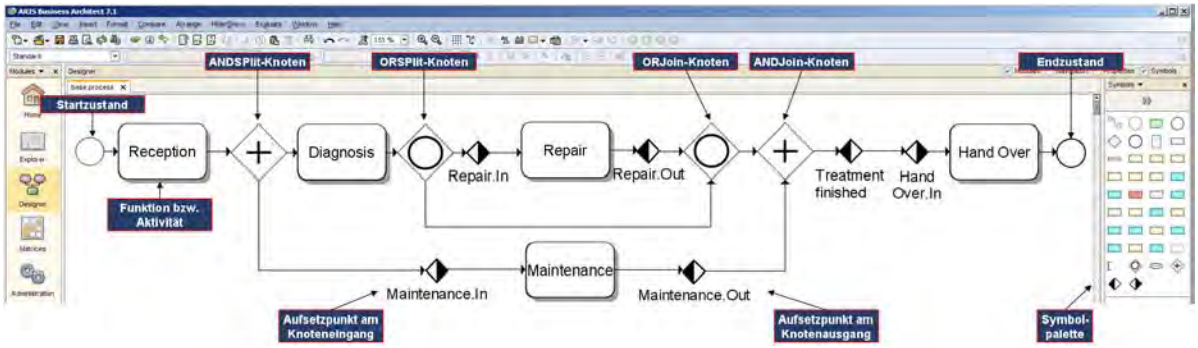


Abbildung 10.3: Screenshot eines Basisprozesses mit Aufsetzpunkten

Änderungsoperationen und Optionen. Zur Abbildung von Optionen verwenden wir innerhalb eines Optionsmodells sog. Pools.[1] Diese erlauben uns das Gruppieren mehrerer Prozesselemente zu einem Objekt. In einem Pool fassen wir Änderungsoperationen als Liste bzw. Sequenz von „Lanes" zusammen. Lanes sind ebenfalls Container von Prozesselementen und können, im Vergleich zu Pools, beliebig ineinander verschachtelt werden. Das heißt, in einem Pool können beliebig viele Lanes angegeben werden. Auf diese Weise können wir in einer Option (Pool) beliebig viele Änderungsoperationen (Lanes) modellieren und innerhalb eines Optionsmodells beliebig viele Optionen definieren. Um eine neue Änderungsoperation zu modellieren (vgl. Anforderung 10.2), müssen wir für eine neue Lane angeben, um welchen Operationstyp (d.h. `INSERT`, `DELETE`, `MOVE` oder `MODIFY`) es sich handelt. Dazu verwenden wir jeweils verschiedene Prozesselemente bzw. Operationssymbole, welche wir zuvor im ARIS Symbol Designer entworfen und anschließend in das `Provop-Option`-Metamodell importiert haben. Über diese Symbole können wir erkennen, um welchen Änderungsoperationstyp es sich handelt und welche Parameter angegeben sind.

Modellieren einer `INSERT`-Operation. Zur Modellierung einer `INSERT`-Operation wird das entsprechende Symbol aus der Symbolpalette in eine Lane gezogen. Innerhalb der Lane wird das einzufügende BPMN-Fragment modelliert. Alle Ein- und Ausgänge des Fragments werden mit Anker-Symbolen verknüpft.[2] Für jeden Anker wird das Mapping auf einen Aufsetzpunkt

[1] Zu Optionsmodellen siehe auch Abschnitt 9.2.1.3.

[2] Die Anker-Symbole der `INSERT`-Operation sind ebenfalls im ARIS-Symbol-Designer entstanden.

als Parameter angegeben, der unterhalb des Anker-Symbols angezeigt wird. Ein Beispiel zeigt Abbildung 10.4.

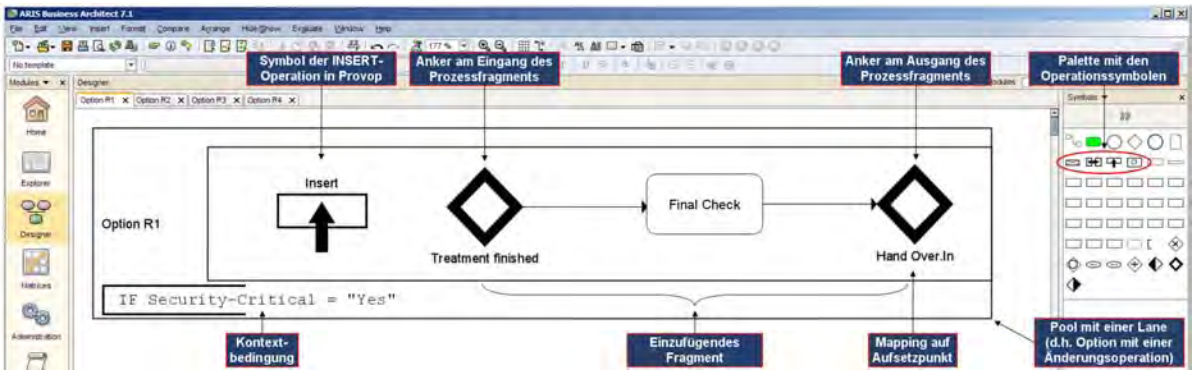


Abbildung 10.4: Screenshot einer `INSERT`-Operation

Modellieren einer `DELETE`-Operation. Zur Modellierung einer `DELETE`-Operation wird das entsprechende Symbol aus der Symbolpalette in eine Lane gezogen. Bei einer `DELETE`-Operation mit Prozesselement-ID erstellen wir das zu löschende Element des Basisprozesses als sog. Variantenkopie innerhalb der Lane der `DELETE`-Operation. Bei einer Variantenkopie wird in ARIS ein Originalobjekt als neues Objekt (d.h. Variante) in der Datenbank erstellt und jeweils ein Verweis von der Variante auf das Originalobjekt sowie vom Originalobjekt zur Variante erzeugt. Bei einer DELETE-Operation mit Aufsetzpunkten, werden die Start- und Ende-Aufsetzpunkte des zu löschenden Fragments mit Anker-Symbolen angegeben.[3] Um den Bezug zwischen Start- und Ende-Anker deutlich zu machen, verbinden wir die jeweiligen Paare mit einer Kante. Je ein Beispiel für eine `DELETE`-Operation, basierend auf Prozesselement-ID bzw. Aufsetzpunkten, zeigt Abbildung 10.5.

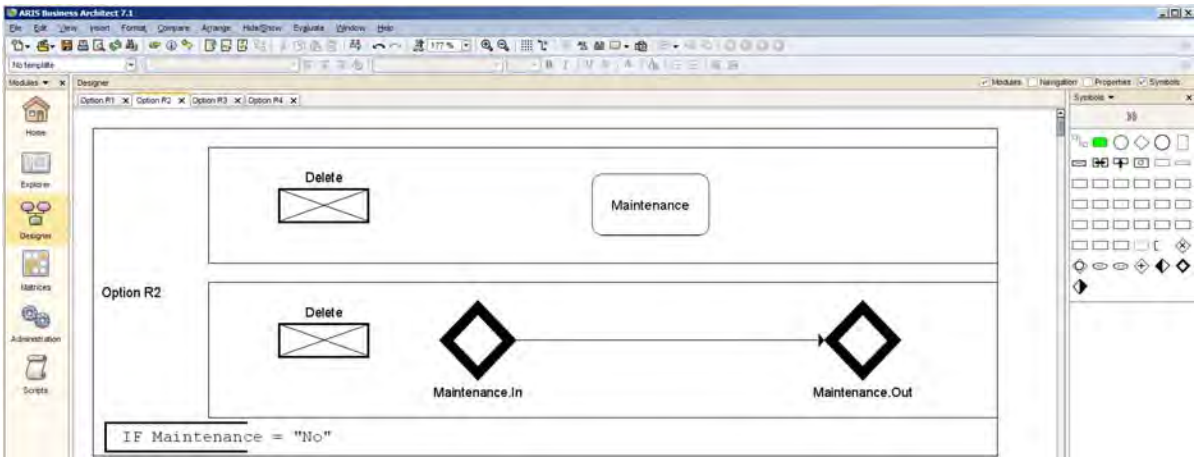


Abbildung 10.5: Screenshot einer `DELETE`-Operation

Modellieren einer `MOVE`-Operation. Zur Modellierung einer `MOVE`-Operation wird das `MOVE`-Operations-Symbol aus der Symbolpalette in eine Lane gezogen. Für die Verschiebung werden Anker angegeben, die auf Aufsetzpunkte im Basisprozess abgebildet werden. Dieses Mapping wird als Parameter der Anker spezifiziert und unterhalb der Anker angezeigt. Die Anker für

[3] Da wir im Basisprozess explizit zwischen Aufsetzpunkten an Ein- bzw. Ausgängen von Knoten, durch entsprechende Symbole, unterscheiden müssen, müssten wir bei einem Löschen über Aufsetzpunkte, neben dem korrekten Bezeichner eines Aufsetzpunktes, auch dessen Typ explizit berücksichtigen. Um die Modellierung an dieser Stelle zu vereinfachen, verwenden wir hier das Symbol der Anker.

die alte und die neue Position im Basisprozess, werden jeweils paarweise mit einer Kante verbunden. Die Bezeichner „OLD_Position“ und „NEW_Position“ visualisieren Start- und Zielposition der Verschiebung im Basisprozess. Ein Beispiel für eine `MOVE`-Operation zeigt Abbildung 10.6.

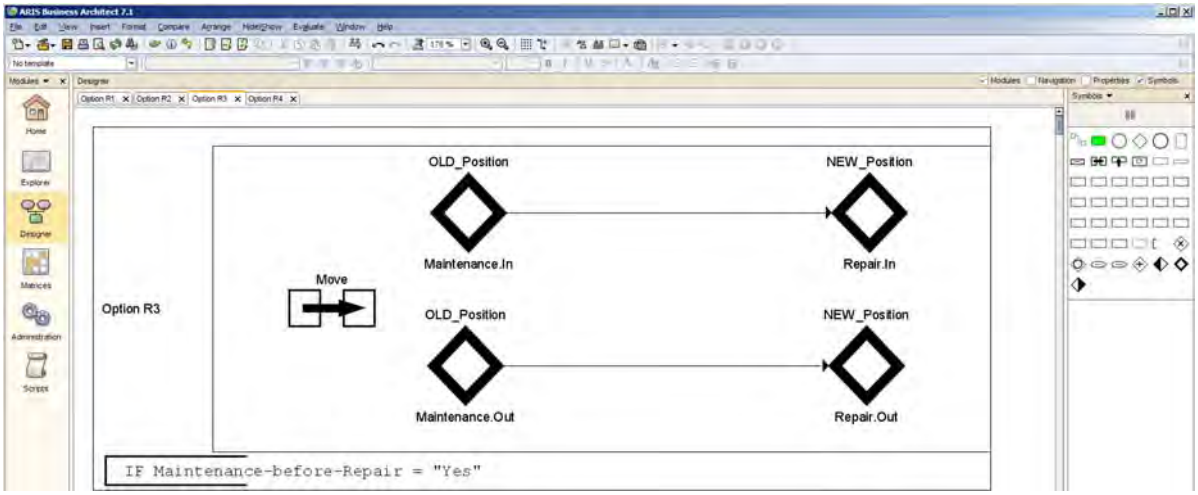


Abbildung 10.6: Screenshot einer `MOVE`-Operation

Modellieren einer `MODIFY`-Operation. Zur Modellierung einer `MODIFY`-Operation wird das entsprechende Symbol aus der Symbolpalette in eine Lane gezogen. Das zu ändernde Fragment wird, analog zur `DELETE`-Operation, als Variantenkopie innerhalb der Lane angegeben. Das zu ändernde Attribut wird als zusätzliches Objekt (sog. Satellitenobjekt) dargestellt. Die Attributänderung wird als Beschriftung des Objekts dargestellt. Ein Beispiel für eine `MODIFY`-Operation zeigt Abbildung 10.7.

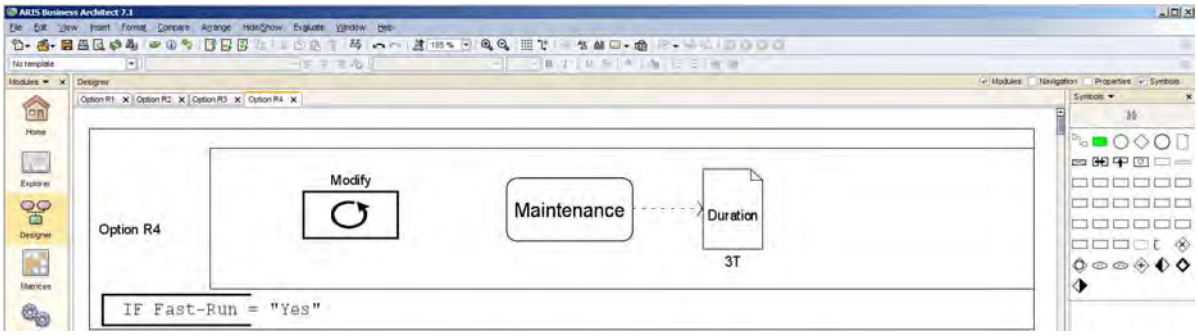


Abbildung 10.7: Screenshot einer `MODIFY-Operation`

10.3.3 Realisierung der Konfigurationskonzepte

Im Folgenden präsentieren wir die Realisierung der Provop-Konzepte zur Konfiguration von Prozessvarianten. Wir gehen zunächst auf technische Aspekte ein und beschreiben anschließend die Benutzersicht.

10.3.3.1 Technische Aspekte

Anstoßen der Konfiguration. Startet der Variantenverantwortliche die Konfiguration eines Prozessmodells über das Kontextmenü des Basisprozesses, so wird ein Report ausgeführt. Dieser ist als ARIS-Skript implementiert, welches im Prinzip eine Erweiterung von ECMA-Skript

um ARIS-spezifische Funktionen darstellt.[4] Der Report prüft zunächst, ob ein Basisprozess, d.h. ein Modell vom Typ `Provop-Basemodel`, modelliert worden ist. Ist dies der Fall, werden anschließend alle modellierten Optionen identifiziert. Das heißt, die Namen aller Modelle vom Typ `Provop-Option`, die sich im gleichen Ordner des Basisprozesses befinden, werden aufgelistet und dem Variantenverantwortlichen zur Auswahl angezeigt. Nachdem dieser Optionen manuell oder kontextbasiert ausgewählt hat, werden das Ergebnismodell als Kopie des Basismodells erstellt und die ausgewählten Optionen angewendet. Das Ergebnismodell wird im spezifizierten Zielverzeichnis unter dem angegebenen Namen abgelegt.

Kopieren des Basisprozesses. Zur Generierung eines Ergebnismodells in ARIS erstellen wir zunächst eine Kopie des Basisprozesses und wenden dann auf diese Kopie die ausgewählten Optionen an. Für die Erstellung einer Kopie gibt es mehrere Möglichkeiten. So kann ein Prozessmodell als *Ausprägungskopie*, *Definitionskopie* oder *Variantenkopie* eines Prozesses angelegt werden. Eine Ausprägungskopie führt dazu, dass alle Elemente des Basisprozesses mit den Elementen des Ergebnismodells logisch verknüpft sind und Änderungen, unabhängig davon ob sie am Original oder an der Kopie erfolgen, jeweils weitergegeben werden. Das heißt, dass z.B. beim Löschen eines Elements aus dem Ergebnismodell auch das Original aus dem Basisprozess gelöscht wird. Ein solches Verhalten ist nicht wünschenswert und Ausprägungskopien sind somit ungeeignet für unseren Ansatz. Eine Alternative stellen Definitionskopien dar. In diesem Fall werden die Prozesselemente als neue Objekte in der Datenbank von ARIS erstellt. Eine Verknüpfung zum Original gibt es nicht. Änderungen werden somit nicht weitergegeben. Nachteilig ist hier, dass keine Information über den Basisprozess mehr gegeben ist. Da die Elemente eine neue Prozesselement-ID erhalten, ist es außerdem nicht mehr ohne weiteres möglich, ID-basierte Änderungsoperationen anzuwenden. In diesem Fall ist eine zusätzliche Verwaltung der alten und neuen Prozesselement-IDs erforderlich. Wir umgehen diese Problemstellung, indem wir das Ergebnismodell als Variantenkopie des Basisprozesses anlegen. Diese Kopierfunktion in ARIS entspricht einer Definitionskopie, wobei allerdings Variante und Original einander eindeutig zugeordnet sind. Über diese Information können wir nun die originalen Prozesselement-IDs identifizieren. Änderungen an Original oder Variante werden nicht wechselseitig weitergegeben.

Anwenden der Änderungsoperationen. Nachdem das kopierte Ergebnismodell angelegt ist, untersuchen wir die ausgewählten Optionsmodelle. Dazu identifizieren wir jeden modellierten Pool (d.h. jede modellierte Option) in einem Optionsmodell und suchen jeweils nach den Lanes bzw. Änderungsoperationen. Für jede Lane können wir den jeweiligen Änderungsoperationstyp eindeutig am entsprechenden Operationssymbol erkennen. Ist kein solches Symbol in der Lane enthalten, ist sie fehlerhaft modelliert und es wird eine entsprechende Benutzermitteilung erzeugt. Abhängig vom identifizierten Operationssymbol werden anschließend die weiteren Prozesselemente der Lane (z.B. Fragmente, Anker) ausgewertet. Die Änderungsoperationen arbeiten dann wie in Abschnitt 5.4 beschrieben (vgl. Anforderung 10.3).

10.3.3.2 Benutzersicht

Über das Kontextmenü eines Prozessmodells wird die Konfiguration eines Ergebnismodells angestoßen. Es wird ein Wizard geöffnet, der aus drei Schritten besteht. Im ersten Schritt werden der Name und das Zielverzeichnis des Ergebnismodells erfragt. Außerdem werden dem Variantenverantwortlichen alle anwendbaren Optionen anzeigt (vgl. Abbildung 10.8). Ist die

[4] ECMA-Script ist eine sog. Scripting Sprache und im Internet weit verbreitet. Sie wird standardisiert und weiterentwickelt von Ecma International und liegt derzeit in der Spezifikation ECMA-262 vor [Int09]. Dialekte von ECMA-Script sind z.B. JavaScript und ActionScript.

Funktion „Select Options manually“ ausgewählt, kann er über Kombi-Boxen die relevanten Optionen, die auf den Basisprozess angewendet werden sollen, manuell auswählen.

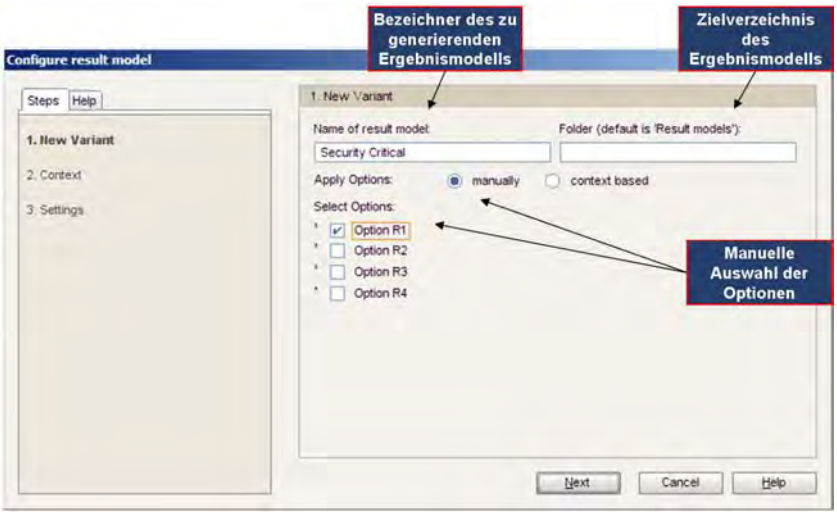


Abbildung 10.8: Screenshot zur manuellen Auswahl der Optionen

Wählt der Variantenverantwortliche die Funktion „Select Options context based“ aus, können die relevanten Optionen abhängig vom gegebenen Kontext bestimmt werden. Dieser Kontext wird dann im zweiten Schritt des Wizards angegeben (vgl. Abbildung 10.10).[5]

Zur Angabe des aktuellen Kontextes werden die Variablen des Kontextmodells angezeigt. Wird eine dieser Variablen selektiert, werden in der daneben angegebenen Liste alle Werte ihres spezifischen Wertebereichs aufgelistet. Durch Markieren eines Wertes und durch Klicken des Hinzufügen-Buttons (Doppelpfeil nach rechts) werden die Variable und ihr Wert der aktuellen Kontextbeschreibung hinzugefügt. Gleichzeitig wird die Menge der in diesem Kontext relevanten Optionen bestimmt und in einer darunter stehenden Liste ausgegeben.

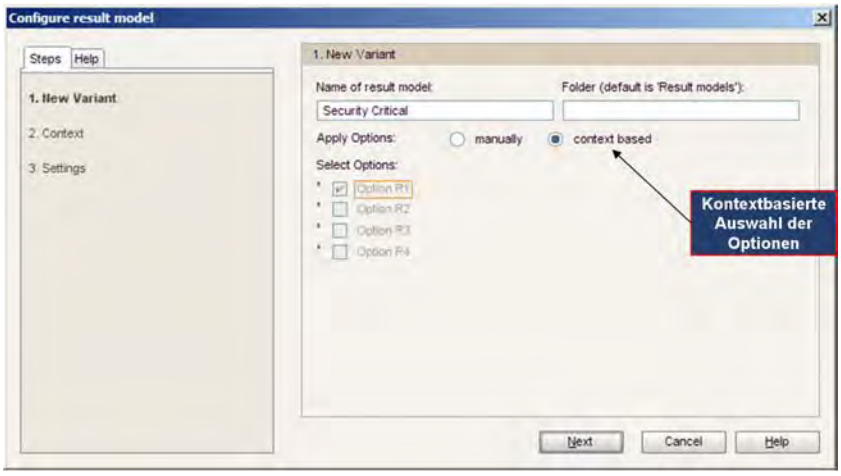


Abbildung 10.9: Screenshot zur kontextbasierten Auswahl der Optionen

Hat der Variantenverantwortliche seine manuelle oder kontextbasierte Auswahl beendet und dies durch Klicken des „Next“ Buttons bestätigt, kann er in Schritt 3 zwischen verschiedenen Darstellungsoptionen für das Ergebnismodell wählen (vgl. Anforderung 10.4). Hat er

[5]Dieser Schritt entfällt bei einer manuellen Auswahl der Optionen.

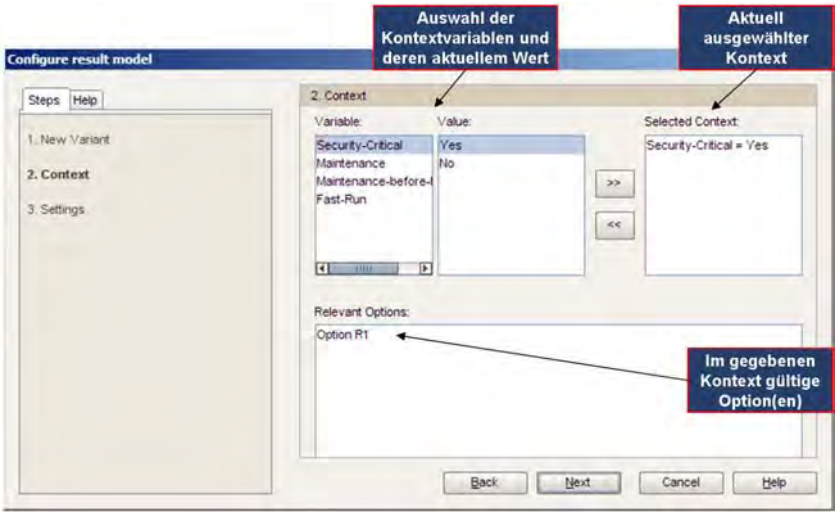


Abbildung 10.10: Screenshot zur Angabe des aktuellen Kontextes

die Darstellungsoption „Delete adjustment points“ angegeben, werden nach Anwendung der Änderungsoperationen alle Aufsetzpunkte aus dem Ergebnismodell entfernt und der Graph anschließend wieder zusammengeführt. Wählt der Variantenverantwortliche die Darstellungsoption „Simplify result model“ (vgl. Abbildung 10.11), werden nach Anwendung der Änderungsoperationen die diskutierten Vereinfachungsoperationen auf das generierte Ergebnismodell angewendet.

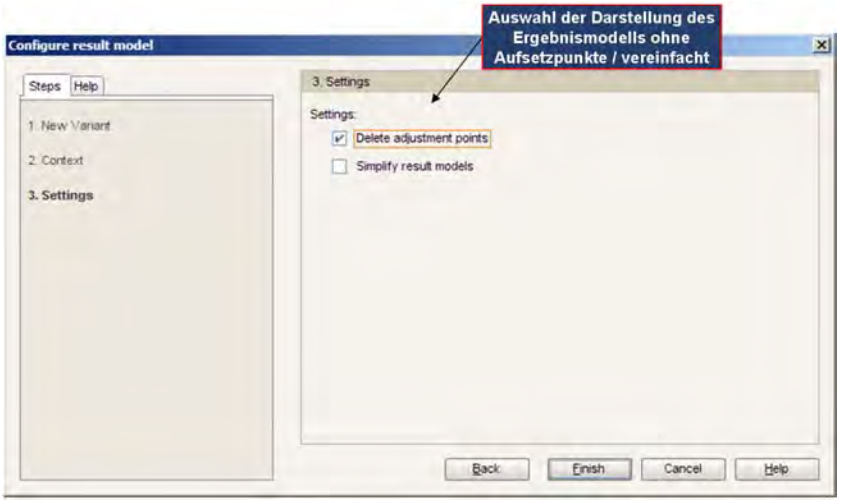


Abbildung 10.11: Screenshot zur Auswahl der Darstellungsoptionen des Ergebnismodells

Beendet der Variantenverantwortliche seine Auswahl von Optionen und Darstellungsoptionen durch Klicken des „Finish“-Buttons, wird das Ergebnismodell erstellt. Dabei werden die ausgewählten Optionen bzw. die darin beschriebenen Änderungsoperationen, entsprechend der in Abschnitt 5.4 vorgestellten Art und Weise, auf den Basisprozess angewendet. Das Ergebnismodell wird dann im angegebenen Zielverzeichnis abgelegt und entsprechend der gewählten Darstellungsoptionen angezeigt. Danach terminiert der Report. Abbildung 10.12 zeigt beispielhaft das Ergebnismodell, das nach Anwendung der in Abbildung 10.4 dargestellten Änderungsoperation resultiert.[6]

[6]Die Vereinfachungsoperationen führen in diesem Beispiel dazu, dass die Aktivität `Final Check` sequentiell eingefügt werden kann und nicht in einem parallelen Ausführungpfad zu einer einzelnen Kante liegt.

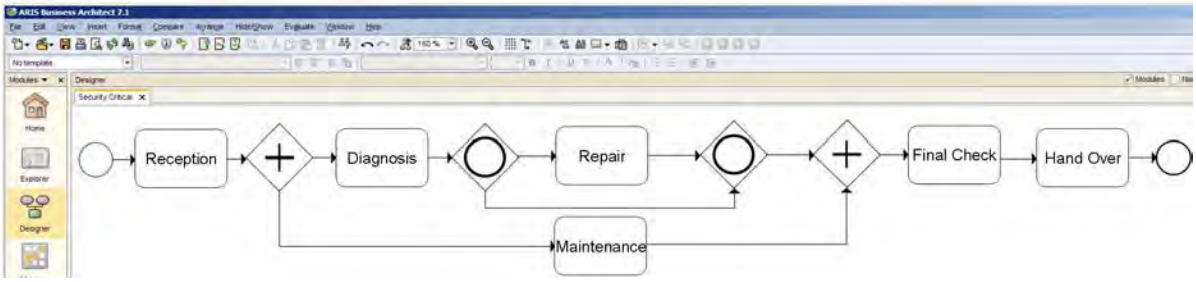


Abbildung 10.12: Screenshot eines Ergebnismodells ohne Aufsetzpunkte und vereinfacht

Wir erstellen im Prototyp nicht automatisch alle möglichen Ergebnismodelle, sondern immer nur ein spezifisches Variantenmodell. Auf diese Weise können wir eine schlechte Performance und lange Berechnungszeiten für die Generierung der gesamten Prozessfamilie vermeiden. Eine optimale Lösung für das Erzeugen von Ergebnismodellen, welche in ARIS allerdings nicht umsetzbar ist, stellt das proaktive Cachen von Ergebnismodellen dar. In diesem Fall würden alle Ergebnismodelle vorab konfiguiert werden. Dieser Vorgang bzw. die Ergebnismodelle selbst, bleiben allerdings vor dem Variantenverantwortlichen verborgen und werden unsichtbar unter der Modellierungsoberfläche des Werkzeugs verwaltet. Stößt der Variantenverantwortliche eine Konfiguration an, kann dann auf die bereits konfigurierten Ergebnismodelle zugegriffen werden (z.B. über einen Abgleich der anzuwendenden Optionsmenge).

10.4 Prototyp zur Ausführung von Prozessvarianten

Die zweite Stufe des Prototypen stellt die Ausführung von Prozessvarianten nach dem Provop-Ansatz dar. Ziel dieser Komponente des Prototypen ist es, die Abbildung der Provop-spezifischen Konstrukte auf eine Prozessausführungssprache zu zeigen. Im Folgenden stellen wir dazu wichtige Entwurfsentscheidungen vor und beschreiben dann die Realisierung der Ausführungskonzepte.

10.4.1 Entwurfsentscheidungen

Weiterentwicklung des WebSphere Integration Developers. Zur Übertragung des Provop-Ansatzes müssen wir die Spezifika des entsprechenden Prozess-Metamodells beachten und berücksichtigen, welche Funktionalität vom WfMS unterstützt wird. So finden sich wesentliche Unterschiede bei der Ausführungssemantik, der Abbildung von Datenflüssen und den Strukturierungsregeln für Kontrollflüsse (z.B. Zyklen, Blockstrukturierung). Auf diese Weise ergeben sich bestimmte Einschränkungen für den Provop-Ansatz, d.h. es können ggf. nicht alle Konzepte, so wie von uns in Provop beschrieben, ohne Nachimplementierung oder Erweiterung des WfMS, umgesetzt werden. Im Rahmen einer Diplomarbeit wurden verschiedene Systeme evaluiert [Mau09]. Schließlich fiel die Wahl auf den WebSphere Integration Developer. Wesentliche Entscheidungsfaktoren sind die Mächtigkeit der zugrunde liegenden Prozessbeschreibungssprache (d.h. BPEL) und die einfache Nutzung der integrierten Testumgebung auf dem WebSphere Process Server.

Systemwechsel. Die Verwendung von ARIS zur Realisierung der Modellierungs- und Konfigurationskomponente führt an dieser Stelle zu einem Systemwechsel. Außerdem ergibt sich ein Wechsel der Prozessbeschreibungssprache von BPMN zu BPEL. Es ist nicht Ziel des Prototypen, die fehlenden Systemschnittstellen zwischen ARIS und WebSphere Integration Developer zu realisieren. Daher betrachten wir diese Problemstellung nicht weiter.

10.4.2 Realisierung der Ausführungskonzepte

Im Folgenden präsentieren wir die Realisierung der Ausführungskonzepte von Provop.

