ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik

Institut für Datenbanken und Informationssysteme

Anwender- und aufgabenzentrierte Auswertung der Erfüllung semantischer Constraints in Prozess-Management-Systemen

Diplomarbeit an der Universität Ulm

Vorgelegt von:
Philipp Merkel
philipp.merkel@uni-ulm.de

Gutachter:
Prof. Dr. Peter Dadam
Prof. Dr. Manfred Reichert

Betreuerin:
Linh Thao Ly

2010

SeaFlows

Fassung 20. Juli 2010

Zusammenfassung

Durch den Einsatz von Business Process Management (BPM) zur Steuerung betrieblicher Abläufe eröffnen sich neue Möglichkeiten, die Einhaltung von Vorgaben und Regeln in Geschäftsprozessen über deren gesamten Lebenszyklus hinweg zu überprüfen. Insbesondere bei der Prüfung eines Prozesses in der Entwurfsphase genügt es dabei nicht, anzugeben, ob eine Regel vom Prozess erfüllt oder verletzt wird – schließlich ist es häufig der Fall, dass die Erfüllung vom Verlauf der Prozessausführung abhängt. Außerdem ist bei umfangreichen Prozessen und Regeln die Ursache für eine Verletzung möglicherweise nicht auf den ersten Blick ersichtlich. Daher ist eine detaillierte Visualisierung der Ergebnisse notwendig, aus der erkennbar ist, in welchen Fällen und aus welchen Gründen eine Verletzung einer Regel auftritt. Dabei muss die Darstellung möglichst übersichtlich, klar und für den Anwender verständlich erfolgen, um Fehler bei der Interpretation der Ergebnisse zu vermeiden und eine schnelle Behebung der Verletzungen zu ermöglichen.

Diese Diplomarbeit untersucht verschiedene Einsatzszenarien für Integritätsregeln im Prozessmanagement und leitet daraus Anforderungen an die Darstellung der Auswertungsergebnisse ab. Es wird eine Ergebnisstruktur eingeführt, die die Ergebnisse verschiedener Auswertungsalgorithmen flexibel aufnehmen kann. Anschließend wird das Konzept einer Nutzerschnittstelle vorgestellt, die es Anwendern erlaubt, diese Ergebnisse ihren Aufgaben und Anforderungen entsprechend zu analysieren, ihre Implikationen zu verstehen und Handlungsbedarf zu erkennen. Dabei liegt der Fokus dieser Arbeit auf der in die Prozessdarstellung integrierten Visualisierung der Ergebnisse. Hierdurch wird erkennbar, bei welchen möglichen Verläufen der Prozessausführung es zu Verletzungen der Integritätsregeln kommt und wie genau diese entstehen. Es werden verschiedene Anpassungsmöglichkeiten vorgestellt, um auch in umfangreichen Prozessen eine übersichtliche Darstellung zu ermöglichen. Die konzipierte Schnittstelle wurde in eine funktionsfähige Demonstrator-Anwendung umgesetzt, deren Aufbau ebenfalls erläutert wird.

Die in dieser Arbeit vorgestellten Konzepte leisten einen Beitrag, um den Einsatz von Integritätsregeln im BPM zu vereinfachen und neue Anwendungsmöglichkeiten zu eröffnen.

Inhaltsverzeichnis

Zusammenfassung **iii**

1 Einleitung **1**

1.1 Grundbegriffe . . . 2
1.2 Aufgabenstellung . . . 4
1.3 Verwandte Arbeiten . . . 6
1.4 Aufbau dieser Arbeit . . . 7

2 Grundlagen **9**

2.1 Auswertungssubjekt . . . 10
2.1.1 ADEPT-Prozessbeschreibungssprache . . . 10
2.1.2 Ausführungsspuren . . . 15
2.2 Aktivitätentypen . . . 18
2.3 SeaFlows-Integritätsregeln . . . 19
2.3.1 Regelgraphen . . . 20
2.3.2 Prädikatenlogische Formalisierung . . . 24
2.3.3 Erweiterung der prädikatenlogischen Regeln . . . 27
2.3.4 Interpretation der Integritätsregeln . . . 29
2.4 Auswertungsalgorithmen . . . 31
2.4.1 Graphbasierte Untersuchung . . . 31
2.4.2 Modellbasierte Untersuchung . . . 32
2.4.3 Weitere Verfahren . . . 32
2.5 Systembestandteile . . . 33

3 Anforderungsanalyse **35**

3.1 Nutzergruppen . . . 36
3.1.1 Aufgaben . . . 37
3.1.2 Spezifische Anforderungen . . . 40

3.2 Anwendungsszenarien . . . 41
3.2.1 Einsatzzwecke . . . 42
3.2.2 Auswertungszeitpunkte . . . 46
3.3 Anforderungen an das System . . . 49
3.3.1 Grundlegendes . . . 49
3.3.2 Regelklassen . . . 49
3.3.3 Ergebnisauswertung . . . 51
3.3.4 Ergebnisdarstellung . . . 53

4 Ergebnisstruktur **57**
4.1 Anforderungen . . . 58
4.2 Voraussetzungen . . . 59
4.3 Aufbau . . . 60
4.4 Gesamterfülltheit . . . 62
4.5 Regelverletzungen . . . 63
4.5.1 Herleitung . . . 64
4.5.2 Definition . . . 67
4.6 Ergebnisunschärfe . . . 72
4.7 Ergebnisbeispiele . . . 75

5 Nutzerschnittstelle **77**
5.1 Ziele . . . 78
5.2 Grundlagen . . . 79
5.3 Endbenutzeransicht . . . 79
5.4 Gesamtlistenansicht . . . 80
5.5 Regelansicht . . . 83
5.6 Prozessansicht . . . 84
5.6.1 Herausforderungen . . . 85
5.6.2 Übersichtsliste . . . 86
5.6.3 Textuelle Regelverletzungen . . . 87
5.6.4 Anpassung der Prozessdarstellung . . . 89
5.6.5 Spurlinien . . . 92
5.6.6 Gesamterfülltheitsmodus . . . 97
5.6.7 Regelverletzungsmodus . . . 101

6 Praktische Umsetzung **105**
6.1 Grundlegendes . . . 106
6.2 Aktivitätentypen . . . 107
6.3 Integritätsregeln . . . 107
6.4 Prozessmodelle . . . 108
6.5 Ausführungsspuren . . . 108
6.5.1 Definierte Zweigeinschränkungen . . . 110
6.5.2 Unscharfe Zweigeinschränkungen . . . 112
6.5.3 Knotenausführungen . . . 113
6.6 Ergebnisstruktur . . . 113
6.6.1 Gesamterfülltheit . . . 113
6.6.2 Regelverletzungen . . . 114
6.7 Nutzerschnittstelle . . . 115
6.7.1 Prozessdarstellung . . . 116
6.7.2 Prozessmodell . . . 119
6.7.3 Auswertungsergebnis . . . 124
6.7.4 Übersichtsliste . . . 129
6.8 Auswertungsalgorithmus . . . 129
6.8.1 Anbindung . . . 130
6.8.2 Beispielalgorithmus . . . 131

7 Zusammenfassung und Ausblick **135**
7.1 Erweiterungsmöglichkeiten . . . 136
7.2 Schlusswort . . . 139

Literaturverzeichnis **141**

Abbildungsverzeichnis **146**

Definitionsverzeichnis **147**

Beispielverzeichnis **149**

Index **154**

1 Einleitung

In der Vergangenheit wurden Prozesse und Abläufe in der Geschäftswelt üblicherweise entweder manuell koordiniert, was die Kontrolle ihrer Umsetzung erschwerte, oder waren (vollständig oder teilweise) in den verwendeten Software-Systemen fest implementiert, was Anpassungen der Prozesse aufwändig machte. Daher haben sich in den letzten Jahren Prozessmanagement-Systeme etabliert. Diese erlauben es, Prozesse explizit zu beschreiben und auf Grundlage dieser Beschreibung ausführen zu lassen. So erhöht sich die Flexibilität im Vergleich zu fest implementierten Prozessen, da bei Änderungen im Betriebsablauf oder Geschäftsumfeld Prozessbeschreibungen und – bei der Verwendung moderner, sog. adaptiver Prozessmanagement-Systeme – auch einzelne Prozessausführungen leicht an die neuen Anforderungen angepasst werden können.

Durch die explizite Beschreibung und die Ablaufkontrolle, die solche Systeme bieten, eröffnen sich auch neue Möglichkeiten, die Einhaltung von Vorgaben und Regeln, die sich z. B. aus Verträgen, Spezifikationen oder Vorschriften ergeben, in einem Prozess sicherzustellen und so seine Korrektheit zu überprüfen. Durch die erhöhte Flexibilität, die adaptive Prozessmanagement-Systeme bieten, gewinnt die Überprüfung der Einhaltung von Regeln im Prozess auch an Notwendigkeit, damit durch Prozessänderungen nicht versehentlich Fehler entstehen, die bei der ursprünglichen Beschreibung des Prozesses vermieden wurden. Daher ist es sinnvoll, die Korrektheit der Prozesse möglichst durchgehend zu überprüfen, von Beginn des Prozessentwurfs über die Ausführung bis zur Optimierung und Anpassung. So können Fehler in der Prozessspezifikation bereits im Vorfeld gefunden werden, aber auch bei der Ausführung oder Änderungen am Prozess entstehende Probleme werden zum jeweiligen Zeitpunkt erkannt.

Insbesondere bei der Überprüfung eines Prozesses in der Entwurfsphase genügt es nicht, anzugeben, ob eine Regel durch diesen Prozess erfüllt oder verletzt wird – schließlich ist es häufig der Fall, dass die Erfüllung vom Verlauf der Prozessausführung abhängt, beispielsweise wenn mehrere alternative Ausführungsmöglichkeiten existieren. Außerdem ist bei umfangreichen Prozessen und Regeln möglicherweise nicht auf den ersten Blick ersicht-

lich, aus welchem Grund eine Regel verletzt ist. Daher ist eine ausführlichere Darstellung des Ergebnisses notwendig, aus der – wenn möglich – erkennbar ist, in welchen Fällen und aus welchen Gründen eine Verletzung einer Regel auftritt. Dabei muss die Darstellung möglichst übersichtlich, klar und für den Anwender verständlich erfolgen, um Fehler bei der Interpretation des Ergebnisses zu vermeiden und eine schnelle Behebung der Verletzung zu ermöglichen.

Üblicherweise stellen die Spezifikation und Überprüfung von Regeln zusätzliche Arbeitsschritte bei der Entwicklung eines Prozesses dar. Daher muss der hierfür benötigte Aufwand möglichst gering gehalten werden, wozu eine klare und an die Aufgaben und Ziele der Anwender angepasste Darstellung einen wichtigen Beitrag leistet.

Die Entwicklung einer Nutzerschnittstelle, die eine solche übersichtliche, aufgabenangemessene Darstellung der Auswertungsergebnisse ermöglicht, ist Aufgabe dieser Diplomarbeit.

1.1 Grundbegriffe

In diesem Abschnitt werden einige Begriffe und Konzepte, die in dieser Arbeit verwendet werden, kurz vorgestellt.

Die rechnerunterstützte Verwaltung und Kontrolle von Geschäftsprozessen wird als **Prozessmanagement** oder *Business Process Management* (BPM) bezeichnet. Als Grundlage der Ausführung von Prozessen dienen dabei üblicherweise Prozessbeschreibungen in Graphform, die *Prozessvorlagen* oder *Prozessmodelle* genannt werden. Wird ein Prozess auf Basis einer solchen Vorlage ausgeführt, wird diese einzelne Ausführung als *Prozessinstanz* des Modells bezeichnet. Die Ausführung von Prozessinstanzen wird üblicherweise protokolliert, die entstehenden Protokolle werden als *Ausführungs-Logs* bezeichnet.

Einige moderne BPM-Systeme, etwa *ADEPT* [26], bieten die Möglichkeit, einzelne Prozessinstanzen während ihrer Ausführung zu bearbeiten und an Sonderfälle oder sonstige im Vorfeld nicht eingeplante Gegebenheiten anzupassen. Solche Änderungen an Prozessinstanzen werden als *Ad-hoc-Änderungen* bezeichnet. Ebenfalls kann unterstützt werden, Änderungen an Prozessmodellen, die durchgeführt werden, während Instanzen der Modelle ausgeführt werden, auf diese Instanzen zu übertragen. Dies wird als *Propagierung* der Änderungen auf die jeweiligen Instanzen bzw. als *Schemaevolution* bezeichnet.

Die Regeln, deren Überprüfung in Prozessmodellen und -instanzen eine Grundlage dieser Arbeit darstellt, werden im Folgenden als (semantische) **Integritätsregeln** bezeichnet. In der Geschäftswelt werden solche Regeln, die beispielsweise betriebsinterne Vorschriften, zu erfüllende Normen und Vorgaben oder vertragliche Regelungen beschreiben, häufig unter dem Begriff *Business Rules* zusammengefasst.

Die Information, ob bzw. in welchem Ausmaß in einem Prozessmodell oder einer Prozessinstanz eine solche Regel erfüllt ist, wird in dieser Arbeit als *Erfülltheit* der Regel im Modell/der Instanz bzw. als *Compliance* des Modells/der Instanz zu der Regel bezeichnet.

Der **Lebenszyklus** eines Geschäftsprozesses (*Business Process Lifecycle*) wird nach [32] in vier Phasen eingeteilt: Die Phase *Process Design*, in der ein Prozess neu entwickelt oder ein bestehender Prozess angepasst wird, die Phase *System Configuration*, in der der Prozess als Prozessmodell im BPM-System umgesetzt wird, die Phase *Process Enactment*, in der der Prozess ausgeführt wird, und die Phase *Diagnosis*, in der auf Grundlage der erfolgten Ausführungen Verbesserungsmöglichkeiten untersucht werden. Da die Prüfung von Integritätsregeln erst erfolgen kann, wenn die Modellierung von Prozessen im BPM-System begonnen hat, betrachten wir anstelle der ersten beiden Phasen eine neue Phase *Prozessentwicklung*. Falls die konzeptionelle Entwicklung des Prozesses bereits mit Unterstützung des BPM-Systems erfolgt, entspricht dies der ersten Phase des ursprünglichen Lebenszyklus, wobei die zweite Phase dann entfällt; Andernfalls erfolgt die Entwicklung des Prozessmodells auf Grundlage einer bereits vorliegenden Prozessspezifikation, dies entspricht dann der zweiten Phase im Business Process Lifecycle. Zudem unterscheiden wir bei der Prozessentwicklung, ob die Modellierung eines neuen Prozesses erfolgt oder ein bestehender, bereits modellierter Prozess überarbeitet wird (vgl. Abb. 1.1).

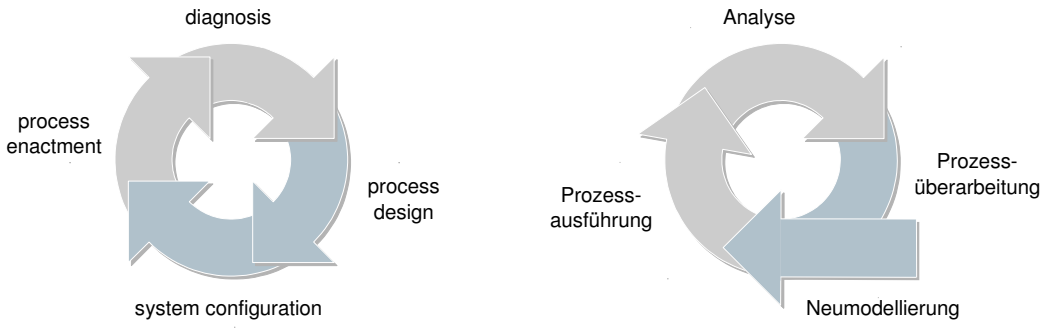


Abbildung 1.1: Business Process Lifecycle nach [32] (links) und modifizierte Variante (rechts – dort nur im BPM-System ablaufende Vorgänge)

1.2 Aufgabenstellung

Aufgabe dieser Diplomarbeit ist die Untersuchung und Entwicklung benutzerorientierter Konzepte zur Repräsentation und Visualisierung der Ergebnisse der Prüfung von Geschäftsprozessen auf die Erfülltheit von Integritätsregeln. Im Speziellen werden folgende Themenbereiche untersucht (die Auflistung wurde direkt der Aufgabenstellung entnommen):

- Use Cases für die Prüfung der Compliance mit Integritätsregeln und daraus resultierende Anforderungen an die Visualisierung und Benutzerinteraktion
- Analyse möglicher Quellen für Incompliance und Konzeption einer adäquaten Ergebnisstruktur, die eine benutzerorientierte Auswertung und Darstellung der Prüfergebnisse erlaubt
- Anforderungen an die Prüfkomponente und Untersuchung der Anbindung der Ergebnisstruktur an mögliche Auswertungsalgorithmen bzw. deren Ergebnisse
- Prototypische Umsetzung der Konzepte durch die Implementierung der Ergebnisstruktur in einem Demonstrator

Als Grundlage hierfür wird die Prozessbeschreibungssprache *ADEPT* sowie der Regelmodellierungsansatz des *SeaFlows*-Projekts verwendet. Ziel dieser Arbeit ist es, ein tieferes Verständnis für die an die Visualisierung gestellten Anforderungen und sich hieraus ergebende Folgen für die Prüfung der Prozessintegrität zu schaffen und basierend auf diesen Anforderungen konkrete Konzepte zur benutzerorientierten Ergebnisdarstellung anzubieten.