Technische Aspekte. Zur Übertragung eines Ergebnismodells in den WebSphere Integration Developer, müssen Provop-spezifische Konstrukte in BPEL-Code transformiert werden. Dabei werden die `OPSplit`-Knoten, welche durch eine statische `INSERT`- oder `MOVE`-Operation erzeugt werden, mit einer `ANDSplit`-Semantik realisiert. `OPSplit`-Knoten, welche durch dynamisch angewandte Änderungsoperationen resultieren, werden als `ORSplit`-Knoten abgebildet. Die spezifischen Abfragen zur Einhaltung der Kontextbedingungen, Atomarität und Optionsbeziehungen werden über *Java Code Snippets* realisiert. Die ausgehenden Kanten des `ORSplit`-Knotens basieren dann auf dem Rückgabewert dieser Abfrage. Der `OPJoin`-Knoten wird als `ORJoin`-Knoten dargestellt. Die speziellen Eingangsbedingung des `OPJoin`-Knotens wird in Form von Transitionsbedingungen der eingehenden Kanten abgebildet. Die statischen Kontextvariablen werden als sog. `Business Objects` abgebildet, die dem Prozess als Input-Daten geliefert werden. Die dynamischen Kontextvariablen können zur Laufzeit geändert werden. Dabei unterscheiden wir nicht zwischen Prozessdaten und externen Daten (z.B. in einer Datenbank).

Benutzersicht. Die komplexen Abfragen zur dynamischen Anwendung von Optionen im Verlauf der Ausführung einer Prozessinstanz laufen vollständig im Hintergrund ab und bleiben somit vor dem Endanwender verborgen. Es ist allerdings möglich, mit Hilfe des Workflow-Client, in diesem Fall dem *Business Process Choreographer Explorer*, die getroffenen Entscheidungen in einer spezifischen Ansicht nachzuvollziehen.

10.5 Zusammenfassung

Der Provop-Prototyp bildet die Phasen Modellierung, Konfiguration und Ausführung von Prozessvarianten ab. Die Modellierung und Konfiguration der Prozessvarianten findet durch eine Weiterentwicklung des ARIS Business Architects statt [HBR09a, RRHB09]. Anschließend erfolgt die technische Detaillierung und Ausführung der Prozessvarianten im WebSphere Integration Developer bzw. Process Server [IBM09, Mau09]. Die technische Realisierung zeigt, dass die Umsetzung der entwickelten Konzepte auch auf Basis kommerzieller Prozessmanagement-Systeme ohne wesentliche Einschränkungen möglich ist (vgl. Anforderung 10.5).

Der Provop-Prototyp wird in unseren Fallstudien eingesetzt. Die Modellierung von Optionen und Änderungsoperationen hat sich als praktikabler Ansatz erwiesen. Zukünftig wird der Prototyp in weiteren Fallstudien verwendet sowie entsprechend evaluiert und weiterentwickelt (vgl. Anforderung 10.6).

11 Fallstudien

Neben der prototypischen Realisierung bildet die Anwendung der Konzepte in Fallstudien eine wichtige Säule der Validation des Provop-Ansatzes. Konkret haben wir vier Fallstudien in den Domänen Automobilindustrie, Stadtverwaltung und Gesundheitswesen durchgeführt. Das vorliegende Kapitel stellt diese Fallstudien vor und erläutert, welche Erkenntnisse wir aus der Anwendung des Provop-Ansatzes gewonnen haben.

Kapitel 11 gliedert sich wie folgt: Abschnitt 11.1 motiviert die Notwendigkeit von Fallstudien für die Validation unseres Provop-Ansatzes. Anschließend beschreibt Abschnitt 11.2 unsere Vorgehensweise bei der Durchführung und Auswertung der Fallstudien. Darauf aufbauend stellen wir die vier Fallstudien in den Abschnitten 11.3 bis 11.6 vor. Abschnitt 11.7 bietet eine abschließende Diskussion und Zusammenfassung.

11.1 Motivation

Zur Absicherung der von uns entwickelten Provop-Konzepte ist eine Validation mittels Fallstudien erforderlich. Ziel dieser Fallstudien ist es, qualitative Aussagen in Bezug auf die praktische Anwendbarkeit und Übertragbarkeit der Provop-Konzepte zu gewinnen. Das heißt, die Anwendbarkeit von Provop in realen Anwendungsszenarien soll geprüft werden. Um die Generizität und Übertragbarkeit der Provop-Konzepte auf andere Domänen zu demonstrieren, ist eine gewisse Heterogenität der Fallstudien wünschenswert. Das heißt, es sollten Fallstudien aus unterschiedlichen Domänen betrachtet werden, die jeweils unterschiedliche Anwendungsszenarien beschreiben sowie unterschiedliche Prozess-Metamodelle, Komplexität der Prozessmodelle und Anzahl von Prozessvarianten aufweisen. Weiter soll ein Vergleich zur bisherigen Handhabung von Prozessvarianten auf Grundlage konventioneller Methoden (vgl. Abschnitt 4.2) möglich sein. Schließlich möchten wir mit Hilfe der Fallstudien bestehende Limitationen von Provop und geeignete Erweiterungsmöglichkeiten aufzeigen.

Neben qualitativen Analysen sollen in gewissem Umfang auch quantitative Aussagen getroffen werden, etwa bzgl. der Anzahl von Optionen bzw. Änderungsoperationen, die zur Beschreibung der Prozessvarianten des jeweiligen Szenarios erforderlich sind.

Tabelle 11.1 fasst wesentliche Anforderungen bzw. Ziele der Fallstudien zusammen.

Tabelle 11.1: Anforderungen an die Fallstudien

Qualitative Aussagen
Anforderung 11.1 *Die Übertragbarkeit des Provop-Ansatzes auf verschiedene Domänen soll demonstriert werden.*
Anforderung 11.2 *Die Anwendbarkeit bzw. Praktikabilität des Ansatzes ist nachzuweisen.*
Anforderung 11.3 *Ein Vergleich zwischen konventionellen Ansätzen für das Variantenmanagement und Provop soll Vorteile des Provop-Ansatzes dokumentieren.*
Anforderung 11.4 *Limitationen und Erweiterungsmöglichkeiten des Provop-Ansatzes sollen identifiziert werden.*
Quantitative Aussagen
Anforderung 11.5 *Quantitative Aussagen zur benötigten Anzahl an Optionen und Änderungsoperationen sollen für die verschiedenen Szenarien getroffen werden.*

11.2 Vorgehensweise

Zur Validation der entwickelten Konzepte haben wir vier Fallstudien durchgeführt. Die erste Fallstudie referenziert den Prozess zur Bearbeitung eines Produktänderungsantrags in der Automobilindustrie. Für diesen Prozess wird in der Empfehlung des Verbandes der *Automobilindustrie* (VDA) ein Referenzprozess vorgeschlagen [VDA05]. Wir untersuchen in dieser ersten Fallstudie, inwieweit dieser Referenzprozess und die in [VDA05] beschriebenen Ablaufvarianten mit Provop abbildbar sind. Die zweite Fallstudie entstammt ebenfalls der Domäne *Automobilindustrie*. Sie betrifft den Berechnungsprozess zur Absicherung digitaler Prototypen in der Fahrzeugentwicklung. Hier wird in erster Linie untersucht, wie eine angepasste Variante der Provop-Methodik genutzt werden kann, um eine Vielzahl an Prozessvarianten zu einem SOLL-Prozess zu konsolidieren. Unsere dritte Fallstudie stammt aus der Domäne *Stadtverwaltungen*. Sie basiert auf der in [GWJV+09] vorgestellten Fallstudie zur Konfiguration von Referenzprozessmodellen. Spezifische Prozessvarianten der Domäne *Stadtverwaltungen* werden hier durch den Ansatz der Referenzprozessmodellierung abgebildet. Unsere Fallstudie zeigt die jeweiligen Vor- und Nachteile des in [GWJV+09] vorgestellten Ansatzes auf und vergleicht ihn mit einer Lösung nach dem Provop-Ansatz. Die vierte Fallstudie wird in der Domäne *Gesundheitswesen* durchgeführt. Hier werden die organisatorischen Prozesse zur Abwicklung medizinischer Untersuchungen in einem Krankenhaus sowie deren Varianten betrachtet und im Provop-Prototypen abgebildet. Für jede Fallstudie ergeben sich eine unterschiedliche Anzahl an Prozessvarianten sowie spezifische Problemstellungen. Für eine einheitliche Darstellung der Ergebnisse orientieren wir uns an den folgenden Fragestellungen:

- **Szenario des Prozesses:**
  - Welches Szenario adressiert die Fallstudie? Wie ist der generelle Prozessablauf?
  - Welche Prozessvarianten treten auf? Welche Entscheidungen im Prozessmodell sind normale Entscheidungen, die für alle Prozessvarianten getroffen werden müssen, und welche bilden variante Abläufe (d.h. Prozessvarianten) ab?
  - Wie werden Prozessvarianten in den untersuchten Anwendungsfällen bisher gehandhabt? Welche konkreten Probleme ergeben sich daraus?

- **Abbildung der Prozessvarianten mit Provop:**
  - Welche Faktoren bzw. Kontextvariablen beeinflussen die Variantenbildung? Welche Werte können diese Variablen annehmen?
  - Wie ist der Basisprozess zu wählen? Existiert bereits ein geeigneter Standardprozess für die Domäne?
  - Wie können die identifizierten Varianten mittels Optionen abgebildet werden? Lässt sich leicht ein Bezug zwischen den Optionen und dem Kontext herstellen?
- **Gewonnene Erkenntnisse:**
  - Welche konkreten Erkenntnisse können aus der Fallstudie gewonnen werden?
  - Welche Limitationen oder Erweiterungsmöglichkeiten für den Provop-Ansatz lassen sich in diesem speziellen Fall erkennen?

11.3 Fallstudie 1: VDA-Empfehlung 4965 - Engineering Change Management

In der Domäne *Automobilindustrie* ist das Thema *Engineering Change Management (ECM)* äußerst relevant. Für jede Änderung, die an einem Bauteil vorgenommen werden soll, ist ein gewisser ECM-Prozess zu durchlaufen. Dabei werden, abhängig von bestimmten Faktoren, unterschiedliche Prozessvarianten ausgeführt. Da Bauteile auch von Zulieferern entwickelt und produziert werden, muss bei Änderungen ein besonderes Augenmerk auf die Kommunikation zwischen den internen und externen Partnern gelegt werden. Ziel der VDA-Empfehlung 4965 ist die Unterstützung der ECM-Prozesse, insbesondere hinsichtlich der Kommunikation von Änderungsinformationen zwischen Automobilherstellern und Zulieferern unter Nutzung von Standards [VDA05].

Der Fokus unserer ersten Fallstudie liegt auf einer ausgewählten Phase des ECM-Prozesses – dem *Engineering Change Request (ECR)*. Hierzu betrachten wir den Referenzprozess sowie mögliche Prozessvarianten. Die VDA-Empfehlung beschriebt nicht, wie die Prozessvarianten in der Praxis gehandhabt werden können.

11.3.1 Szenario des Änderungsmanagementprozesses

Abbildung 11.1 zeigt die Phasen eines ECM-Prozesses nach [VDA05]: In der ersten Phase (*Identification of Potential for Change*) werden die Änderungen identifiziert und ihr Nutzen bewertet. Die Auslöser für Änderungen können vielfältig sein. Neben der Anpassung und Reaktion auf geänderte Rahmenbedingungen (z.B. gesetzliche Regelungen) sind auch Verbesserungsvorschläge und Innovationen häufige Ursachen für Produktänderungen. In der zweiten Phase (*Development of Alternative Solutions*) werden verschiedene Lösungsalternativen für die Änderung erarbeitet und analysiert sowie ggf. Vorentscheidungen getroffen. Anschließend werden die Lösungsvorschläge in der dritten Phase (*Specification and Decision of Change*) im Detail spezifiziert und entsprechende Bewertungen und Stellungnahmen aus den betroffenen Fachbereichen eingeholt. Am Ende dieser Phase steht eine Entscheidung für oder gegen die Umsetzung der Änderung. Im ersten Fall wird in den sich anschließenden Phasen vier und fünf (*Engineering / Manufacturing Implementation of Change*) die Umsetzung der Änderung vorgenommen.

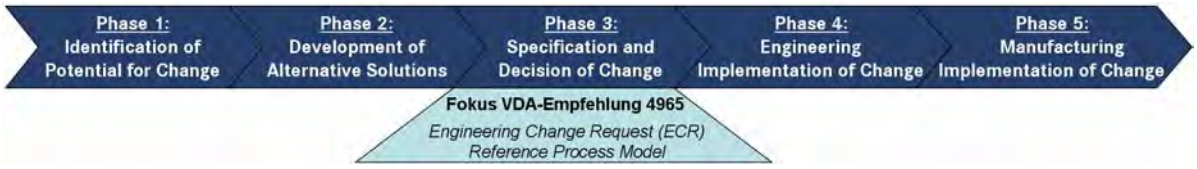


Abbildung 11.1: Phasen des ECM Prozesses aus [VDA05]

Die von uns betrachtete VDA-Richtlinie detailliert die dritte Phase des ECM-Prozesses (d.h. den Engineering Change Request (ECR)) und spezifiziert dazu einen ECR-Referenzprozess (vgl. Abbildung 11.2). Der Prozess beginnt mit dem Empfang eines Änderungsantrags in der Aktivität <`Inquiry of ECR`>. Anschließend wird der Änderungsantrag in Aktivität <`Creation of ECR`> detailliert und für die nachfolgenden Schritte vorbereitet. Es folgt die technische Analyse und Bewertung der Änderung in den Aktivitäten <`Technical Analysis of ECR`> und <`Commenting on ECR`>. Der Änderungsantrag wird danach im Rahmen der Aktivität <`Approval of ECR`> entweder genehmigt oder zurückgewiesen.

Für den ECR-Referenzprozess aus Abbildung 11.2 werden in der VDA-Empfehlung folgende Ablaufvarianten beschrieben:

Abbruch. Wird ein Änderungsantrag direkt zurückgewiesen oder zurückgenommen, kann die Ausführung des ECR-Prozesses nach Ausführung der Aktivitäten <`Inquiry of ECR`> oder <`Creation of ECR`> abgebrochen werden. Erscheint eine Änderung als nicht mehr aussichtsreich, kann die Bearbeitung des Änderungsantrages auch während der Ausführung der Aktivitäten <`Technical Analysis of ECR`> bis <`Approval of ECR`> vorzeitig beendet werden. Abbildung 11.2 zeigt die Abbruchvarianten durch den Übergang in einen Endzustand bzw. durch einen unterbrechbaren Bereich mit Abbruchsignal (d.h. Signal <`Cancel ECR`>).

Rücksprung. Rücksprünge erlauben das Wiederholen von Aktivitäten, d.h. es werden indirekt Schleifen abgebildet. In einem ECR-Prozess sind Rücksprünge erforderlich, wenn die bisherigen Angaben und Informationen zu einem ECR unvollständig sind oder sich neue Rahmenbedingungen ergeben. Je nachdem während der Ausführung welcher Aktivität der Bedarf für einen Rücksprung identifiziert wird, sind unterschiedliche Rücksprünge bzw. Wiedereinstiegspunkte relevant. Abbildung 11.2 zeigt die für den ECR-Prozess spezifizierten Rücksprünge `R1` bis `R3`.

Schnelldurchlauf. Zur Verkürzung der Durchlaufzeit eines ECR-Prozesses werden drei verschiedene „Schnelldurchläufe“ definiert:

- **Schnelle Ausführung:** Die Ausführung der Aktivitäten soll schneller erfolgen, ohne jedoch Aktivitäten auszulassen.
- **Schätzwertbasierte Genehmigung:** In diesem Fall kann das Einholen von Stellungnahmen entfallen (d.h. Aktivität <`Commenting on ECR`> wird ausgelassen). Es wird stattdessen mit den Schätzwerten der vorherigen Aktivitäten gearbeitet.
- **Vorabgenehmigung:** Bei dringenden Änderungen kann eine Vorabgenehmigung dazu führen, dass die Umsetzung einer Änderung (d.h. bestimmte Aktivitäten der Phasen 4 und 5) bereits parallel zu Phase 3 des ECM-Prozesses stattfindet. Das heißt, sie erfolgt außerhalb bzw. parallel zum ECR-Prozess aus Abbildung 11.2.

Weitere Prozessvarianten. Der Referenzprozess aus Abbildung 11.2 beinhaltet lediglich *komplexe Aktivitäten*, d.h. jede dieser Aktivitäten ist durch einen Sub-Prozess beschrieben. Diese Sub-Prozesse können wiederum in unterschiedlichen Varianten vorliegen. So wird z.B. in der

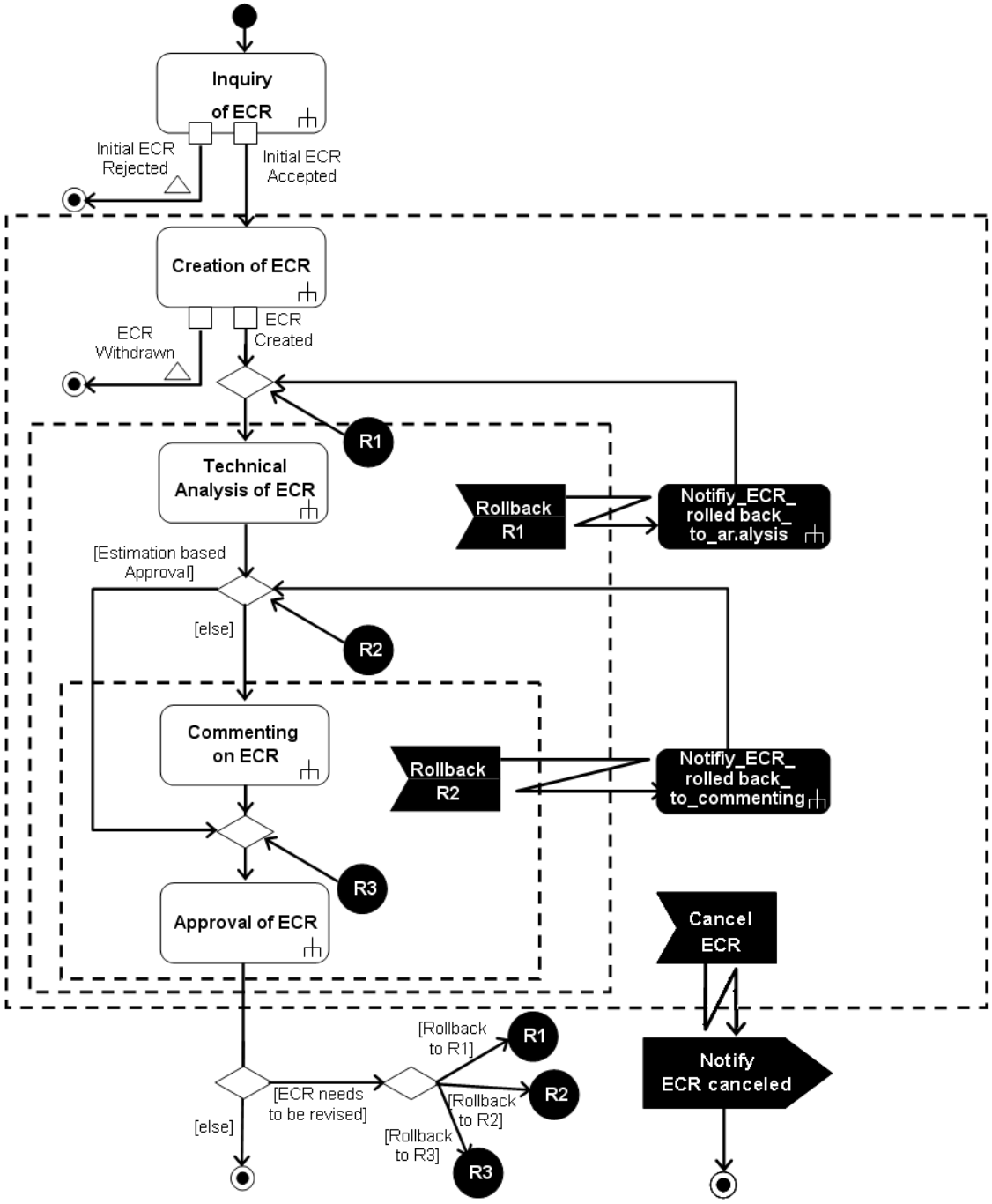


Abbildung 11.2: ECR-Referenzprozess aus [VDA05] (UML 2.0 Aktivitätendiagramm)

komplexen Aktivität <`Approval of ECR`> ein anderer Ablauf gewählt (d.h. ein anderer Subprozess ausgeführt), wenn eine externe Genehmigung benötigt wird oder eine zusätzliche Prüfung durchgeführt werden soll (d.h. *<ECR Completeness Check>*). Insgesamt ergeben sich für die Aktivität <`Technical Analysis of ECR`> zwei Subprozesse und für die Aktivitäten <`Commenting on ECR`> und <`Approval of ECR`> je drei Subprozesse.[1] Aus Platzgründen verzichten wir an dieser Stelle auf eine Beschreibung dieser Prozessvarianten und verweisen auf die VDA-Empfehlung für weitere Details.

[1] Dabei stellt je ein Subprozess den Standardprozess dar.

11.3.2 Abbildung der Prozessvarianten mit Provop

Es ist im vorliegenden Fall naheliegend, den ECR-Referenzprozess als Basisprozess zu wählen. Für die oben skizzierten Ablaufvarianten ist nun zu prüfen, ob sie „normale“ Verzweigungen darstellen, d.h. für alle Prozessvarianten relevant sind, oder ob sie variantenspezifisch sind.

Nicht alle beschriebenen Ablaufvarianten des ECR-Prozesses sollten als Prozessvarianten mit Optionen abgebildet werden: Zwar ist die Abbildung von Abbrüchen mit Optionen möglich, allerdings können die Abbruchentscheidungen erst zur Laufzeit getroffen werden und sind prinzipiell für alle Prozessvarianten relevant. In diesem Fall müsste das Einfügen eines Endknotens für jede gültige Kontextbeschreibung dynamisch vorgenommen werden. Entsprechend wären die eingefügten Endknoten dann Teil jedes Ergebnismodells. Explizite Optionen bringen daher keinen Mehrwert im Vergleich zur direkten Modellierung der Abbruchvariante im Basisprozess.

Eine analoge Beobachtung kann für Rücksprünge gemacht werden. Diese können zwar ebenfalls durch Optionen abgebildet werden, allerdings sind Rücksprünge ebenfalls für alle Prozessvarianten und in einem dynamischen Kontext relevant. Hinzu kommt im Falle der Rücksprünge auch die Möglichkeit der Unterbrechung laufender Aktivitäten. Solche Abbrüche sind nicht durch Provop-Änderungsoperationen abbildbar. Stattdessen müssen hier, wie in der VDA-Empfehlung beschrieben, Ausnahmebehandlungen definiert werden [CCPP99, HA00, RvdAtH06].

Varianten des ECR-Prozesses, die mittels Optionen abgebildet werden können, sind die oben erwähnten Schnelldurchläufe:

- **Schnelle Ausführung:** Die Dauer der zu verkürzenden Aktivitäten wird durch `MODIFY`-Operationen angepasst (vgl. Abbildung 11.3a).
- **Schätzwertbasierte Genehmigung:** Das Auslassen der Aktivität <`Commenting on ECR`> wird durch eine `DELETE`-Operation abgebildet (vgl. Abbildung 11.3b). Abbildung 11.4 zeigt das entsprechende Variantenmodell.
- **Vorabgenehmigung:** Die vorzeitige Umsetzung der Änderung wird durch eine `MOVE`-Operation der Umsetzungsaktivitäten erzielt. Da die zu verschiebende Aktivität außerhalb des ECR-Referenzprozesses, d.h. in einer späteren Phase, liegt und die VDA-Empfehlung diese Phasen nicht im Detail spezifiziert verzichten wir an dieser Stelle auf die Darstellung der Option.

Insgesamt kann der Kontext der Prozessvarianten leicht identifiziert werden. Er ergibt sich in erster Linie durch entsprechende Datenfelder im Änderungsantrag. So wird z.B. für eine schätzwertbasierte Genehmigung im Datenfeld <`ECR_CLASSIFICATION.Description`> des Änderungsantrags der Wert `[estimation based approval]` spezifiziert. Da der Wert dieser Kontextvariable erst zur Laufzeit festgelegt wird, ist ihr Modus als dynamisch anzugeben.

Insgesamt können die beschriebenen Schnelldurchläufe und die erwähnten Ablaufvarianten der komplexen Aktivitäten durch 8 Optionen abgebildet werden. Mit diesen können wir 72 Prozessvarianten abdecken. Dieses Ergebnis resultiert, da die Optionen der Schnelldurchläufe, aufgrund ihrer Abhängigkeit von unterschiedlichen Werten der Kontextvariable <`ECR_CLASSIFICATION.Description`>, nicht kombinierbar sind. Das heißt es kann immer nur eine Schnelldurchlauf-Option angewendet werden. Das gleiche gilt für Optionen, die jeweils die unterschiedlichen Sub-Prozesse aus dem Standard-Sub-Prozess der komplexen Aktivitäten ableiten. Hier kann pro komplexer Aktivität maximal eine Option angewendet werden.

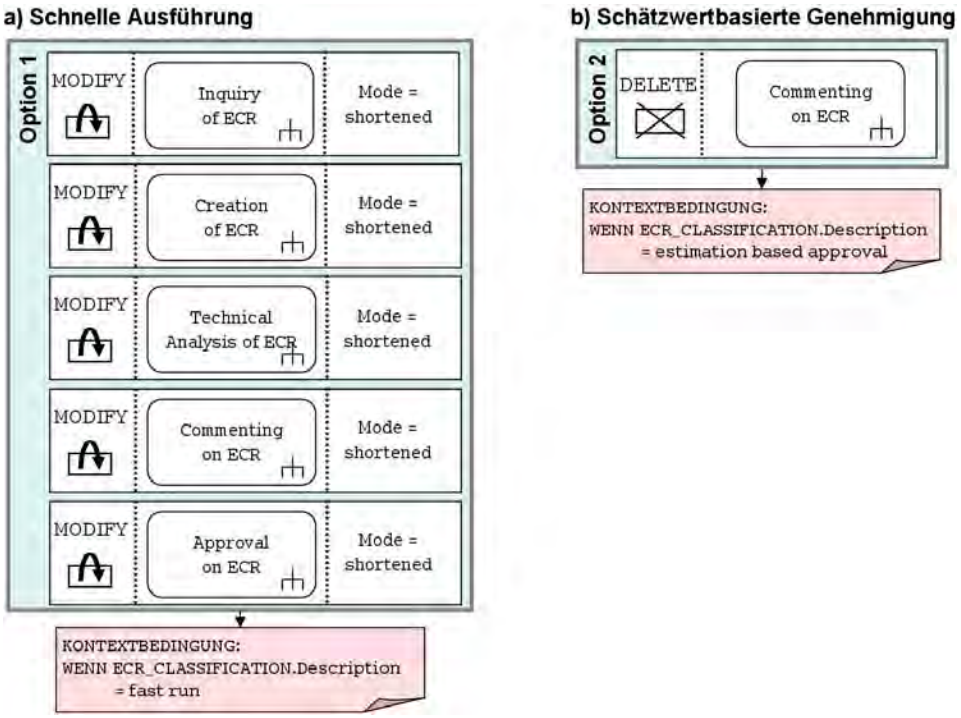


Abbildung 11.3: Optionen zur Abbildung der Schnelldurchlaufvarianten

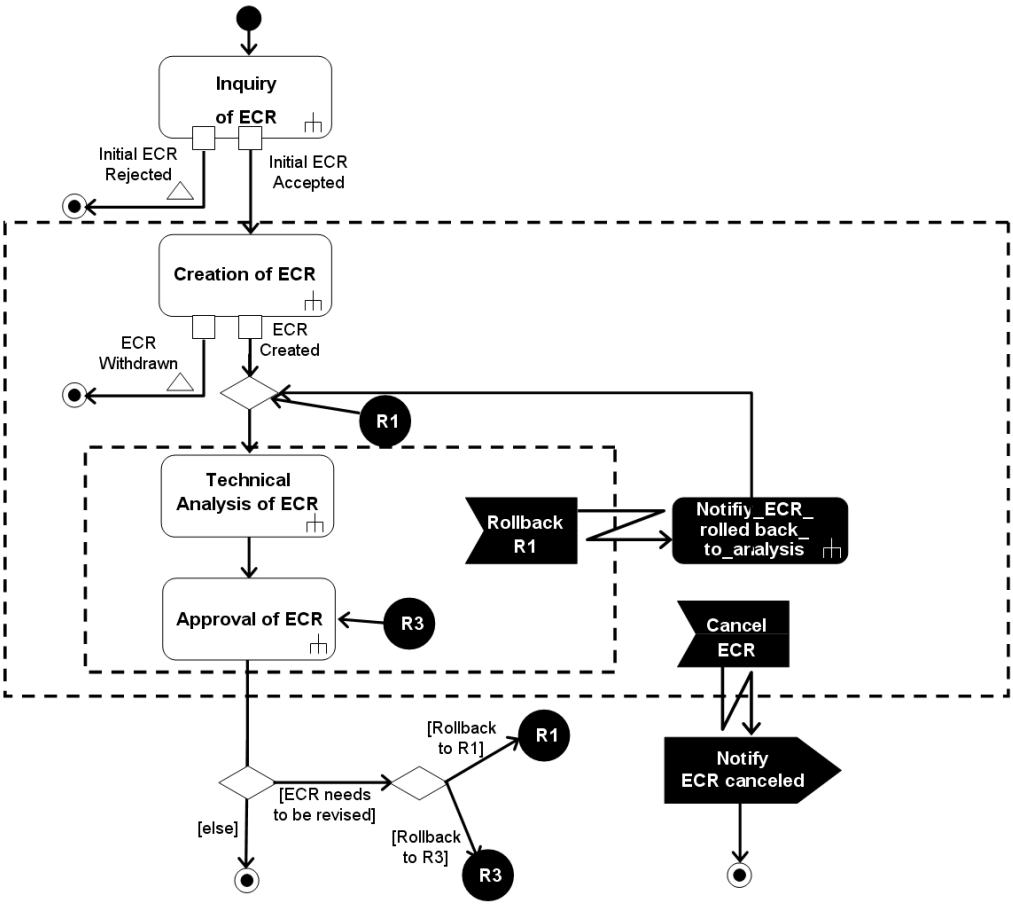


Abbildung 11.4: Prozessvariante für eine schätzwertbasierte Genehmigung

11.3.3 Gewonnene Erkenntnisse

Die Fallstudie zeigt gut, für welche Arten von Ablaufvarianten die Modellierung von Optionen nicht sinnvoll ist. So ist weder die Beschreibung von Abbruchvarianten noch von Rücksprüngen für eine Anpassung durch Optionen geeignet, da hier kein wirklicher Mehrwert resultiert. Außerdem zeigt sich an diesen Beispielen, dass diese identischen Änderungsoperationen (z.B. Einfügen von Endknoten und Rücksprungkanten) nicht generisch modelliert werden können, so dass keine einheitliche Änderungsoperation zur Abbildung aller Abbruch- bzw. Rücksprungvarianten verwendet werden kann.

Die Fallstudie hat weiter gezeigt, dass für bestimmte Ablaufvarianten, vor allem auf einer tiefen Beschreibungsebene des Prozessmodells, eine Abbildung nach dem Provop-Ansatz nicht nur möglich ist, sondern auch einfach und intuitiv umsetzbar. Sowohl der Basisprozess als auch die zu beschreibenden Optionen und Kontextabhängigkeiten konnten wir direkt der VDA-Empfehlung entnehmen.

Bereits durch eine geringe Anzahl an Optionen kann eine große Anzahl an Prozessvarianten abgebildet werden. Die Kontextabhängigkeit führt zu einer deutlichen Reduktion der Anzahl ableitbarer Prozessvarianten, die auch in der Praxis relevant sind. So ist die Zahl möglicher Prozessvarianten ohne Berücksichtigung des Kontextes 2^8, da für jede der acht Optionen einzeln über ihre Anwendung bzw. Nichtberücksichtigung entschieden werden kann.[2] Davon werden allerdings nur 72 Prozessvarianten benötigt.

11.4 Fallstudie 2: Berechnungsprozess zur digitalen Absicherung

In der Produktentwicklung werden im Rahmen des *Computer-Aided Engineering (CAE)* digitale Prototypen eines Fahrzeugs erstellt. Diese werden eingesetzt, um bereits zu einem frühen Zeitpunkt in der Entwicklung eines Fahrzeugs bestimmte Fahrzeugfunktionen digital abzusichern und somit die Qualität des Fahrzeugs zu erhöhen. Digitale Prototypen sind wesentlich kostengünstiger als Hardware-Prototypen und werden daher breit verwendet. Berechenbare Fahrzeugfunktionen sind u.a. Crashverhalten, Aerodynamik, Klimakontrolle, Geräuschentwicklung, Vibration und Rauigkeit. Die Berechnung dieser Funktionen wird von verschiedenen Fachbereichen vorgenommen. Bisher gibt es dazu kein standardisiertes Datenmanagement-System. Dadurch arbeiten die Fachbereiche nicht auf einer gemeinsamen Datenbasis bzw. einem gemeinsamen digitalen Prototyp. Die ganzheitliche Bewertung und Optimierung aller berechenbaren Fahrzeugfunktionen, analog zum Vorgehen bei Hardware-Prototypen, ist somit nicht möglich. Das Ziel des Projektes *Computer-Aided Engineering – Engineering Data Management (caEdm)* ist es daher, alle Simulationsdaten in einem System zu verwalten [Naw09]. Dies soll die Wiederverwendung der Daten ermöglichen sowie die Nachvollziehbarkeit und Qualität der Dokumentation von Berechnungsergebnissen ermöglichen. Dazu soll ein einheitlicher, stringenter Berechnungsprozess für alle Fahrzeugfunktionen definiert werden.

11.4.1 Szenario des Berechnungsprozesses

Abbildung 11.5 zeigt die sieben Phasen des Berechnungsprozesses: In Phase 1 wird zunächst festgelegt, welche Berechnung durchgeführt werden soll und welche Daten dazu benötigt

[2] In diesem Fall gehen wir davon aus, dass die Optionen, wie im Szenario des ECR Prozesses der Fall, kommutativ sind. Das heißt es ergeben sich aufgrund der Anwendungsreihenfolge keine strukturell oder semantisch unterschiedlichen Ergebnismodelle.

werden. Anschließend werden die benötigten Daten in Phase 2 beschafft und in Phase 3 entsprechend aufbereitet, so dass sie anschließend im richtigen Format vorliegen. In Phase 4 werden die Daten um spezifische Informationen der Berechnungsdisziplin angereichert und das Ziel der Berechnung definiert. Darauf aufbauend kann in Phase 5 die eigentliche Berechnung durchgeführt werden. Die gewonnenen Ergebnisse werden anschließend in Phase 6 ausgewertet und in Phase 7 an diverse Gremien kommuniziert.

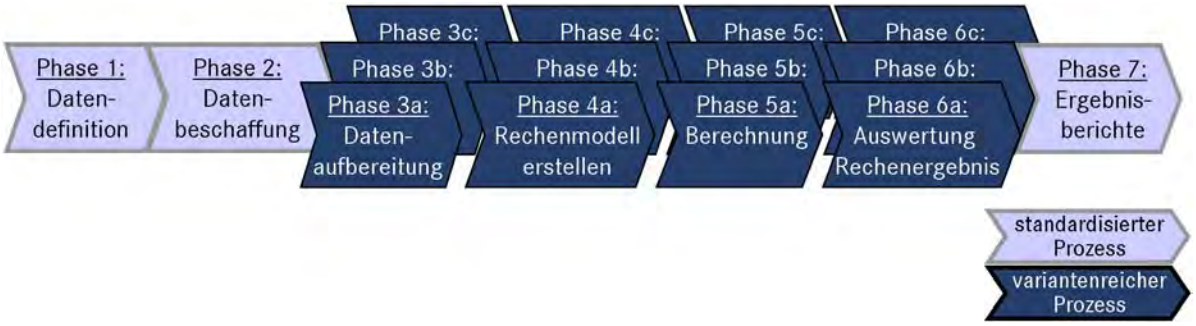


Abbildung 11.5: Phasen des Berechnungsprozesses

Die Phasen 1, 2 und 7 werden bereits durch einen einheitlichen Prozess abgebildet. Für die Phasen 3 bis 6 ist dies bisher nicht der Fall. Hier ist die Variantenvielfalt enorm. Sie ergibt sich u.a. aus den zahlreichen verschiedenen und voneinander unabhängigen Berechnungsdisziplinen (z.B. Crashverhalten und Aerodynamik). Diese Disziplinen gibt es zudem jeweils für die Geschäftsbereiche bzw. Fahrzeugtypen Bus, Van, PKW und LKW. Darüber hinaus verwenden die verschiedenen Fachbereiche sowie die einzelnen Berechner innerhalb eines Bereichs unterschiedliche Simulationsmethoden und -werkzeuge.[3] Insgesamt ergeben sich auf diese Weise mehrere Dutzend Prozessvarianten. Eine Vereinheitlichung all dieser Prozessvarianten zu einem gemeinsamen Standard ist nicht sinnvoll. Um dennoch nicht für jede einzelne Prozessvariante eine eigenständige IT-Lösung entwickeln zu müssen, ist es Ziel von caEdm, die Gemeinsamkeiten der Prozessvarianten zu identifizieren und diese einheitlich abzubilden. Darüber hinaus sollen Bereiche mit ähnlichen Prozessen auf einen gemeinsamen *SOLL-Prozess* konsolidiert werden. Hieraus resultiert eine reduzierte Anzahl an (Soll-) Prozessvarianten und damit auch ein deutlich reduzierter Entwicklungsaufwand. Für das Management von Prozessvarianten in caEdm ergeben sich hieraus vier Aufgaben:

1. Erhebung der Prozessvarianten des Berechnungsprozesses in den einzelnen Fachbereichen (sog. IST-Prozesse)
2. Identifikation von konsolidierbaren (d.h. ähnlichen) IST-Prozessen
3. Konsolidierung der IST-Prozesse zu SOLL-Prozessen
4. Abbildung der SOLL-Prozesse in einem IT-System

Der Schwerpunkt der Unterstützung des Projektes caEdm durch Provop liegt auf den Teilaufgaben 1 bis 3. Die Implementierung des SOLL-Prozesses in einem IT-System ist nicht Gegenstand dieser Fallstudie.

[3]Die Anzahl verschiedener Simulationsmethoden und -werkzeuge liegt bei mehreren Dutzend. Beispielhaft genannt seien hier Matlab, Fortran, Medina und Catia.

11.4.1.1 Bisheriger Umgang mit Prozessvarianten

In einer frühen Interviewphase werden die Prozesse zweier Fachbereiche entsprechend des Mehr-Modell-Ansatzes erhoben. Bereits für diese beiden Prozessvarianten gestaltet sich die Modellierung mit existierenden Werkzeugen schwierig. Das Ausmodellieren der beiden ähnlichen Prozesse führt zu starken Redundanzen. Des Weiteren resultieren Prozessmodelle, die zwar generell sehr ähnlich ablaufen, aber aufgrund der spezifischen Begriffswelten der einzelnen Fachbereiche unterschiedlich beschrieben sind. Das heißt, die Prozessvarianten besitzen unterschiedliche Detaillierungsgrade und keine einheitliche Bezeichnung ähnlicher oder identischer Prozesselemente. Der Vergleich im Sinne einer Konsolidierung ist bereits bei zwei Prozessvarianten schwierig. Bei mehreren dutzend Prozessvarianten, wie im Falle des Berechnungsprozesses, ist eine Konsolidierung ausmodellierter Prozessmodelle nicht mehr praktikabel.

11.4.1.2 Projektunterstützung durch den Provop-Ansatz

Um eine Vielzahl an Prozessen effizient zu einem Soll-Prozess zu konsolidieren, wird der Provop-Ansatz adaptiert. Da im vorliegenden Projekt der ARIS Business Architect als Modellierungswerkzeug sowie ereignisgesteuerte Prozessketten (EPK) als Modellierungssprache gesetzt sind, verzichten wir auf eine direkte Anwendung des Provop Prototypen. Stattdessen übertragen wir den Provop-Ansatz entsprechend. Die zu konsolidierenden Prozessvarianten werden auf Basis der EPK in einer Spaltendarstellung angezeigt. Abbildung 11.6 zeigt beispielhaft drei Ausschnitte aus den Spaltendarstellungen verschiedener SOLL-Prozesse und ihrer Prozessvarianten. Der SOLL-Prozess ist jeweils links im Bild dargestellt. Es werden zunächst nur Funktionen[4] abgebildet, ohne einen expliziten Kontrollfluss vorzugeben. Es gilt implizit eine sequentielle Ausführung der Funktionen von oben nach unten. Verzweigungen werden in dieser Darstellung durch annotierte Split- und Join-Knoten übertragen. Diese Spaltendarstellung erlaubt eine platzsparende Modellierung des SOLL-Prozesses (d.h. Basisprozesses) und seiner Prozessvarianten in weiteren Spalten. Dabei findet im Prinzip über die Zeilen ein Vergleich der Prozessvarianten mit dem SOLL-Prozess statt. Das heißt wir betrachten nun für jede Funktion des SOLL-Prozesses, ob es eine entsprechende Funktion in der Prozessvariante gibt. Mögliche Ergebnisse dieser Analyse sind:

- **Identische Funktionen:** Sind die verglichenen Funktionen semantisch identisch, wird im Feld der Prozessvariante keine Funktion aufgeführt.
- **Zu konsolidierende Funktionen:** Sind die verglichenen Funktionen zwar semantisch identisch, tragen aber unterschiedliche Bezeichner, müssen diese bei der Überarbeitung des SOLL-Prozesses konsolidiert werden. Wir modellieren dazu in der Spalte der Prozessvariante eine Funktion, welche mit der zugehörigen Funktion im SOLL-Prozess über eine Kante verbunden wird.
- **Zusätzliche Funktionen im IST-Prozess:** Existiert eine Funktion im IST-Prozess, die keiner Funktion des SOLL-Prozesses zugeordnet werden kann, ist diese entweder überflüssig oder muss in der nächsten Überarbeitung des SOLL-Prozesses in diesen eingearbeitet werden. Im ersten Fall wird die Funktion nur in der Zeile des IST-Prozesses aufgeführt und als entfernbar dargestellt. Im zweiten Fall wird die Funktion an der entsprechenden Position im IST- und SOLL-Prozess aufgeführt und farbig hervorgehoben.
- **Zusätzliche Funktionen im SOLL-Prozess:** Existiert eine Funktion im SOLL-Prozess, die keiner Funktion des IST-Prozesses zugeordnet werden kann, ist diese Funktion

[4] In EPKs werden Aktivitäten bzw. Prozessschritte als Funktionen bezeichnet [Sch98].

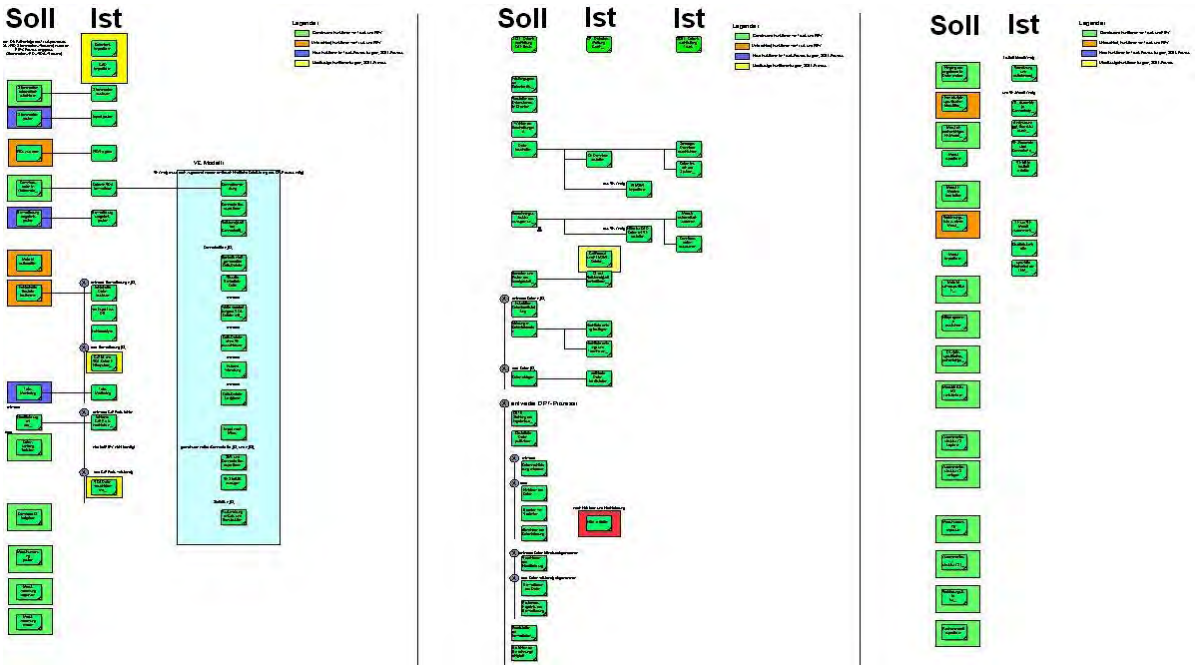


Abbildung 11.6: Spaltendarstellung von SOLL- und IST-Prozessen mit Hervorhebung der Änderungen

entweder überflüssig oder als neue Funktion zukünftig relevant. Entsprechend wird sie durch unterschiedliche Farben im SOLL-Prozess hervorgehoben.

- **Spezialisierung und Generalisierung:** Existiert eine Funktion im SOLL-Prozess, die durch mehrere Funktionen im IST-Prozesses abgebildet wird, liegt eine Spezialisierung vor. Die Funktionen werden im IST-Prozess gebündelt und mit der Funktion im SOLL-Prozess über eine Kante verknüpft. Dieses Vorgehen entspricht im umgekehrten Fall einer Generalisierung durch den IST-Prozess und wird analog zur Spezialisierung dargestellt.

11.4.2 Abbildung der Prozessvarianten mit Provop

Neben der angepassten Provop-Methodik zur direkten Unterstützung des Projektes caEdm, wird nebenläufig untersucht, wie der vorliegende Berechnungsprozess mittels Provop abgebildet werden kann. Aufgrund der Unvollständigkeit der Daten (das Projekt caEdm läuft noch über einen Zeitraum von mehreren Jahren) und der Größe des Prozessmodells (der Berechnungsprozess besteht aus ca. 150-200 Knoten) verzichten wir an dieser Stelle auf die graphische Abbildung des Basisprozesses und zugehöriger Optionen.

Basisprozess

Der Basisprozess für die nichtvarianten Phasen 1, 2 und 7 (vgl. Abbildung 11.7), ist bereits durch einen einheitlichen Prozess gegeben. Für die variantenbehafteten Phasen 3 bis 6 kann ein generischer Basisprozess identifiziert werden, der die wesentlichen Arbeitsschritte umfasst. Die konsolidierten SOLL-Prozesse stellen dann jeweils eine Variante des Prozesses dar. Abbildung 11.7 zeigt diese Struktur in einer Übersicht.

Optionen

Die notwendigen Optionen sind durch eine Modelldifferenzbildung zwischen Basisprozess und SOLL-Prozess ermittelbar. Durch die bereits dokumentierten Differenzen zwischen IST-

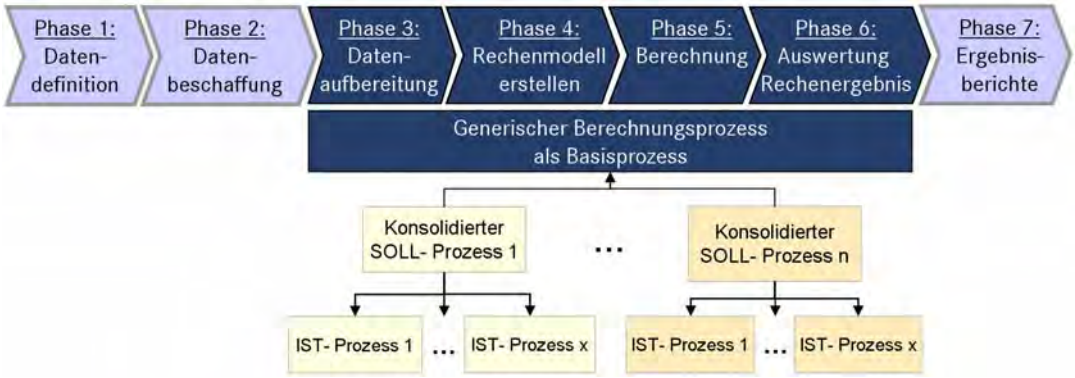


Abbildung 11.7: Struktur der Lösung

und SOLL-Prozess in der EPK-Spaltendarstellung, können diese direkt auf Optionen abgebildet werden. Dabei erlaubt der spaltenweise Vergleich der IST-Prozesse auch die Identifikation von Wiederverwendungspotentialen einzelner Änderungsoperationen, so dass nicht unbedingt alle Änderungen einer Spalte (d.h. eines IST-Prozesses), grob granular, d.h. zu einer Option, zusammengefasst werden müssen.

Kontextmodell des Berechnungsprozesses

Die starke Varianz des Berechnungsprozesses spiegelt sich in dessen Kontextmodell wider; Tabelle 11.2 zeigt daher nur einen Ausschnitt. Die aufgeführten Kontextvariablen sind wesentliche Einflussfaktoren des Berechnungsprozesses, d.h. sie führen zu Prozessvarianten. Einige der Kontextvariablen sind für die varianten Prozessdaten relevant, d.h. abhängig von den verwendeten Entwurfsmethoden und Berechnungswerkzeugen werden unterschiedliche Datenformate produziert.

Tabelle 11.2: Ausschnitt aus dem Kontextmodell des Berechnungsprozesses

Variablenname	Wertebereich	Modus
Berechnungsdisziplin	Crashverhalten, Geräuschentwicklung, Vibration, Rauigkeit, Abgasrückführung, Thermische Absicherung, Powertrain, Betriebsfestigkeit, Passive Sicherheit, Energiemanagement, Mehrkörpersimulation, Strukturmechanik, Ventil- und Steuertrieb, ...	statisch
Fahrzeugtyp	Bus, Van, PKW, LKW	statisch
Designmethoden	FEM, MBS, CFD, FVM	statisch
Berechnungswerkzeuge	Matlab, Fortran, CATIA, PERMAS, Femfat, Hypergraph, FEKO, Elkin, PROSTAR, Polyworks, Texware, WLG, Midas Pre, Midas Post, ANSA, Auto-Mesh, Dads, CableMod, Ventil, EnSight, FFT AN, POSRAD, Simpack, Simdrive, VL Motion, StarCD-ICE, Shiva, Adams	dynamisch
...	...	...

Die Zuordnung von IST-Prozessen zu einem Kontext, wird im Projekt caEdm durch entsprechende Benennung der Modelle bzw. der Spalten in der EPK-Spaltendarstellung umgesetzt. Die Kontextvariablen und deren Wertebereiche werden parallel erhoben. Zur Abbildung der Kontextabhängigkeit von Optionen in Provop, können diese Modell- und Spaltenbezeichner direkt in Form von Kontextbedingungen formuliert und mit den entsprechenden Optionen verknüpft werden.

11.4.3 Gewonnene Erkenntnisse

Im Bereich der Berechnungsprozesse zur digitalen Absicherung liegt eine Vielzahl komplexer Prozessvarianten vor. Die bisherige Erhebung dieser Varianten in separaten Prozessmodellen hat sich als schwierige Aufgabe herausgestellt. Die Konsolidierung der Variantenmodelle wird aber nicht nur durch die vorliegende Modellkomplexität erschwert, sondern auch wegen unterschiedlicher Detaillierungsgrade der Prozessmodelle und unterschiedlichen Bezeichnern für identische Modellelemente. Um diesen Praxisproblemen zu begegnen, adaptieren wir den Provop-Lösungsansatz, wobei nicht auf den Provop-Prototypen zurückgegriffen werden kann. Stattdessen ist es unser Ziel, basierend auf heutigen Prozessmodellierungswerkzeugen, eine Erhebung bzw. Abbildung der Prozessvarianten gemäß dem Provop-Konzept vorzunehmen. Das Ergebnis ist eine Spaltendarstellung des Basisprozesses und notwendiger Anpassungen (d.h. Optionen) in weiteren Spalten. Die Hervorhebung von Änderungen, in Anlehnung an die in Abschnitt 9.2.1 vorgestellten Darstellungsformen, hat sich als hilfreich erwiesen. Die Spaltendarstellung erlaubt einen übersichtlichen Vergleich mehrerer Prozessvarianten und zeigt zugleich die Unterschiede dieser Prozessvarianten in hohem Detaillierungsgrad auf.

Im Rahmen der Fallstudie werden auch die Änderungstypen *Spezialisierung* und *Generalisierung* von Aktivitäten betrachtet. Dabei stellt vor allem die Spezialisierung einen relevanten Fall für eine Änderungsoperation zur Abbildung von „Varianten von Varianten“ dar [MCH07]. In Provop ist eine Spezialisierung prinzipiell durch Kombination einer `DELETE`- und `INSERT`-Operation abbildbar. Eine explizite Spezialisierungsoperation bietet allerdings eine größere Transparenz für den Modellierer und abstrahiert von den zugrunde liegenden Änderungsoperationen. Die Generalisierung von Aktivitäten durch eine Änderungsoperation erscheint nicht als Anwendungsfall des Variantenmanagements, ist aber ein relevanter Aspekt hinsichtlich der geeigneten Visualisierung von Prozessmodellen [BRB07, Bob08].

Ein wichtiger Schwerpunkt des Projektes caEdm liegt auf der Erhebung varianter Datenmodelle und Systeme. Deren Abbildung, nach dem Provop-Ansatz, wird nicht weiter betrachtet.

11.5 Fallstudie 3: Registrierungsprozess von Stadtverwaltungen

In [GWJV+09] präsentieren die Autoren eine Fallstudie aus der Domäne *Stadtverwaltung* zur Validation ihres Ansatzes der Referenprozessmodellierung. Dazu werden in vier niederländischen Stadtverwaltungen die Registrierungsprozesse für folgende Szenarien erhoben: *Eheschließung*, *Anerkennung der Vaterschaft eines ungeborenen Kindes*, *Anmeldung eines Neugeborenen* und *Sterbefall*. Durch die wenig restriktiven zentralen Richtlinien, ergeben sich vier unterschiedliche Varianten pro Registrierungsprozess.

Die Autoren zeigen die konkreten Ergebnisse ihrer Fallstudie für den Prozess zur Anerkennung der Vaterschaft. Die Studie hat gezeigt, dass die erhobenen Prozessvarianten in einem gemeinsamen Referenzprozessmodell mit der erweiterten Prozessbeschreibungssprache *configurable Yet Another Workflow Language (cYAWL)* abgebildet werden können [GvdAJVlR07, GvdAJV08]. Es wird außerdem gezeigt, wie die bereits in [RLS+07, GvdAJVlR07] vorgestellten Methoden zur fragebogenbasierten Konfiguration von Prozessvarianten eingesetzt werden können, um die jeweiligen Prozessvarianten aus dem Referenzprozessmodell abzuleiten.

Ziel unserer Fallstudie ist es, die in [GWJV+09] veröffentlichten Prozessvarianten nach dem Provop-Ansatz abzubilden und einen direkten Vergleich der beiden Ansätze zu führen.

11.5.1 Szenario des Registrierungsprozesses zur Anerkennung der Vaterschaft

Um die Vaterschaft vor Geburt eines Kindes amtlich anzuerkennen, ist ein spezieller Registrierungsprozess erforderlich. Dieser ist in sechs Phasen eingeteilt (vgl. Abbildung 11.8): Der Prozess beginnt in Phase 1 mit dem Stellen des Antrags auf Anerkennung der Vaterschaft und der Prüfung der Identität des Antragstellers. Anschließend werden in Phase 2 personenbezogene Daten abgefragt, z.B. Familienstand und Wohnort des Antragstellers. Nach Prüfung des Antrags wird dieser in Phase 3 entweder bewilligt oder abgelehnt. Bei Ablehnung terminiert der Prozess. In Phase 4 werden letzte Formalien geprüft. So wird erfragt, ob das Kind das erste Kind aus der Beziehung ist. Danach ist der Name des Kindes auszuwählen. Abschließend wird eine Urkunde ausgestellt, kopiert und abgelegt.

Abbildung 11.8: Phasen des Registrierungsprozesses

11.5.1.1 Varianten des Registrierungsprozesses

Aufgrund der wenig restriktiven, zentralen Vorgaben zur Bearbeitung eines Antrags auf Anerkennung der Vaterschaft eines ungeborenen Kindes, haben sich in den einzelnen Stadtverwaltungen unterschiedliche Prozesse etabliert. Diese sind an die spezifischen Rahmenbedingungen sowie die historisch gewachsenen Strukturen der einzelnen Behörden angepasst. Das heißt die Varianten unterscheiden sich in Bezug auf die Reihenfolge der Bearbeitungsschritte. So können wir, bezogen auf die oben beschriebenen Phasen des Registrierungsprozesses, die folgenden vier Phasenmodelle für die Varianten identifizieren (vgl. Abbildung 11.9)[5]:

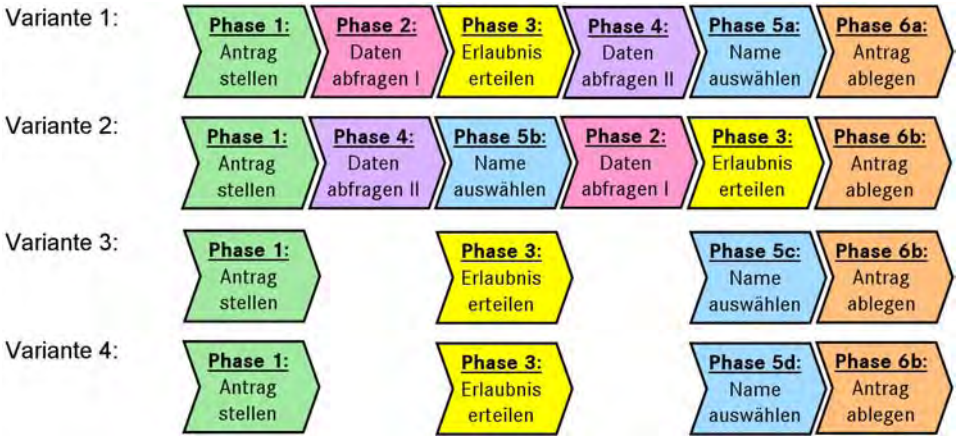


Abbildung 11.9: Phasen der Prozessvarianten

Die Varianten unterscheiden sich bzgl. der Ausführungsreihenfolge der Phasen. Darüber hinaus werden in manchen Varianten ganze Phasen ausgelassen oder bestimmte Phasen variieren zwischen den Varianten stark. Phase 5 etwa wird in jeder Stadtverwaltung anders gehandhabt (hier ausgedrückt durch Phasen 5a, 5b, 5c und 5d). Phase 6 wiederum wird entweder als Phase 6a oder Phase 6b ausgeführt. Die Bearbeitungsschritte in den Phasen 1 bis 4 sind in allen vier Prozessvarianten identisch, allerdings variiert die Reihenfolge der Phasen 2 und 4.

[5]Die Farbe der Phasen entspricht keiner speziellen Semantik, sondern dient lediglich der besseren Unterscheidbarkeit.

Abbildungen B.1 bis B.4 in Anhang B zeigen die Prozessmodelle der vier Varianten im Details. Wir haben die verschiedenen Phasen für eine bessere Vergleichbarkeit der Variantenmodelle entsprechend hervorgehoben.

11.5.1.2 Bisheriger Umgang mit dem Registrierungsprozesses

Bisher werden die Registrierungsprozesse dezentral, d.h. in Verantwortung der jeweiligen Gemeinden, modelliert und verwaltet. Es gibt kein zentral vorgegebenes Prozessmodell. Aufgrund der geringen Komplexität der Prozessvarianten (max. 22 Knoten pro Prozessgraph), werden alle alternativen Abläufe (z.B. unterschiedliche Vorgehensweise bei der Namensauswahl für ausländische Antragsteller) direkt in die Modelle integriert. Im Prinzip stellt jedes der vier Variantenmodelle eine Realisierung des Ein-Modell-Ansatzes dar, während auf die Gesamtzahl aller Stadtverwaltungen betrachtet ein Mehr-Modell-Ansatz vorliegt (vgl. Abschnitt 4.2.1 und 4.2.2). Letzteres bedeutet, dass bei grundlegenden Gesetzesänderungen, etwa wenn zusätzliche Daten und Dokumente des Antragstellers vorliegen müssen, die Variantenmodelle der einzelnen Kommunen jeweils separat angepasst werden müssen.

11.5.1.3 Abbildung der Prozessvarianten mit cYAWL

Aufgrund der Nachteile der bisherigen Handhabung der Registrierungsprozesse beschreibt [GWJV+09] einen Ansatz, um für alle Gemeinden ein Referenzprozessmodell zu definieren. Die einzelnen Varianten entstehen durch Konfiguration des Referenzprozessmodells. Zur Konfiguration der Variantenmodelle stehen die Operationen `Blocking` und `Hiding` zur Verfügung [GvdAJVlR07, GvdAJV08]. Diese erlauben es insbesondere, einzelne Prozessfragmente aus dem Referenzprozessmodell zu entfernen, um eine konkrete Variante abzuleiten. Abbildung 11.10 zeigt das Referenzprozessmodell, abgebildet als cYAWL-Modell, für den Registrierungsprozess zur Anerkennung der Vaterschaft eines ungeborenen Kindes. Die Konfigurationsdetails sind hier ausgeblendet.

Für die benutzerseitige Konfiguration der Varianten wird eine Fragebogenmethode eingesetzt [RGDvdA07, RLS+07]. Hier werden freitextliche Fragen beschrieben, deren Antwort zu einem *Domain Fact* führt (vgl. Abbildung 11.11). Die Domain Facts werden dann, in einer internen Tabelle, auf die konfigurierten Knoten des Referenzprozessmodells abgebildet. Auf diese Weise kann ein konkretes Variantenmodell durch Auswerten des Fragebogens erstellt werden.

11.5.2 Abbildung der Prozessvarianten mit Provop

Die Varianten des Registrierungsprozesses können auch mit Hilfe des Provop-Ansatzes abgebildet werden. Dazu kann ein beliebiger Basisprozess definiert werden. In Anlehnung an den Ansatz der Referenzprozessmodellierung [GWJV+09], besteht eine Möglichkeit darin den Basisprozess als Obermenge aller Prozessvarianten zu definieren. In diesem Fall resultieren im Prinzip die gleichen Konfigurationsschritte wie bei dem in [GWJV+09] beschriebenen Ansatz. Das heißt die Operationen des Blockings und Hidings werden auf Provop-`DELETE`-Operationen abgebildet. Kontextinformationen können aus den Domain Facts abgeleitet werden bzw. ergeben sich durch Angabe der spezifischen Stadtverwaltung.