Insgesamt soll diese Arbeit einen Beitrag zur Verbesserung der Benutzerschnittstelle der Komponenten zur Regelmodellierung und -prüfung leisten und so die umfassende Unterstützung von Integritätsregeln in adaptiven BPM-Systemen weiter voran zu bringen.

In Abbildung 1.2 ist die Aufgabenstellung graphisch dargestellt: Als Grundlagen dienen existente Konzepte zur Beschreibung von *Integritätsregeln* und Geschäftsprozessen (letztere bilden die Basis des *Auswertungssubjekts*) sowie *Auswertungsalgorithmen* zur Prüfung der Regeln. Diese Arbeit untersucht mittels Use Cases die *Anforderungen* der Anwender und stellt eine *Ergebnisstruktur* sowie Konzepte für ihre Repräsentation an der *Nutzerschnittstelle* vor, die schließlich im *Demonstrator* prototypisch umgesetzt werden.

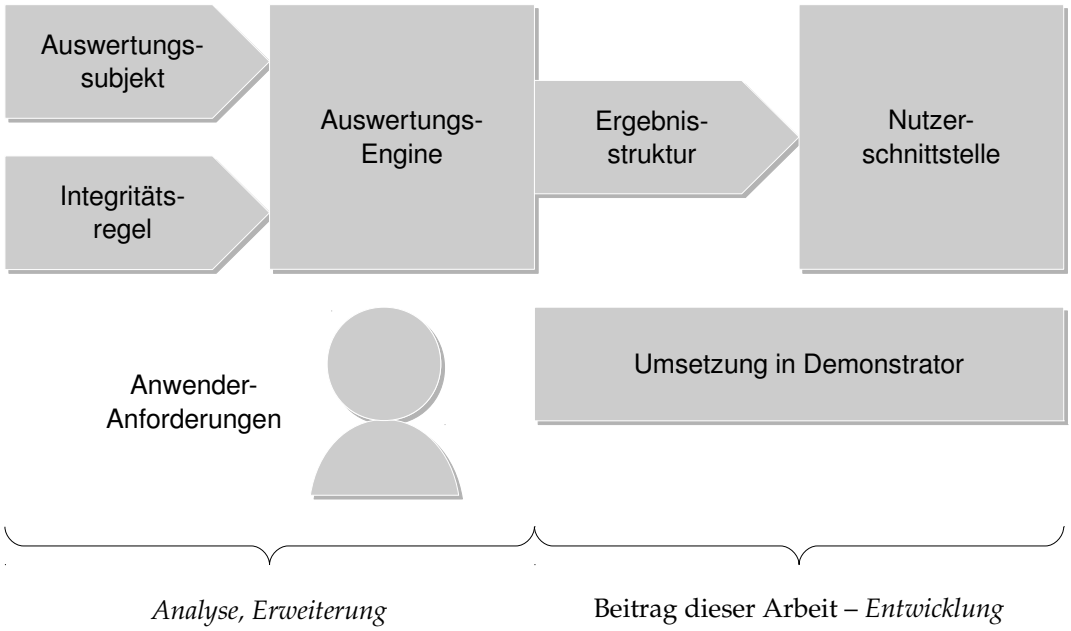


Abbildung 1.2: Schematische Darstellung der Aufgabenstellung

1.3 Verwandte Arbeiten

Eine grundlegende Maßnahme zur Sicherstellung von Korrektheit in Prozessen stellt die Wahrung der *formalen* Integrität einer Prozessbeschreibung in Bezug auf Eigenschaften wie Terminierung der Ausführung oder Erreichbarkeit aller Zustände dar, was entweder durch nachträgliche Überprüfung erfolgen kann, wie in [31], oder durch Verhinderung der Erzeugung inkorrekter Modelle durch die Struktur der Prozessbeschreibung (*Correctness by design*), was z. B. im ADEPT-System vorgesehen ist [8]. Bei ADEPT wird zusätzlich die Korrektheit des Datenflusses sichergestellt, indem überprüft wird, ob in einem Prozessmodell Datenobjekte gelesen werden, bevor sie geschrieben wurden [26].

Die Überprüfung der *semantischen* Korrektheit von Prozessen, welche die Grundlage dieser Arbeit darstellt, bildet die nächste Stufe der Integritätssicherung. Ähnliche Bestrebungen finden sich bereits in anderen Gebieten der Informatik, etwa bei der Sicherstellung der korrekten Funktionsweise von Software durch Programmbeweise oder der Überprüfung der Konsistenz von in UML beschriebenen Softwarespezifikationen mit OCL-Regeln [4]. Im Prozessmanagement gibt es Ansätze zur Überprüfung spezifischer Typen von Integritätsbedingungen, von einfachen zeitlichen Einschränkungen [22] bis zu komplexen Regelsätzen wie geschäftlichen Verträgen [16].

Ansätze für allgemeinere Compliance-Überprüfungen, verbunden mit einer graphischen Darstellung der Regeln, finden sich in [18]. Eine Einordnung verschiedener Arten von Frameworks für die Unterstützung von Compliance im Prozessmanagement und einen Überblick über die Anforderungen an diese liefert [12]. Einen umfassenden Ansatz zur Überprüfung von Integritätsregeln über Prozessen in den verschiedenen Phasen des Prozesslebenszyklus bieten die im *SeaFlows*-Projekt entwickelten Ergebnisse [19, 20, 21], wo durch die Verwendung einer verständlichen Pattern-basierten Regelbeschreibung insbesondere auch ein Fokus auf Benutzerfreundlichkeit gelegt wird. Das in diesen Arbeiten eingeführte Regelmodell bildet eine wichtige Grundlage für diese Diplomarbeit.

Für die formale Beschreibung von Integritätsregeln gibt es verschiedene Ansätze. Neben der in dieser Arbeit verwendeten Prädikatenlogik erster Stufe [29] können auch andere Logiken wie temporale Logik (etwa im *Model Checking* [6]) oder speziell an das Anwendungsgebiet angepasste Logiken wie *Formal Contract Logic* (FCL) [15] eingesetzt werden.

Die Visualisierung der Auswertungsergebnisse einer Compliance-Überprüfung stellt den Hauptbestandteil dieser Arbeit dar. Eine Herausforderung ist dabei, dem Anwender zu

ermöglichen, auch bei umfangreichen Prozessmodellen die Übersicht zu behalten. Ähnliche Fragestellungen finden sich auch in anderen Bereichen, in denen umfangreiche Objekte dargestellt werden müssen, etwa bei der Visualisierung von Programmabläufen, wofür interessante Konzepte entwickelt wurden, welche z. B. Abstraktion [5] oder »semantischen Zoom« [23] verwenden. Ähnliches wird in dieser Arbeit durch die – manuelle bzw. automatische – Zusammenfassung von Prozessteilen erreicht, die eine übersichtliche Darstellung ermöglicht und dem Anwender erlaubt, manuell bestimmte Elemente genauer zu untersuchen. Auch von Nutzen bei der Visualisierung komplexer Daten ist die Verwendung mehrerer verschiedener Ansichten, zwischen denen der Anwender manuell wechseln kann [10], was in dieser Arbeit ebenfalls eingesetzt wird. Ansätze für die Visualisierung der Auswertungsergebnisse von Integritätsregeln über Prozessmodellen finden sich in [2], wo ein auf Patterns basierender Ansatz verwendet wird. Die Verwendung eines Gegenbeispiels zur Angabe des Auswertungsergebnisses bei verletzten Regeln wird in [18] angesprochen. Ähnliche Prinzipien werden in dieser Arbeit bei der Darstellung von Regelverletzungen durch Regelgraph-Fragmente eingesetzt.

Die in dieser Arbeit entwickelte Visualisierung hat zum Ziel, Anwendern zu ermöglichen, durch Verletzung von Integritätsregeln entstehende Fehler in Prozessen (wenn möglich bereits vor, aber auch während der Ausführung) zu erkennen und verstehen und sie so zu befähigen, diese Fehler manuell zu beheben. Einen anderen Ansatz stellen Versuche dar, Regelverletzungen zur Laufzeit automatisch zu beheben [13, 3], was bislang jedoch hauptsächlich für einfache Integritätsregeln möglich ist oder wenn eindeutige Behebungsmöglichkeiten existieren.

1.4 Aufbau dieser Arbeit

Der verbleibende Teil dieser Arbeit ist folgendermaßen aufgebaut:

In *Kapitel 2* werden zunächst bestehende Konzepte vorgestellt, die die **Grundlage** für diese Arbeit bilden. Dazu gehört eine einheitliche Struktur für verschiedene Auswertungssubjekte, über denen die Integritätsregeln überprüft werden können, die Beschreibungssprache der Prozessmodelle, der Aufbau der auszuwertenden Integritätsregeln, mögliche Auswertungsalgorithmen sowie im System benötigte Software-Komponenten. Dabei werden auch Einschränkungen, Erweiterungen und Anpassungen, welche für die Verwendung der Konzepte in dieser Arbeit gelten, sowie die verwendeten Formalisierungen beschrieben.

Anschließend wird in *Kapitel 3* eine Analyse der unterschiedlichen Nutzergruppen eines BPM-Systems mit Regelunterstützung durchgeführt und es werden verschiedene Anwendungsszenarien für die Nutzung eines solchen Systems vorgestellt. Daraus werden **Anforderungen** abgeleitet, die das System – insbesondere die Nutzerschnittstelle zur Analyse der Auswertungsergebnisse – erfüllen muss, um die Anwender bei der Durchführung ihrer Aufgaben zu unterstützen.

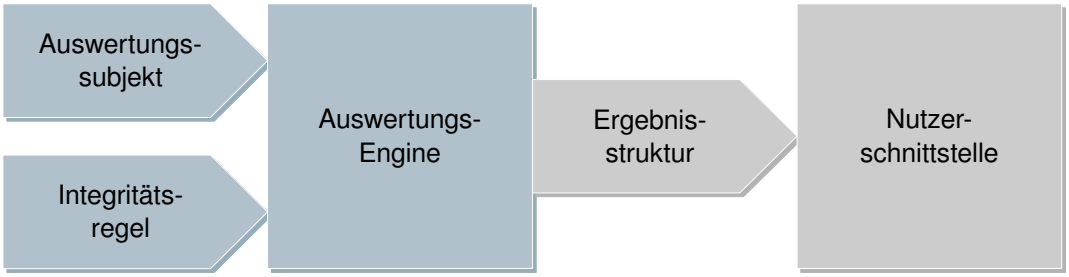
In *Kapitel 4* wird eine allgemeine **Ergebnisstruktur** vorgestellt, welche die Ergebnisse verschiedener Auswertungsalgorithmen flexibel aufnehmen kann und die Grundlage für eine anwenderfreundliche Darstellung bildet.

Kapitel 5 beschreibt eine **Nutzerschnittstelle**, mit der die in der Ergebnisstruktur vorliegenden Auswertungsergebnisse den Aufgaben des Anwenders entsprechend dargestellt werden können und die eine interaktive Anpassung zulässt, um auch bei umfangreichen Ergebnissen eine übersichtliche Darstellung zu ermöglichen.

In *Kapitel 6* wird anschließend die **Umsetzung** der Ergebnisse dieser Arbeit in eine funktionsfähige Demonstrator-Anwendung beschrieben. Insbesondere behandelt dieser Teil der Arbeit auch, wie die in Kapitel 4 konzeptionell beschriebene Ergebnisstruktur in tatsächliche Datenstrukturen umgesetzt werden kann, und stellt einen einfachen Auswertungsalgorithmus vor.

Abschließend werden in *Kapitel 7* die Ergebnisse der Arbeit kurz zusammengefasst und mögliche Erweiterungen vorgestellt.

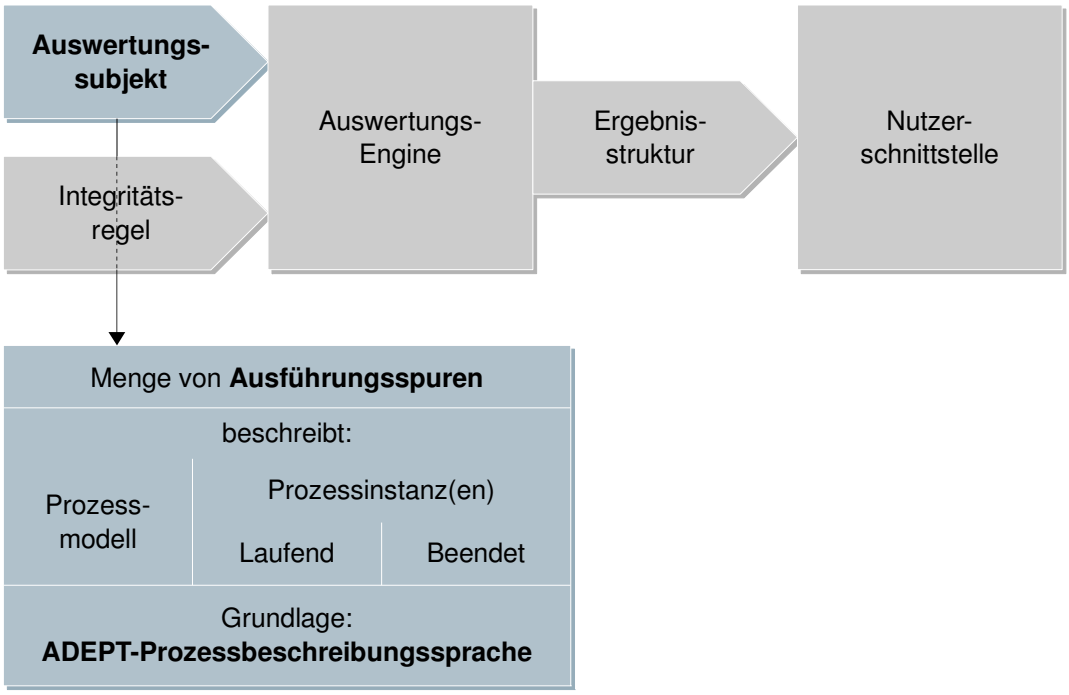
2 Grundlagen



Für die Darstellung der Erfülltheit von Integritätsregeln über Geschäftsprozessen zur interaktiven Analyse durch den Anwender werden einige grundlegende Elemente benötigt. Hierfür werden in dieser Arbeit verschiedene existente Konzepte eingesetzt, die teilweise angepasst und erweitert werden.

In Abschnitt 2.1 wird die in dieser Arbeit verwendete Prozessbeschreibungssprache sowie das Konzept der Ausführungsspuren vorgestellt, die als Auswertungssubjekt die Grundlage für die Regelauswertung bilden. Das Bindeglied zwischen Auswertungssubjekt und Integritätsregeln stellen Aktivitätentypen dar, die in Abschnitt 2.2 erläutert werden. In Abschnitt 2.3 wird das SeaFlows-Regelmodell eingeführt, das in in dieser Arbeit zur Beschreibung der Integritätsregeln dient. Anschließend werden in Abschnitt 2.4 verschiedene Algorithmen angesprochen, die zur Auswertung der Regeln eingesetzt werden können. In Abschnitt 2.5 werden schließlich Systemkomponenten vorgestellt, die zum praktischen Einsatz von Integritätsregeln in BPM-Systemen benötigt werden.

2.1 Auswertungssubjekt



Damit Prozesse in BPM-Systemen verwendet werden können, müssen sie zunächst in einer wohldefinierten Sprache beschrieben werden. Eine solche *Prozessbeschreibungssprache* alleine genügt jedoch noch nicht zur durchgängigen Auswertung von Integritätsregeln über den gesamten Lebenszyklus hinweg – es wird zusätzlich eine Struktur benötigt, die die dynamischen Eigenschaften einzelner Ausführungen dieser Prozesse abbilden kann. Diese Struktur bildet das **Auswertungssubjekt**, über dem die Auswertung der Regeln vor, während und nach der Prozessausführung erfolgt.

In Abschnitt 2.1.1 wird die Prozessbeschreibungssprache des ADEPT-Projekts vorgestellt, die in dieser Arbeit zur Beschreibung der Prozesse eingesetzt wird. Die bei der Ausführung von in dieser Sprache modellierten Prozessen entstehenden Ausführungsspuren beschreiben die dynamischen Ausführungseigenschaften und werden in Abschnitt 2.1.2 definiert.

2.1.1 ADEPT-Prozessbeschreibungssprache

Als Beschreibungssprache für Prozesse verwenden wir in dieser Arbeit die **Well-Structured-Marking-Netze** (WSM-Netze) aus dem ADEPT-Projekt, die u. a. in [26], [27] und [28]

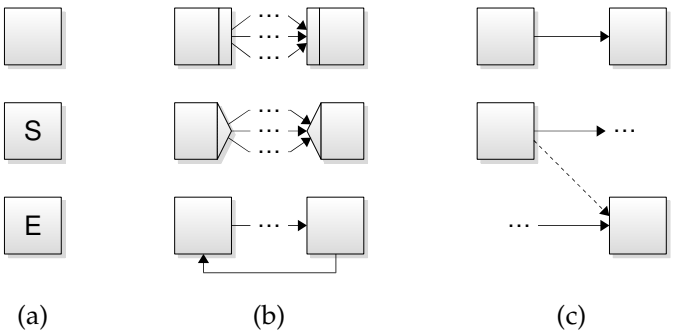


Abbildung 2.1: Knoten- und Kantentypen in ADEPT (jeweils von oben nach unten):

(a) Einfacher Knoten, Startknoten, Endknoten

(b) AND-Split/-Join, XOR-Split/-Join, Schleifenstart/-ende

(c) Kontrollflusskante (durchgezogen), Synchronisationskante (gestrichelt)

vorgestellt werden. Mit Hilfe dieser Beschreibungssprache werden der Kontroll- und Datenfluss eines Prozesses in Form eines Prozessmodells beschrieben. Grundlage eines solchen Modells ist ein gerichteter Graph, dessen Knoten einzelne Schritte im Prozess beschreiben. Die Verbindung der Knoten durch sog. **Kontrollflusskanten** gibt den möglichen Kontrollfluss an, indem beschrieben wird, welche Knoten in welcher Kombination bzw. Reihenfolge ausgeführt werden können. Bei der Ausführung eines Knotens wird die ihm zugeordnete **Aktivität** (die ggf. auch eine *Nullaktivität* sein kann, die keine Aktion durchführt) ausgeführt. Die Ergebnisse dieser Arbeit sind unabhängig davon, in welcher Weise solche Aktivitäten genau ausgeführt werden bzw. für welche Aktionen sie stehen. Genauere Ausführungen hierzu sind u. a. in [26] zu finden.

Es existieren verschiedene Typen von Knoten. **Normale Knoten** haben jeweils eine ein- und eine ausgehende Kontrollflusskante. Zusätzlich gibt es genau einen **Startknoten**, der nur eine ausgehende Kante, und einen **Endknoten**, der nur eine eingehende Kante besitzt. Weitere Knotentypen sind **Split-** und **Joinknoten** sowie **Schleifenstart-** und **-endknoten**. Bei den Split- und Joinknoten wird zudem zwischen AND- und XOR-Splits/-Joins unterschieden. Ein Splitknoten hat eine eingehende und mindestens eine ausgehende Kante, ein Joinknoten mindestens eine eingehende und genau eine ausgehende Kante. Ein Schleifenstartknoten hat zwei eingehende und eine ausgehende, ein Schleifenendknoten zwei ausgehende und eine eingehende Kante, wobei eine dieser Kanten den Schleifenendknoten

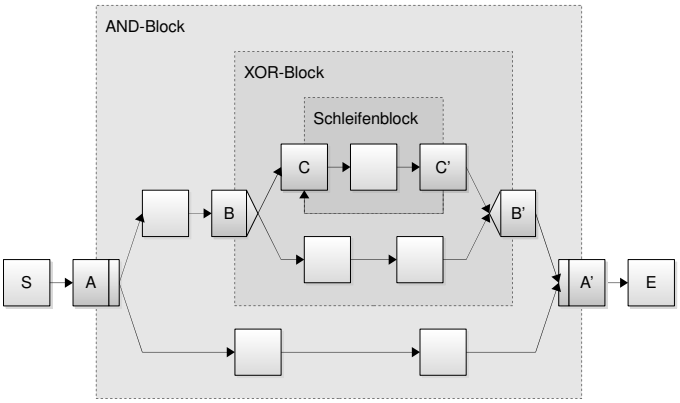


Abbildung 2.2: Blockstrukturierung in einem ADEPT-Prozessmodell

direkt mit dem Startknoten verbindet – diese Kante wird als **Rücksprungkante** bezeichnet. Abb. 2.1 a), b) zeigt die graphischen Darstellungen der verschiedenen Knotentypen.

Diese speziellen Knoten sind in Form von verschachtelten Blöcken organisiert. Je ein Split- und Joinknoten bzw. Schleifenstart- und -endknoten umschließen einen Block. Alle Knoten, die sich dazwischen befinden, liegen innerhalb des Blocks. Für diese Arbeit gilt die Einschränkung, dass ein Block, der mit einem AND-Split beginnt, auch mit einem AND-Join enden muss – analog muss ein Block, der mit einem XOR-Split beginnt, auch mit einem XOR-Join enden (vgl. auch [9], wo diese Einschränkung ebenfalls gilt). Die Blöcke sind hierarchisch verschachtelt und dürfen sich nicht überschneiden, es darf also nicht vorkommen, dass ein Start- bzw. Split-Knoten innerhalb eines Blocks liegt, der zugehörige End- bzw. Join-Knoten jedoch außerhalb (oder umgekehrt).

Beispiel 2.1. *Abb. 2.2 zeigt ein beispielhaftes ADEPT-Prozessmodell mit Hervorhebung der Blockstruktur. Knoten A stellt einen AND-Split dar, A' den zugehörigen Join-Knoten; bei Knoten B und B' handelt es sich um XOR-Split- bzw. Join-Knoten. Knoten C und C' stellen Schleifenstart- und -endknoten dar. Knoten S und E sind die Start- bzw. Endknoten des Prozessmodells.*

Blöcke werden, entsprechend den sie umschließenden Knoten, als XOR-, AND- oder Schleifenblöcke bezeichnet. Bei XOR- und AND-Blöcken gilt: Jeder Weg über Kontrollflusskanten, der von einem Nachfolger des Split-Knotens zu einem Vorgänger des Join-Knotens führt, wird als Zweig dieses Blocks bezeichnet. Zusätzlich kann ein XOR-Block einen leeren Zweig enthalten, also eine direkte Kante vom Split- zum Join-Knoten.

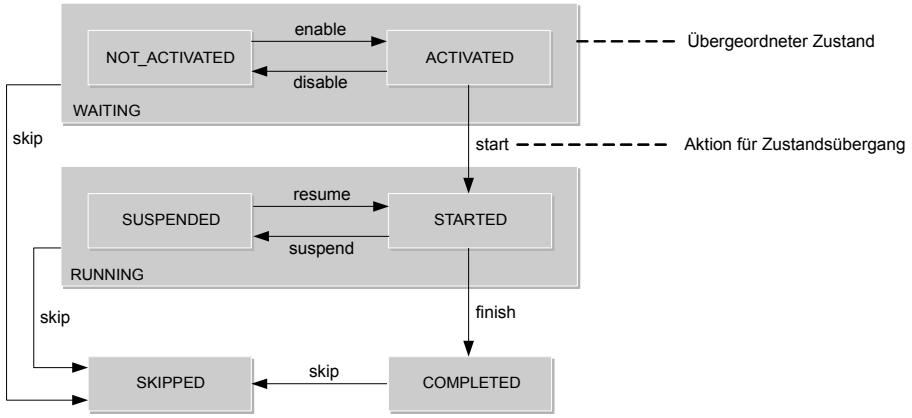


Abbildung 2.3: Das Zustandsmodell eines Knotens in ADEPT (aus [28], S. 61)

Zusätzlich zu den Kontrollflusskanten existieren sog. **Synchronisationskanten** (*Sync-Kanten*). Diese gerichteten Kanten können zwei beliebige Knoten verbinden, die auf verschiedenen Zweigen eines AND-Blocks liegen. Dabei können die Knoten auf diesen Zweigen durchaus innerhalb weiter verschachtelter XOR- oder AND-Blöcke liegen. Allerdings sind keine Sync-Kanten erlaubt, die in Schleifen hinein oder aus ihnen heraus führen. Graphisch werden Sync-Kanten als Pfeile mit gestrichelten Linien dargestellt, während Kontrollflusskanten durchgezogene Linien besitzen (vgl. Abb. 2.1 c).

Bei der **Ausführung** eines Prozesses werden die Knoten entsprechend der Spezifikation des Modells ausgeführt. Dabei durchläuft ein Knoten verschiedene Zustände. Ausgangszustand ist `NOT_ACTIVATED`. Ein Knoten kann ausgeführt werden, sobald er den Zustand `ACTIVATED` angenommen hat. Während der Ausführung durchläuft er weitere Zustände, bevor er nach Abschluss der Ausführung den Zustand `COMPLETED` annimmt (sofern bei der Ausführung kein Fehler auftritt – der Fehlerfall wird in dieser Arbeit nicht unterstützt, aber in Abschnitt 7.1 angesprochen). Das (vereinfachte) ADEPT-Zustandsmodell ist in Abb. 2.3 dargestellt.

Die Ausführung einer Instanz des durch das Modell beschriebenen Prozesses beginnt stets mit dem Startknoten, der hierzu den Zustand `ACTIVATED` annimmt. Sobald der Knoten `COMPLETED` ist, erhält der über die ausgehende Kontrollflusskante nachfolgende Knoten den Zustand `ACTIVATED` und kann somit als nächstes ausgeführt werden. Dasselbe gilt für Join-Knoten, Schleifenstartknoten und normale Knoten: Sobald solch ein Knoten den

Zustand COMPLETED erreicht, erhält sein Nachfolgeknoten den Zustand ACTIVATED, außer, wenn dieser Nachfolger ein AND-Join-Knoten ist.

Für einen XOR-Split-Knoten gilt, dass beim Erreichen des Zustands COMPLETED genau ein nachfolgender Knoten den Zustand ACTIVATED einnimmt. Die anderen Zweige werden abgewählt und alle enthaltenen Knoten in den Zustand SKIPPED versetzt.[1] Die Entscheidung für die Zweigauswahl kann auf verschiedene Weise getroffen werden, für diese Arbeit wird hiervon abstrahiert und nur das Ergebnis, die Auswahl des entsprechenden Zweiges, betrachtet.

Sobald ein AND-Split-Knoten als COMPLETED markiert wird, erhalten alle seine Nachfolgeknoten den Zustand ACTIVATED, es werden also alle Zweige des Blocks parallel ausgeführt. Ein AND-Join-Knoten erhält erst den Zustand ACTIVATED, wenn alle seine Vorgängerknoten ausgeführt wurden, also den Zustand COMPLETED erreicht haben.

Für einen Schleifenendknoten gilt: Sobald dieser COMPLETED ist, wird entweder sein Nachfolgerknoten auf ACTIVATED gesetzt (außer es ist ein AND-Join-Knoten, bei dem noch nicht alle Vorgänger abgeschlossen sind) oder die Schleife wird erneut ausgeführt. In letzterem Fall erhält der zugehörige Schleifenstartknoten erneut den Zustand ACTIVATED und alle anderen im Schleifenblock enthaltenen Knoten, inklusive des Schleifenendknotens, werden wieder in den Zustand NOT_ACTIVATED gesetzt. Jede Ausführung einer Schleife wird als *Iteration* der Schleife bezeichnet. Die Ausführung des gesamten Prozesses endet, sobald der Prozess-Endknoten den Zustand COMPLETED erreicht.

Die Synchronisationskanten schränken die Ausführungsreihenfolge der durch sie verbundenen Knoten ein. Es gilt folgende Semantik: Wenn eine Sync-Kante von a nach b existiert, darf die Ausführung von b erst begonnen werden, wenn a abgeschlossen wurde oder sichergestellt ist, dass a nicht mehr (bzw. erst wieder in einer späteren Iteration einer umgebenden Schleife) zur Ausführung kommt.

Außerdem kann in einem ADEPT-Prozessmodell mit speziellen Datenobjekten und Datenkanten der Datenfluss zwischen verschiedenen Knoten modelliert werden. Da Datenfluss in dieser Arbeit jedoch nicht berücksichtigt wird, wird hierauf nicht weiter eingegangen. Einen Einblick in mögliche Erweiterungen der Ergebnisse dieser Arbeit, unter anderem auch um die Unterstützung von Datenfluss, gibt Abschnitt 7.1.

[1] In ADEPT ist alternativ vorgesehen, dass zunächst alle Nachfolgerknoten aktiviert werden und die Auswahl für den Zweig erst erfolgt, wenn einer dieser Knoten gestartet wird – dann werden die anderen Zweige abgewählt. Für die Zwecke dieser Arbeit sind beide Möglichkeiten gleichwertig – der entscheidende Effekt ist, dass nur einer der Zweige tatsächlich ausgeführt wird.

2.1.2 Ausführungsspuren

Als Ergebnis jeder Ausführung eines Prozesses ergibt sich eine **Ausführungsspur**. Diese beschreibt die Zeitpunkte, zu denen Knoten aus dem Prozessmodell gestartet und beendet wurden. Da ein Knoten aufgrund von Schleifen ggf. mehrmals ausgeführt werden kann, wird jede einzelne Ausführung gesondert in die Spur aufgenommen.

Hierzu werden die Ausführungen annotiert – eine **Knotenausführung** ergibt sich als Tupel aus dem ausgeführten Knoten und einer Liste von Iterationsnummern, die angibt, wie oft die den Knoten umgebenden Schleifen bereits ausgeführt wurden. Der erste Eintrag der Liste beschreibt dabei, wie oft die äußerste den Knoten umgebenden Schleife bereits vollständig ausgeführt wurde, der zweite Eintrag die Anzahl der bisherigen Iterationen der zweitäußersten Schleife innerhalb der derzeitigen Iteration der äußersten Schleife, usw. Diese Angabe ist dank der expliziten Schleifenkonstrukte in den ADEPT-WSM-Netzen möglich, da hierbei wiederholte Ausführungen von Knoten stets strukturiert erfolgen.

Definition 2.1 (Knotenausführung)**.** Eine **Knotenausführung** x eines Knotens n ist definiert als ein Tupel $x = (n, it)$, mit $it = [it_1, \ldots, it_l]$. l entspricht dabei der Anzahl der n im Prozessmodell umgebenden Schleifenblöcke, $it_i \in \mathbb{N}_0 \forall i \in \{1 \ldots l\}$. Ist $l = 0$, also der Knoten in keiner Schleife enthalten, so wird $it = \epsilon$ geschrieben.