Da sich die Varianten des Registrierungsprozesses in erster Linie durch eine unterschiedliche Reihenfolge der einzelnen Phasen auszeichnen (vgl. Abbildung 11.9), ist der Einsatz von `MOVE`-Operationen hilfreich. Das heißt, wir beschreiben den Basisprozess nicht als Obermenge aller Prozessvarianten, sondern wählen das Prozessmodell von Variante 1 als Basisprozess

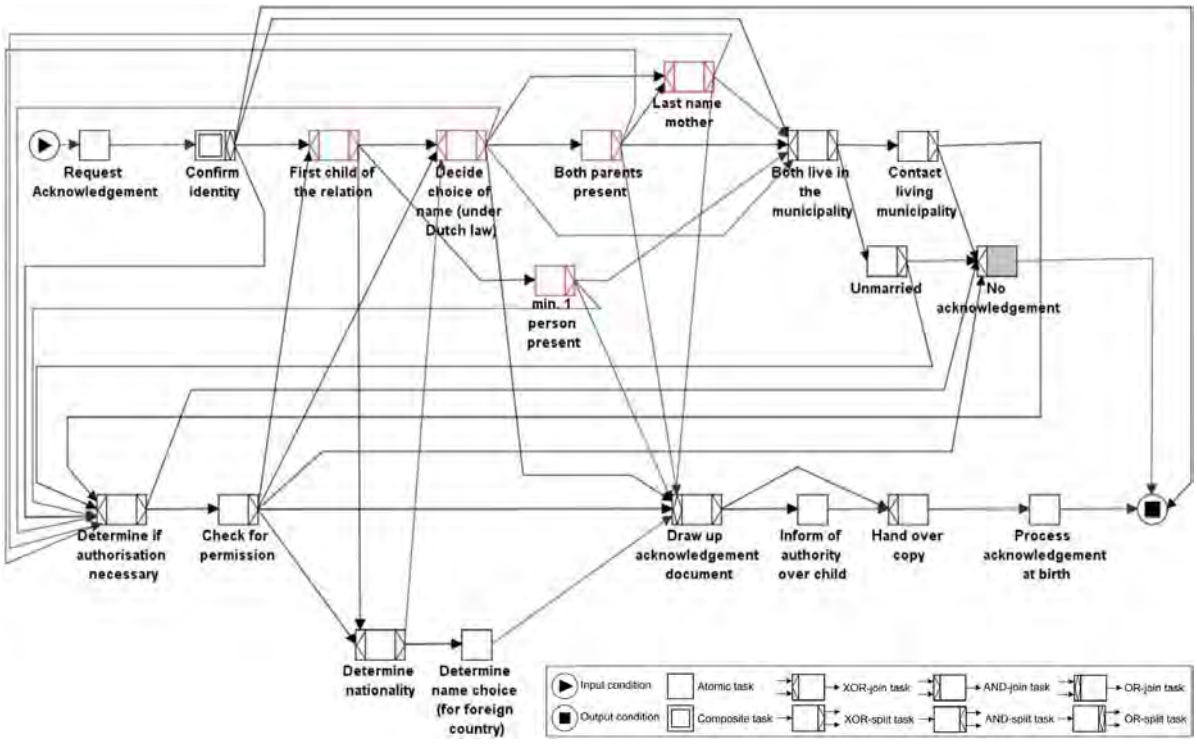


Abbildung 11.10: Referenzprozessmodell für die Anerkennung der Vaterschaft eines ungeborenen Kindes in YAWL-Notation nach [GWJV+09]

(vgl. Abbildung 11.12). Dies ist sinnvoll, da Variante 1 alle Phasen enthält und eine starke Ähnlichkeit zu den anderen Prozessvarianten aufweist.[6]

Die Optionen zur Abbildung der übrigen Varianten können nun aus der Differenz der Variantenmodelle abgeleitet werden. Auf Ebene der Phasenmodelle der Varianten ergeben sich die in Abbildung 11.13 dargestellten Optionen.

Option 1 verschiebt die Phasen 2 und 4 des Basisprozesses nach Phase 5. Optionen 2, 3 und 4 löschen jeweils einzelne Schritte aus dem Basisprozess.[7] Die vier Prozessvarianten (vgl. Abbildung B.1 bis Abbildung B.4) ergeben sich dann wie folgt:

- **Variante 1:** keine Option anwenden.
- **Variante 2:** Optionen 1 und 3 anwenden.
- **Variante 3:** Optionen 1, 2 und 4 anwenden.
- **Variante 4:** Optionen 1, 2 und 5 anwenden.

Dieser feingranulare Ansatz ermöglicht die Wiederverwendung von Optionen. Werden der Basisprozess allerdings zentral definiert und die Optionen (d.h. die Information zur Konfiguration der Prozessvarianten) dezentral in den jeweiligen Gemeinden verwaltet, so ist ein grobgranulares Schneiden der Änderungsoperationen sinnvoller, auch wenn sich hieraus starke Redundanzen ergeben. Unter der Annahme, dass Variante 1 der vorgegebene Basisprozess ist, ergeben sich die in Abbildung 11.14 dargestellten Optionen.

[6] Es ist auch denkbar, die Variante 3 als Basisprozess zu wählen. In diesem Fall entfernen wir uns allerdings zu sehr vom Gedanken des Referenzprozesses, da hier `INSERT`-Operationen zum Einfügen ganzer Prozess-Phasen erforderlich werden.

[7] Hier kann durch Angabe von Prozessfragmenten die Anzahl der Optionen noch reduziert werden. Aus Gründen der Nachvollziehbarkeit haben wir darauf allerdings verzichtet.

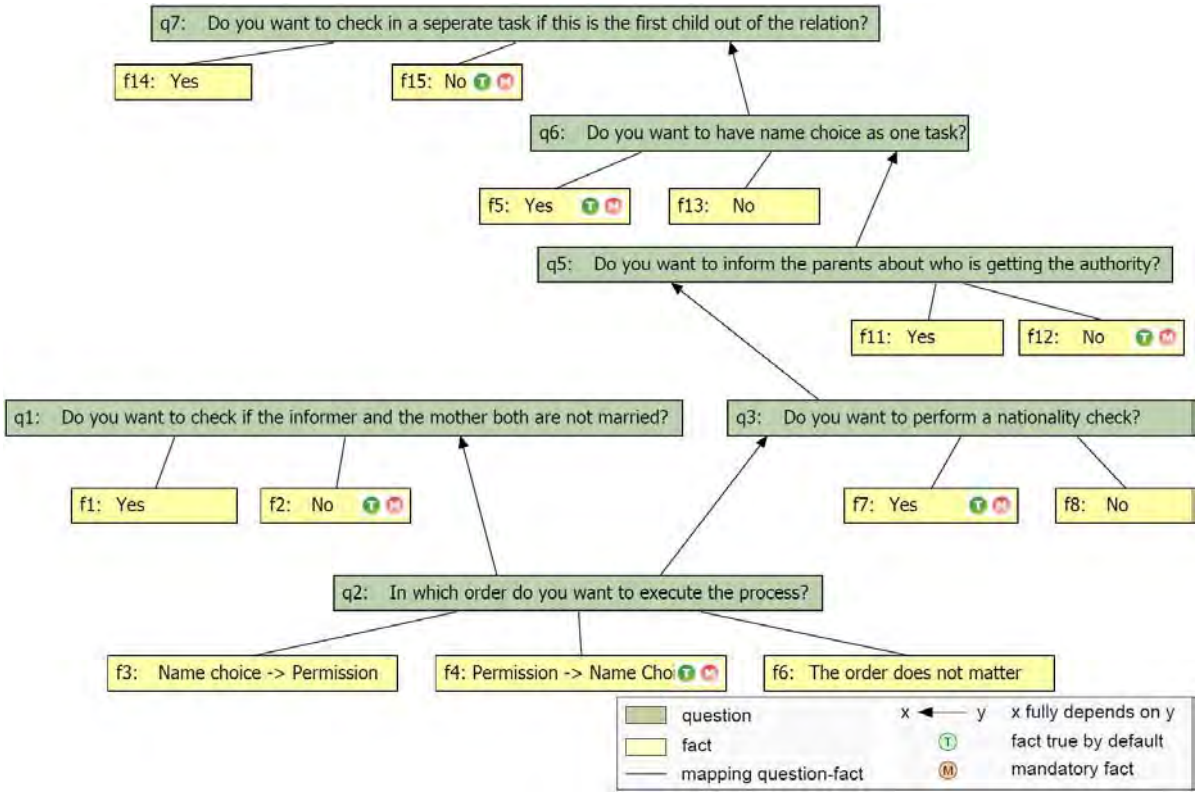


Abbildung 11.11: Domain Facts des Registrierungsprozesses nach [GWJV+09]

Die vier Prozessvarianten (vgl. Abbildung B.1 bis Abbildung B.4) können wie folgt konfiguriert werden:

- **Variante 1:** keine Option anwenden.
- **Variante 2:** Option 1 anwenden.
- **Variante 3:** Option 2 anwenden.
- **Variante 4:** Option 3 anwenden.

Der Kontext der Optionen bzw. der Prozessvarianten wird bei diesem Ansatz ausschließlich durch die jeweilige Gemeinde bestimmt. Das bedeutet, dass keine Kontextmodelle und Kontextabhängigkeiten mehr zur Konfiguration der Prozessvarianten gepflegt werden müssen. Es werden stattdessen immer alle Optionen einer Gemeinde auf den Basisprozess angewendet.

11.5.3 Vergleich von Referenzprozessmodellierung mit Provop

Der Vorteil der Referenzprozessmodellierung nach [GWJV+09], im Vergleich zu einem Mehr-Modell-Ansatz, besteht darin, dass zentral vorgegebene Änderungen direkt an einem Modell vorgenommen werden können. Dabei stellt die Fragebogenmethode einen sehr guten Ansatz zur abstrakten Konfiguration von Prozessvarianten aus einem Referenzprozess dar. Der Variantenverantwortliche kann bei der Konfiguration vom zugrunde liegenden Prozessmodell abstrahieren. Diese Arbeitserleichterung seitens des Variantenverantwortlichen resultiert allerdings in einem Mehraufwand für den Variantenmodellierer. Letzterer muss für alle konfigurierbaren Knoten die entsprechende Abbildung auf *Domain Facts* beschreiben. Die Ablauflogik des Prozesses ist ohne Kenntnis zur Konfiguration der Knoten nicht mehr eindeutig bzw. direkt aus dem Modell ablesbar. Nachteilig ist ebenfalls, dass durch die unterschiedliche Reihenfolge der Aktivitäten bzw. Phasen in den einzelnen Prozessvarianten und dem Fehlen

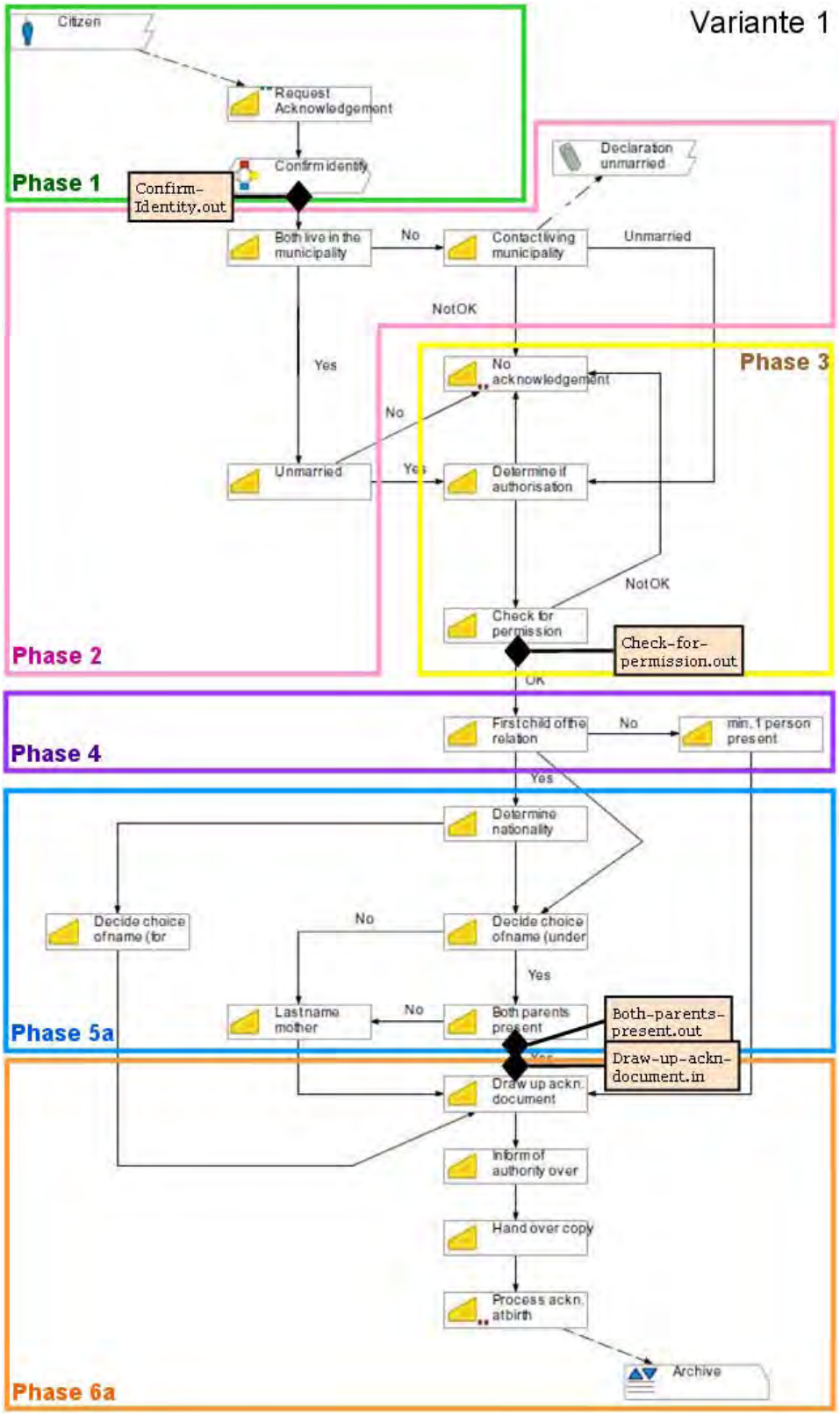


Abbildung 11.12: Variante 1 als Basisprozess

einer Verschiebeoperation eine komplexe Kontrollfluss-Struktur des Referenzprozessmodells resultiert.

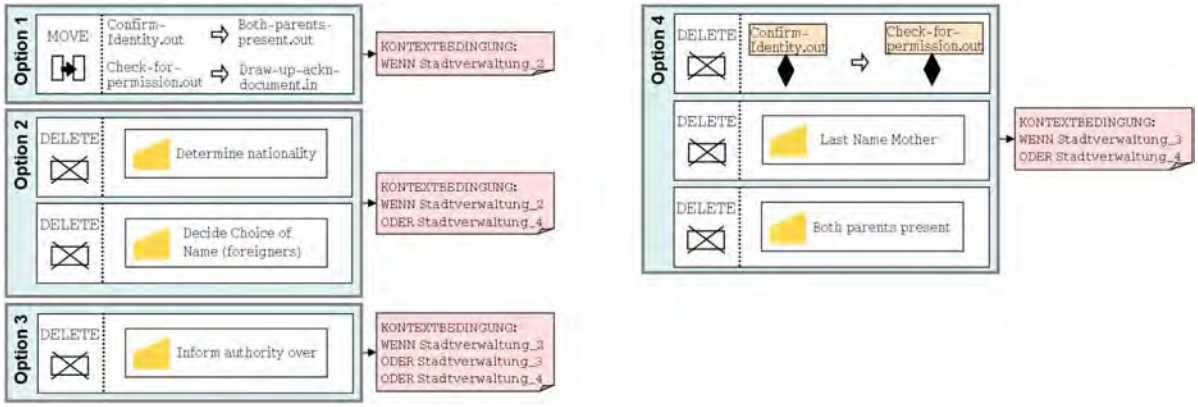


Abbildung 11.13: Feingranulare Optionen für den Registrierungsprozess

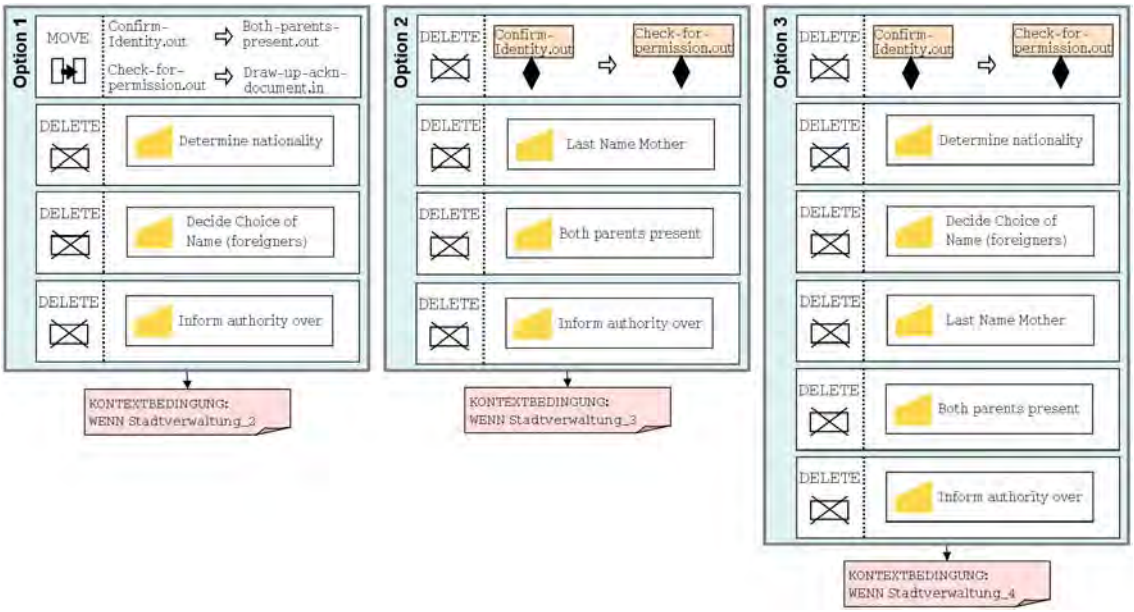


Abbildung 11.14: Grobgranulare Optionen für den Registrierungsprozess

Ein weiterer wichtiger Aspekt der Referenzprozessmodellierung ist die dezentrale Verwaltung der Prozessmodelle durch die einzelnen Stadtverwaltungen. Generell ergibt sich durch einen Ein-Modell-Ansatz die Situation, dass alle Prozessvarianten von einer zentralen Stelle (d.h. einem Referenzprozessverantwortlichem) zu einem Referenzprozessmodell integriert werden müssen. Das heißt das Referenzprozessmodell muss alle spezifischen Ablaufvarianten der Gemeinden berücksichtigen, um nicht zu restriktiv oder ungenau zu sein. Dabei ist zu hinterfragen, ob es für die verschiedenen Stadtverwaltungen überhaupt erstrebenswert ist, ihre spezifischen Prozesse von zentraler Stelle einsehen, erheben und dokumentieren zu lassen oder ob die Einhaltung der wenigen Richtlinien und Vorgaben nicht ausreicht.

Ein wesentlicher Vorteil von Provop ist, dass neben dem Löschen bzw. Auslassen von Aktivitäten auch das Verändern der Aktivitätenreihenfolge in einem Prozessmodell möglich ist. Die von Provop angebotene `MOVE`-Operation, welche das Verschieben von Prozessfragmenten ermöglicht, erlaubt somit einfach strukturiertere (Referenz-)Prozessmodelle, da doppelt modellierte Aktivitäten oder Rücksprünge, wie in [GWJV+09] erforderlich, überflüssig werden. Ein weiterer Vorteil von Provop im Szenario der Registrierungsprozesse ist, dass ein dezentrales Verwalten der Prozessvarianten möglich wird, auch wenn von einem zentralen

Referenzprozessmodell für alle Gemeinden ausgegangen werden kann. Dazu ist nur das Wissen über die modellierten Prozesselemente (z.B. Aufsetzpunkte, Aktivitäten) erforderlich und kein internes Fachwissen, hinsichtlich der Konfigurationen von Knoten und entsprechender *Domain Facts*.

Änderungen am Basisprozess können darüber hinaus leicht nachgezogen werden. Im Ansatz der Referenzprozessmodellierung ist an dieser Stelle eine Neukonfiguration aller Variantenmodelle, mit Hilfe der Fragebogenmethode, erforderlich. Des Weiteren bietet Provop hinsichtlich der Festlegung des Basisprozesses mehr Freiheiten als ein Referenzprozessmodell. So kann der Basisprozess auch als Standardprozess, Schnittmenge aller Varianten oder als häufigste Prozessvariante gewählt werden (vgl. Abschnitt 5.2). Nachteil der dezentralen Verwaltung der Prozessvarianten ist jedoch, dass eine Standardisierung und Harmonisierung erschwert wird. Hierzu müssten die Optionen der Stadtverwaltungen auf Gemeinsamkeiten untersucht werden und langfristig die Freiheitsgrade der Variantenmodellierer vor Ort (d.h. in den Gemeinden) eingeschränkt werden.

11.5.4 Gewonnene Erkenntnisse

Die unterschiedlichen Reihenfolgen der Phasen des Registrierungsprozesses zeigen den sinnvollen Einsatz von `MOVE`-Operationen. Diese führen, im Vergleich zur ausschließlichen Verwendung von `DELETE`-Operationen im Referenzprozessmodell-Ansatz, zu einer besseren Les- und Handhabbarkeit des Basisprozesses.

Die Fallstudie zeigt ein Szenario für die verteilte Modellierung von Prozessvarianten.[8] Dabei ist der Provop-Ansatz gut geeignet, um einerseits die zentrale Entwicklung eines Basisprozesses und andererseits seine dezentrale Anpassung mittels Optionen zu erlauben. Diese Struktur führt zu einer sehr guten Skalierbarkeit des Ansatzes: Für jede Stadtverwaltung muss jeweils nur eine Option modelliert und gepflegt werden. Diese Option besteht in den aufgeführten Beispielen, trotz den oft größeren strukturellen Änderungen am Basisprozess, aus maximal sechs Änderungsoperationen.

11.6 Fallstudie 4: Untersuchungsprozess in Kliniken

Durch mehrjährige Tätigkeit eines Projektmitarbeiters in einer medizinischen Universitätsklinik, konnten detaillierte Erkenntnisse über die dort typischen Prozessabläufe gewonnen werden. Das betrachtete Klinikum setzt sich aus verschiedenen spezialisierten Kliniken zusammen, etwa einer Kinderklinik, einer Frauenklinik und einer Medizinischen Klinik (Innere Medizin).

Im Rahmen der durchgeführten Fallstudie haben wir organisatorische Abläufe für das diagnostische Prozedere untersucht. Therapeutische Prozesse werden im Folgenden nicht betrachtet. Aus Platzgründen stellen wir die Prozesse zum Teil in abstrahierter Form dar.

11.6.1 Szenario des Untersuchungsprozesses

In einer Klinik existieren verschiedene Prozessbereiche. Einer dieser Bereiche ist die Diagnostik. Ein Untersuchungsprozess kann in die folgenden vier Phasen eingeteilt werden (vgl. Abbildung 11.15): Zunächst ordnet eine Krankenstation bzw. ein Stationsarzt eine Untersuchung an. Anschließend wird die Untersuchung vorbereitet, z.B. wird der Patient über die Untersuchung aufgeklärt oder es werden Proben entnommen. Danach erfolgt die eigentliche

[8] Ein vergleichbares Szenario haben wir in Abschnitt 2.2 bereits mit der Domäne des Werkstattprozess erläutert.

Durchführung der Untersuchung. Abschließend wird der Befund erstellt und an die Krankenstation bzw. den Stationsarzt zurückgemeldet.

Abbildung 11.15: Phasen des Untersuchungsprozesses

11.6.1.1 Varianten des Untersuchungsprozesses

Für den Untersuchungsprozess existieren zwei Typen von Prozessvarianten: Die Patienten- und die Laboruntersuchung. Wir beschreiben im Folgenden die Standardprozesse dieser beiden Typen und stellen jeweils die wichtigsten Prozessvarianten vor. Abbildung 11.16 zeigt die Struktur der Prozessvarianten.

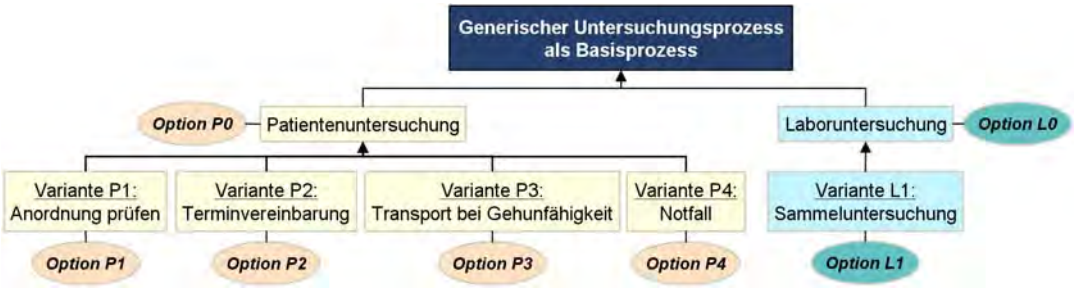


Abbildung 11.16: Hierarchie der Varianten des Untersuchungsprozesses

Patientenuntersuchung

Abbildung 11.17 zeigt den organisatorischen Ablauf einer Patientenuntersuchung. Dieser startet mit der Anordnung der Untersuchung (z.B. CT, Röntgen, EKG). Anschließend werden die Untersuchung bei der betreffenden Stelle (z.B. Röntgen) angefordert und ein Termin vereinbart. Danach wird der Patient über die Untersuchung aufgeklärt und entsprechend vorbereitet. Nach Abruf des Patienten wird er in der Untersuchungsstelle lokal vorbereitet, bevor die eigentliche Untersuchung schließlich stattfindet. Danach werden der Patient nachversorgt und der Befund erstellt. Dieser wird an die anfordernde Stelle zurück übermittelt, gelesen und validiert.

Für diesen Standardprozess der Patientenuntersuchung existieren variantenspezifische Abweichungen, die auch kombiniert auftreten können. Die wesentlichen davon sind nachfolgend aufgeführt.

- **Variante P1 (Anordnung prüfen):** Abhängig von der Untersuchungsart und den Risiken der Untersuchung (invasive Untersuchungen sind z.B. risikoreicher als nicht-invasive), muss die angeordnete Untersuchung zunächst von einem Facharzt geprüft werden. Diese Prüfung findet nach dem Vier-Augen-Prinzip statt. Ist die Untersuchung aufgrund der Risiken oder anderer Faktoren nicht möglich, bricht der Prozess an dieser Stelle ab.

- **Variante P2 (Terminvereinbarung):** Abhängig von der leistungserbringenden Stelle findet die Terminvereinbarung in unterschiedlicher Art und Weise statt. So ist es in manchen Funktionsbereichen (bzw. Untersuchungsstellen) üblich, einen Termin zu vereinbaren, während andere nur eine Registrierung des Patienten durch die anfordernde

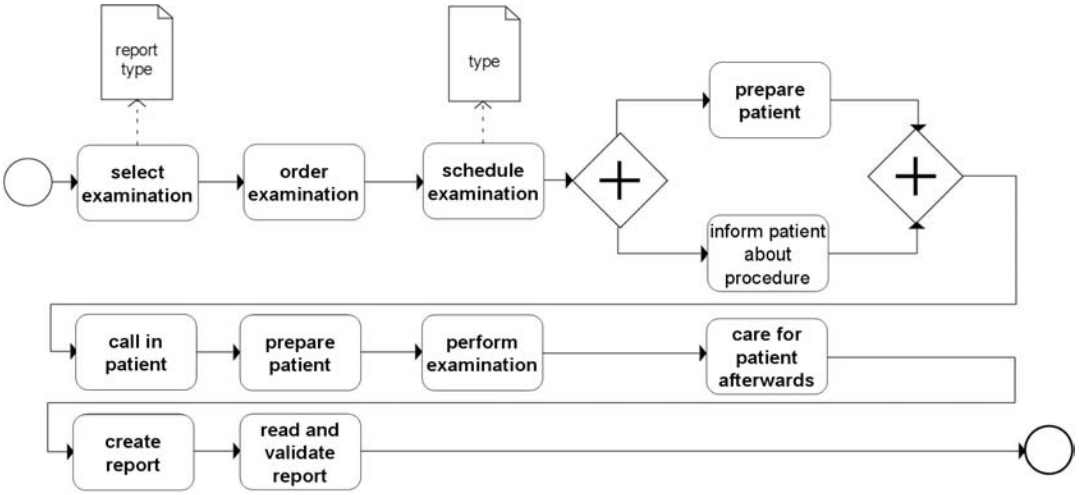


Abbildung 11.17: Standardprozess der Patientenuntersuchung

Krankenstation erfordern. Es gibt außerdem Untersuchungsstellen, die für bestimmte Untersuchungen gar keine Anmeldung erfordern. Die Art der Terminvereinbarung korreliert anschließend mit dem Patientenabruf. Dieser ist meist nur dann erforderlich, wenn zuvor ein Termin vereinbart worden oder eine Registrierung erfolgt ist.

- **Variante P3 (Transport bei Gehunfähigkeit):** Es ist unter Umständen erforderlich, nicht gehfähige Patienten zur Untersuchung zu transportieren. Dieser Transport muss dann, im Anschluss an die Untersuchung, auch zurück auf die Krankenstation erfolgen. Gehfähige Patienten müssen im Regelfall nicht transportiert werden.
- **Variante P4 (Notfall):** Liegt ein Notfall vor, so muss der Befund einer Untersuchung sofort an die Krankenstation zurückgemeldet werden. Das heißt, er wird dem Patienten direkt oder – beim Transport eines nicht gehfähigen Patienten – dem Personal mitgegeben. Die Befundart wird bei der Anordnung einer Untersuchung vom Stationsarzt spezifiziert.

Neben diesen Varianten können auch Kombinationen auftreten. Zum Beispiel können unterschiedliche Terminvereinbarungen getroffen werden, unabhängig davon, ob der Patient transportiert werden muss oder nicht. Weitere Varianten sind: Abbruch der Untersuchungsdurchführung (etwa wenn der Patient nicht nüchtern ist, obwohl die Untersuchung dies erfordert) und Untersuchungsart (z.B. CT, Röntgen, EKG). Hieraus resultieren unterschiedliche Tätigkeiten bei der Patientenvorbereitung, Patientenaufklärung und Durchführung der Untersuchung. Aus Platzgründen verzichten wir auf eine Betrachtung dieser Varianten.

Ausnahmefälle, etwa dass ein Patient erst durch eine Untersuchung gehunfähig wird (z.B. aufgrund Nachwirkungen einer Betäubung), werden nicht als Prozessvarianten gehandhabt. Insgesamt ergeben sich im vorliegenden Szenario 16 Prozessvarianten für die Patientenuntersuchung, mit maximal 16 Knoten (d.h. Aktivitäten und Strukturknoten).