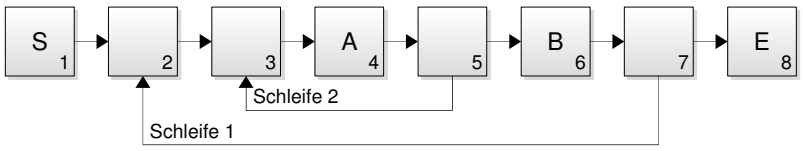


Abbildung 2.4: Prozessmodell mit verschachtelten Schleifen

Beispiel 2.2. *Betrachten wir eine beispielhafte Ausführung des Prozessmodells aus Abb. 2.4: Nach dem Start wird Knoten 1 ausgeführt. Dieser Knoten ist in keiner Schleife enthalten, daher handelt es sich um die Knotenausführung* $(1, \epsilon)$*. Anschließend wird Knoten 2 ausgeführt, welcher sich in einer Schleife befindet, welche bisher noch nie vollständig ausgeführt wurde. Daher wird diese Knotenausführung als* $(2, [0])$ *beschrieben. Knoten 3 befindet sich in Schleife 2, die in dieser Iteration von Schleife 1 noch nie ausgeführt wurde, daher handelt es sich um die Knotenausführung* $(3, [0, 0])$*. Gleiches gilt für Knoten 4 und 5. Die Schleife wird nun wiederholt, daher wird Knoten 3 erneut ausgeführt. Da Schleife 2 in dieser Iteration von Schleife 1 bereits einmal ausgeführt wurde, wird diese Ausführung des Knotens mit* $(3, [0, 1])$ *bezeichnet, analog für Knoten 4 und 5. Nun wird Schleife 2*

verlassen und Knoten 6 ausgeführt. Es ergibt sich als Knotenausführung $(6, [0])$*, analog für Knoten 7. Wird Schleife 1 nun wiederholt, wird Knoten 2 erneut ausgeführt, diesmal als Knotenausführung* $(2, [1])$*. Nun wird Knoten 3 ausgeführt. Innerhalb dieser Iteration von Schleife 1 wurde Schleife 2 noch nicht ausgeführt, daher handelt es sich um die Knotenausführung* $(3, [1, 0])$*, etc.*

Konkret wird eine Ausführungsspur in dieser Arbeit – als Erweiterung der Definition aus [19] – als chronologisch sortierte Liste von Ereignissen betrachtet, die jeweils Start bzw. Ende einer Knotenausführung angeben.

Definition 2.2 (Potentielle Ausführungsspur)**.** Für die Zwecke dieser Arbeit ist eine **Potentielle Ausführungsspur** t der Länge l über einer Menge von Knoten N definiert als eine endliche Folge $\langle e_i \rangle, i \in \{1 \dots l\}$ von Ereignissen $e_i = (nodeEx_i, type_i, time_i)$ mit $type_i \in \{\text{start}, \text{end}\}$. $time_i \in \mathbb{R}$ gibt die vergangenen Zeiteinheiten seit Beginn der Ausführung an, wobei die verwendete Zeiteinheit beliebig ist. $nodeEx_i$ ist eine Knotenausführung.

Dabei muss gelten: $time_1 = 0$, sowie $\forall e_i \in t$:

- Falls $i > 1 : time_i \geq time_{i-1}$ (Die Ereignisse stehen in chronologischer Reihenfolge).
- Falls $type_i = \text{end}$: $\exists\, ! \, e_j \in t : j < i \wedge nodeEx_j = nodeEx_i \wedge type_i = \text{start}$ (Zu jedem Endereignis existiert genau ein Startereignis derselben Knotenausführung, das vor ihm in der Liste auftritt).
- Falls $type_i = \text{start}$: $\exists\, ! \, e_j \in t : j > i \wedge nodeEx_j = nodeEx_i \wedge type_j = \text{end}$ (Zu jedem Startereignis existiert genau ein korrespondierendes Endereignis, das später auftritt).

Definition 2.3 (Ausführungsspur)**.** Sei M ein Prozessmodell, t eine potentielle Ausführungsspur über der Menge aller Knoten in M. Dann heißt t **Ausführungsspur** zu M, wenn t durch die Ausführung von M entstehen kann, d. h. wenn M so ausgeführt werden kann, dass für alle $e_i = ((node_i, it_i), type_i, time_i)$ aus t gilt: Zum Zeitpunkt $time_i$ nach Start der Ausführung von M wird die Ausführung von Knoten $node_i$

- gestartet, falls $type_i = \text{start}$.
- beendet, falls $type_i = \text{end}$.

Dabei entspricht der jeweilige Wert von it_i der Ausführungshistorie der $node_i$ umgebenden Schleifen zum entsprechenden Zeitpunkt.

$traces(M)$ gibt die Menge aller möglichen Ausführungsspuren zu M an.

Über der Menge aller Ausführungsspuren sind Funktionen definiert, deren Werte sich für eine Ausführungsspur $t = \langle e_i \rangle$ nach obigem Aufbau folgendermaßen ergeben:

- $nodeExs(t) = \{x \mid \exists e_i \in t : nodeEx_i = x\}$
- $nodes(t) = \{n \in N \mid \exists it : (n, it) \in nodeExs(t)\}$
- $nodeExStartPos(t, n) = i$, wenn $nodeEx_i = n, type_i =$ start $(i \in \{1 \dots l\})$.
- $nodeExEndPos(t, n) = i$, wenn $nodeEx_i = n, type_i =$ end $(i \in \{1 \dots l\})$.
- $nodeExStartTime(t, n) = time_i$, wenn $nodeEx_i = n, type_i =$ start $(i \in \{1 \dots l\})$.
- $nodeExEndTime(t, n) = time_i$, wenn $nodeEx_i = n, type_i =$ end $(i \in \{1 \dots l\})$.

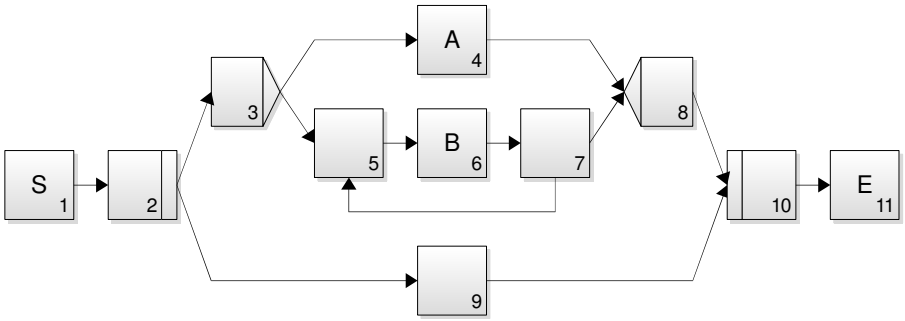


Abbildung 2.5: Beispiel-Prozessmodell

Beispiel 2.3. *Abb. 2.5 zeigt ein einfaches ADEPT-Prozessmodell. Im Folgenden zwei beispielhafte Ausführungsspuren zu diesem Modell:*

- $\Big\langle ((1,\epsilon), start, 0); ((1,\epsilon), end, 1); ((2,\epsilon), start, 1); ((2,\epsilon), end, 1); ((3,\epsilon), start, 3);$
  $((3,\epsilon), end, 3); ((9,\epsilon), start, 4); ((4,\epsilon), start, 5); ((9,\epsilon), end, 8); ((4,\epsilon), end, 14);$
  $((8,\epsilon), start, 15); ((8,\epsilon), end, 15); ((10,\epsilon), start, 15); ((10,\epsilon), end, 17); ((11,\epsilon), start, 18);$
  $((11,\epsilon), end, 19)\Big\rangle$
- $\Big\langle ((1,\epsilon), start, 0); ((1,\epsilon), end, 0); ((2,\epsilon), start, 2); ((2,\epsilon), end, 3); ((3,\epsilon), start, 4);$
  $((3,\epsilon), end, 4); ((5,[0]), start, 4); ((5,[0])), end, 5); ((6,[0]), start, 6); ((9,\epsilon), start, 7);$
  $((6,[0]), end, 9); ((7,[0]), start, 10); ((7,[0]), end, 10); ((5,[1]), start, 11);$
  $((5,[1]), end, 11); ((6,[1]), start, 13); ((9,\epsilon), end, 14); ((6,[1]), end, 15);$
  $((7,[1]), start, 15); ((7,([1]), end, 17); ((8,\epsilon), start, 17); ((8,\epsilon), end, 17);$
  $((10,\epsilon), start, 19); ((10,\epsilon), end, 22); ((11,\epsilon), start, 22); ((11,\epsilon), end, 23)\Big\rangle$

2.2 Aktivitätentypen

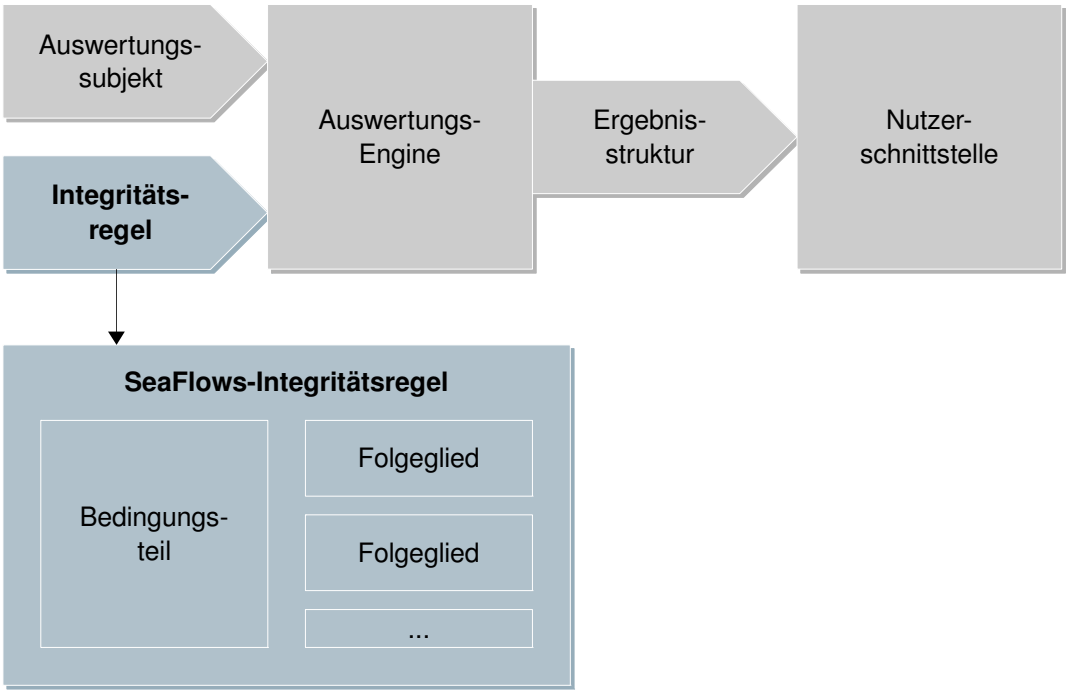
Wie bereits in Abschnitt 2.1.1 erwähnt, wird jedem Knoten in einem Prozessmodell eine **Aktivität** zugeordnet [26]. Für die Auswertung von SeaFlows-Integritätsregeln werden diese Aktivitäten in ein System von **Aktivitätentypen** eingeordnet, die als Verbindung zwischen den spezifischen Aktivitäten eines Prozessmodells und den abstrakten Aktivitäten einer Integritätsregel dienen [20]. Ein Aktivitätentyp kann ähnlich einem Interface in einer objektorientierten Programmiersprache wie *Java* [14] betrachtet werden. Wie ein *Marker-Interface* in Java definiert ein Aktivitätentyp keinerlei Programmlogik oder Funktionalität, sondern stellt lediglich eine Klassifikation dar.

Für diese Arbeit gehen wir davon aus, dass jeder Aktivität in einem Prozessmodell ein solcher Aktivitätentyp zugeordnet ist, zudem können beliebige weitere (abstrakte) Aktivitätentypen definiert werden. Jeder Aktivitätentyp kann Subtyp eines oder mehrerer abstrakter Aktivitätentypen sein, die dann seine Obertypen darstellen (sog. *Mehrfachvererbung*). Dabei sind in der Struktur der Obertypen keine Zyklen erlaubt, d. h. ein Aktivitätentyp kann nicht (im transitiven Abschluss der Subtyp-Relation) Subtyp von sich selbst sein.

Wie bei Subtyp-Strukturen üblich gilt bei der Verwendung von Aktivitätentypen in Integritätsregeln die Typersetzungseigenschaft, d. h. eine Aktivität von einem beliebigen Subtyp des Aktivitätentyps A ist ebenfalls vom Typ A und kann überall eingesetzt werden, wo eine Aktivität vom Typ A erwartet wird.

Beispiel 2.4. *So kann beispielsweise ein abstrakter Aktivitätentyp »Kunde benachrichtigen« erzeugt werden, als dessen Subtypen die Aktivitätentypen »Kunde anrufen«, »E-Mail an Kunden schreiben« und »Fax an Kunden schreiben« definiert werden. Fordert nun eine Regel, dass in einem Prozess eine Aktivität vom Typ »Kunde benachrichtigen« vorkommt, kann dies durch eine Aktivität eines beliebigen Subtyps erfüllt werden.*

2.3 SeaFlows-Integritätsregeln



Die in dieser Arbeit betrachteten **Integritätsregeln** basieren auf der im *SeaFlows*-Projekt entwickelten Regelform, die im Folgenden vorgestellt wird. Diese verwendet als Grundlage der Auswertung die Ausführungsspuren, die während der Ausführung eines Prozesses entstehen. Somit können die Integritätsregeln nach [19] über einem Prozess zu jedem Zeitpunkt in seinem (in Abschnitt 1.1 vorgestellten) Lebenszyklus ausgewertet werden.

Zur Entwurfszeit wird das Prozessmodell durch die Menge aller von ihm erzeugbaren Ausführungsspuren repräsentiert, so dass die Auswertung der Integritätsregeln auf dieser Basis bereits vor der Ausführung des Prozesses erfolgen kann. *Zur Laufzeit* kann jederzeit der Verlauf der Ausführung einer Instanz des Prozesses in Form einer unvollständigen Ausführungsspur angegeben werden, die das bisherige Verhalten der Instanz repräsentiert. Die Auswertung der Regeln erfolgt dann über allen vollständigen Spuren, die auf Grundlage des Prozessmodells (bzw. einer modifizierten Variante, wenn Ad-hoc-Änderungen durchgeführt wurden) nach dem bisherigen Ausführungsverlauf noch möglich sind. *Nach einer Ausführung* des Prozesses steht schließlich die tatsächliche Ausführungsspur dieser Prozessinstanz fest, auf der dann eine nachträgliche Analyse der Erfülltheit der Regeln erfolgen kann.

Jede Integritätsregel gibt eine Forderung an, die jede der betrachteten Ausführungsspuren erfüllen muss. Im Rahmen dieser Arbeit liegt dabei der Fokus auf der strukturellen Ebene. Es werden Bedingungen angegeben, die für die Knoten in der Ausführungsspur gelten müssen – in dieser Arbeit sind solche Bedingungen Auftreten/Nichtauftreten, Reihenfolgen und zeitliche Eigenschaften von Knoten mit bestimmten Aktivitätentypen. Das Modell bietet jedoch Möglichkeiten zur Erweiterung, um weitere Ebenen, etwa Daten oder Ressourcen, zu berücksichtigen. Mögliche Erweiterungen und ihre Umsetzbarkeit in Bezug auf die Beiträge dieser Arbeit werden in Abschnitt 7.1 angesprochen.

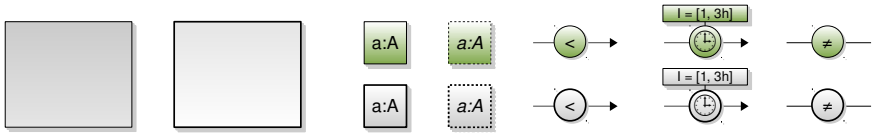
In Abschnitt 2.3.1 werden wir zunächst eine graphbasierte Beschreibung für diese Integritätsregeln vorstellen, die sich leicht in eine anwenderfreundliche Darstellung umsetzen lässt, welche ebenfalls in diesem Abschnitt beschrieben wird. Außerdem wird die Semantik der Regeln hier informell erläutert. Anschließend wird in Abschnitt 2.3.2 eine prädikatenlogische Repräsentation für dieselben Regeln eingeführt sowie dargestellt, auf welche Weise diese der graphbasierten entspricht. In Abschnitt 2.3.3 wird diese logische Darstellung für die Zwecke dieser Arbeit leicht erweitert, ohne ihre Semantik zu verändern. In Abschnitt 2.3.4 stellen wir schließlich eine Struktur zur Interpretation der prädikatenlogisch beschriebenen Regeln vor, durch welche die Semantik der Regeln bei der Auswertung über Ausführungsspuren formal beschrieben wird.

2.3.1 Regelgraphen

Im Rahmen des SeaFlows-Projekts wurde von Ly et al. in [20] eine graphbasierte Darstellung von Integritätsregeln eingeführt, die als Grundlage für diese Arbeit dient und im Folgenden vorgestellt wird.

Ein **Regelgraph** stellt ein Pattern dar, das jede Ausführungsspur, über der es ausgewertet wird, erfüllen muss, damit die Regel als erfüllt gilt. Er besteht aus mehreren Teilen – einem **Bedingungsteil** und einem oder mehreren **Folgegliedern**. Der Bedingungsteil und die Folgeglieder enthalten jeweils weitere Pattern, die eine Bedingung bzw. Forderung der Regel repräsentieren. Alle Folgeglieder bilden gemeinsam den sog. **Folgeteil**. Im Bedingungsteil und den Folgegliedern sind **Knoten** und **Kanten** enthalten. Der Bedingungsteil kann auch leer sein. Ist nur ein einziges Folgeglied vorhanden, kann auch dieses leer sein.

Jeder Knoten hat einen Namen und einen Typ, der einem Aktivitätentyp entspricht. Ein Knoten kann als **Occurrence-** oder **Absence-Knoten** definiert sein sowie entweder dem Be-



(a) (b) (c) (d) (e) (f) (g)

Abbildung 2.6: Graphische Darstellung der Elemente eines Regelgraphen:

(a) Bedingungsteil (b) Folgeteil
(c) Occurrence-Knoten (d) Absence-Knoten
(e) Reihenfolgekante (f) Zeitkante (g) Ungleichheitskante

Bei (c)–(g) jeweils oben: Bedingungsteil, unten: Folgeteil.

dingungsteil oder einem Folgeglied zugeordnet sein. Eine Kante ist jeweils zwischen zwei Knoten definiert und ebenfalls entweder dem Bedingungsteil oder einem Folgeglied zugeordnet. Als mögliche Kantenarten sind ungerichtete **Ungleichheitskanten** sowie gerichtete **Reihenfolge-** und **Zeitkanten** definiert, wobei zu jeder Zeitkante ein abgeschlossenes Intervall ($\subseteq \mathbb{R}_0^+$, also nur im positiven Bereich) angegeben wird. Wenn zwei Knoten durch eine Zeitkante verbunden sind, muss es ebenfalls eine gleichgerichtete Reihenfolgekante zwischen den Knoten geben.

Eine Kante im Bedingungsteil kann nur Knoten aus dem Bedingungsteil verbinden, von denen mindestens einer ein Occurrence-Knoten ist. Eine Kante in einem Folgeglied kann zwischen Knoten aus dem Bedingungsteil, Knoten aus dem entsprechenden Folgeglied sowie zwischen einem Knoten aus dem Bedingungsteil und einem aus dem Folgeglied (in beliebiger Richtung) erfolgen, wobei wieder mindestens ein Occurrence-Knoten enthalten sein muss. Enthält eine Kante im Folgeteil einen Knoten aus dem Bedingungsteil, muss dieser in jedem Fall ein Occurrence-Knoten sein. Abb. 2.7 zeigt die erlaubten Kantenverbindungen (unter Verwendung der im Folgenden eingeführten graphischen Darstellung).

Darstellung

Der Bedingungsteil und die Folgeglieder werden in der graphischen Darstellung der Regel durch je eine Box dargestellt, die das jeweilige Pattern enthält. Knoten werden durch Kästen dargestellt, die ihren Namen und Typ in Textform enthalten. Zur Unterscheidung zwischen Occurrence- und Absence-Knoten besitzen letztere eine gestrichelte Außenlinie

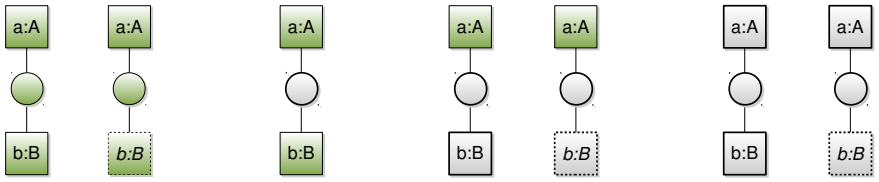


Abbildung 2.7: Erlaubte Kanten in Regelgraphen.

Die Richtung (von *a* nach *b*, von *b* nach *a* oder ungerichtet) und der Typ der Kante ist dabei jeweils beliebig. Bei (a) und (b) ist die Kante dem Bedingungsteil, bei (c) bis (g) einem Folgeglied zugeordnet.

und kursiven Text. Knoten aus dem Bedingungsteil besitzen einen grünen Hintergrund und eine dünne Umrandungslinie, Knoten aus dem Folgeteil einen grauen Hintergrund und eine dicke Umrandungslinie (vgl. Abb. 2.6). Knoten aus dem Folgeteil werden innerhalb der Box ihres jeweiligen Folgeglieds dargestellt, Knoten aus dem Bedingungsteil innerhalb von dessen Box. Außerdem können Knoten aus dem Bedingungsteil zusätzlich als Kopien in beliebigen Folgeteil-Boxen dargestellt sein. Gerichtete Kanten werden durch Pfeile, ungerichtete Kanten durch Linien dargestellt, die die jeweiligen Knoten verbinden. Dabei müssen Kanten aus dem Bedingungsteil stets komplett innerhalb von dessen Box liegen, Folgeteil-Kanten innerhalb der Box des jeweiligen Folgeglieds. Wenn eine Folgeteil-Verbindung also an einer oder zwei Bedingungsteil-Knoten ansetzt, müssen diese als Kopien im jeweiligen Folgeglied dargestellt werden und die Kantenverbindung muss zwischen den Kopien erfolgen, wie in Abb. 2.8 dargestellt ist.

Die Pfeile bzw. Linien sind durch ein Symbol annotiert, das die Art der Kante darstellt. Reihenfolgekanten werden dabei durch ($<$), Zeitkanten durch ($\clockwise$) und Ungleichheitskanten durch ($\neq$) gekennzeichnet. Der Hintergrund der Symbole wird dabei bei einer Bedingungsverbindung mit derselben Farbe und Umrandung wie Bedingungsknoten, ansonsten mit Farbe und Umrandung der Folgeknoten dargestellt. Bei Zeitkanten wird zudem das Intervall in Textform angegeben (vgl. Abb. 2.6).

Semantik

Die Regel ist über einer Ausführungsspur genau dann erfüllt, wenn für jede Belegung der Occurrence-Knoten aus dem Bedingungsteil mit Knotenausführungen aus der Aus-

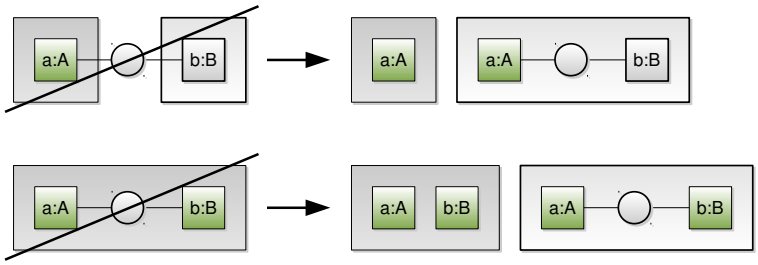


Abbildung 2.8: Kanten aus dem Folgeteil mit Knoten aus dem Bedingungsteil. Links: ungültige Darstellung, rechts: korrekte Darstellung.

führungsspur, mit der das Pattern des Bedingungsteils erfüllt ist, auch eine Belegung der Occurrence-Knoten mindestens eines Folgeglieds existiert, mit der dieses Folgeglied erfüllt ist. Eine Belegung der Knoten ist eine Abbildung, die jedem Knoten aus dem Regelgraphen eine Knotenausführung aus der Ausführungsspur zuordnet, die dem Typ des Regelknotens (bzw. einem Subtyp) entspricht.

Das Pattern eines Regelglieds (Bedingungsteil bzw. Folgeglied) ist für eine Belegung der Occurrence-Knoten erfüllt, wenn gilt:

- Für jede dem entsprechenden Regelglied zugeordnete Kante zwischen zwei **Occurrence-Knoten** gilt, dass die den Regelknoten zugewiesenen Knotenausführungen die durch die Kante repräsentierte Forderung erfüllen.
- Für jeden **Absence-Knoten** aus dem Regelglied gilt: es existiert *keine* Knotenausführung n in der Ausführungsspur, deren Aktivitätenyp dem des Absence-Knotens (oder einem Subtyp) entspricht und bei der für jede Kante zwischen dem Absence-Knoten und einem Occurrence-Knoten die Forderung der Kante durch n und die dem Occurrence-Knoten zugewiesene Knotenausführung m erfüllt wird.

Ein leerer Bedingungsteil gilt als immer erfüllt, ein leeres Folgeglied als nie erfüllt.

Eine Reihenfolgekante zwischen zwei Knoten in der Regel repräsentiert die Forderung, dass die ihnen zugewiesenen Knotenausführungen n und m in dieser Reihenfolge ausgeführt werden, also dass n in der Ausführungsspur beendet wird, bevor die Ausführung von m gestartet wird. Eine Zeitkante mit Intervall I fordert, dass die Zeit zwischen dem Ende von n und dem Start von m im angegebenen Intervall liegt. Eine Ungleichheitskante

fordert, dass beiden Regelknoten unterschiedliche Knotenausführungen zugeordnet sind (die aber demselben Prozessknoten entstammen können).

Eine exakte, logische Definition der Erfülltheit ergibt sich durch die prädikatenlogische Formalisierung in Abschnitt 2.3.2 in Verbindung mit der Interpretation aus Abschnitt 2.3.4.

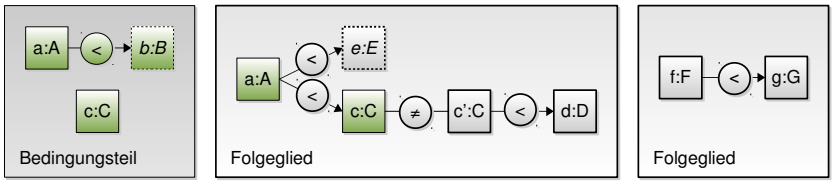


Abbildung 2.9: Beispielregel

Beispiel 2.5. *In Abb. 2.9 ist der Graph einer Regel dargestellt, die sich wie folgt beschreiben lässt:*

Wenn in einer Ausführungsspur eine Ausführung a eines Knotens vom Typ A und eine Ausführung c eines Knotens vom Typ C vorkommt und nach a kein Knoten vom Typ B ausgeführt wird, gilt eine der folgenden Aussagen:

- *Es wird nach a kein Knoten vom Typ E ausgeführt und Knoten c wird erst nach a ausgeführt. Außerdem ist noch eine weitere Ausführung c' eines Knotens vom Typ C enthalten sowie ein Knoten vom Typ D, der nach Ende der Ausführung von c' gestartet wird.*

oder

- *Es kommt in der Ausführungsspur ein Knoten vom Typ F vor und nach diesem wird ein Knoten vom Typ G ausgeführt.*

2.3.2 Prädikatenlogische Formalisierung

Um formale Untersuchungen zu ermöglichen, lässt sich einem Regelgraphen, wie in [20] beschrieben, eine prädikatenlogische Formel zuordnen, die dieser Regel entspricht. Generell gilt, dass Knoten aus dem Regelgraphen in Variablen und einstellige Typisierungsprädikate, Kanten in zweistellige Prädikate umgesetzt werden. Eine solche prädikatenlogische Regel r ist folgendermaßen aufgebaut:

$$r := aq(a \longrightarrow (c_1 \vee c_2 \vee \ldots))$$

Die **Bedingungs-Quantifizierung** aq enthält für jeden Occurrence-Knoten mit Namen v_i aus dem Bedingungsteil des Regelgraphen eine allquantifizierte Variable $\forall v_i$:

$$aq := \forall v_1 \forall v_2 \ldots$$

Der **Bedingungsteil** a hat folgenden Aufbau:

$$a := (apospred_1 \wedge apospred_2 \wedge \ldots \wedge anegitem_1 \wedge anegitem_2 \wedge \ldots)$$

Dabei sind die $apospred_*$ **positive Prädikate**: Für jeden Occurrence-Knoten v mit Typ A aus dem Bedingungsteil des Regelgraphen ist ein Prädikat $apospred_i = C_A(v)$ enthalten. Für jede Bedingungsteil-Kante zwischen zwei Occurrence-Knoten v und w ist ein Prädikat $apospred_i = X(v, w)$ enthalten, wobei für X das jeweils der Kantenart entsprechende Prädikat eingesetzt wird (die Prädikate werden weiter unten definiert). Die $anegitem_*$ werden als **Negativelemente** bezeichnet. Dabei ist für jeden Absence-Knoten v vom Typ A aus dem Bedingungsteil des Graphen ein folgendermaßen aufgebautes Element enthalten:

$$anegitem_i := \forall v \neg (C_A(v) \wedge anegpred_{i,1} \wedge anegpred_{i,2} \wedge \ldots)$$

Die $anegpred_{i,*}$ ergeben sich folgendermaßen: Für jede Kante zwischen dem Absence-Knoten v und einem anderen Knoten w ist ein passendes Prädikat $anegpred_{i,j} = X(v, w)$ bzw. $anegpred_{i,j} = X(w, v)$, je nach Richtung der Kante, enthalten.

Wenn der Bedingungsteil des Regelgraphen leer ist, ist $a := true$.

Für jedes **Folgeglied** aus dem Graphen ist in r ein Element c_i enthalten:

$$c_i := (ex_i : cpospred_{i,1} \wedge cpospred_{i,2} \wedge \ldots \wedge cnegitem_{i,1} \wedge cnegitem_{i,2} \wedge \ldots)$$

Die **Folgeglied-Quantifizierung** ex_i enthält für jeden Occurrence-Knoten namens $v_{i,j}$ aus dem Folgeglied eine existenzquantifizierte Variable $\exists v_{i,j}$:

$$ex_i := \exists v_{i,1} \exists v_{i,2} \ldots$$

Die positiven Prädikate $cpospred_{i,*}$ ergeben sich wie zuvor: Für jeden im entsprechenden Folgeglied des Regelgraphen enthaltenen Occurrence-Knoten mit Namen v und Aktivitätentyp A ist ein Prädikat $cpospred_{i,j} = C_A(v)$ enthalten. Für jede Kante im Folgeglied

zwischen zwei Occurrence-Knoten namens v und w (auch, wenn eine oder beide aus dem Bedingungsteil stammen) ist ein Prädikat $cpospred_{i,j} = X(v, w)$ enthalten, wobei für X das jeweils passende Prädikat eingesetzt wird. Auch die Negativelemente $cnegitem_{i,*}$ sind wie zuvor zusammengesetzt, für jeden im Folgeglied enthaltenen Absence-Knoten v mit Aktivitätentyp A ist ein Element

$$cnegitem_{i,j} := \forall v \neg (C_A(v) \wedge cnegpred_{i,j,1} \wedge cnegpred_{i,j,2} \wedge \ldots)$$

vorhanden. Die $cnegpred_{i,j,*}$ ergeben sich wieder, indem für jede Kante zwischen dem Absence-Knoten v und einem anderen Knoten w (auch, wenn dieser aus dem Bedingungsteil stammt) ein passendes Prädikat $cnegpred_{i,j,k} = X(v, w)$ bzw. $cnegpred_{i,j,k} = X(w, v)$, je nach Richtung der Kante, enthalten ist.

Wenn das Folgeglied im Regelgraphen leer ist, ist $c_i := false$.

Die Kanten aus dem Graphen werden folgendermaßen in Prädikate umgesetzt:

- Eine Reihenfolgekante ($<$) zwischen v und w wird in das Prädikat $O(v, w)$ umgesetzt.
- Eine Ungleichheitskante ($\neq$) zwischen v und w wird in das Prädikat $U(v, w)$ umgesetzt.
- Eine Zeitkante ($\oplus$) zwischen v und w mit einem Zeitintervall I ($I = [a, b]$ oder $I = [a, \infty), a \geq 0, b \geq a$) wird in das Prädikat $T_I(v, w)$ umgesetzt.

Das Intervall in T_I gibt dabei die Zeit an, die zwischen dem Ende der Ausführung von v und dem Start von w vergehen darf. Ein Intervall der Form $[a, \infty)$ beschreibt einen Mindestabstand, ein Intervall der Form $[0, b]$ einen Maximalabstand und ein allgemeines Intervall $[a, b]$ einen Mindest- und Maximalabstand.

Ein *Typisierungsprädikat* $C_A(v)$, wie es in der obigen Formalisierung mehrmals verwendet wird, legt die Typisierung der durch den Regelknoten v repräsentierten Knotenausführung mit Aktivitätentyp A fest.