Laboruntersuchung

Abbildung 11.18 zeigt den Standardprozess der Laboruntersuchung. Der Prozess startet mit der Anordnung einer Laboruntersuchung. Anschließend wird die entsprechende Probe entnommen (z.B. Blut, Urin) und an das Labor versandt. Dort wird die Probe untersucht und der Befund erstellt. Dieser wird anschließend an die Krankenstation zurückgemeldet und vom anfordernden Arzt validiert.

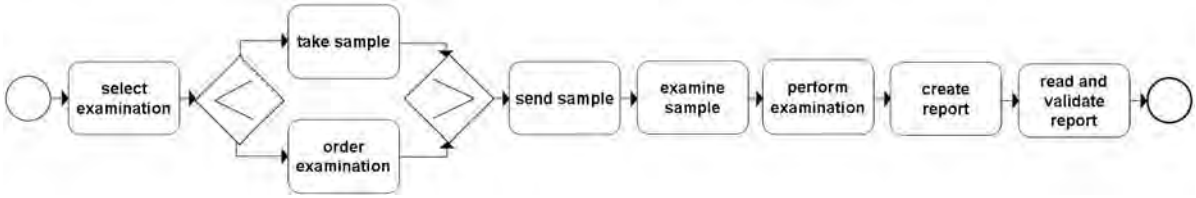


Abbildung 11.18: Standardprozess der Laboruntersuchung

Neben diesem Standardprozess der Laboruntersuchung existiert noch eine wesentliche Prozessvariante:
Variante L1 (Sammeluntersuchung): Für manche Laboruntersuchungen ist es erforderlich, periodisch (z.B. einmal täglich über einen Zeitraum von einer Woche oder dreimal täglich über zwei Tage hinweg) eine Probe zu sammeln und an das Labor zu versenden. In diesem Fall wird die Untersuchung einmalig angeordnet und angefordert, und es resultiert auch nur ein einziger Befund. Dazwischen werden die Schritte der Probenentnahme, das Versenden der Probe und die Analyse der Probe iteriert. Die Ergebnisse der jeweiligen Einzelbefunde werden im Labor zwischengespeichert. Dieser Befund ist anders strukturiert als ein normaler Laborbefund, z.B. werden bei solchen Sammeluntersuchungen dann Minimal- und Maximalwerte angegeben oder die Werte über die Zeit betrachtet.

In Summe ergeben sich für die Labouruntersuchung somit 2 Prozessvarianten, mit maximal 10 Knoten (d.h. Aktivitäten, Struktur- und Schleifenknoten).

11.6.1.2 Bisheriger Umgang mit Prozessvarianten

Bisher werden die zahlreichen Varianten des Untersuchungsprozesses nach dem Mehr-Modell-Ansatz gehandhabt (vgl. Abschnitt 4.2.1). Dies bedeutet, dass für jede Variante ein separat erstelltes und zu pflegendes Prozessmodell existiert. Insgesamt sind dies 20 Variantenmodelle. Die Abbildung der Varianten, nach dem Mehr-Modell-Ansatz, führt zu den bekannten Problemen, wie schlechte Wartbarkeit der Variantenmodelle und fehlender Transparenz zu variantenspezifischen Abweichungen und deren Rahmenbedingung (d.h. Kontext). Außerdem stellt sich die Implementierung der Prozessvarianten als schwierig und aufwendig heraus, da keine Wiederverwendung von Prozessfragmenten stattfindet. Die resultierenden Variantenmodelle sind außerdem zu starr, um die dynamischen Arbeitsabläufe in einer Klinik abzubilden. So ist z.B. ein Wechsel zwischen verschiedenen Variantenmodellen nur durch Abbruch und Neustart einer anderen Prozessinstanz möglich. Abhilfe schafft hier die Nutzung eines adaptiven WfMS wie ADEPT, welches dynamische Änderungen zur Laufzeit unterstützt [DR09, RRKD05]. Diese dynamischen Änderungen entsprechen jedoch nicht vollständig den Zielen des Managements von Prozessvarianten.

11.6.2 Abbildung der Prozessvarianten mit Provop

Im Rahmen unserer Fallstudie wird der Provop-Prototyp verwendet, um die Prozessvarianten zu modellieren und zu konfigurieren. Im Folgenden beschreiben wir die konkrete Abbildung der Varianten unter Verwendung entsprechender Screenshots.[9]

Basisprozess. Es existieren zwei Typen von Untersuchungsprozessen: Labor- und Patientenuntersuchungen. Um diese Typen abzubilden, können wir entweder zwei verschiedene Basisprozesse modellieren oder alle Varianten mit einem Basisprozess abbilden. Im Rahmen

[9] Aus Platzgründen zeigen wir nur einen Bildausschnitt und nicht die gesamte Oberfläche des Werkzeugs.

der Fallstudie haben wir uns für letztgenannte Vorgehensweise entschieden, da die Abläufe der Prozesse ähnlich zueinander sind. Wir stellen den Basisprozess daher als Schnittmenge der beiden Standardprozesse für Patienten- und Laboruntersuchung dar. Er besteht aus fünf Aktivitäten.[10]

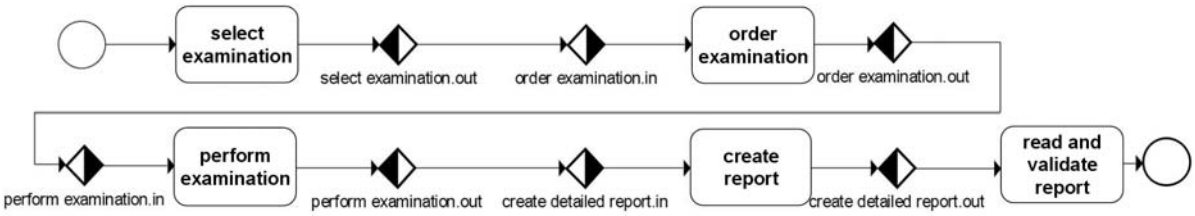


Abbildung 11.19: Generischer Untersuchungsprozess als Basisprozess

Kontextmodell des Untersuchungsprozesses. Basierend auf den identifizierten Prozessvarianten bzw. den Rahmenbedingungen in denen sie erforderlich werden, können wir das Kontextmodell des Untersuchungsprozesses beschreiben (vgl. Tabelle 11.3). Wir fassen dazu die Kontextvariablen beider Typen von Prozessvarianten (d.h. Patienten- und Laboruntersuchungen) zusammen. Die Zugehörigkeit zu einem Typ spezifizieren wir in der Kontextvariable *examination*.

Tabelle 11.3: Ausschnitt aus dem Kontextmodell des Untersuchungsprozesses

Variablenname	Wertebereich	Modus
examination	patient examination, laboratory test	static
sample	none, single sample, multiple samples	static
complications risk	high, medium, low	static
emergency	yes, no	static
ambulatory	yes, no	static
appointment	appointment, registration, none	static

Bestimmte Wertekombinationen von Kontextvariablen aus dem Kontextmodell des Untersuchungsprozesses sind in der Praxis nicht relevant. Das heißt für diese muss keine korrekte Prozessvariante definiert werden. Um die entsprechenden Kombinationen auszuschließen, beschreiben wir folgende Kontextregeln:

Regel 1:
WENN examination = „laboratory test“
DANN appointment = „none“

Regel 2:
WENN examination = „patient examination“ DANN sample = „none“

Regel 3:
WENN emergency = „yes“ DANN sample <> „multiple samples“

Optionen. Wie erläutert, haben wir einen generischen Basisprozess zur Ableitung der Prozessvarianten festgelegt. Dieser muss zur Ableitung der spezifischen Prozessvarianten durch Optionen angepasst werden. Abbildung 11.16 gibt dazu einen Überblick zur Hierarchie der Prozessvarianten und die zur Abbildung dieser Prozessvarianten notwendigen Optionen. Aus dieser Darstellung resultiert eine hierarchische Strukturierung der Optionen, welche wir mit Hilfe explizit definierter Hierarchiebeziehungen beschreiben. Die Kontextbedingungen

[10] Aus Gründen der besseren Nachvollziehbarkeit haben wir die Anzahl der erforderlichen Aufsetzpunkte im Basisprozess und in den Optionen nicht auf ein Minimum reduziert.

der Optionen sind ebenfalls gut aus dieser Hierarchie ableitbar. Abbildung 11.20 zeigt die modellierten Optionen.

Option L0 wird für alle Varianten einer Patientenuntersuchungen angewendet. Sie besteht aus drei Änderungsoperationen: Operation 1 und 2 fügen zunächst zwei Prozessfragmente in den Basisprozess ein, die spezifische Aktivitäten für eine Patientenuntersuchung umfassen (z.B. `prepare patient` und `inform patient about procedure`). Des Weiteren ändert Operation 3 die Befundart entsprechend ab.

Insgesamt sind zur Abbildung aller Prozessvarianten 7 Optionen erforderlich. Diese bestehen jeweils aus maximal drei Änderungsoperationen. Es werden alle von Provop angebotenen Änderungsoperationstypen verwendet.

Ableitung der Ergebnismodelle. Durch Modellierung des Basisprozesses und den zugehörigen Optionen, können wir die Ergebnismodelle der 18 Prozessvarianten ableiten (d.h. 2 Varianten der Laboruntersuchung und 16 Varianten der Patientenuntersuchung – der Basisprozess stellt keine gültige Prozessvariante dar). Die Kombinierbarkeit der Optionen erlaubt nicht nur die Abbildung der oben beschriebenen Varianten, sondern auch Kombinationen davon (z.B. Variante für gehfähige Patienten und keine Terminvereinbarung). Aus Platzgründen verzichten wir an dieser Stelle auf die Darstellung aller ableitbaren Prozessvarianten und zeigen in Abbildung 11.21 exemplarisch das Ergebnismodell einer Patientenuntersuchung eines gehunfähigen Patienten der einen Notfall darstellt. Ein zweites Beispiel zeigt Abbildung 11.22 für eine Patientenuntersuchung ohne Terminvereinbarung.

11.6.3 Gewonnene Erkenntnisse

Der Untersuchungsprozess in einer Klinik stellt bereits einen relativ komplexen Prozess mit vielen Abweichungen dar. Diese Abweichungen entstehen u.a. aufgrund der historisch gewachsenen Vorgehensweisen verschiedener Stationen oder den spezifischen Patientendaten (z.B. Gehfähigkeit). In Provop sind diese varianten Prozessabläufe sehr gut mit Hilfe eines Basisprozesses und seinen zugehörigen Optionen abbildbar. Die hohen Pflege- und Wartungsaufwände der ehemals ausmodellierten Prozessvarianten können deutlich reduziert werden. Die Anzahl der zu modellierenden Prozesselemente wird auf ein Minimum reduziert. Dabei bleibt die Verwendung von Optionen, die jeweils 1-3 Änderungsoperationen umfassen, durch deren hierarchische Strukturierung übersichtlich. Eine Erweiterung der Menge unterstützter Ablaufvarianten des Patienten- oder Laboruntersuchungsprozesses kann durch weitere Optionen flexibel erfolgen.

Da die Rahmenbedingungen der einzelnen Prozessvarianten bekannt sind, können wir das spezifische Kontextmodell des Untersuchungsprozesses leicht erfassen. Darüber hinaus sind wir in der Lage für die modellierten Optionen zu beschreiben, in welchem konkreten Kontext sie gültig werden. Somit ist eine automatische Konfiguration der Varianten des Untersuchungsprozesses möglich.

Die Fallstudie hat auch aufgezeigt, an welchen Stellen der Provop-Ansatz erweitert werden kann, um spezifischere Anwendungsfälle abzudecken. So beschreiben wir in unserer Fallstudie, mit Hilfe der Hierarchiebeziehung zwischen Optionen, verschiedene Gruppen von Prozessvarianten. Diese Gruppen bzw. „Kontextvariablenbäume“ sind im Provop-Kontextmodell nicht direkt abbildbar. Stattdessen müssen wir zahlreiche Kontextregeln beschreiben, welche die möglichen Kombinationen von Variablenwerten einschränken. Dies wäre über eine hierarchische Struktur des Kontextmodells ggf. überflüssig.

Ein weiterer Aspekt betrifft die transparente Abbildung variantenbehafteter Daten. Die Datenvarianz der Untersuchungsprozesse resultiert aus verschiedenen Rahmenbedingungen. So existieren z.B. dutzende Untersuchungsarten (z.B. Röntgenuntersuchung, Blutwertanlayse), welche jeweils ein anderes Procedere erfordern, mit unterschiedlichen Formularen und unterschiedlichen geforderten bzw. produzierten Daten. Dies ergibt sich u.a. daraus, dass bei manchen Untersuchungsstellen die Befunde diktiert werden, während andere eine EDV-gestützte (strukturierte) Aufnahme der Ergebnisse verfolgen [KZR+94]. Außerdem unterscheiden nicht alle Bereiche zwischen einem Kurzbefund und einem ausführlichen Befund und die Daten liegen nicht immer in einer strukturierten Form vor. Nicht nur für den Befund existiert eine solche Vielzahl von Datenvarianten. Weitere Beispiele sind die untersuchungsspezifischen Formulare zur Patientenaufklärung und die Datenformate bei der EDV-gestützten Terminvereinbarung. In Provop sind variante Formulare und Daten bisher nur in Form von verschiedenen Attributwerten und entsprechenden `MODFIY`-Operationen abbildbar. Diese Lösung ist im Szenario des Untersuchungsprozesses bzw. allgemein bei einer großer Datenvarianz noch nicht ausreichend, da die Transparenz über die Daten verloren geht. Eine wichtige Erkenntnis aus dieser Fallstudie ist daher, dass der Provop-Ansatz um eine Betrachtung varianter Daten bzw. Datenflüsse erweitert werden muss, um zukünftig auch die Herausforderungen großer Datenvarianzen zu bewältigen.

11.7 Diskussion und Zusammenfassung

Im Rahmen dieser Arbeit haben wir in verschiedensten Domänen insgesamt vier Fallstudien durchgeführt. Diese zeigen, dass der Provop-Ansatz eine ausreichende Mächtigkeit bietet, um den heutigen Herausforderungen des Managements von Prozessvarianten zu begegnen. In allen Fallstudien konnte der Basisprozess und die zugehörigen Optionen leicht identifiziert werden. Darüber hinaus hat sich gezeigt, dass der Bezug zwischen Optionen und dem spezifischen Kontext einer Prozessvariante leicht hergestellt werden kann und den Variantenverantwortlichen bei der Konfiguration von Prozessvarianten unterstützt. Dabei stellen die expliziten Optionsbeziehungen nicht nur ein gutes Mittel zur Abbildung struktureller Abhängigkeiten dar, sondern sie erlauben auch die hierarchische Strukturierung der modellierten Optionen.

Die Fallstudien haben allerdings auch gezeigt, um welche Aspekte der Provop-Ansatz noch erweitert werden sollte. Ein Aspekt ist hier die hierarchische Strukturierung der Variablen eines Kontextmodells. Ein anderer Aspekt betrifft die Unterstützung varianter Datenstrukturen für den Austausch von Daten zwischen Aktivitäten. Hier ergeben sich in der Praxis noch viele offene Punkte, die bisher nicht ausreichend betrachtet wurden. Dabei stellt die Änderbarkeit von Attributen in Provop bereits einen ersten Lösungsansatz dar.

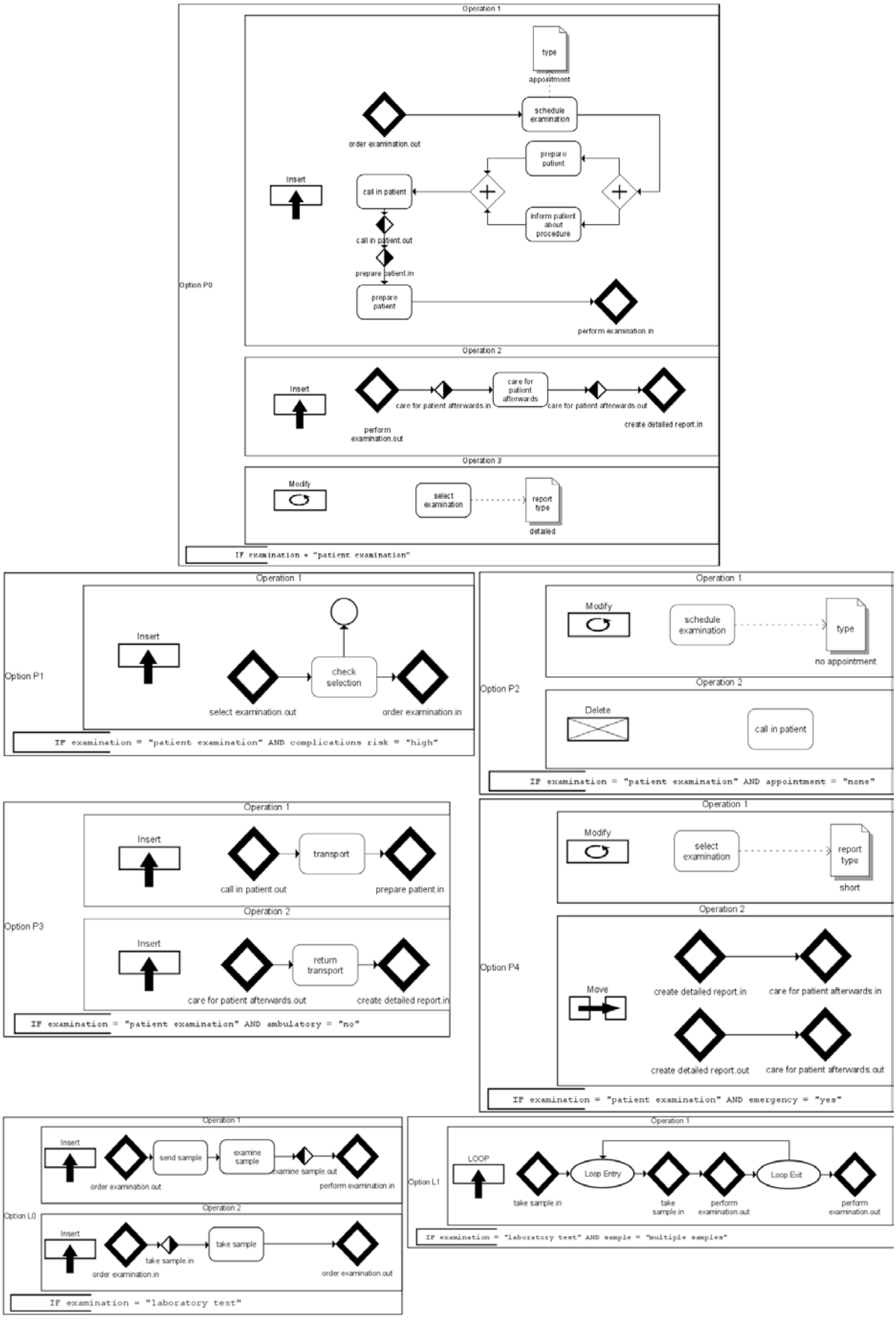


Abbildung 11.20: Option zur Abbildung der Varianten des Untersuchungsprozesses

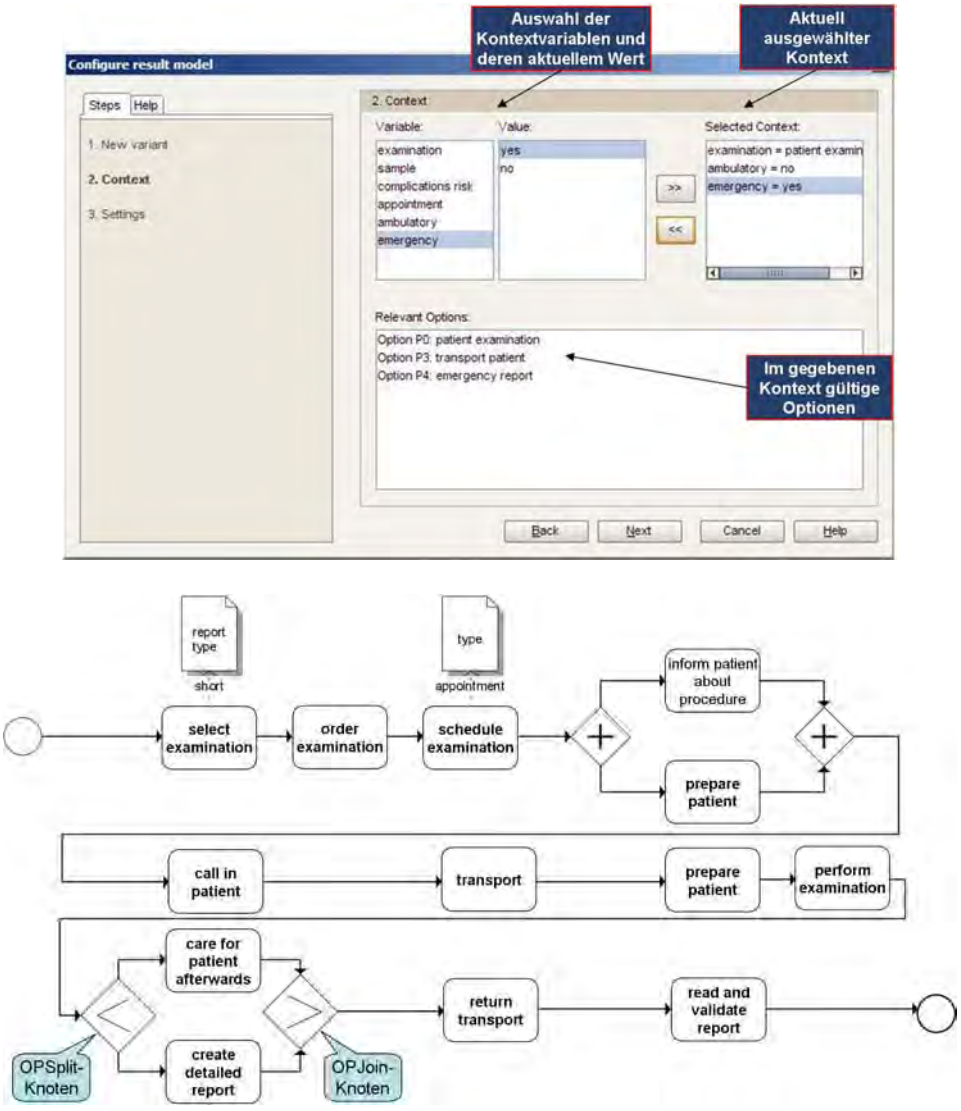


Abbildung 11.21: Ergebnismodell nach Anwendung der Optionen P0, P3 und P4

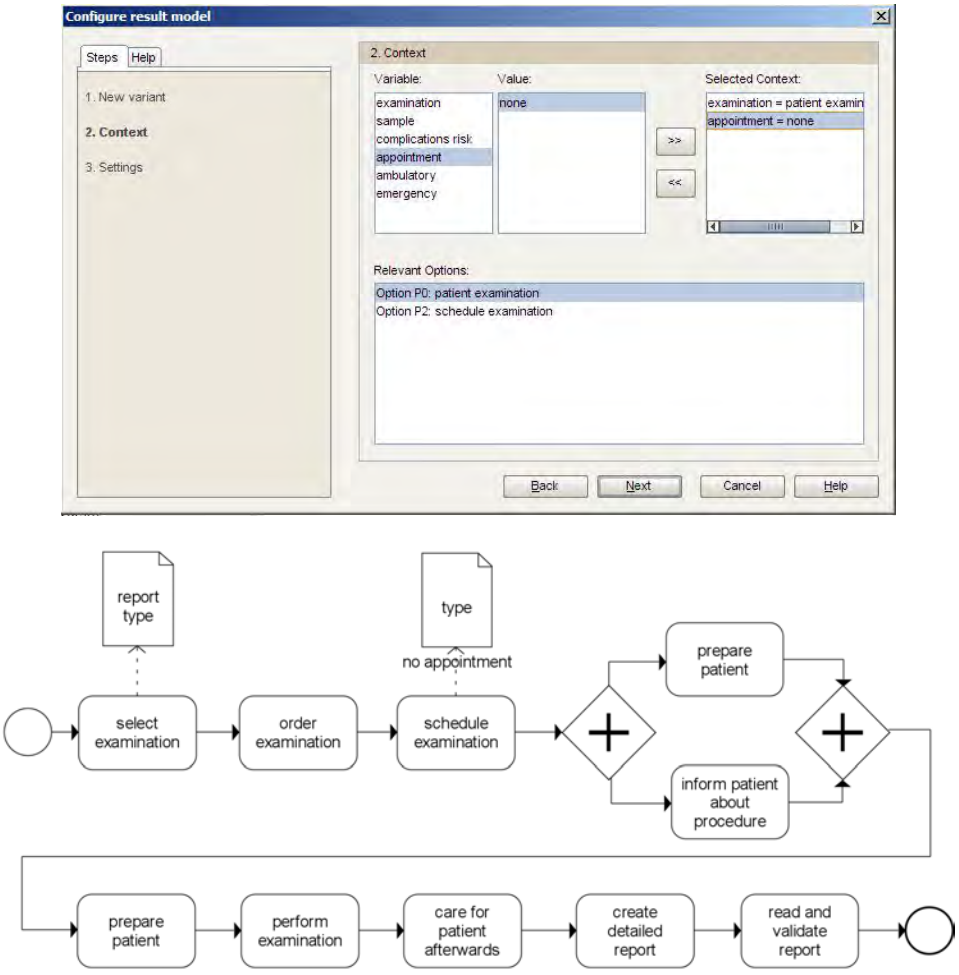


Abbildung 11.22: Ergebnismodell nach Anwendung der Optionen P0 und P2

Teil IV:
Diskussion und Zusammenfassung

12

Weitere verwandte Arbeiten

In den vorangehenden Kapiteln haben wir den Provop-Ansatz im Detail vorgestellt. In diesem Zusammenhang haben wir den aktuellen Stand der Technik zu den jeweils behandelten Einzelthemen am Ende der einzelnen Kapitel umfassend diskutiert. Abbildung 12.1 gibt nochmals einen Überblick zu wissenschaftlichen Ansätzen entlang der Phasen des Prozesslebenszyklus. Kapitel 12 erläutert die wichtigsten, der in Abbildung 12.1 hervorgehoben Arbeiten gesamtheitlich und grenzt sie gegenüber Provop ab.

Tabelle 12.1: Überblick der wichtigsten verwandten Ansätze

	Prozess-Modellierung	Konfiguration von Varianten	Prozess-Ausführung	Prozess-Evolution
Prozess-Management	▪ ***Referenzprozess-modellierung*** ▪ Varianten Repository mit Such-Strategien ▪ Sichtenkonzepte für Ein-Modell-Ansätze ▪ Prozessaggregation ▪ Workflow, Data und Change Patterns	▪ ***Konfigurierbare Referenzprozessmodelle (cEPC, cYAWL)*** ▪ Prozesskonfiguration durch Spezialisierung ▪ Kontexte zur block-orientierten Prozess-Individualisierung ▪ Algorithmen zur Verifikation von Prozessmodellen	▪ ***Dynamische Änderung der Prozessinstanz (ADEPT u. ProCycle)*** ▪ Sukzessive Instanziierung von Subworkflows (z.B. Worklets) ▪ Deklarative Prozesse (z.B. DECLARE, AGENTWork)	▪ ***Mining von Referenzprozessen aus Prozessvarianten (MinAdept)*** ▪ ***Schemaevolution (ADEPT)*** ▪ Refaktorierung von Prozessmodellen
Weitere Ansätze	▪ UML Variabilitäts-Mechanismen ▪ Versionierung von Softwareprodukten	▪ Softwarekonfiguration (Feature-Diagramme, AND/OR Graphen) ▪ Produktkonfiguration / Stücklisten		
Kapitel	*Kapitel 5*	*Kapitel 6*	*Kapitel 7*	*Kapitel 8*

Das Kapitel gliedert sich wie folgt: Die Abschnitte 12.1 und 12.2 stellen die beiden wichtigsten Forschungsprojekte vor, die im Kontext von Provop relevant sind. Anschließend fasst Abschnitt 12.3 das Kapitel zusammen.

12.1 Modellierung und Konfiguration von Referenzprozessmodellen

Die relevantesten Arbeiten, in Bezug auf den Provop-Lösungsansatz stammen aus dem Forschungsgebiet der konfigurierbaren Referenzprozessmodelle [RLS+07, RvdA07, GvdAJVlR07, GvdAJV08, Ros09]. Die Forschungsaktivitäten der *Business Process Management Research Group* der Queensland University of Technology (Australien) werden in enger Kollaboration mit der *Business Process Management Research Group* der Technischen Universität Eindhoven (Niederlande) durchgeführt.

Zwischen den Arbeiten dieser Forschungsgruppen und Provop bestehen an mehreren Stellen Berührungspunkte. Der Ansatz der konfigurierbaren Referenzprozessmodelle erlaubt nicht nur die Modellierung von Prozessvarianten in einem Prozessmodell, sondern auch deren Konfiguration, basierend auf Kontextinformationen. Die Ansätze wurden zunächst, basierend auf *configurable Event driven Process Chains (cEPC)*, entwickelt [RvdA07]. Dazu wird die Standard-EPK-Notation um eine Konfigurationslogik in Form von Labels erweitert, welche mit den konfigurierbaren Prozesselementen verknüpft werden (vgl. Abbildung 12.1). In einem Label ist beschrieben, welche spezifischen Anforderungen und Richtlinien für die Konfiguration gelten. Dabei können auch Abhängigkeiten zwischen mehreren konfigurierbaren Knoten definiert werden. Ein konfigurierbarer Knoten kann dann im Variantenmodell verbleiben, optional sein oder ausgelassen werden.

Dieser Ansatz wurde zur Ausführung von konfigurierten Prozessmodellen auf die *Yet Another Workflow Language (YAWL)* übertragen, indem YAWL zu *configurable YAWL (cYAWL)* erweitert wurde [GvdAJVlR07, Ros09]. Zur Konfiguration des Referenzprozesses beschreiben die Autoren in [GvdAJVlR07, Ros09] das *(optional) Blocking* und das *(optional) Hiding* konfigurierbarer Knoten bzw. Ein-/ Ausgangsports.[1] Blocking erlaubt es, ganze Ausführungspfade im konfigurierten Proessmodell auszulassen. Wird eine Aktivität blockiert, so werden die nachfolgenden Aktivitäten des selben Ausführungspfades ebenfalls blockiert. Auf diese Weise können semantische und strukturelle Abhängigkeiten zwischen einer blockierten Aktivität und ihren Nachfolgern berücksichtigt werden. Das Hiding von Aktivitäten erlaubt es im Gegensatz dazu, eine einzelne Aktivität auszulassen, ohne dass davon nachfolgende Aktivitäten beeinflusst werden.

In [vdADG+08] wird beschrieben, wie die Korrektheit von konfigurierbaren Prozessvarianten gewährleistet werden kann. Dabei gehen die Autoren von einem korrekten Referenzprozessmodell aus. Nach jedem Konfigurationsschritt resultiert wieder ein korrektes Prozessmodell. Das heißt, die möglichen Anpassungen (Blocking, Hiding) ändern nicht die Korrektheitseigenschaft eines Prozessmodells. Auf diese Weise wird für alle aus dem gegebenen Referenzprozessmodell konfigurierbaren Prozessmodelle Korrektheit gewährleistet.

Zur Konfiguration von cEPCs und cYAWL-Modellen wurde eine Fragebogenmethode entwickelt, welche die komfortable Ableitung einer Prozessvariante aus dem Referenzprozessmodell erlaubt [RLS+07, RGDvdA07, Ros09]. Ein Fragebogen umfasst dabei eine Menge von geschlossenen Fragen[2], die in natürlicher Sprache verfasst sind. Jede Antwortmöglichkeit entspricht dabei einem sog. *Domain Fact*. Alle Domain Facts beschreiben dann im Prinzip den konkreten Kontext einer zu konfigurierenden Prozessvariante. Mit Hilfe einer Mapping-Tabelle zur Abbildung von Domain Facts auf konfigurierbare Knoten des Referenzprozessmodells wird die kontextabhängige Konfiguration umgesetzt (vgl. Abbildung 12.2). Die Domain

[1] Ein- bzw. Ausgangsports entsprechen den Ein- bzw. Ausgängen von Knoten, an denen Kontrollflusskanten ein- bzw. ausgehen.

[2] Das heißt alle möglichen Antworten sind vorgegeben. Eine freitextliche Antwort, wie bei sog. offenen Fragen, ist nicht möglich.

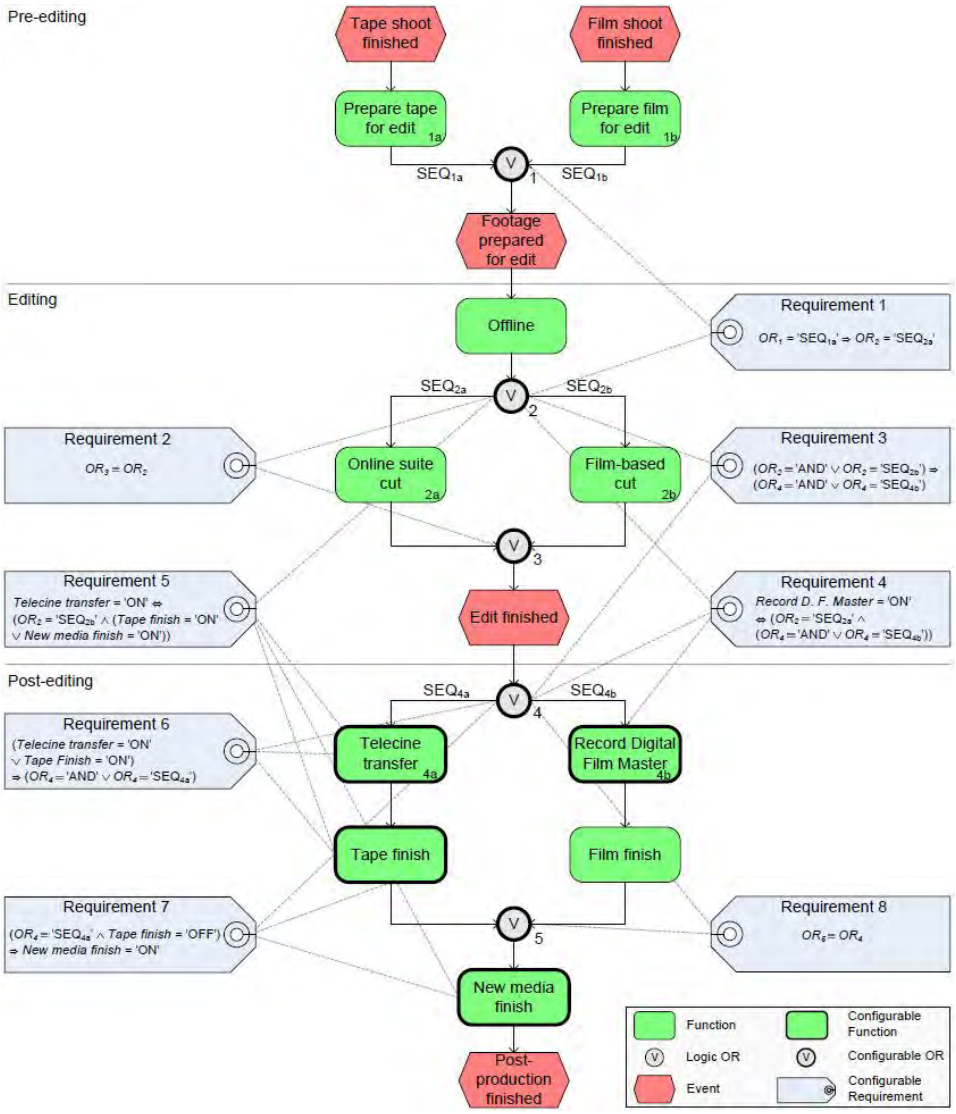


Abbildung 12.1: Referenzprozess der Filmnachbearbeitung nach [RLS+07]

Facts werden bei der Konfiguration strukturiert erhoben, d.h. es wird eine Reihenfolge für die Fragen vorgegeben und abhängig von vorangehenden Antworten werden bestimmte Fragen ggf. ausgelassen. Diese Methode wurde sowohl für cEPCs als auch für cYAWL-Modelle implementiert. Die Ergebnisse der Forschungsaktivitäten sind im Werkzeug *Synergia* [RG09] realisiert und wurden in verschiedenen Fallstudien validiert [RtHRS08, GWJV+09].

Zwischenfazit

Generell stellt die Referenzprozessmodellierung eine Optimierung des konventionellen Ein-Modell-Ansatzes zur Modellierung und Konfiguration von Prozessvarianten dar (vgl. Abschnitt 4.2.2). Allerdings bleiben die Varianten auch bei diesem verbesserten Ansatz in der Ablauflogik des Prozessmodells versteckt und sind somit für den Variantenmodellierer nicht wirklich transparent. Des Weiteren bietet dieser Ansatz, im Gegensatz zu Provop, keine Möglichkeit zum Erweitern des generischen Referenzprozesses zur Abbildung einer konkreten Prozessvariante an. Auch das Verschieben von Prozesselementen ist im Prinzip nur durch redundante Modellierung von Prozesselementen an verschiedenen Positionen und das spätere Auslassen überflüssiger Prozesselemente möglich. Unsere Fallstudie in der Domäne Stadtver-

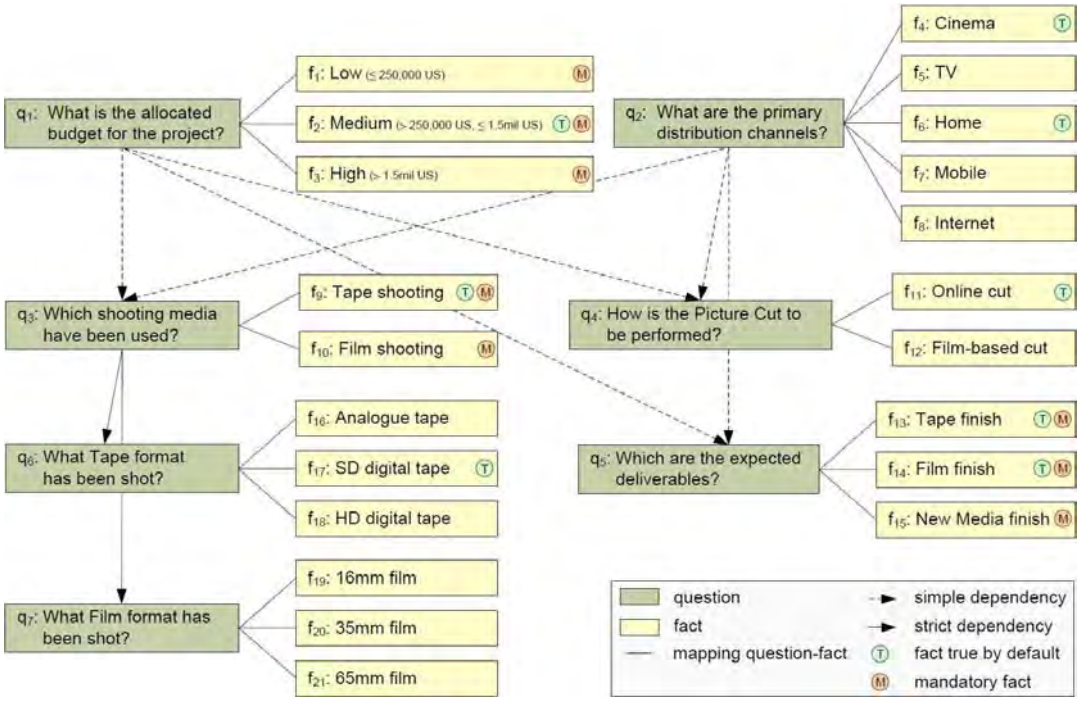


Abbildung 12.2: Domain Facts der Filmnachbearbeitung nach [RLS+07]

waltungen (vgl. Abschnitt 11.5) – die auch in Synergia abgebildet wird [GWJV+09, RG09] – hat gezeigt, dass die Provop-`MOVE`-Operation die Komplexität eines Referenz- bzw. Basisprozesses reduziert.

Im Vergleich zur Referenzprozessmodellierung, setzt Provop keinen korrekten Basisprozess voraus, da dies für manche Anwendungsszenarien zu restriktiv ist. Entsprechend führt auch die Anwendung von Änderungsoperationen in Provop nicht unbedingt zu einem korrekten Ergebnismodell. Die Korrektheit der Prozessvarianten einer Prozessfamilie kann allerdings mit Hilfe der Provop-Korrektheitsprüfung gewährleistet werden.

Die Fragebogenmethode erlaubt eine Abstraktion von der eigentlichen Konfiguration von Prozessvarianten und ähnelt dem kontextbasierten Ansatz von Provop. So kann der gegebene Kontext in Provop prinzipiell auch mittels Fragebogen erhoben werden.

Der beschriebene Ansatz der konfigurierbaren Referenzprozessmodelle deckt die ersten beiden Phasen des Prozesslebenszyklus ab. Aspekte der flexiblen Ausführung von Prozessvarianten werden bisher zwar durch optionales Blocking und Hiding angedacht [GvdAJVlR07], aber nicht weiter vertieft. Somit stellt dieser Ansatz keine durchgängige Lösung dar, wie es bei Provop der Fall ist.

12.2 Fallbasierte Adaption zur flexiblen Ausführung von Prozessen

Verwandte Arbeiten zur Ausführung und Evolution von Prozessvarianten entstammen auch dem *ADEPT* Forschungsprojekt, welches am Institut für Datenbanken und Informationssysteme an der Universität Ulm betrieben wird [DR09, Rei09]. Ziel dieses Forschungsprojektes ist die Entwicklung eines flexiblen Prozessmanagement-Systems, das die dynamische Änderung von Prozessinstanzen auch während der Laufzeit erlaubt [RRKD05, RKL+06, DR09]. Dabei können autorisierte Endanwender durch höherwertige Änderungsoperationen, etwa

das Hinzufügen, Löschen oder Verschieben von Aktivitäten, die Prozessinstanz dynamisch adaptieren (vgl. [WRR08, WSR09]).

ADEPT verfolgt das Prinzip der „Correctness by Construction“. Das heißt, bereits bei der Modellierung der Prozesse wird deren syntaktische Korrektheit (inklusive Ausschluss von Deadlocks oder fehlerhaften Datenflüssen) sichergestellt. Zusätzlich wird im Rahmen des Projekts *SeaFlows* ein Rahmenwerk entwickelt, das die Beschreibung von semantischen Beschränkungen für Prozessmodelle erlaubt und deren Einhaltung sowohl bei der Modellierung als auch bei der dynamischen Änderung von Prozessinstanzen prüft [LRD06, LRD08b, LRMGD09].

Eine Weiterentwicklung von ADEPT stellt *ProCycle* dar [WWRD07, WRWRM09]. Die Autoren folgen hier dem Konzept des *Case-based Reasoning* (dt.: fallbasiertes Schließen). Dabei werden neben dynamischen Anpassungen einer Prozessinstanz auch die Gründe für diese Anpassungen in Form von Frage-Antwort-Paaren gespeichert. Die dokumentierten Fälle werden in einer „Fallbasis“ verwaltet. Diese dient einerseits zur Detektion von Prozessverbesserungen, andererseits zur Unterstützung von Endanwendern bei Prozessanpassungen in ähnlichen Situationen. Im letzteren Anwendungsfall heißt dies, dass zur Laufzeit bei Auftreten von Ausnahmen in der Fallbasis nach ähnlichen Fällen gesucht wird und die dokumentierten Anpassungen direkt übernommen werden können. Das heißt, das Wissen über Kontext und erforderliche Anpassungen wird hier in einer wiederverwendbaren Form gespeichert.

Neben der spontanen bzw. fallbasierten Anpassung von Prozessinstanzen, stellt die Evolution des Prozesschemas auf Typebene eine weitere Grundfunktion dar. So kann der Prozessverantwortliche in ADEPT das Prozessschema eines Prozesstyps anpassen und anschließend auf laufende Prozessinstanzen propagieren. Dabei werden Korrektheitsaspekte berücksichtigt, z.B. werden nur solche Änderungen propagiert, die mit dem aktuellen Ausführungszustand einer Prozessinstanz verträglich sind [RRD03, RRD04a, RRD04c, RRD04b]. Erweiterte Korrektheitsbetrachtungen finden sich in [RMRW08b].

In weiteren Forschungsarbeiten im Rahmen des ADEPT-Projektes wird untersucht, wie Informationen aus Änderungshistorien, in welchen alle Änderungen zur Laufzeit protokolliert werden, für eine geeignete Prozessevolution genutzt werden können [RRJK06, GRRvdA06, GRMR+08]. Durch Auswertung von Änderungshistorien können Anpassungs- und Optimierungsempfehlungen für Prozessmodelle abgleitet werden [LRWc08, RJR07]. Im *MinAdept* Projekt, welches ebenfalls als Weiterentwicklung bzw. Vertiefung des ADEPT Projektes verstanden werden kann, wird z.B. betrachtet, wie die durch Änderungshistorien gewonnenen Informationen über tatsächlich ausgeführte Prozessvarianten genutzt werden können, um einen optimalen Referenzprozess zu erzeugen. Optimal bedeutet hier, dass zwischen Referenzprozess und allen Varianten eine minimale durchschnittliche Modelldifferenz (hier gemessen durch höherwertige Änderungsoperationen) besteht [LRWb08, LRWd08], so dass der durchschnittliche Konfigurationsaufwand bei der Ableitung einer konkreten Variante minimiert wird.

Die Kernkonzepte des ADEPT-Projektes, d.h. Adaption von Prozessinstanzen und Schemaevolution, wurden in einem Prozessmanagement-System, der AristaFlow BPM-Suite [DRRM+09], implementiert.

Zwischenfazit:

Die Phasen der Modellierung und Konfiguration von Prozessvarianten werden in ADEPT über konventionelle Methoden abgedeckt. Das heißt, entweder müssen alle Prozessvarianten in separaten Modellen erstellt werden oder sie werden als Ablaufvarianten in einem Prozessmodell integriert. Allerdings reduziert die Möglichkeit der dynamischen Adaption von

Prozessinstanzen zur Laufzeit die Notwendigkeit der „Ausmodellierung“ aller möglichen Varianten. Das heißt, diese können bei Bedarf auch erst zur Laufzeit hergestellt werden. Im Gegensatz zu Provop finden solche Laufzeitkonfigurationen nicht automatisch statt, sondern erfordern das Eingreifen der Benutzer.

Berührungspunkte zwischen ADEPT und Provop ergeben sich hinsichtlich der Evolution von Prozessvarianten. So unterstützt die fallbasierte Anpassung von Prozessvarianten in ProCycle die dynamische Konfiguration von Prozessvarianten auf Basis höherwertiger Änderungsoperationen. Prinzipiell ist dieser Ansatz auch zu einer statischen Konfiguration von Prozessvarianten geeignet.

12.3 Zusammenfassung

Die Analyse verwandter Arbeiten in diesem und den vorangehenden Kapiteln hat gezeigt, dass das Management von Prozessvarianten in der wissenschaftlichen Literatur bisher nicht in ausreichendem Maße betrachtet wird. Es existieren nur wenige Ansätze aus dem Prozessmanagement, die sich überhaupt mit Prozessvarianten und ihren Facetten in Theorie und Praxis beschäftigen. Eine durchgängige Lösung über alle Phasen des Prozesslebenszyklus hinweg, wie sie in Provop realisiert wird, findet sich in keinem Ansatz. Allerdings existieren für viele Einzelaspekte von Provop, etwa der Korrektheit von Prozessvarianten oder deren adäquaten Darstellung, interessante Arbeiten, wenn auch mit anderem Fokus. Bezogen auf den Gesamtansatz haben diese Arbeiten nur wenige Berührungspunkte mit Provop.

13

Zusammenfassung und Ausblick

Das Management von Prozessen spielt in Unternehmen eine zunehmend wichtigere Rolle. Die Effizienz und Qualität der Geschäftsprozesse sowie ihre kontinuierliche Weiterentwicklung und Optimierung ist deshalb entscheidend für den Unternehmenserfolg. In diesem Zusammenhang ist die effektive Handhabung von Prozessvarianten unabdingbar, um die Variabilität der Unternehmensabläufe adäquat abbilden zu können. Existierende Prozessmodellierungswerkzeuge bieten allerdings keine ausreichende Unterstützung für diese Aufgabe. Sie erlauben das Management von Prozessvarianten ausschließlich über deren Ausmodellierung in separaten Prozessmodellen oder deren Integration in einem Prozessmodell, unter Verwendung alternativer Ausführungspfade. Diese rudimentäre Unterstützung führt zu verschiedenen Problemstellungen: Neben den hohen Aufwänden für die Modellierung und Wartung der Prozessmodelle, bleibt die eigentliche Varianz der Prozesse intransparent, d.h. sie wird in der Ablauflogik oder in separaten Prozessmodellen versteckt. Darüber hinaus können die Varianten in keiner Relation zu ihrem spezifischen Anwendungsfall gebracht werden. Mit anderen Worten, der Kontext einer Prozessvariante wird nicht berücksichtigt.

13.1 Zusammenfassung

Die vorliegende Arbeit liefert ein durchgängiges Lösungskonzept für das Management von Prozessvarianten. Die wichtigsten Beiträge unseres Provop-Ansatzes lassen sich wie folgt zusammenfassen:

- **Operationale Modellierung von Prozessvarianten:** Provop ermöglicht ein änderungsbasiertes Variantenmanagement. Zu diesem Zweck werden explizite Änderungsoperationen verwendet, um zu beschreiben, welche Abweichungen eine Prozessvariante zum gegebenen Basisprozess aufweist. Dieser Ansatz ist angelehnt an die typische Vorgehensweise bei der Modellierung neuer Prozessvarianten, die meist durch Kopieren und Anpassen eines existierenden Prozessmodells entstehen.
- **Transparente Modellierung von Prozessvarianten:** Ein wichtiges Ziel bei der Entwicklung des Provop-Ansatzes ist das Erreichen transparenter Prozessvarianten. Transparenz wird in unserem Ansatz durch explizite Änderungsoperationen bzw. deren Gruppierung

zu Optionen erreicht. Dadurch ist für den Variantemodellierer bzw. -verantwortlichen transparent, welche Änderungen am Basisprozess vorgenommen werden müssen, um konkrete Prozessvarianten abzuleiten. Darüber hinaus erlauben es Aufsetzpunkte eines Basisprozesses, hervorzuheben, an welchen Stellen Anpassungen vorgenommen werden können.

- **Konfiguration von Prozessvarianten in Abhängigkeit von den gegebenen Rahmenbedingungen (d.h. Prozesskontext):** Provop unterstützt die Konfiguration von Prozessvarianten, abhängig von ihrem Anwendungskontext. Dazu legen wir für jeden Prozesstyp ein intuitives domänenspezifisches Kontextmodell fest. Auf Basis dieses Kontextmodells können wir definieren, welche Optionen bzw. Änderungsoperationen in welchem spezifischen Kontext angewendet werden sollen. Dies erleichtert die Konfiguration der Prozessvarianten und erlaubt darüber hinaus die Abstraktion von den eigentlichen Änderungen. Das heißt, der Variantenverantwortliche kann, ohne Detailwissen über die Änderungsoperationen, eine Prozessvariante konfigurieren.

- **Flexible Ausführung der Prozessvarianten in einem Workflow-Management-System:** Der für einen Prozesstyp angegebene Kontext kann sich üblicherweise während der Ausführung eines Variantenmodells ändern. Provop berücksichtigt dies, durch die Beschreibung dynamischer Kontextvariablen bzw. der vorgeplanten Anwendung von Optionen. Dabei können bereits zum Zeitpunkt der Konfiguration alle zur Laufzeit relevanten Prozessvarianten gemeinsam in einem Prozessmodell repräsentiert werden. Zur Laufzeit kann flexibel zwischen verschiedenen (korrekten) Prozessvarianten gewechselt werden.

- **Gewährleistung der Korrektheit:** Provop gewährleistet, dass die konfigurierbaren Varianten einer Prozessfamilie strukturell und semantisch korrekt sind. Dazu bestimmen wir zunächst mit Hilfe der Kontextabhängigkeit der Optionen und den definierten Optionsbeziehungen, welche konfigurierbaren Prozessvarianten semantisch sinnvoll sind. Anschließend prüfen wir nur für diese die Einhaltung der spezifischen Korrektheitseigenschaften ihres zugrunde liegenden Prozess-Metamodells (z.B. Deadlockfreiheit, Zyklenfreit). Des Weiteren stellen wir zur Laufzeit sicher, dass nur korrekte Prozessvarianten ausgeführt werden. Damit stellt Provop die stabile Ausführung der Prozessvarianten sicher.

- **Unterstützung von Prozessvarianten über den gesamten Prozesslebenszyklus hinweg:** Provop betrachtet den gesamten Prozesslebenszyklus, begonnen bei der Modellierung von Prozessvarianten, über deren korrekte Konfiguration sowie Ausführung und schließlich Evolution. Diese Durchgängigkeit ist ein wichtiger Mehrwert von Provop gegenüber konventionellen Ansätzen des Variantenmanagements.

- **Generischer Ansatz, unabhängig von Fachprozessen und Prozess-Metamodellen:** Prozessvarianten sind kein Phänomene, die auf bestimmte Prozesstypen oder Disziplinen begrenzt sind. Vielmehr existiert die Problemstellung einer adäquaten Abbildung von Varianten für die meisten Prozesstypen. Der Provop-Lösungsansatz wurde deshalb mit dem Ziel entwickelt, generisch auf verschiedene Prozesstypen und in beliebigen Domänen anwendbar zu sein. Aus diesem Grund haben wir uns in Provop nicht auf ein Prozess-Metamodell festgelegt, d.h. wir haben keine spezialisierte Lösung für ein bestimmtes Prozess-Metamodell realisiert.

13.2 Ausblick

Das Management von Prozessvarianten mittels Provop bietet noch weitere Herausforderungen, die in dieser Arbeit aufgedeckt werden, deren Bearbeitung aber den Rahmen dieser Arbeit gesprengt hätte.

In unseren Fallstudien haben wir gelernt, dass in bestimmten Fällen eine hierarchische Abbildung von Kontextvariablen wünschenswert ist. In zukünftigen Arbeiten könnte daher geprüft werden, inwieweit diese Hierarchien im Kontextmodell abgebildet werden können und welche Aspekte sich daraus für seine Verwendung ergeben. Eine weitere Erkenntnis ist, dass der Aspekt variantenbehafteter Datenflüsse eines Prozessmodells in Provop bisher nicht ausreichend berücksichtigt wird. Es sollte daher geprüft werden, um welche Änderungsoperationen Provop erweitert werden muss, damit auch eine hohe Datenvarianz adäquat gehandhabt werden kann.

Heute werden Prozessvarianten in Unternehmen nach dem Ein- oder Mehr-Modell-Ansatz abgebildet. Um diese Prozessvarianten auf den Provop-Ansatz zu migrieren, ist ein Methodikleitfaden erforderlich. Dieser sollte beschreiben, wie ausgehend von der durch einen Ein- oder Mehr-Modell-Ansatz geprägten Prozesslandschaft, ein Basisprozess mit zugehörigen Optionen identifiziert werden kann. Darüber hinaus sollte dieser Methodikleitfaden beschreiben, wie ein Domänen-spezifisches Kontextmodell erhoben werden kann.

Eine interessante Fragestellung ist, inwieweit eine Übertragung des Provop-Ansatzes für Prozessmodelle auf weitere Modell- oder Diagrammtypen möglich bzw. sinnvoll ist. Es ist z.B. vorstellbar, ein Datenmodell, abhängig von einem spezifischen Anwendungskontext, zu konfigurieren. Solche Anwendungsmöglichkeiten erfordern ggf. zusätzliche Änderungsoperationen (z.B. Aggregation, Spezialisierung), um die Provop entsprechend erweitert werden müsste.

Zur Validation des Provop-Ansatzes ist eine Weiterentwicklung des Prototypen sinnvoll. Bisher nicht realisierte Aspekte sind z.B. ein Editor zur Festlegung der Optionsbeziehungen und die Möglichkeit der dynamischen Anwendung von Änderungsoperationen. Die in weiteren Fallstudien gewonnenen Anforderungen und Erkenntnisse würden zudem die Weiterentwicklung und Optimierung des Provop-Ansatzes ermöglichen und weitere Forschungsfragestellungen aufdecken.

Literatur

[AAAN+05] M. Abi-Antoun, J. Aldrich, N. Nahas, B. Schmerl und D. Garlan: *Differencing and Merging of Architectural Views*. Technischer Bericht, Software Research Int'l School of Computer Science Carnegie Mellon Universität, 2005.

[AP03] M. Alanen und I. Porres: *Difference and Union of Models*. Technischer Bericht, TUCS Turku Centre for Computer Science Abo Akadami Universität, 2003.

[AP08] A. Awad und F. Puhlmann: *Structural Detection of Deadlocks in Business Process Models*. In: *Business Information Systems*, S. 239–250, 2008.

[AtHEvdA06] M. Adams, A. H. M. ter Hofstede, D. Edmond und W. M. P. van der Aalst: *Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows*. In: *Proc. of the 14th Int'l Conf. on Cooperative Information Systems (CoopIS)*, LNCS 4275, S. 291–308, 2006.

[AtHvdAE07] M. Adams, A. H. M. ter Hofstede, W. M. P. van der Aalst und D. Edmond: *Dynamic, Extensible and Context-Aware Exception Handling for Workflows*. In: *Proc. of the 15th Int'l Conf. on Cooperative Information Systems (CooplS)*, LNCS 4803, S. 95–112, 2007.

[Bab86] W. A. Babich: *Software Configuration Management*. Addison-Wesley, 1986.

[BACR08] B. Burmeister, M. Arnold, F. Copaciu und G. Rimassa: *BDI-agents for Agile Goal-oriented Business Processes*. In: *Proc. of the 7th Int'l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, S. 37–44, 2008.

[Bau05] T. Bauer: *Integration von prozessorientierten Anwendungen*. In: *Proc. of the 2nd GI-Workshop on Enterprise Application Integration (EAI)*, CEUR Workshop Pro. 141, 2005.

[BB01] F. Bachmann und L. Bass: *Managing Variability in Software Architectures*. In: *Proc. of the Int'l Symposium on Software Reusability*, S. 126–132, 2001.

[BBa07] T. Bauer und R. Bobrik: *Applikationsübergreifendes Monitoring von Geschäftsprozessen*. EMISA Forum, 27(1):14–19, 2007.

[BBb07] R. Bobrik und T. Bauer: *Towards Configurable Process Visualizations with Proviado*. In: *Proc. of the 16th IEEE Int'l Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, S. 367–369, 2007.

[BBG+05] J. Bayer, W. Buhl, C. Giese, T. Lehner, A. Ocampo, F. Puhlmann, E. Richter, A. Schnieders, J. Weiland und M. Weske: *PESOA - Process Family Engineering - Modeling Variant-rich Processes*. Technischer Bericht 18/2005, Hasso-Plattner-Institut, Potsdam, 2005.

[BBR06] R. Bobrik, T. Bauer und M. Reichert: *Proviado - Personalized and Configurable Visualizations of Business Processes*. In: *Proc. of the 7th Int'l Conf. on E-Commerce and Web Technologies (EC-Web)*, LNCS 4082, S. 61–71 2006.

[BD99] T. Bauer und P. Dadam: *Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows*. In: *Proc. of Workshop Informatik*, CEUR Workshop Pro. 24, S. 25–32, 1999.

[Bec00] K. Beck: *Extreme Programming Explained*. Addison-Wesley, 2000.

[BGGB01] M. Becker, L. Geyer, A. Gilbert und K. Becker: *Comprehensive Variability Modeling to Facilitate Efficient Variability Treatment*. In: *Proc. of the 4th Int'l Workshop on Product Family Engineering (PFE)*, S. 294–303, 2001.

[Bob08] R. Bobrik: *Konfigurierbare Visualisierung komplexer Prozessmodelle*. Doktorarbeit, Universität Ulm, 2008.

[BP09] S. Buchwald und T. B. R. Pryss: *IT-Infrastrukturen für flexible, service-orientierte Anwendungen - ein Rahmenwerk zur Bewertung*. In: *13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)*, LNI P-144, S. 524–543, 2009.

[BPM09] *Business Process Modeling Notation (BPMN) Version 1.2*, 2009.

[BRB05] R. Bobrik, M. Reichert und T. Bauer: *Requirements for the Visualization of System-Spanning Business Processes*. In: *Proc. of the 16th Int'l Workshop on Database and Expert Systems Applications (DEXA)*, S. 948–954, 2005.

[BRB07] R. Bobrik, M. Reichert und T. Bauer: *View-Based Process Visualization*. In: *Proc. of the 5th Int'l Conf. on Business Process Management (BPM)*, LNCS 4714, S. 88–95, 2007.

[Bro03] J. Brocke: *Referenzmodellierung - Gestaltung und Verteilung von Konstruktionsprozessen*. Doktorarbeit, Westfälische Wilhelms-Universität Münster, 2003.

[BSBB06] B. Burmeister, H.-P. Steiert, T. Bauer und H. Baumgärtel: *Agile Processes Through Goal- and Context-Oriented Business Process Modeling*. In: *Business Process Management Workshops, held in Conjunction with 4th Int'l Conf. on Business Process Management (BPM)*, LNCS 4103, S. 217–228, 2006.

[Car08] J. Cardoso: *Business Process Control-Flow Complexity: Metric, Evaluation, and Validation*. Int'l Journal of Web Services Research, 5(2):49–76, 2008.

[CCPP96] F. Casati, S. Ceri, B. Pernici und G. Pozzi: *WIDE: Workflow Model and Architecture*. Technischer Bericht CTIT Technical Report 96-19, Universität Twente, 1996.

[CCPP99] F. Casati, S. Ceri, S. Paraboschi und G. Pozzi: *Specification and Implementation of Exceptions in Workflow Management Systems*. ACM Trans. Database Syst., 24(3):405–451, 1999.

[CE00] K. Czarnecki und U. Eisenecker: *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, 2000.

[CGP00] E. M. Clarke, O. Grumberg und D. A. Peled: *Model Checking*. MIT Press, 2000.

[Coa08] W. M. Coalition: *Workflow Management Coalition Workflow Standard: Process Definition Interface – XML Process Definition Language*, 2008.

[CW97] R. Conradi und B. Westfechtel: *Towards a Uniform Version Model for Software Configuration Management*. In: *Proc. of the Int'l Workshop on System Configuration Management (SCM), Workshop held in conjunction with the Int'l Conf. on System Configuration Management (ICSE)*, LNCS 1235, S. 1–17, 1997.

[CW98] R. Conradi und B. Westfechtel: *Version Models for Software Configuration Management*. ACM Computing Surveys, 30(2):232–282, 1998.

[Dad05] P. Dadam: *Flexibles Workflow-Management mit ADPET2*. Heidelberger Innovationsforum, 2005.