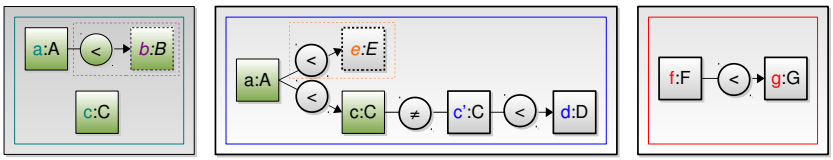


Abbildung 2.10: Beispielregel mit Hervorhebung der Regelglieder und Negativelemente

Beispiel 2.6. *Für das Beispiel aus Abb. 2.10 ergibt sich folgende prädikatenlogische Formel:*

$$r = \forall a \forall c \Big(\big(C_A(a) \wedge C_C(c) \wedge \forall b \neg (C_B(b) \wedge O(a,b)) \big) \longrightarrow$$
$$\Big(\big(\exists c' \exists d \big(C_C(c') \wedge C_D(d) \wedge O(a,c) \wedge U(c,c') \wedge O(c',d) \wedge \forall e \neg (C_E(e) \wedge O(a,e)) \big) \big) \vee$$
$$\big(\exists f \exists g \big(C_F(f) \wedge C_G(g) \wedge O(f,g) \big) \big) \Big) \Big)$$

2.3.3 Erweiterung der prädikatenlogischen Regeln

Für die Zwecke dieser Arbeit ist es sinnvoll, die Folgeglieder einer Regel weiter zu unterteilen. Um möglichst genau angeben zu können, welcher Teil einer Regel verletzt ist, muss diese zuvor in ihre elementaren Forderungen zerlegt werden. Hierzu wird ein Folgeglied so in eine oder mehrere Und-verknüpfte **Existenzgruppen** aufgeteilt, dass die Aufteilung vollständig ist (das Folgeglied also nur noch aus Existenzgruppen besteht), die Semantik beibehalten wird und die Existenzgruppen minimal, also nicht weiter aufspaltbar sind. Dazu werden alle Variablen und Negativelemente, die (bzw. bei Negativelementen deren Variablen) durch Prädikate »verbunden« sind, zu jeweils einer Existenzgruppe zusammengefasst. Im Regelgraph entspricht dies dem Zusammenfassen aller durch Folgeglied-Kanten verbundenen Knoten zu jeweils einer Gruppe (Kanten aus dem Folgeglied, die nur Bedingungs-Knoten verbinden, bilden jeweils eine einzelne Gruppe).

Formal werden die Existenzgruppen $eg_{i,*}$ aus einem Folgeglied c_i in der prädikatenlogischen Regel wie folgt gebildet: Wenn in einem positiven Prädikat $cpospred_{i,j}$ oder einem Negativelement $cnegitem_{i,j}$ nur freie Variablen aus dem Bedingungsteil (aq) vorkommen, bildet es eine eigene Existenzgruppe. Die restlichen Existenzgruppen bilden sich folgendermaßen: Die Menge der Variablen in ex_i wird in Teilmengen aufgeteilt, so dass gilt: zwei Variablen sind in derselben Teilmenge enthalten **genau dann, wenn** es ein Prädikat $cpospred_{i,k}$

oder ein Negativelement $cnegitems_{i,k}$ im Folgeglied gibt, in dem beide Variablen enthalten sind. Für jede dieser Teilmengen ist dann eine Existenzgruppe $eg_{i,j}$ definiert der Form

$$eg_{i,j} := [egex_{i,j}(egpospred_{i,j,1} \wedge egpospred_{i,j,2} \wedge \ldots \wedge egnegitem_{i,j,1} \wedge egnegitem_{i,j,2} \wedge \ldots)].$$

$egex_{i,j}$ enthält Existenzquantifizierungen für alle Variablen aus der Teilmenge:

$$egex_{i,j} := \exists v_{i,j,1} \exists v_{i,j,2} \ldots$$

Die $egpospred_{i,j,*}$ bzw. $egnegitem_{i,j,*}$ bestehen aus allen $cpospred_{i,*}$ bzw. $cnegitem_{i,*}$, die eine Variable aus der entsprechenden Teilmenge enthalten. Nach dieser Aufteilung ist das Folgeglied c_i dann folgendermaßen aufgebaut:

$$c_i := (eg_{i,1} \wedge eg_{i,2} \wedge \ldots).$$

Diese Aufspaltung erzeugt minimale Existenzgruppen: Würde man eine solche Gruppe weiter unterteilen, müssten, da alle enthaltenen Variablen durch Prädikate »verbunden« sind, die Existenzquantoren aus diesen Gruppen nach außen gezogen werden, da Variablen dann in mehreren Gruppen referenziert würden. In diesem Fall wäre die Aufspaltung aber nicht mehr vollständig, da die Existenzquantoren dann außerhalb der Gruppen lägen.

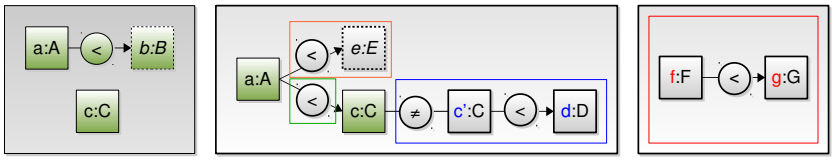


Abbildung 2.11: Beispielregel mit farbiger Hervorhebung der Existenzgruppen

Beispiel 2.7. *Mit Existenzgruppen ergibt sich für das Beispiel aus Abb. 2.11 folgende Formel:*

$$r = \forall a \forall c \Big(\big(C_A(a) \wedge C_C(c) \wedge \forall b \neg (C_B(b) \wedge O(a,b)) \big) \longrightarrow$$
$$\Big(\big([O(a,c)] \wedge [\exists c' \exists d (C_C(c') \wedge C_D(d) \wedge U(c,c') \wedge O(c',d))] \wedge [\forall e \neg (C_E(e) \wedge O(a,e))] \big) \vee$$
$$\big([\exists f \exists g (C_F(f) \wedge C_G(g) \wedge O(f,g))] \big) \Big) \Big)$$

2.3.4 Interpretation der Integritätsregeln

Eine wie in Abschnitt 2.3.2 bzw. 2.3.3 beschrieben als prädikatenlogische Formel vorliegende Regel kann nun formal über einer Ausführungsspur ausgewertet werden. Hierfür muss eine Grundmenge angegeben werden, die die Werte beschreibt, welche die Variablen in der Formel annehmen können, zudem müssen den Prädikatensymbolen in der Formel tatsächliche Funktionen über den Elementen der Grundmenge zugeordnet werden. Im Folgenden werden die notwendigen Funktionen sowie die Zuordnungsfunktion definiert.

Definition 2.4 (Interpretation der Prädikatensymbole)**.** Sei t eine Ausführungsspur wie in Definition 2.3. Sei weiter r eine Regel in prädikatenlogischer Form wie in Abschnitt 2.3.2 bzw. 2.3.3, $activityTypes(r)$ die Menge der in r in C_A-Prädikaten verwendeten Aktivitätentypen und $intervals(r)$ die Menge der in r in T_I-Prädikaten verwendeten Intervalle.

Dann sind für $A \in activityTypes(r), I \in intervals(r), x = (n_x, it_x) \in nodeExs(t), y \in nodeExs(t)$ folgende Funktionen definiert:

$$containsActivity_{t,A}(x) = \begin{cases} \text{true} & \text{falls Knoten } n_x \text{ von Typ } A \text{ (oder einem Subtyp)} \\ \text{false} & \text{andernfalls.} \end{cases}$$

$$execOrder_t(x, y) = \begin{cases} \text{true} & \text{falls } nodeExEndPos(t, x) < nodeExStartPos(t, y) \\ \text{false} & \text{andernfalls.} \end{cases}$$

$$execInterval_{t,I}(x, y) = \begin{cases} \text{true} & \text{falls } nodeExStartTime(t, y) - nodeExEndTime(t, x) \in I \\ \text{false} & \text{andernfalls.} \end{cases}$$

$$unequal_t(x, y) = \begin{cases} \text{true} & \text{falls } x \neq y \\ \text{false} & \text{andernfalls.} \end{cases}$$

Weiter definieren wir die Interpretationsfunktion $predAssignment_{t,r}$ als Funktion über der Menge der Prädikatensymbole in r, die jedem dieser Symbole eine Funktion zuweist:

- $\forall A \in activityTypes(r) : predAssignment_{t,r}(C_A) = containsActivity_{t,A}$
- $predAssignment_{t,r}(O) = execOrder_t$
- $\forall I \in intervals(r) : predAssignment_{t,r}(T_I) = execInterval_{t,I}$
- $predAssignment_{t,r}(U) = unequal_t$

Die folgende Definition führt darauf aufbauend die semantische Interpretation der Integritätsregeln ein.

Definition 2.5 (Interpretation einer Integritätsregel)**.** Sei r eine Regel wie in Abschnitt 2.3.2 bzw. 2.3.3 und t eine Ausführungsspur wie in Definition 2.3. So ist

$$struct_{t,r} = (nodeExs(t), predAssignment_{t,r})$$

die semantische Interpretation für die Regel r mit der Grundmenge $nodeExs(t)$ und der Prädikateninterpretationsfunktion $predAssignment_{t,r}$. Die Regel r gilt somit genau dann als über t erfüllt, wenn $struct_{t,r} \models r$, also $struct_{t,r}$ ein Modell für r ist.

Beispiel 2.8. *Wir betrachten die erste Ausführungsspur aus Beispiel 2.3 (S. 17). Für diese Ausführungsspur t_{bsp} gilt:*

$$nodeExs(t_{bsp}) = \{(1,\epsilon),(2,\epsilon),(3,\epsilon),(4,\epsilon),(9,\epsilon),(8,\epsilon),(10,\epsilon),(11,\epsilon)\}.$$

Des Weiteren betrachten wir die einfache Regel

$$r_{bsp} := \forall a\big(C_A(a) \longrightarrow (\exists b : C_B(b) \wedge O(a,b))\big).$$

Nach Definition 2.4 sieht die Zuweisungsfunktion $predAssignment_{t_{bsp},r_{bsp}}$ folgendermaßen aus:

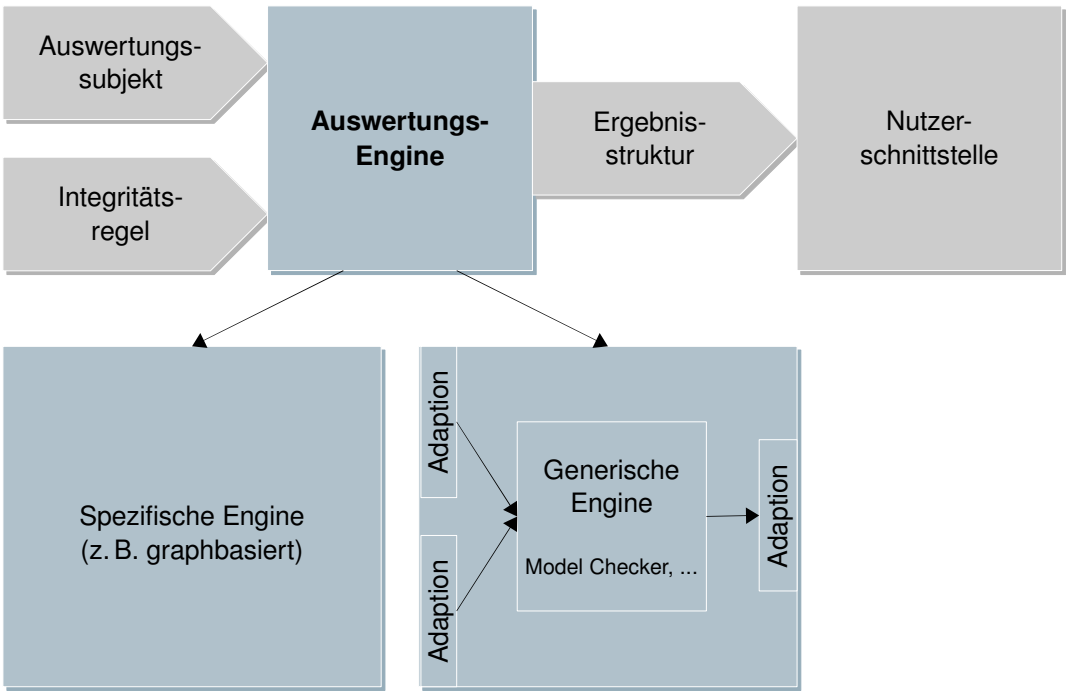
$$predAssignment_{t_{bsp},r_{bsp}}(C_A) = containsActivity_{t_{bsp},A}$$
$$predAssignment_{t_{bsp},r_{bsp}}(C_B) = containsActivity_{t_{bsp},B}$$
$$predAssignment_{t_{bsp},r_{bsp}}(O) = execOrder_{t_{bsp}}.$$

Somit ist

$$struct_{t_{bsp},r_{bsp}} = (nodeExs(t_{bsp}), predAssignment_{t_{bsp},r_{bsp}})$$

die semantische Interpretation für die Regel. Damit wird die Regel über dieser Ausführungsspur als ***nicht erfüllt*** *ausgewertet, denn mit einer Bindung von a an $(4,\epsilon)$ ist $containsActivity_{t_{bsp},A}(a)$ erfüllt, jedoch existiert keine Knotenausführung b in der Spur, mit der $containsActivity_{t_{bsp},B}(b)$ und $execOrder_{t_{bsp}}((4,\epsilon),b)$ erfüllt wäre.*

2.4 Auswertungsalgorithmen



Für die Auswertung einer Integritätsregel über einem Auswertungssubjekt sind verschiedenste Algorithmen denkbar, die sich für die praktische Nutzung in sog. Auswertungs-Engines umsetzen lassen – Software-Komponenten, die für die Durchführung der Regelauswertung zuständig sind. Im Folgenden werden zwei prinzipielle Möglichkeiten für verwendbare **Auswertungsalgorithmen** beispielhaft vorgestellt.

2.4.1 Graphbasierte Untersuchung

Da sowohl Prozessmodelle als auch Integritätsregeln als Graphen darstellbar sind, lassen sich für die Auswertung Algorithmen einsetzen, die direkt auf den Prozess- und/oder Regelgraphen arbeiten. Hierfür können an die Art der Graphen angepasste Algorithmen entwickelt werden, die übliche Graph-Aktionen wie das Durchlaufen der Knoten entlang der Kanten, das Bilden von Teilgraphen oder das Zusammenfassen von Graphen einsetzen, um die Erfülltheit von Regeln zu ermitteln. Die Ergebnisse, die solche Algorithmen liefern, können vielfältig sein. Sinnvoll ist jedoch die Angabe von einem oder mehreren Teilgraphen des Prozessmodellgraphen, so dass die Vereinigung der durch die Graphen

repräsentierten Ausführungsspuren den Spuren entspricht, in denen die überprüfte Regel erfüllt (oder verletzt) ist. Dabei kann ein Teilgraph ggf. um weitere Kanten erweitert werden, beispielsweise zusätzliche Synchronisationskanten zwischen Knoten enthalten, wenn die Regel nur erfüllt ist, wenn diese Knoten in der angegebenen Reihenfolge ausgeführt werden.

Ein Algorithmus dieses Typs wird beispielsweise in der prototypischen Implementierung der Ergebnisse dieser Arbeit verwendet. Er wird in Abschnitt 6.8.2 konzeptionell beschrieben. Die dort verwendeten Zweigeinschränkungen stellen Umsetzungen von Teilgraphen des Prozessmodells dar, die in einer solchen Einschränkung verwendeten Bedingungen entsprechen zusätzlichen Kanten im Graphen.

2.4.2 Modellbasierte Untersuchung

Eine weitere Möglichkeit zur Untersuchung der Compliance von Integritätsregeln über Prozessmodellen und -instanzen stellt die Verwendung modellbasierter Verfahren dar. Dabei werden Integritätsregeln als logische Formeln betrachtet, als Beschreibungsmechanismen bieten sich beispielsweise Prädikatenlogik oder temporale Logik an. Eine prädikatenlogische Beschreibung von Integritätsregeln wurde in Abschnitt 2.3.2 vorgestellt. Das Auswertungssubjekt wird ebenfalls in ein formales Modell überführt, etwa einen Automaten oder ein Zustandsmodell.

Liegen Regel und Auswertungssubjekt in kompatiblen formalen Beschreibungen vor, lassen sich diese durch die Auswertungs-Engine – mit spezifischen Algorithmen oder durch Verwendung externer Model-Checking-Systeme [6] – analysieren. Als Ergebnis einer solchen Auswertung wird üblicherweise zurückgegeben, ob die Regeln und das Auswertungssubjekt kompatibel sind, sowie, wenn dies nicht der Fall ist, ein *Gegenbeispiel* angegeben – eine Belegung der Variablen des Regelausdrucks, mit der eine Inkompatibilität zwischen Auswertungssubjekt und Regel besteht.