[Dav93] F. D. Davis: *User Acceptance of Information Technology: System Characteristics, User Perceptions and Behavioral Impacts*. Int'l Journal of Man-Machine Studies, 38(3):475–487, 1993.

[DBDK04] P. J. Dickinson, H. Bunke, A. Dadej und M. Kraetzl: *Matching graphs with unique node labels*. In: *Pattern Anal. Appl.*, 7:243–254, 2004.

[DGHW07] M. Dumas, A. Grosskopf, T. Hettel und M. T. Wynn: *Semantics of Standard Process Models with OR-Joins*. In: *On the Move to Meaningful Internet Systems: OTM 2007*, LNCS 4803, S. 41–58, 2007.

[DKRR98] H. Davulcu, M. Kifer, C. R. Ramakrishnan und I. V. Ramakrishnan: *Logic Based Modeling and Analysis of Workflows*. In: *Proc. of the 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, S. 25–33. ACM Press, 1998.

[DM08] G. Decker und J. Mendling: *Instantiation Semantics for Process Models*. In: *Proc. of the 6th Int'l Conf. on Business Process Management (BPM)*, LNCS 5240, S. 164–179. 2008.

[DR98] P. Dadam und M. Reichert: *The ADEPT WfMS Project at the University of Ulm*. In: *Proc. of the 1st European Workshop on Workflow and Process Management (WPM)*, 1998.

[DR09] P. Dadam und M. Reichert: *The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support - Challenges and Achievements*. Computer Science - Research and Development, 23(2), 2009.

[DRRM+09] P. Dadam, M. Reichert, S. Rinderle-Ma, K. Goeser, U. Kreher und M. Jurisch: *Von ADEPT zur AristaFlow BPM Suite - Eine Vision wird Realität: „Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen*. EMISA Forum, 29(1):9–28, 2009.

[DSNB04] S. Deelstra, M. Sinnema, J. Nijhuis und J. Bosch: *COSVAM: A Technique for Assessing Software Variability in Software Product Families*. In: *Proc. of the 20th IEEE Int'l Conf. on Software Maintenance (ICSM)*. IEEE Computer Society, S. 458–462, 2004.

[DvdA04] J. Dehnert und W. M. P. van der Aalst: *Bridging The Gap Between Business Models And Workflow Specifications*. Int'l Journal of Cooperative Information Systems, 13(3):289–332, 2004.

[DvdAtH05] M. Dumas, W. van der Aalst und A. ter Hofstede: *Process-aware Information Systems: Bridging People and Software Through Process Technology*. Wiley, 2005.

[EL96] J. Eder und W. Liebhart: *Workflow Recovery*. In: *Proc. of the 4th Int'l Conf. on Cooperative Information Systems (CoopIS)*, S. 124–134, 1996.

[End04] R. Endl: *Regelbasierte Entwicklung betrieblicher Informationssysteme - Gestaltung flexibler Informationssysteme durch explizite Modellierung der Geschäftslogik*. Doktorarbeit, Universität Bern, 2004.

[Esh02] R. Eshuis: *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. Doktorarbeit, Universität Twente, 2002.

[FL04] P. Fettke und P. Loos: *Referenzmodellierungsforschung - Langfassung eines Aufsatzes*. Technischer Bericht, Johannes Gutenberg-Universität Mainz, 2004.

[Fow00] M. Fowler: *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 2000.

[FRF01] A. Fent, H. Reiter und B. Freitag: *Design for Change: Evolving Workflow Specifications in ULTRAflow*. In: *Proc. of the 14th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 2348, S. 516–534, 2001.

[Gad05] A. Gadatsch: *Grundkurs Geschäftsprozess-Management*. Vieweg Verlag, 2005.

[GHS95] D. Georgakopoulos, M. F. Hornick und A. P. Sheth: *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*. Distributed and Parallel Databases, 3:119–153, 1995.

[GL06] V. Gruhn und R. Laue: *Complexity metrics for business process models*. In: *Proc. of the 9th Int'l Conf. on Business Information Systems (BIS)*, LNI 85, S. 1–12, 2006.

[GLK+07] U. Greiner, S. Lippe, T. Kahl, J. A. J. Ziemann und Frank-Walter: *Enterprise Interoperability*. Kapitel Designing and Implementing Cross-Organizational Business Processes - Description and Application of a Modelling Framework, S. 137–147. Springer, 2007.

[GRMR+08] C. W. Günther, S. Rinderle-Ma, M. Reichert, W. van der Aalst und J. Recker: *Using process mining to learn from process changes in evolutionary systems*. Int'l Journal of Business Process Integration and Management, 3:61–78, 2008.

[Gro00] B. R. Group: *Defining Business Rules What are they really?* Technischer Bericht 1.3, Business Rules Group, 2000.

[GRRvdA06] C. W. Günther, S. Rinderle, M. Reichert und W. M. P. van der Aalst: *Change Mining in Adaptive Process Management Systems*. In: *Proc. of the 14th Int'l Conf. on Cooperative Information Systems (CoopIS)*, LNCS 4275, S. 309–326, 2006.

[GvdAJV08] F. Gottschalk, W. M. P. van der Aalst und M. H. Jansen-Vullers: *Mining Reference Process Models and Their Configurations*. In: *On the Move to Meaningful Internet Systems: OTM 2007*, LNCS 5333, S. 263–272, 2008.

[GvdAJVlR07] F. Gottschalk, W. M. P. van der Aalst, M. H. Jansen-Vullers und M. la Rosa: *Configurable Workflow Models*. In: *Int'l Journal of Cooperative Information Systems*. 2007.

[GWJV+09] F. Gottschalk, T. A. C. Wagemakers, M. H. Jansen-Vullers, W. M. P. van der Aalst und M. L. Rosa: *Configurable Process Models: Experiences from a Municipality Case Study*. In: *Proc. of the 21st Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 5565, S. 486–500, 2009.

[HA00] C. Hagen und G. Alonso: *Exception Handling in Workflow Management Systems*. IEEE Transactions on Software Engineering, 26(10):943–958, 2000.

[HBR07] A. Hallerbach, T. Bauer und M. Reichert: *Managing Process Variants in the Process Life Cycle*. Technischer Bericht TR-CTIT-07-87, Universität Twente, 2007.

[HBR08a] A. Hallerbach, T. Bauer und M. Reichert: *Anforderungen an die Modellierung und Darstellung von Prozessvarianten*. Datenbank Spektrum, 24:48–58, 2008.

[HBR08b] A. Hallerbach, T. Bauer und M. Reichert: *Modellierung und Darstellung von Prozessvarianten in Provop*. In: *Modellierung 08 - GI-Fachtagung*, LNI 127, S. 41–56, 2008.

[HBR08c] A. Hallerbach, T. Bauer und M. Reichert: *Context-based Configuration of Process Variants*. In: *Proc. of the 3rd Int'l Workshop on Technologies for Context-Aware Business Process Management (TCoB)*, S. 31-40, 2008.

[HBR09a] A. Hallerbach, T. Bauer und M. Reichert: *Capturing Variability in Business Process Models: The Provop Approach*. In: *Int'l Journal of Software Process Improvement and Practice*. Wiley InterSience, 2009. (forthcoming).

[HBR09b] A. Hallerbach, T. Bauer und M. Reichert: *Configuration and Management of Process Variants*. In: M. Rosemann und J. V. Brocke (Herausgeber): *Handbook on Business Process Management*. Springer-Verlag, 2009. (forthcoming).

[HBR09c] A. Hallerbach, T. Bauer und M. Reichert: *Correct Configuration of Process Variants in Provop*. Technischer Bericht UIB-2009-03, Universität Ulm, 2009.

[HBR09d] A. Hallerbach, T. Bauer und M. Reichert: *Issues in Modeling Process Variants with Provop*. In: *Proc. of the 4th Int'l Workshop on Business Process Design (BPD), Workshop held in conjunction with the Int'l Conf. on Business Process Management (BPM)*, LNBIP 17, S. 56–67, 2009.

[HBR09e] A. Hallerbach, T. Bauer und M. Reichert: *Guaranteeing Soundness of Configurable Process Variants in Provop*. In: *Proc. of the 11th IEEE Conf. on Commerce and Enterprise Computing (CEC)*, S. 98–105, 2009.

[Hel07] T. Hellberg: *Einkauf mit SAP MM - Prozesse, Funktionen, Customizing*. SAP PRESS, 2007.

[Her99] T. Herrmann: *Flexible Präsentation von Prozeßmodellen*. In: U. Arend, E. Eberleh und K. Pitschke (Herausgeber): *Software-Ergonomie '99, Design von Informationswelten, Gemeinsame Fachtagung des German Chapter of the ACM, der Gesellschaft für Informatik (GI) und der SAP AG*, Berichte des German Chapter of the ACM, 53:123–136, Teubner, 1999.

[HKMS94] H. Herbst, G. Knolmayer, T. Myrach und M. Schlesinger: *The specification of business rules: A comparison of selected methodologies*. In: *IFIP WG8.1 Working Conf. on Methods and Associated Tools for the Information Systems Life Cycle*, IFIP Transactions, A-55:29–46, Elsevier, 1994.

[HP99] H. Hoheisel und M. Pfahrer: *Ein temporales Regel-Repository zur Unterstützung evolutionärer Workflow-Modellierung*. In: A.-W. Scheer und M. Nüttgens (Herausgeber): *Electronic Business Engineering / 4. Int'l Tagung Wirtschaftsinformatik*. Physica-Verlag, 1999.

[HP03] G. Halmans und K. Pohl: *Communicating the Variability of a Software-Product Family to Customers*. Software and System Modeling, 2(1):15–36, 2003.

[IBM08] IBM: *IBM WebSphere Business Modeller 6.2*, 2008.

[IBM09] IBM: *IBM WebSphere Process Server 6.2 - IBM WebSphere Integration Developer 6.2*, 2009.

[IDS08] IDS Scheer AG: *ARIS Platform Method 7.1.*, 2008.

[Int09] ECMA International: *ECMAScript Language Specification – ECMA-262*, 2009.

[Jar05] M. Jaring: *Variability Engineering as an Integral Part of the Software Product Family Development Process*. Doktorarbeit, Reichsuniversität Groningen, 2005.

[JFJ+96] N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien und M. E. Wiegand: *Agent-Based Business Process Management*. Int'l Journal of Cooperative Information Systems, 5(2&3):105–130, 1996.

[JWH00] G. Joeris, H. Wache und O. Herzog: *MOKASSIN - Innovative Technologien zum Prozeßmanagement und zur semantischen Datenintegration*. KI Journal, 14(1):52–55, 2000.

[KPP06] D. Kolovos, R. Paige und F. Polack: *Merging Models with the Epsilon Merging Language (EML)*. In: *Proc. of the 9th Int'l Conf. on Model Driven Engineering Languages and Systems (MoDELS)*, LNCS 4199, S. 215–229, 2006.

[KR96] M. Kamath und K. Ramamritham: *Correctness issues in workflow management*. In: *Distributed Systems Engineering*, S. 213–221, 1996.

[KZR+94] K. Kuhn, T. Zemmler, M. Reichert, D. Roesner, O. Baumiller und H. Knapp: *An Integrated Knowledge-Based System to Guide the Physician During Structured Reporting.* Methods of Information in Medicine, 33:417–422, 1994.

[LDC+89] A. Lie, T. Didriksen, R. Conradi, E.-A. Karlsson, S. O. Hallsteinsen und P. Holager: *Change Oriented Versioning*. In: *Proc. of the 2nd European Software Engineering Conf. (ESEC)*, LNCS 387, S. 191–202, 1989.

[LR99] F. Leymann und D. Roller: *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 1999.

[LR07] R. Lenz und M. Reichert: *IT Support for Healthcare Processes - Premises, Challenges, Perspectives*. Data and Knowledge Engineering, 61(1):39–58, 2007.

[LRD06] L. T. Ly, S. Rinderle und P. Dadam: *Semantic Correctness in Adaptive Process Management Systems*. In: *Proc. of the 4th Int'l Conf. on Business Process Management (BPM)*, LNCS 4102, S. 193–208, 2006.

[LRD08b] L. T. Ly, S. Rinderle und P. Dadam: *Integration and verification of semantic constraints in adaptive process management systems*. Data and Knowledge Engineering, 64(1):3–23, 2008.

[LRDtH+08] M. La Rosa, M. Dumas, A. ter Hofstede, J. Mendling und F. Gottschalk: *Beyond Control-Flow: Extending Business Process Configuration to Roles and Objects*. In: *Proc. of the 27th Int'l Conf. on Conceptual Modeling (ER)*, LNCS 5231, S. 199–215, 2008.

[LRMGD09] L. T. Ly, S. Rinderle-Ma, K. Göser und P. Dadam: *On Enabling Integrated Process Compliance with Semantic Constraints in Process Management Systems*. Information Systems Frontiers, 2009.

[LRWa08] C. Li, M. Reichert und A. Wombacher: *Discovering Reference Models by Mining Process Variants Using a Heuristic Approach*. In: *Proc. of the 7th Int'l Conf. on Business Process Management (BPM)*, LNCS 5701, 2009.

[LRWb08] C. Li, M. Reichert und A. Wombacher: *Discovering Reference Process Models by Mining Process Variants*. In: *Proc. of the 6th IEEE Int'l Conf. on Web Services (ICWS)*, S. 45–53, 2008.

[LRWc08] C. Li, M. Reichert und A. Wombacher: *Mining Based on Learning from Process Change Logs*. In: *Proc. of the 4th Int'l Workshop on Business Process Intelligence (BPI), Workshop held in conjunction with Business Process Management (BPM) Conf.*, LNBIP 17, S. 121–133, 2008.

[LRWd08] C. Li, M. Reichert und A. Wombacher: *Mining Process Variants: Goals and Issues (Short Paper)*. In: *Proc. of the 5th IEEE Int'l Conf. on Services Computing (SCC)*, S. 573–576, 2008.

[LRWe08] C. Li, M. Reichert und A. Wombacher: *On Measuring Process Model Similarity based on High-level Change Operations*. In: *Proc. of the 27th Int'l Conf. on Conceptual Modeling (ER)*, LNCS 5231, S. 248–264, 2008.

[LRWf09] C. Li, M. Reichert und A. Wombacher: *What are the Problem Makers: Ranking Activities According to their Relevance for Process Changes*. In: *Proc. of the 7th IEEE Int'l Conf. on Web Services (ICWS)*. IEEE Computer Society Press, 2009.

[LS06] R. Lu und S. Sadiq: *Managing Process Variants as an Information Resource*. In: *Proc. of the 4th Int'l Conf. on Business Process Management (BPM)*, LNCS 4102, S. 426–431, 2006.

[Mau09] C. Mauroner: *Ausführung variantenbehafteter Workflow-Modelle abhängig vom Prozesskontext*. Diplomarbeit, Universität Ulm, 2009.

[MCH07] T. Malone, K. Crowston und G. Herman: *Organizing Business Knowledge - The MIT Process Handbook*. MIT Press, 2007.

[MELS95] K. Misue, P. Eadest, W. Lai und K. Sugiyama: *Layout Adjustment and the Mental Map*. Visual Languages and Computing, 6:183–210, 1995.

[Men08] J. Mendling: *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*. LNBIP 6, 2008.

[MGMR02] S. Melnik, H. Garcia-Molina und E. Rahm: *Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching*. In: *Proc. of the 18th Int'l Conf. on Data Engineering (ICDE)*, IEEE Computer Society, S. 117–128, 2002.

[MGR03] R. Müller, U. Greiner und E. Rahm: *AGENTWORK: A Workflow System Supporting Rule-based Workflow Adaptation*. Technischer Bericht, Universität Leipzig, 2003.

[MHHR06] D. Müller, J. Herbst, M. Hammori und M. Reichert: *IT Support for Release Management Processes in the Automotive Industry*. In: *Proc. of the 4th Int'l Conf. on Business Process Management (BPM)*, LNCS 4102, S. 368–377, 2006.

[MID08] MID: *Innovator*, 2008.

[MNN05] J. Mendling, G. Neumann und M. Nüttgens: *Yet Another Event-driven Process Chain - Modeling Workflow Patterns with yEPCs*. Enterprise Modelling and Information Systems Architectures (EMISA), 1:3–13, 2005.

[MR08] M. Z. Muehlen und J. Recker: *How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation*. In: *Proc. of the 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 5074, S. 465–479, 2008.

[MRH08] D. Mueller, M. Reichert und J. Herbst: *A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures*. In: *Proc. of the 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 5074, S. 48–63, 2008.

[MRH07] D. Mueller, M. Reichert und J. Herbst: *Data-driven Modeling and Coordination of Large Process Structures*. In: *Proc. of the 15th Int'l Conf. on Cooperative Information Systems (CoopIS)*, LNCS 4803, S. 131–149, 2007.

[MRB08] B. Mutschler, M. Reichert und J. Bumiller: *Unleashing the Effectiveness of Process-oriented Information Systems: Problem Analysis, Critical Success Factors and Implications*. IEEE Transactions on Systems, Man, and Cybernetics, 38(3):280–291, 2008.

[Mun96] B. P. Munch: *HiCoV: Managing the Version Space*. In: *Proc. of the Int'l Workshop on System Configuration Management (SCM), Workshop held in conjunction with the Int'l Conf. on System Configuration Management (ICSE)*, LNCS 1167, S. 110–126, 1996.

[Naw09] A. Nawotki: *Wege zum professionellen CAE-Datenmanagement –Towards Professional CAE Data Management*. In: *Medina User Forum*, 2009.

[OAS07] OASIS: *Web Service Business Process Execution Language Version 2.0*, 2007.

[Opd92] W. F. Opdyke: *Refactoring Object-Oriented Frameworks*. Doktorarbeit, Universität Illinois, 1992.

[Par94] D. L. Parnas: *Software aging*. In: *Proc. of the 16th IEEE Int'l Conf. on Software Engineering (ICSE)*, S. 279–287, 1994.

[Par98] H. Partsch: *Requirements-Engineering systematisch - Modellierung für softwaregestützte Systeme*. Springer, 1998.

[Pes08] M. Pesic: *Constraint-Based Workflow Management Systems: Shifting Control to Users*. Doktorarbeit, TU Eindhoven, 2008.

[Pet81] J. L. Peterson: *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.

[PSSvdA07] M. Pesic, M. H. Schonenberg, N. Sidorova und W. M. P. van der Aalst: *Constraint-Based Workflow Models: Change Made Easy*. In: *Proc. of the 15th Int'l Conf. on Cooperative Information Systems (CoopIS)*, LNCS 4803, S. 77–94, 2007.

[PSvdA07] M. Pesic, H. Schonenberg und W. M. P. van der Aalst: *DECLARE: Full Support for Loosely-Structured Processes*. In: *Proc. of the 11th IEEE Int'l Enterprise Distributed Object Computing Conf. (EDOC)*, S. 287–300, 2007.

[PSWW05] F. Puhlmann, A. Schnieders, J. Weiland und M. Weske: *PESOA - Variability Mechanisms for Process Models*. Technischer Bericht 17/2005, Hasso-Plattner-Institut Potsdam, 2005.

[RAE04] N. Russell, Arthur und D. Edmond: *Workflow Data Pattern*. Technischer Bericht FIT-TR-2004-01, Queensland University of Technology, 2004.

[RAvdAM06] N. Russell, Arthur, W. M. P. van der Aalst und N. Mulyar: *Workflow Control-Flow Patterns: A Revised View*. Technischer Bericht BPM-06-22, BPM Center, 2006.

[RB07] G. Rimassa und B. Burmeister: *Achieving Business Process Agility in Engineering Change Management with Agent Technology*. In: *Proc. of the 8th AI*IA/TABOO Joint Workshop "From Objects to Agents": Agents and Industry: Technological Applications of Software Agents (WOA)*, S. 1–7, 2007.

[RD98] M. Reichert und P. Dadam: *ADEPTflex–Supporting Dynamic Changes of Workflows Without Losing Control*. Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, 10(2):93–129, 1998.

[Rei00] M. Reichert: *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. Doktorarbeit, Universität Ulm, 2000.

[Rei09] M. Reichert: *A Thing Called "Fluid Process" - Beyond Rigidity in Business Process Support*. In: *Proc. of the 3rd Int'l Workshop EMISA*, LNI P-152, Koellen-Verlag, 2009.

[RG09] M. L. Rosa und F. Gottschalk: *Synergia - comprehensive tool support for configurable process models*. In: *Proc. of the 7th Int'l Conf. on Business Process Management (Demonstration Program)*, 2009.

[RGDvdA07] M. L. Rosa, F. Gottschalk, M. Dumas und W. M. P. van der Aalst: *Linking Domain Models and Process Models for Reference Model Configuration*. In: *Business Process Management Workshops*, LNCS 4928, S. 417–430, 2007.

[Rin04] S. Rinderle: *Schema Evolution in Process Management Systems*. Doktorarbeit, Universität Ulm, 2004.

[RJR07] S. Rinderle, M. Jurisch und M. Reichert: *On Deriving Net Change Information From Change Logs - The DELTALAYER-Algorithm*. In: *12. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)*, S. 364–381, 2007.

[RKL+06] S. Rinderle, U. Kreher, M. Lauer, P. Dadam und M. Reichert: *On Representing Instance Changes in Adaptive Process Management Systems*. In: *Proc. of the 15th IEEE Int'l Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, S. 297–304, 2006.

[RLS+07] M. L. Rosa, J. Lux, S. Seidel, M. Dumas und A. H. M. ter Hofstede: *Questionnaire-driven Configuration of Reference Process Models*. In: *Proc. of the 19th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 4495, S. 424-438, 2007.

[RMRW08a] S. Rinderle-Ma, M. Reichert und B. Weber: *On the Formal Semantics of Change Patterns in Process-aware Information Systems*. In: *Proc. of the 27th Int'l Conf. on Conceptual Modeling (ER)*, LNCS 5231, S. 279–293, 2008.

[RMRW08b] S. Rinderle-Ma, M. Reichert und B. Weber: *Relaxed Compliance Notions in Adaptive Process Management Systems*. In: *Proc. of the 27th Int'l Conf. on Conceptual Modeling (ER)*, LNCS 5231, S. 232–247, 2008.

[RMvdT09] H. A. Reijers, R. S. Mans und R. A. van der Toorn: *Improved model management with aggregated business process models*. Data Knowledge and Engineering, 68(2):221–243, 2009.

[Ros03] R. G. Ross: *The Business Rule Approach*. IEEE Computer, 36(5):85–87, 2003.

[Ros09] M. L. Rosa: *Managing Variability in Process-Aware Information Systems*. Doktorarbeit, Queensland University of Technology, 2009.

[RR06] S. Rinderle und M. Reichert: *Data-Driven Process Control and Exception Handling in Process Management Systems*. In: *Proc. of the 18th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 4001, S. 273–287, 2006.

[RDB03] M. Reichert, P. Dadam und T. Bauer: *Dealing with forward and backward jumps in workflow management systems*. In: *Int'l Journal Software and Systems Modeling (SOSYM)*, 2(1), S. 37–58, 2003.

[RRD03] M. Reichert, S. Rinderle und P. Dadam: *On the Common Support of Workflow Type and Instance Changes under Correctness Constraints*. In: *Proc. of the 11th Int'l Conf. on Cooperative Information Systems (CooplS)*, LNCS 2888, S. 407–425, 2003.

[RRD03] S. Rinderle, M. Reichert und P. Dadam: *Evaluation Of Correctness Criteria For Dynamic Workflow Changes.* In: *Proc. of the 1st Int'l Conf. on Business Process Management (BPM)*, LNCS 2678, S. 41–57, 2003.

[RRD04a] S. Rinderle, M. Reichert und P. Dadam: *Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey.* Data and Knowledge Engineering, 50(1):9–34, 2004.

[RRD04b] S. Rinderle, M. Reichert und P. Dadam: *Disjoint And Overlapping Process Changes: Challenges, Solutions, Applications*. In: *Proc. of the 12th Int'l Conf. on Cooperative Information Systems (CoopIS)*, LNCS 3290, S. 101–120, 2004.

[RRD04c] S. Rinderle, M. Reichert und P. Dadam: *On Dealing With Structural Conflicts Between Process Type and Instance Changes*. In: *Proc. of the 2nd Int'l Conf. on Business Process Management (BPM)*, LNCS 3080, S. 274–289. 2004.

[RRHB09] M. Reichert, S. Rechtenbach, A. Hallerbach und T. Bauer: *Extending a Business Process Modeling Tool with Process Configuration Facilities - The Provop Demonstrator*. In: *Proc. of the 7th Int'l Conf. on Business Process Management (Demonstration Program)*, 2009.

[RRJK06] S. Rinderle, M. Reichert, M. Jurisch und U. Kreher: *On Representing, Purging, and Utilizing Change Logs in Process Management Systems*. In: *Proc. of the 4th Int'l Conf. Business Process Management (BPM)*, LNCS 4102, S. 241–256, 2006.

[RRKD05] M. Reichert, S. Rinderle, U. Kreher und P. Dadam: *Adaptive Process Management with ADEPT2*. In: *Proc. of the 21st Int'l Conf. on Data Engineering (ICDE)*, S. 1113–1114, 2005.

[RRvHZ00] C. Rupprecht, T. Rose, E. van Halm und A. Zwegers: *Project-specific Process Configuration in Virtual Enterprises*. In: *Proc. of the 4th Int'l Conf. on the Design of Information Infrastructure Systems for Manufacturing (DIISM)*, IFIP 191, S. 46–53, 2000.

[RtHRS08] M. L. Rosa, A. ter Hofstede, M. Rosemann und K. Shortland: *Bringing Process to Post Production*. In: *Proc. of the Int'l Conf. on Creating Value: Between Commerce and Commons*, 2008.

[Rup02] C. Rupprecht: *Ein Konzept zur projektspezifischen Individualisierung von Prozessmodellen*. Doktorarbeit, Universität Fridericiana Karlsruhe, 2002.

[RV04] H. A. Reijers und I. T. P. Vanderfeesten: *Cohesion and Coupling Metrics for Workflow Process Design*. In: *Proc. of the 2nd Int'l Conf. on Business Process Management (BPM)*. LNCS 3080, S. 290-305, 2004.

[RvdA07] M. Rosemann und W. van der Aalst: *A Configurable Reference Modeling Language*. Information Systems, 32:1–23, 2007.

[RvdAtH06] N. Russell, W. M. P. van der Aalst und A. H. M. ter Hofstede: *Workflow Exception Patterns*. In: *Proc. of the 18th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 4001, S. 288–302, 2006.

[RWR06] S. Rinderle, A. Wombacher und M. Reichert: *On the Controlled Evolution of Process Choreographies*. In: *Proc. of the 22nd Int'l Conf. on Data Engineering (ICDE)*, 2006.

[Sch97] A.-W. Scheer: *Wirtschaftsinformatik : Referenzmodelle für industrielle Geschäftsprozesse*. Springer, 1997.

[Sch97] R. Schütte: *Grundsätze ordnungsgemäßer Referenzmodellierung: Konstruktion konfigurations- und anpassungsorientierter Modelle*. Doktorarbeit, Universität Münster, 1997.

[Sch98] A.-W. Scheer: *ARIS- Modellierungsmethoden, Metamodelle, Anwendungen*. Springer, 1998.

[Sci94] E. Sciore: *Versioning and Configuration Management in an Object-Oriented Data Model*. VLDB Journal, 3(1):77–106, 1994.

[SG08] C.-M. Seiler und M. Grauer: *Einsatz von Konfigurations- und Variantenmanagement in der Einzelfertigung*. In: *Multikonferenz Wirtschaftsinformatik*, 2008.

[SHTB07] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux und Y. Bontemps: *Generic semantics of feature diagrams*. Computer Networks, 51(2):456–479, 2007.

[Sin95] M. P. Singh: *Semantical Considerations on Workflows: An Algebra for Intertask Dependencies*. In: *Proc. of the 5th Int'l Workshop on Database Programming Languages*, 1995.

[SOS05] S. W. Sadiq, M. E. Orlowska und W. Sadiq: *A framework for constraint specification and validation in flexible workflows*. Information Systems, 30(5):349–378, 2005.

[SvGB05] M. Svahnberg, J. van Gurp und J. Bosch: *A taxonomy of variability realization techniques*. Software - Practice and Experience, 35:705–754, 2005.

[SZ06] J. Siegeris und A. Zimmermann: *Workflow Model Compositions Preserving Relaxed Soundness*. In: *Proc. of the 4th Int'l Conf. on Business Process Management (BPM)*, LNCS 4102, S. 177–192, 2006.

[TBB08] F. Touré, K. Baïna und K. Benali: *An Efficient Algorithm for Workflow Graph Structural Verification*. In: *On the Move to Meaningful Internet Systems: OTM 2008*, LNCS 5333, S. 392–408, 2008.

[TBG08] F. Touré, K. Baïna und W. Gaaloul: *Toward a Hybrid Algorithm for Workflow Graph Structural Verification*. In: *Proc. of the 10th Int'l Conf. on Enterprise Information Systems (ICEIS)*, S. 442–447, 2008.

[Tho06] O. Thomas: *Das Referenzmodellverständnis in der Wirtschaftsinformatik: Historie, Literaturanalyse und Begriffsexplikation*. Technischer Bericht 187, Institut für Wirtschaftsinformatik im Deutschen Forschungszentrum für Künstliche Intelligenz, 2006.

[UB08] T. Unger und T. Bauer: *Towards a Standardized Task Management*. In: *Multikonferenz Wirtschaftsinformatik (MKWI)*, 2008.

[UML09] *OMG Unified Modeling Language (OMG UML) Version 2.2*, 2009.

[vdA97] W. M. P. van der Aalst: *Verification of Workflow Nets*. In: W. van der Aalst (Herausgeber): *Proc. of the 18th Int'l Conf. on Application and Theory of Petry Nets (ICATPN)*, LNCS 1248, S. 407–426, 1997.

[vdA98] W. M. P. van der Aalst: *The Application of Petri Nets to Workflow Management*. The Journal of Circuits, Systems and Computers, 8(1):21–66, 1998.

[vdA00] W. M. P. van der Aalst: *Workflow Verification: Finding control-flow errors using petrinet-based techniques*. In: *Business Process Management, Models, Techniques, and Empirical Studies*, 2000.

[VDA05] VDA Recommendation 4965 T1: *Engineering Change Management (ECM) - Part 1: Engineering Change Request (ECR) Version 1.1*, 2005.

[vdAB02] W. M. P. van der Aalst und T. Basten: *Inheritance of Workflows: An Approach to Tackling Problems Related to Change*. Technischer Bericht, TU Eindhoven, 2002.

[vdADG+05] W. M. P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann und M. H. Jansen-Vullers: *Configurable Process Models as a Basis for Reference Modeling*. In: *Proc. of the Int'l Workshop on Business Process Management, Workshop held in Conjunction with the Int'l Conf. on Business Process Management (BPM)*, LNCS 3812, S. 512–518, 2005.

[vdADG+08] W. M. P. van der Aalst, M. Dumas, F. Gottschalk, A. H. M. ter Hofstede, M. la Rosa und J. Mendling: *Correctness-Preserving Configuration of Business Process Models*. In: *Proc. of the 11th Int'l Conf. on Fundamental Approaches to Software Engineering (FASE)*, LNCS 4961, S. 46–61, 2008.

[vdAdMW06] W. M. P. van der Aalst, A. K. A. de Medeiros und A. J. M. M. Weijters: *Process Equivalence: Comparing Two Process Models Based on Observed Behavior*. In: *Proc. of the 4th Int'l Conf. on Business Process Management (BPM)*, LNCS 4102, S. 129–144, 2006.

[vdAGRR06] W. M. P. van der Aalst, C. Günther, J. Recker und M. Reichert: *Using Process Mining to Analyze and Improve Process Flexibility*. In: *Proc. of the CAISE*06 Workshop on Business Process Modelling, Development, and Support (BPMDS)* CEUR Workshop Proceedings 263, 2006.

[vdAPS09] W. M. P. van der Aalst, M. Pesic und H. Schonenberg: *Declarative workflows: Balancing between flexibility and support*. Computer Science - Research and Development (CSRD), 23(2):99–113, 2009.

[vdAtH05] W. M. P. van der Aalst und A. H. M. ter Hofstede: *YAWL: yet another workflow language*. Information Systems, 30(4):245–275, 2005.

[vdAvDH+03] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm und A. J. M. M. Weijters: *Workflow mining: A survey of issues and approaches*. Data and Knowledge Engineering, 47(2):237–267, 2003.

[vDMvdA06] B. F. van Dongen, J. Mendling und W. M. P. van der Aalst: *Structural Patterns for Soundness of Business Process Models*. In: *Proc. of the 10th IEEE Int'l Enterprise Distributed Object Computing Conf. (EDOC)*, S. 116–128, 2006.

[Ver07] M. Verfuurt: *Modeling Business Process Variability - A search for innovative solutions to business process variabilit modeling problems*. Diplomarbeit, Universität Twente, 2007.

[vGBS01] J. van Gurp, J. Bosch und M. Svahnberg: *On the Notion of Variability in Software Product Lines*. In: *Working IEEE / IFIP Conf. on Software Architecture (WICSA 2001)*, S. 45–54, 2001.

[VH01] B. Von Halle: *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. Wiley, 2001.

[VRM+08] I. T. P. Vanderfeesten, H. A. Reijers, J. Mendling, W. M. P. van der Aalst und J. Cardoso: *On a Quest for Good Process Models: The Cross-Connectivity Metric*. In: *Proc. of the 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 5074, S. 480–494, 2008.

[VvdAtH07] H. M. W. Verbeek, W. M. P. van der Aalst und A. H. M. ter Hofstede: *Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Relaxed Soundness and Invariants*. The Computer Journal, 50(3):294–314, 2007.

[VW99] G. Vossen und M. Weske: *The WASA2 Object-Oriented Workflow Management System*. In: *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, S. 587–589, 1999.

[W3C09] W3C Recommendation: *Extensible Markup Language (XML) 1.1*, 2009. zuletzt besucht am 17.06.2009.

[Wes01] M. Weske: *Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System*. In: *Proc. of the Hawaii Int'l Conf. on System Sciences (HICSS)*, 2001.

[Wes07] M. Weske: *Business Process Management - Concepts, Languages, Architectures*. Springer, 2007.

[WEvdAtH05] M. T. Wynn, D. Edmond, W. M. P. van der Aalst und A. H. M. ter Hofstede: *Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets*. In: *Proc. of the 26th Int'l Conf. on Application and Theory of Petri Nets (ICATPN)*, LNCS 3536, S. 423–443, 2005.

[WG05] W. M. P. van der Aalst, M. Weske, D. Grünbauer: *Case Handling: A New Paradigm for Business Process Support*. Data and Knowledge Engineering, 53(2):129–162, 2005.

[WK08] K. Wahler und J. M. Küster: *Predicting Coupling of Object-Centric Business Process Implementations*. In: *Proc. of the 6th Int'l Conf. on Business Process Management (BPM)*, LNCS 5240, S. 148–163, 2008.

[Wom06] A. Wombacher: *Evaluation of technical measures for workflow similarity based on a pilot study*. Technischer Bericht, Universität Twente, 2006.

[WPZW09] B. Weber, J. Pinggera, S. Zugal und W. Wild: *Alaska Simulator - A Journey to Planning*. In: *Proc. of the 10th Int'l Conf. on Agile Processes in Software Engineering and Extreme Programming (XP)*, S. 253–254, 2009.

[WR08] B. Weber und M. Reichert: *Refactoring Process Models in Large Process Repositories*. In: *Proc. of the 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 5074, S. 124–139, 2008.

[WRR07] B. Weber, S. B. Rinderle und M. Reichert: *Change Patterns and Change Support Features in Process-Aware Information Systems*. In: *Proc. of the 19th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, LNCS 4495, S. 574–588, 2007.

[WRR08] B. Weber, M. Reichert und S. Rinderle-Ma: *Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems*. Data and Knowledge Engineering, 66(3):438–466, 2008.

[WRW06] B. Weber, M. Reichert und W. Wild: *Case-Base Maintenance for CCBR-based Process Evolution*. In: *Proc. of the 8th European Conf. on Case-Based Reasoning (ECCBR)*, LNCS 4106, S. 106–120, 2006.

[WRWRM09] B. Weber, M. Reichert, W. Wild und S. Rinderle-Ma: *Providing Integrated Life Cycle Support in Process-Aware Information Systems*. Int'l Journal of Cooperative Information Systems (IJCIS), 18(1):115–165, 2009.

[WSR09] B. Weber, S. Sadiq und M. Reichert: *Beyond Rigidity - Dynamic Process Lifecycle Support: A Survey on Dynamic Changes in Process-aware Information Systems*. Computer Science - Research and Development, 23(2):47-65, 2009.

[WWRD07] B. Weber, W. Wild, M. U. Reichert und P. Dadam: *ProCycle - Integrierte Unterstützung des Prozesslebenszyklus*. In: *KI*, 21(4):9–15, 2007.

[WZPW09] B. Weber, S. Zugal, J. Pinggera und W. Wild: *Experiencing Process Flexibility Patterns with Alaska Simulator*. In: *Proc. of the 7th Int'l Conf. on Business Process Management (Demonstration Program)*, 2009.

[Zen06] C. Zenner: *Durchgängiges Variantenmanagement in der Technischen Produktionsplanung*. Technischer Bericht, Universität des Saarlandes, 2006.

[ZWR01] A. Zündorf, J. P. Wadsack und I. Rockel: *Merging graph-like object structures*. In: *Proc. of the 10th Workshop on Software Configuration Management (SCM)*, 2001.

Teil IV:
Anhang

A

Provop-Funktionsübersicht

Im Folgenden werden die in dieser Arbeit verwendeten Funktionen in alphabetischer Reihenfolge aufgeführt.

calculateDynamicOptions

Diese Funktion identifiziert für eine gegebene Kontextbeschreibung die Menge der statischen und dynamischen Optionen (vgl. Abschnitt 7.2.2).

Function calculateDynamicOptions (In: CtxtDescr, OptionSet; Out: staticOptions, dynamicOptions)
// identifiziere alle als statisch spezifizierten Kontextvariablen des Kontextmodells
V_{stat} = {$StatV_1$, .. , $StatV_n$} ∈ CtxtModel with $StatV_i$.mode = static for i = 1..n
// identifiziere alle als dynamisch spezifizierten Kontextvariablen des Kontextmodells
V_{dyn} = {$DynV_1$, .. , $DynV_m$} ∈ CtxtModel with $DynV_j$.mode = dynamic for j = 1..m

// Initialisierung
staticOptions = ∅
dynamicOptions = ∅

// initialisiere für jede modellierte Option der global verfügbaren Menge `allDefinedOptions`
// die Variablen `alwaysUsed` *und* `neverUsed`
for each option ∈ allDefinedOptions **do**
    alwaysUsed(option) = true
    neverUsed(option) = true

// erzeuge für jede statische Kontextvariable ein Tupel bestehend aus Name und aktuellen Wert
// (gegeben durch die aktuell betrachtete Kontextbeschreibung)
for each i = 1..n **do**
    $StatSimVar_i$:= <$StatV_i$.name, getValue($StatV_i$.name,CtxtDescr)>

// erzeuge für jede dynamische Kontextvariable und für alle möglichen Werte aus ihrem Wertebereich
// je ein Tupel aus dem Namen der Variablen und einem möglichen Wert

```
DynSimVar = ∅
for each j = 1..m do
    for each value ∈ ValueRange(DynV_j) do
        DynSimVar := DynSimVar ∪ {<DynV_j.name, value>}

// simuliere alle dynamisch erreichbaren Kontextbeschreibungen; erzeuge diese aus dem
// Kreuzprodukt der oben angelegten Tupel (d.h. <CtxtVariable.name,value>)
for each CtxtDescr
  ∈ {StatSimVar_1} ×...× {StatSimVar_k} × {DynSimVar_1} ×...× {DynSimVar_m}
    for each Option ∈ allDefinedOptions do
        if contextRuleValid(Option,CtxtDescr) then
            // die Kontextbedingung ist erfüllt, daher wird Option nicht niemals angewandt
            neverUsed(Option) = false
        // die Kontextbedingung ist nicht erfüllt, daher wird Option nicht immer angewandt
        else alwaysUsed(Option) = false

// statische Optionen werden immer angewandt
// dynamische werden in mindestens einem, aber nicht in allen Kontextbeschreibungen angewandt
for each Option ∈ allDefinedOptions do
    if alwaysUsed(Option)= true AND neverUsed(Option) = false then
        staticOptions := staticOptions ∪ {Option}
    if alwaysUsed(Option)= false AND neverUsed(Option) = false then
        dynamicOptions := dynamicOptions ∪ {Option}
```

calculateVariant

Transformiert den Basisprozesses durch Anwendung einer Menge von Optionen zu einem Ergebnismodell.

```
Function calculateVariant (In: OptionSet, BaseProcess; Out: ResultingModel) : ProcessModel
ResultingModel = BaseProcess
error = false
// wende alle Optionen einer gegebenen Optionsmenge an
for each Option ∈ OptionSet do
    // wende alle Änderungsoperationen einer Option an
    for each Operation ∈ Option do
        // prüfe den Typ der Änderungsoperation und rufe entsprechende Funktion auf
        switch (Operation.getParameter(„Type“))
            case insert: error = insert(ResultingModel, Option, Operation)
            case delete: error = delete(ResultingModel, Option, Operation)
            case move: error = move(ResultingModel, Option, Operation)
            case modify: error = modify(ResultingModel, Option, Operation)
    // Prüfe ob Fehler bei der Anwendung der Änderungsoperation aufgetreten sind
    if error = true then
        // schreibe Fehler in Fehlerbericht
        writeErrorList(...)
        break
    else simplifyProcessModel(ResultingModel)
// wurden alle Änderungsoperationen aller Optionen ohne Fehler angewandt,
```

```
// wird das Ergebnismodell zurückgegeben
return ResultingModel
```

checkCompatibilityAtRuntime

Prüft die Kompatibilität der Anwendung (`ToBeApplied = true`) bzw. des Auslassens (`ToBeApplied = false`) einer Option mit harten Optionsbeziehungen zur Laufzeit (vgl. Abschnitt 7.3.2).

```
// Globale Variablen:
// AppliedOptions : Menge aller bisher statisch (stat.) und dynamisch (dyn.) angewandten Optionen
// NotAppliedOptions : Menge aller stat und dyn. ausgelassenen Optionen
// UndefinedOptions : Menge aller dyn. Optionen über deren Anwendung noch nicht entschieden wurde
Function checkCompatibilityAtRuntime (In: Option, ToBeApplied: boolean)
// Entnehme aktuell betrachtete Option aus der Menge der nicht entschiedenen Optionen
TempUndefinedOptions = UndefinedOptions - {Option}
UndefOptPowerSet = getPowerSet(TempUndefinedOptions)
for each Set in UndefOptPowerSet do
    if ToBeApplied = true then // Option soll angewandt werden
        TempAppliedOptions = AppliedOptions ∪ {Option} ∪ Set
    else // Option soll ausgelassen werden
        TempAppliedOptions = AppliedOptions ∪ Set
    if checkOptionConstraints(TempAppliedOptions,true) = true then
        return true // es existiert mindestens eine kompatible Optionsmenge
return false // es existiert keine kompatible Optionsmenge
```

checkOptionConstraints

Prüft die Einhaltung von weichen und bzw. oder harten Optionsbeziehungen für eine gegebene Optionsmenge.

Function checkOptionConstraints (In: OptionSet,IsStrong : boolean) : boolean
checkOptionConstraints
Seien die Optionsbeziehungen global gegeben. Dann ist der Rückgabewert der Funktion `true`, wenn das gegebene OptionSet kompatibel mit all diesen Optionsbeziehungen ist. Andernfalls ist der Rückgabewert `false` und eine entsprechende Fehlermeldung wird erzeugt. Die boolesche Variable IsStrong gibt an, ob nur harte oder auch weiche Optionsbeziehungen geprüft werden sollen. Die entsprechende Information ist als Attribut einer Optionsbeziehung durch die Modellierung gegeben.

checkProcessConsistency

Prüft die Korrektheit eines Prozessmodells (vgl. Abschnitt 6.5).

Function checkProcessConsistency (In: ResultingModel) : boolean
Der Rückgabewert der Funktion ist `true`, wenn das gegebene Ergebnismodell korrekt ist, hinsichtlich der spezifischen Eigenschaften des zugrunde liegenden Prozess-Metamodells. Andernfalls ist der Rückgabewert `false` und eine entsprechende Fehlermeldung wird erzeugt.

contextDescriptionValid

Prüft die Gültigkeit einer Kontextbeschreibung hinsichtlich der Kontextregeln des entsprechenden Kontextmodells.

Function ctxtDescriptionValid (In: CtxtDescr) : boolean
Der Rückgabewert der Funktion ist `true`, wenn die gegebene CtxtDescr (d.h. Kontextbeschreibung) gültig ist hinsichtlich aller definierter Kontextregeln des zugrunde liegenden Kontextmodells. Andernfalls ist der Rückgabewert `false`.

contextRuleValid

Prüft die Relevanz einer Option in einer gegebenen Kontextbeschreibung.

Function contextRuleValid (In: Option, CtxtDescr) : boolean
Der Rückgabewert der Funktion ist `true`, wenn die Kontextbedingung, die einer Option zugewiesen ist, in der gegebenen Kontextbeschreibung erfüllt ist. Andernfalls ist der Rückgabewert `false`.

simplifyProcessModel

Vereinfacht ein gegebenen Prozessmodell.

Function simplifyProcessModel (In: ResultingModel) : ProcessModel
Die Funktion wendet die normalen und provopspezifischen Vereinfachungsoperationen nacheinander an. Da nach Anwendung einer Vereinfachungsoperation ggf. die Anwendung weiterer Vereinfachungsoperationen möglich ist, wird die Anwendbarkeit der Operationen solange geprüft bzw. durchgeführt, bis in einem Durchlauf keine von ihnen mehr angewandt werden kann. Die Funktion gibt dann das vereinfachte Prozessmodell zurück.

sortOptionSet

Sortiert eine gegebene Optionsmenge zu einer Optionsfolge. Dabei sei $SeqConstraint(Option_i, Option_j)$ definiert als Reihenfolgebeziehung zwischen $Option_i$ und $Option_j$ (d.h. $Option_i$ wird vor $Option_j$ auf den Basisprozess angewandt). Sei weiter $SeqConstraint^*(Option_i, Option_j)$ definiert als die transitive Hülle von $SeqConstraint(Option_i, Option_j)$.

Function sortOptionSet (In: OptionsSet; Out: sortedOptionSet) : boolean
if $\exists$ $SeqConstraint^*(Option_i, Option_j)$ AND $\exists$ $SeqConstraint^*(Option_j, Option_i)$ = true **then**
  // es gibt eine zyklische Reihenfolgebeziehungen
  insertInErrorList(...)
  return false
else
  for each $i < j \in (1, \ldots n)$ **do**
    if $\exists$ $SeqConstraint^*(Option_i, Option_j)$
    OR $\left(\nexists\ SeqConstraint^*(Option_i, Option_j)\ \text{AND}\ \nexists\ SeqConstraint^*(Option_j, Option_i)\right)$
    AND $\left(getTimeStamp(Option_i) < getTimeStamp(Option_j)\right)$ = true **then**
      // sortiere die Optionen mit einem beliebigen Sortieralgorithmus
      sortedOptionList = $(Option_1, \ldots, Option_n)$ with $Option_i < Option_j$
  return true

B

Weitere Abbildungen

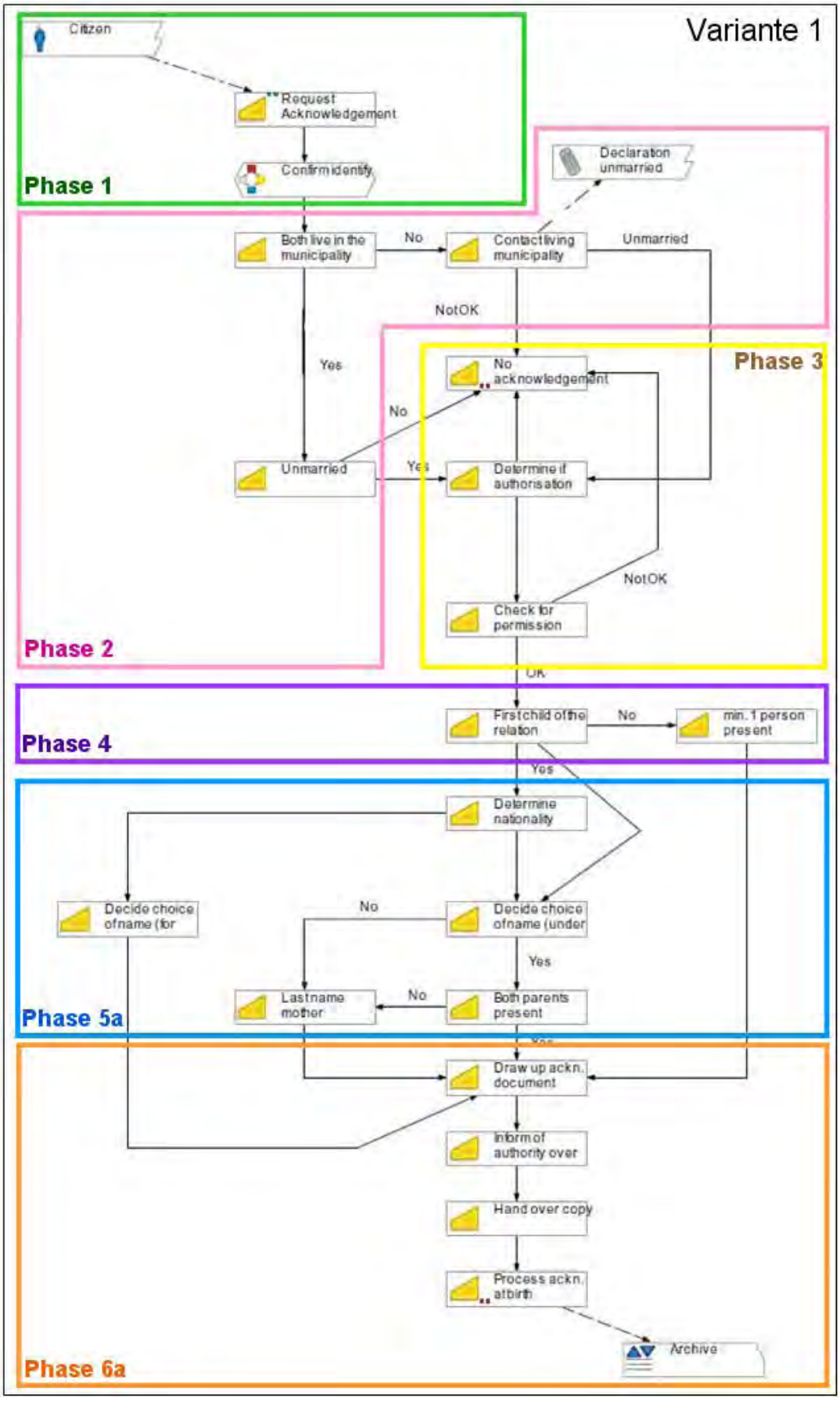


Abbildung B.1: Variante 1 des Registrierungsprozesses nach [GWJV+09]

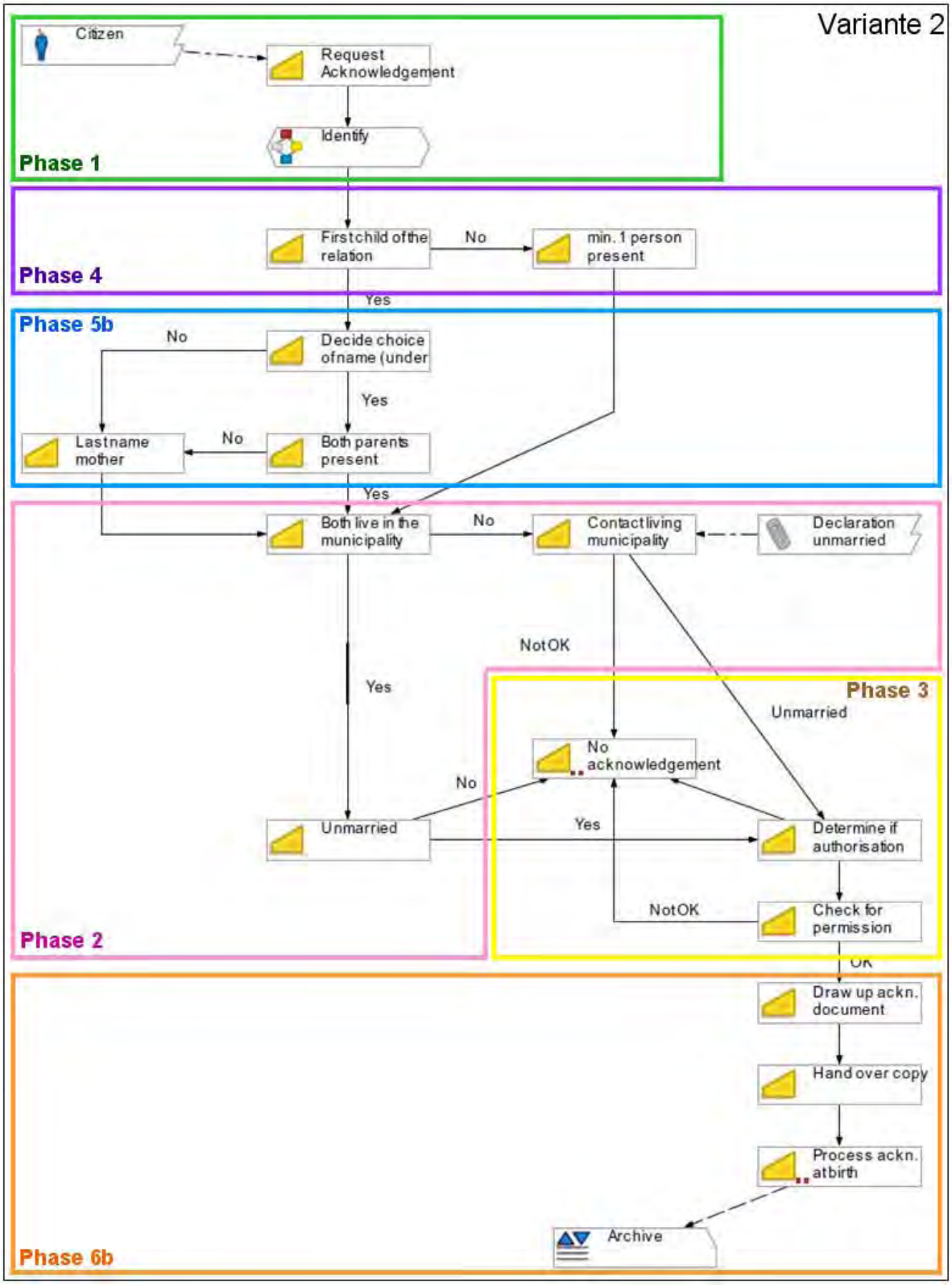


Abbildung B.2: Variante 2 des Registrierungsprozesses nach [GWJV+09]

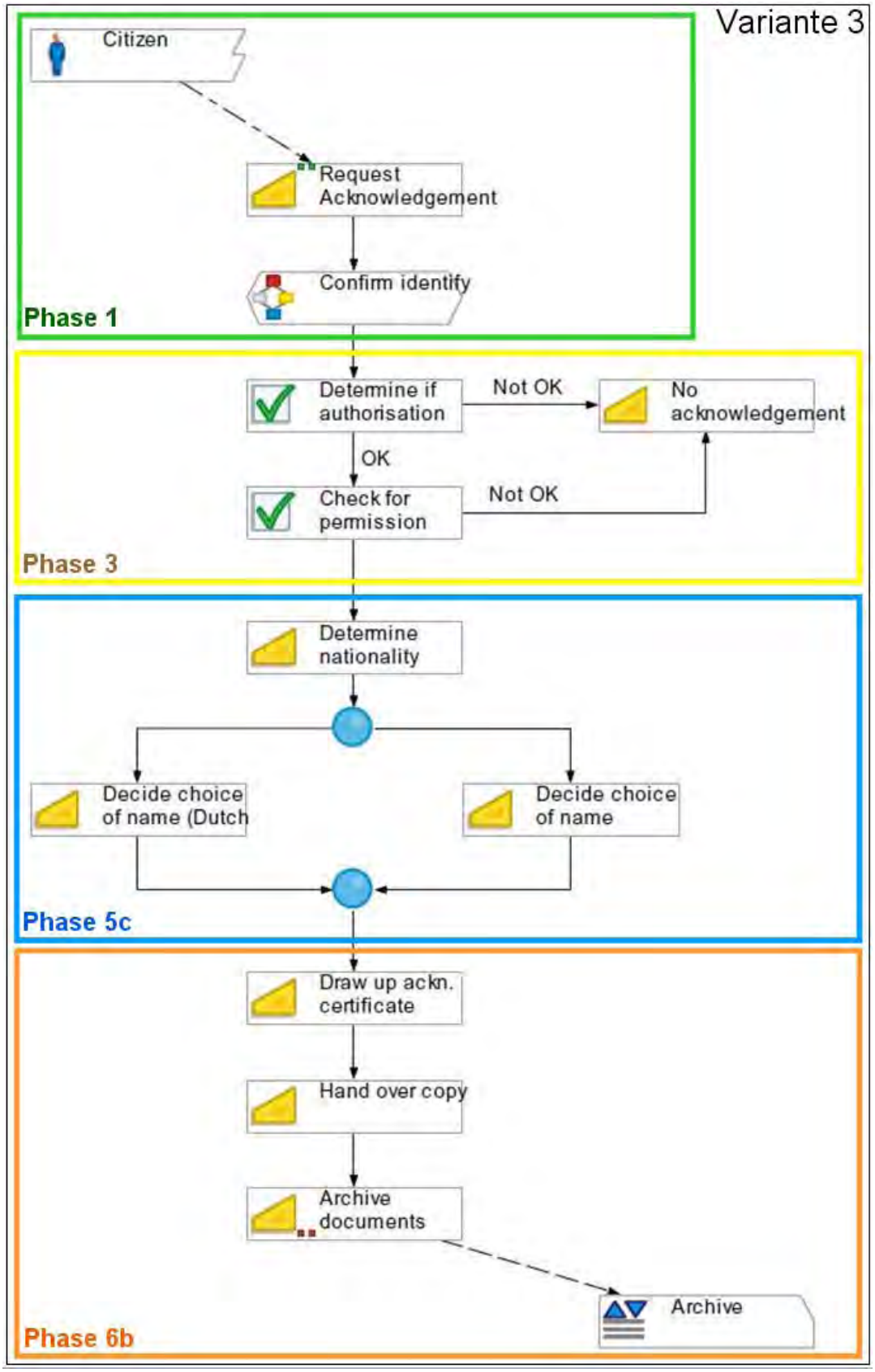


Abbildung B.3: Variante 3 des Registrierungsprozesses nach [GWJV+09]

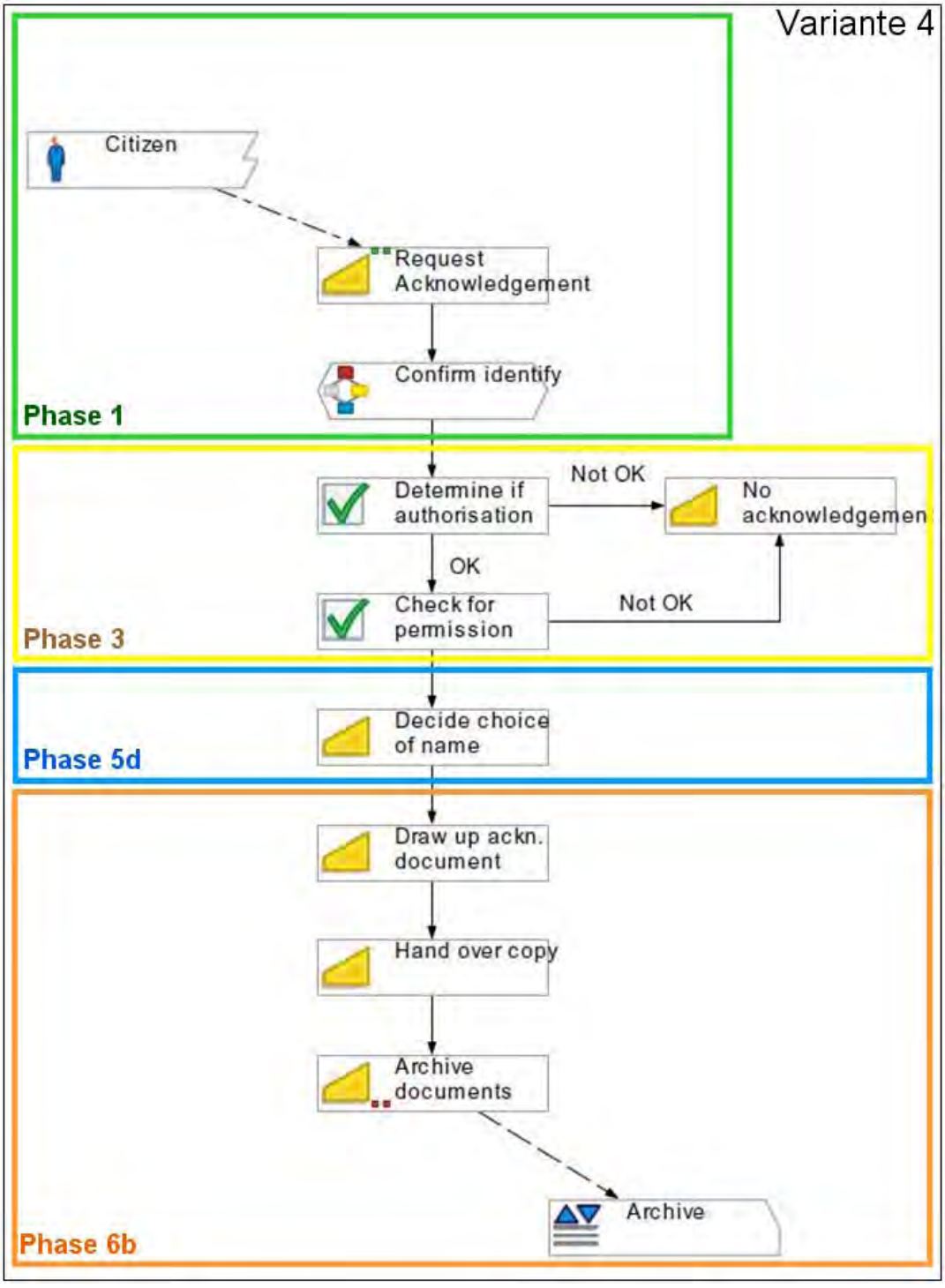


Abbildung B.4: Variante 4 des Registrierungsprozesses nach [GWJV+09]

Abbildungsverzeichnis

Tabellenverzeichnis