2.4.3 Weitere Verfahren

Neben den angesprochenen Mechanismen sind zahlreiche weitere Algorithmen möglich. Ziel dieser Arbeit ist, möglichst unabhängig vom verwendeten Algorithmus zu sein.

2.5 Systembestandteile

Als grundlegende Softwarebestandteile eines Prozessmanagement-Systems mit Regelunterstützung setzen wir in dieser Arbeit Editoren voraus, mit denen die in den vorhergehenden Abschnitten vorgestellten Prozessvorlagen und Integritätsregeln erstellt und bearbeitet und Aktivitätenvorlagen und -typen verwaltet werden können sowie Komponenten, die dazu dienen, laufende Prozesse zu überwachen und Ad-hoc-Änderungen durchzuführen. Eine Anwendung kann auch mehrere dieser Funktionen anbieten. Ein weiterer Bestandteil ist Client-Software zur Durchführung manueller Aktivitäten. Ebenfalls betrachtet wird die im Hintergrund ablaufende Serversoftware.

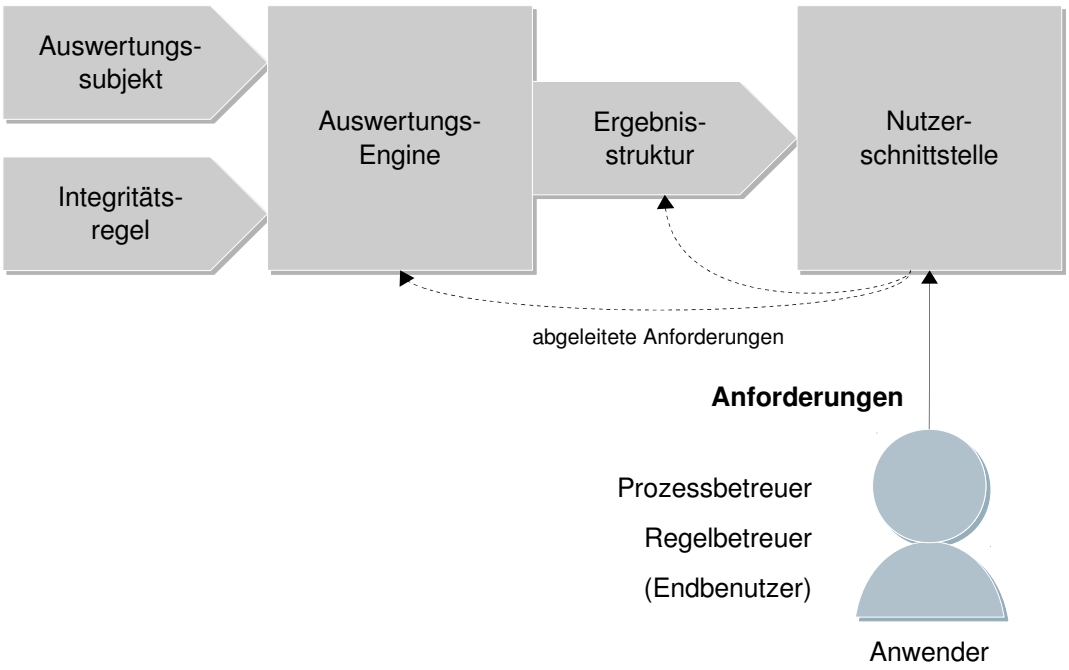
Mit einem **Prozessvorlagen-Editor** ist es dabei möglich, neue Prozessmodelle zu erstellen sowie vorhandene zu bearbeiten. Dabei ist nicht nur die Erzeugung und Manipulation von WSM-Netzen möglich, sondern auch die Zuordnung von Aktivitäten zu den Knoten des Prozesses. Auch können hiermit ggf. Änderungen an einer Prozessvorlage auf laufende Instanzen angewendet werden. Ein **Regeleditor** erlaubt die Erzeugung und Bearbeitung von Regelgraphen und die Zuordnung von Aktivitätentypen zu Variablen aus der Regel. Ein **Aktivitätenvorlagen-Editor** kann zur Erstellung und Verwaltung von ausführbaren Aktivitätenvorlagen genutzt werden, denen dabei verschiedene Aktionen zugeordnet werden können. Mit einem **Aktivitätentypen-Editor** können abstrakte Aktivitätentypen zur Verwendung in Regeln erstellt werden. Außerdem können hier Aktivitätentypen und -vorlagen mit anderen Aktivitätentypen in eine Subtyp-Beziehung gestellt werden. In einer **Überwachungskomponente** ist es unter anderem möglich, den aktuellen Ausführungszustand von einzelnen laufenden Prozessinstanzen anhand ihrer Darstellung als WSM-Netz abzulesen sowie weitere Eigenschaften zu analysieren. In einer **Anwendung zur Ad-hoc-Änderung** von Prozessinstanzen können einzelne laufende Instanzen angepasst werden, entweder auf Basis der Graphdarstellung oder einer vereinfachten Darstellung für Endanwender. In einer **Client-Anwendung** wird dem Endanwender eine *Arbeitsliste* mit durchzuführenden Aktivitäten angezeigt, außerdem sind darin die für die Durchführung der Aktivitäten benötigten Hilfsmittel (Formulare, Spezialanwendungen) verfügbar. Für die Verwaltung der laufenden Prozessinstanzen und die Durchführung der Zustandsübergänge im Prozessmodell ist die **Server-Software** zuständig, mit der Anwender zwar üblicherweise nicht direkt in Kontakt kommen und die daher für die Aspekte dieser Arbeit nur am Rande eine Rolle spielt, die im Hintergrund jedoch ebenfalls an der Sicherstellung der Regelerfüllung, vor allem zur Laufzeit, beteiligt sein kann.

Für die praktische Umsetzung dieser Arbeit in Kapitel 6 wird die AristaFlow-BPM-Suite [1] gewählt. Diese verwendet die in Abschnitt 2.1.1 vorgestellten ADEPT-WSM-Netze zur Prozessbeschreibung. Die Suite enthält den *AristaFlow Process Template Editor*, mit dem Prozessvorlagen erstellt und bearbeitet werden können sowie den *AristaFlow Activity Repository Editor*, mit dem Aktivitätenvorlagen verwaltet werden können. Zur Überwachung laufender Instanzen und der Durchführung von Ad-hoc-Änderungen dient der *AristaFlow Monitor*, einfache Ad-hoc-Änderungen können auch mit dem *AristaFlow Client* durchgeführt werden, der zur Ausführung manueller Aktivitäten dient. Die Ausführung der Prozessinstanzen übernimmt der *AristaFlow Server*.

Für die Umsetzung der dargestellten Integritätsregeln zur Nutzung mit AristaFlow existiert ebenfalls ein Editor, der im Rahmen eines Praktikums an der Universität Ulm erstellt wurde. Dieser erlaubt, unter Verwendung der Aktivitätenvorlagen aus AristaFlow, Regelgraphen, wie sie in Abschnitt 2.3.1 vorgestellt wurden, zu erstellen und in einem auf *XML Metadata Interchange* (XMI) [24] basierenden Format zu speichern.

Für die Verwaltung von Aktivitätentypen für SeaFlows-Regeln existiert derzeit noch kein praktisch umgesetzter Editor. Auf den theoretischen Teil der Arbeit hat dies keine Auswirkungen. Für die praktische Umsetzung im in Kapitel 6 vorgestellten Demonstrator bedeutet es, dass derzeit keine abstrakten Aktivitätentypen verwendet werden können, sondern lediglich Aktivitätentypen, die aus der 1:1-Abbildung von ADEPT-Aktivitätenvorlagen entstehen. Auf die Funktionsweise des Demonstrators hat dies jedoch kein Auswirkungen, da die Verwendung von abstrakten Aktivitätentypen in der Implementierung vorgesehen ist und somit bei Verfügbarkeit eines entsprechenden Editors einfach nachgerüstet werden könnte.

3 Anforderungsanalyse



Um die Entwicklung einer anwender- und aufgabenorientierten Nutzerschnittstelle zu ermöglichen, müssen zunächst die potentiellen Nutzer eines regelbasierten BPM-Systems identifiziert werden und es muss untersucht werden, für welche Zwecke das System einsetzbar sein soll. Daraus lassen sich anschließend Anforderungen ableiten, die das System erfüllen muss.

Hierzu führen wir zunächst in Abschnitt 3.1 eine Analyse der verschiedenen Nutzergruppen durch, die mit dem System in Berührung kommen. In Abschnitt 3.2 werden anschließend Anwendungsszenarien aufgestellt, die aufzeigen, auf welche Weise die Anwender solche regelbasierten Systeme einsetzen können. Darauf aufbauend werden in Abschnitt 3.3 schließlich Anforderungen an das System formuliert.

3.1 Nutzergruppen

Für diese Arbeit werden die Anwender eines BPM-Systems, das Integritätsregeln unterstützt, in drei Nutzergruppen/-rollen gegliedert: Prozessbetreuer, Regelbetreuer und Endanwender.

Ein **Prozessbetreuer** ist für die Erzeugung, Verwaltung, Optimierung und Überwachung von ausführbaren Prozessen zuständig. Dabei kann der Schwerpunkt auf verschiedene Aufgaben gelegt sein: Es ist möglich, dass ein Prozessbetreuer die Erzeugung neuer Prozesse von der Analyse der Realweltprozesse bis hin zur fertigen, ausführbaren Prozessvorlage übernimmt, allerdings können diese Aufgaben auch aufgeteilt oder die Analyse und Untersuchung von Prozess- und Fachexperten bzw. externen Unternehmensberatern außerhalb des Systems durchgeführt werden, so dass der Betreuer nur noch für die Umsetzung des Prozesses im BPM-System zuständig ist. Auch die Überwachung laufender Prozessinstanzen und die Optimierung der Prozesse können von verschiedenen Personen durchgeführt werden.

Für die Zwecke dieser Arbeit werden jedoch alle Aufgaben von der Modellierung des Realweltprozesses über seine Ausführung und den gesamten Prozesslebenszyklus hinweg der Benutzerrolle des Prozessbetreuers zugeordnet, da sie alle dieselbe prozessorientierte Sichtweise auf das System mit sich bringen und auch im AristaFlow-System [9], auf dem die in Kapitel 6 vorgestellte praktische Umsetzung dieser Arbeit basiert, durchgehend unterstützt werden, so dass sie in einem kleineren Unternehmen auch durchaus von einer einzelnen Person durchgeführt werden könnten. Selbstverständlich haben die Ergebnisse dieser Arbeit jedoch auch Gültigkeit, wenn die Benutzerrolle auf mehrere Personen oder Einrichtungen aufgeteilt ist.

Je nach Aufgabenschwerpunkt verfügt der Prozessbetreuer über ein grundlegendes bis vertieftes Fachwissen im Anwendungsgebiet und kennt die Prozesse auch aus der praktischen Anwendung. In jedem Fall ist er mit den im BPM-System umgesetzten Prozessen und ihrer Erzeugung und Änderung vertraut.

Ein **Regelbetreuer** beschreibt, verwaltet und pflegt eine Menge von Integritätsregeln. Die Art dieser Regeln kann, wie in Abschnitt 3.2.1 genauer dargestellt wird, sehr vielfältig sein. Sie reichen von sehr allgemein gültigen, prozessfernen Vorschriften oder Business Rules bis zu für einen Prozess spezifischen Integritätsregeln. Es ist jedoch sinnvoll, wenn jeder Regelbetreuer für die Regeln aus einem bestimmten Bereich zuständig ist und hierbei auch

Fachwissen mitbringt, etwa juristisches Wissen über gesetzliche Regelungen. Bei sehr prozessnahen bzw. prozessspezifischen Regeln, die ggf. auch erst im Laufe der Entwicklung eines Prozesses entstehen, ist typischerweise sinnvoll, wenn es sich bei Regel- und Prozessbetreuer um dieselbe Person handelt.

Wie bereits bei der Rolle des Prozessbetreuers ist auch hier variabel, in welchem Umfang die Regeln bereits mit Hilfe des Systems entworfen werden bzw. ob vorgefertigte Regelbeschreibungen vorliegen. Wiederum soll jedoch – vor allem im Bezug auf weniger umfangreiche, prozessspezifische Regeln – auch der Fall unterstützt werden, in dem ein einzelner Betreuer von der Analyse der Rahmenbedingungen über die Umsetzung der Regeln bis hin zu ihrer Überwachung und Weiterentwicklung alleine zuständig ist.

Die **Endanwender** sind Nutzer, die für die Durchführung der manuellen Schritte in laufenden Instanzen der von Prozessbetreuern definierten Prozesse zuständig sind. Auch im Bereich der Endanwender gibt es ein breites Spektrum des Grads der Systemnutzung, das von der einfachen Bearbeitung der Aufgaben, die in der Arbeitsliste erscheinen bis hin zur aktiven Ad-hoc-Anpassung des Prozesses an einzelne Projekte, Sonderwünsche von Kunden oder Ausnahmesituationen reicht. Endanwender haben üblicherweise ein großes Fachwissen, aber nur geringfügiges Wissen über die technische Realisierung des Prozesses im System.

Manchmal ist es jedoch auch der Fall, dass ein Prozess- bzw. Regelbetreuer das System gleichzeitig auch als Endanwender einsetzt. Dies kann z. B. zutreffen, wenn einzelne Benutzer gesondert geschult werden, um Prozesse ad hoc oder auch planerisch, etwa für einzelne Projekte, anzupassen oder spezifisches Anwenderwissen in die Prozessmodellierung einzubringen. Andererseits können jedoch auch eigentlich als Prozessbetreuer eingesetzte Nutzer als Endanwender auftreten, um den Prozess aus Anwenderperspektive zu testen. In Abb. 3.1 ist graphisch dargestellt, wie alle drei Nutzergruppen überlappen können, wenn einzelne Personen mehreren Gruppen angehören.

3.1.1 Aufgaben

Die Mitglieder der verschiedenen Nutzergruppen arbeiten auf unterschiedliche Weise mit dem System, um ihre verschiedenartigen Aufgaben zu erfüllen. Diese verschiedenen Aufgaben haben einen Einfluss auf die spezifischen Anforderungen der Nutzer und werden daher im Folgenden untersucht.

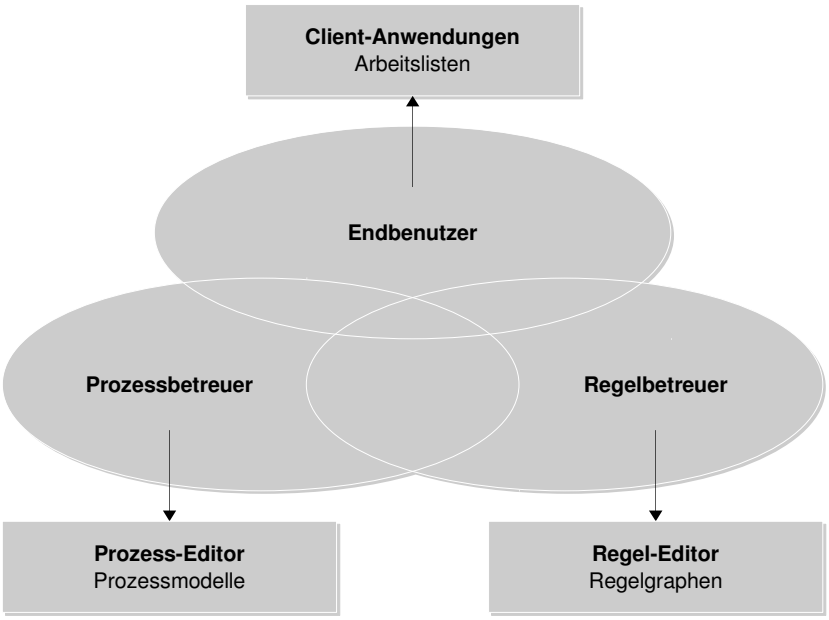


Abbildung 3.1: Betrachtete Nutzergruppen und ihre Interaktion mit dem BPM-System

Eine Hauptaufgabe des **Prozessbetreuers** besteht darin, mit Hilfe des Prozessvorlagen-Editors neue Prozessmodelle zu erstellen. Hierzu hat er entweder einen bereits ausgearbeiteten Prozessentwurf vorliegen, der in einer Prozessmodellierungssprache wie z. B. BPMN [25] erstellt wurde, oder die Modellierung erfolgt erst jetzt mit der Prozessbeschreibungssprache des BPM-Systems. Während die Umsetzung eines bereits modellierten Prozesses – je nach Ausdrucksmächtigkeit der Vorlage – relativ direkt und zielgerichtet erfolgen kann, ist die Vorgehensweise im zweiten Fall wesentlich weniger direkt, nimmt einen längeren Zeitraum in Anspruch und erfordert üblicherweise mehrere Überarbeitungen der modellierten Prozessteile. In jedem Fall sollte nach der Erstellung der Prozessvorlage eine Überprüfung bzw. ein Test erfolgen, ob der umgesetzte Prozess dem Entwurf bzw. dem Realweltprozess entspricht – oder dies bereits während der Umsetzung sichergestellt werden.

Eine weitere wichtige Aufgabe des Prozessbetreuers besteht in der Überwachung der laufenden Prozesse, wenn zuvor erstellte Prozessvorlagen ausgeführt werden. Hierfür werden die Monitoring-Tools des BPM-Systems eingesetzt, die üblicherweise eine dem Prozessvorlagen-Editor ähnelnden Komponente enthalten, in der der Fortschritt laufender Prozesse sowie ihre Struktur, inklusive ggf. enthaltener Ad-hoc-Abweichungen, auf Basis der Pro-

zessvorlage dargestellt wird und analysiert werden kann. Ebenfalls können hier neue Adhoc-Änderungen in Prozessinstanzen durchgeführt werden.

Ein weiterer wichtiger Schritt im Prozesslebenszyklus, dessen Umsetzung in den Zuständigkeitsbereich des Prozessbetreuers fällt, ist die Analyse und Optimierung von Prozessen. Für die Analyse können spezielle Tools verwendet werden, zur Anpassung des Prozesses dient wiederum der Prozessvorlagen-Editor. Auch die Analysetools können teilweise die Prozessvorlage zur Darstellung von Eigenschaften abgeschlossener Prozesse direkt an den jeweiligen Prozessschritten verwenden.

Für den **Regelbetreuer** stellt der Regel-Editor das Hauptwerkzeug dar. Mit diesem kann er neue Integritätsregeln erzeugen und bestehende verändern. In einem vorhergehenden Schritt oder im Zuge der Regelmodellierung muss eine Umsetzung von Realweltregeln in System-Regeln bzw. Regelgraphen erfolgen, was wiederum mehrere Überarbeitungen notwendig machen kann und von der Art der verwendeten Regeln und dem Grad ihrer Strukturierung und Detailliertheit abhängt.

Ähnlich wie bei Prozessvorlagen ist es auch bei Regeln erforderlich, diese während bzw. nach ihrer Modellierung zu testen und bei Bedarf anzupassen. Da Regeln für sich genommen nicht ausführbar sind, ist hierbei für den Regelbetreuer, insbesondere im Zusammenhang mit prozessnahen bzw. prozessspezifischen Regeln, auch eine Betrachtung des Prozesskontexts sinnvoll, damit er die Auswirkungen dieser Regeln und ihre Erfülltheit im Prozess überprüfen kann.

Die Verbindung zwischen Integritätsregeln und Prozessen stellen Aktivitätentypen dar, die in den Regeln referenziert werden und den in den Prozessvorlagen verwendeten Aktivitäten zugeordnet sind. Hier sind verschiedene Vorgehensweisen möglich: Eine Möglichkeit besteht darin, dass Regel- und Prozessbetreuer zunächst unabhängig voneinander Aktivitätentypen erstellen – der Regelbetreuer explizit und unter Nutzung der Vererbungsstruktur von Aktivitätentypen, der Prozessbetreuer implizit durch die Verwendung ausführbarer Aktivitäten, denen jeweils ein gleichnamiger Aktivitätentyp zugeordnet ist. Anschließend können dann den Regel-Aktivitätentypen die Aktivitäten aus dem Prozess zugeordnet werden, indem eine Subtyp-Beziehung zwischen der Aktivität und dem Aktivitätentyp aus der Regel erzeugt wird. Alternativ kann jedoch, insbesondere bei prozessnahen Regeln, auch zunächst von Prozess- und Regelbetreuer ein gemeinsames Vokabular an Aktivitätentypen erstellt werden, das dann von den Regeln und Prozessaktivitäten referenziert wird. Ggf. existieren auch bereits vorgegebene Aktivitätentypen in Form von Ontologien

aus dem jeweiligen Anwendungsgebiet. In jedem Fall wird der Aktivitätentyp-Editor üblicherweise von beiden Benutzerrollen verwendet.

Endanwender arbeiten mit dem BPM-System über die Client-Anwendungen, die größtenteils aufgabenspezifisch sind und üblicherweise von einer generischen Client-Anwendung gesteuert werden. Zur Verwaltung der manuellen Schritte dienen Arbeitslisten. Die generische Client-Anwendung zeigt dem Anwender eine Liste von Schritten aus verschiedenen Prozessinstanzen an, die als nächstes erledigt werden müssen und aus denen der Anwender den nächsten zu erledigenden Schritt auswählen kann, wodurch dann die der Aufgabe zugeordnete Anwendung gestartet wird. Vom BPM-System häufig bereitgestellt wird eine Anwendung, mit der vom Prozessbetreuer spezifizierte Formulare ausgefüllt werden können sowie die Funktionalität, verschiedene spezifische Anwendungen zu starten, mit Eingabedaten aus dem Prozess auszustatten und ihre Ausgabedaten wieder in den Prozess einzubringen.

Wenn das BPM-System Ad-hoc-Abweichungen unterstützt, wie sie etwa in ADEPT möglich sind, kann der Endanwender ggf. einfache Änderungen der laufenden Prozessinstanz in der Client-Anwendung vornehmen. Solche einfachen Änderungen sind etwa das Überspringen von Schritten, das Einfügen neuer Schritte oder das Vorziehen später geplanter Schritte im Prozessablauf [7].

3.1.2 Spezifische Anforderungen

Noch vor der Betrachtung der konkreten Anwendungsszenarien lassen sich aus den Profilen der verschiedenen Nutzergruppen bereits einige spezifische Anforderungen der verschiedenen Anwender an das System und Ziele, die sie mit der Nutzung des Systems verfolgen, ableiten.

Der **Prozessbetreuer** möchte erreichen, dass eine Prozessvorlage den Realweltprozess möglichst genau abbildet und stabil und fehlerfrei läuft sowie sich an vorgegebene Regeln und Einschränkungen hält. Generell steht für ihn der Prozess im Mittelpunkt seines Denkens, daher möchte er ständig Kontrolle über den Prozess behalten und über den Zustand laufender Prozesse informiert sein. Ist eine Regel im Prozess verletzt, möchte er die Ursache hierfür einfach erkennen und den Fehler beheben können. Da er im Allgemeinen weniger Wissen über die Regeln und mehr über den Prozess besitzt, interessiert ihn weniger eine Regel an sich als ihre Auswirkungen auf den Prozess.

Zu den Zielen des **Regelbetreuers** gehört, dass die von ihm betreuten Regeln die Richtlinien, Vorschriften und Randbedingungen, auf denen sie basieren, korrekt abbilden und dass sich alle Prozesse an diese Regeln halten. Zentrales Element für ihn sind die Regeln, je nach Aufgabenverteilung kennt er sich mit Prozessdetails nicht gut aus. Daher hat er die Anforderung, bei Verletzungen von Regeln die Ursache hierfür auch in der Regel zu erkennen, um so möglicherweise fehlerhafte Regeln zu entdecken, die trotz eines korrekten Prozesses zu einer Regelverletzung führen.

Der **Endanwender** möchte, dass die von ihm verwendeten Client-Anwendungen »einfach funktionieren«. Er möchte seine Arbeit erledigen und dabei möglichst nicht von Fehlermeldungen gestört werden. Sollten dennoch einmal Fehler im Prozess auftreten, die seine Intervention erfordern (z. B. da er einen Schritt überspringen möchte, was aber zu einer Regelverletzung führen würde), möchte er keine kryptischen Informationen über für ihn unwichtige Details erhalten, sondern verständliche Hinweise und Informationen darüber, was genau er tun muss, um das aufgetretene Problem zu beheben bzw. mögliche Probleme zu vermeiden, damit er möglichst schnell mit seiner Arbeit fortfahren kann. Da er üblicherweise wenig Wissen über Prozess- oder Regeldetails besitzt und mit der verwendeten Prozessbeschreibungssprache bzw. den Regelgraphen nicht vertraut ist, ist eine Darstellung von Regelverletzungen auf Basis dieser Objekte für ihn im Allgemeinen nicht sinnvoll.

3.2 Anwendungsszenarien

Um die Anforderungen der Anwender an das System im Kontext der verschiedenen Nutzungsmöglichkeiten untersuchen zu können, entwickeln wir zunächst verschiedene Anwendungsszenarien für die Nutzung von Integritätsregeln in BPM-Systemen. Im Folgenden werden verschiedene grundlegende Ziele ihres Einsatzes vorgestellt und mit konkreten Anwendungsbeispielen veranschaulicht. Anschließend wird, ausgehend von diesen konkreten Szenarien, die Nutzung der Integritätsregeln in den verschiedenen Phasen des in Abschnitt 1.1 vorgestellten *Business Process Lifecycle* dargestellt. Alle Beispiele in diesem Abschnitt können durch die in Abschnitt 2.3 vorgestellten SeaFlows-Integritätsregeln modelliert werden.

3.2.1 Einsatzzwecke

Wir definieren fünf grundlegende mögliche Einsatzzwecke von Integritätsregeln im Business Process Management:

- **Einsatzzweck 1:** Überprüfung und Sicherstellung vorgegebener Randbedingungen
- **Einsatzzweck 2:** Testen von Prozesseigenschaften
- **Einsatzzweck 3:** Grundlage der Prozessentwicklung
- **Einsatzzweck 4:** Konsistenthaltung bei Änderungen
- **Einsatzzweck 5:** Ergänzung der Prozessspezifikation

Der erste Einsatzzweck besteht in der **Überprüfung und Sicherstellung vorgegebener Randbedingungen**. Hier geben Regeln bestimmte Einschränkungen an, die ein modellierter Prozess bei seiner Ausführung einhalten muss, um ein korrektes, spezifiziertes Verhalten zu zeigen und Fehler zu vermeiden. Die Regeln dienen als Kontrollmechanismus, mit dem überprüft werden kann, ob ein Prozess den Anforderungen entsprechend korrekt modelliert wurde oder bei seiner Ausführung unerwünschte Situationen auftreten können, die im Produktivbetrieb je nach Anwendungssituation z. B. zu überflüssiger Arbeit, Beeinträchtigung der Kundenzufriedenheit, Geld- oder Zeitverlust, fehlerhaften Endprodukten oder, im schlimmsten Fall – beispielsweise im klinischen Einsatz – Personenschäden führen können. In diesem Fall stellen die Regeln harte Randbedingungen dar, die in jedem Fall eingehalten werden müssen.

Regeln dieser Form können allgemeine Vorgaben sein, etwa im Unternehmen geltende Business Rules, gesetzliche Vorschriften oder interne Policies, technische Randbedingungen, wie etwa eine beschränkte Verfügbarkeit von Ressourcen oder produktionstechnische Reihenfolgevorgaben, oder auch prozessspezifische Richtlinien, etwa Performancevorgaben, die der Prozess einhalten muss oder andere Bestimmungen, die sich beispielsweise aus einem Pflichtenheft ableiten lassen. Die Regeln können generell bereits vor der Modellierung des Prozesses vom Regelbetreuer erstellt werden, allgemeine Regeln können auch bei mehreren Prozessmodellen wiederverwendet werden.

Beispiel 3.1. *Ein Beispiel für eine (sehr einfache) Regel mit diesem Einsatzzweck stellt die Regel »Nach jedem Produktionsschritt muss eine Funktionsprüfung durchgeführt werden« dar, die beispielsweise einer Business Rule entspringen kann, oder die Regel »Das Produkt darf nach Ablauf des Mindesthaltbarkeitsdatums nicht mehr verkauft werden«, die auf gesetzlichen Vorgaben*

basiert. Eine technische Regel könnte z. B. lauten »Wenn das Produkt lackiert wurde und zuvor noch keine Funktionsprüfung durchgeführt wurde, darf diese frühestens zwei Stunden nach Ende des Lackierens durchgeführt werden«. Ein Beispiel für eine spezifische Regel für einen bestimmten Bestellprozess kann mit »Nach Eingang einer Kundenbestellung und vor Versand der Ware muss dem Kunden eine Bestätigungsmail zugeschickt werden« gegeben werden. All diese Regeln sind vorgegebene Randbedingungen, die der Prozess in jedem Fall erfüllen muss – sind sie in einer Prozessinstanz nicht erfüllt, ist dies ein Fehler.

Beim **Testen von Prozesseigenschaften** (Einsatzzweck 2) werden hingegen vom *Prozessbetreuer* für ein Prozessmodell *spezifische* Regeln definiert, mit denen zur Entwurfszeit, Laufzeit oder nach der Ausführung überprüft werden kann, ob der Prozess bestimmte erwünschte Eigenschaften hat. Dabei können Regeln z. B. zur Performanceüberwachung eingesetzt werden, um zu überprüfen, ob laufende Prozesse bestimmte erwünschte Zeitvorgaben einhalten. Eine weitere Anwendungsmöglichkeit ist die Überprüfung, ob der modellierte Prozess bestimmte spezifische Eigenschaften des zugrunde liegenden Realweltprozesses aufweist, indem diese Eigenschaften in Form von Regeln definiert werden. Allgemein ist hierdurch auch die Realisierung von Mechanismen ähnlich den aus Programmiersprachen bekannten *Assertions* möglich. Dabei handelt es sich um zur Entwurfszeit definierte Annahmen über das Prozessverhalten, die zur Laufzeit überprüft werden. Integritätsregeln werden hierbei zu einem Debugging-Werkzeug für Prozesseigenschaften.

Beispiel 3.2. *Ein Beispiel für eine solche Regel zur Überprüfung der Performance eines Prozesses könnte lauten: »Zwischen Eingang einer Bestellung und Versand der Ware sollten maximal 24 Stunden vergehen, wenn die Ware auf Lager ist und keine sonstigen Schwierigkeiten auftreten«. Das Auftreten von sonstigen Schwierigkeiten könnte erkannt werden, indem überprüft wird, ob in einer laufenden Prozessinstanz Aktivitäten vorkommen, die durch Ad-hoc-Änderungen eingefügt wurden. Diese Regel könnte also nur zur Laufzeit ausgewertet werden. Ein Beispiel für eine Assertion ist die Regel »Wenn der Schritt ›Kundengespräch‹ ausgeführt wurde und der Schritt ›Vertrag zugeschickt‹ nicht ausgeführt wurde, sollte der Schritt ›Vertrag unterzeichnen‹ nur ausgeführt werden, wenn zuvor der Schritt ›Vertrag aufsetzen‹ ausgeführt worden war (da ansonsten kein Vertrag vorhanden ist)«.*

Ein dritter möglicher Einsatzzweck besteht darin, Integritätsregeln als **Grundlage der Prozessentwicklung** einzusetzen. Hierbei werden einzelne (möglicherweise bedingte) Eigenschaften eines zu modellierenden Realweltprozesses auf Integritätsregeln abgebildet, die

somit bedingte, parametrisierte Fragmente des Prozesses darstellen. Bei der Prozessmodellierung dienen diese Regeln dann als Grundlage für das Prozessmodell, das so modelliert werden muss, dass es die Regeln erfüllt. Hierbei kann eine schrittweise Verfeinerung eingesetzt werden, um nach und nach Verletzungen der Regeln zu beheben.

Diese Nutzung von Integritätsregeln bietet sich an, wenn noch kein formaler Prozessentwurf existiert. Hier kann es sinnvoll sein, zunächst einzelne Bestandteile und ggf. wiederkehrende Muster in einem Prozess, die unter bestimmten Bedingungen auftreten, in Regeln zu fassen und auf Basis dieser Fragmente einen strukturierten Prozess zu entwickeln. In diesem Fall dienen die Regeln nur als Grundlage für den Prozessentwurf und können anschließend verworfen werden. Jedoch kann es sinnvoll sein, diese (zumindest teilweise) für einen anderen Einsatzzweck (etwa zur Konsistenthaltung) weiterzuverwenden.

Beispiel 3.3. *Als Beispiel für diesen Einsatzzweck sei die Situation angenommen, dass in einem bisher noch manuell durchgeführten und nicht modellierten Bestellungsverarbeitungsprozess stets gilt, dass nach Ankunft einer Bestellung die Verfügbarkeit der Artikel im Warenlager abgerufen wird. Wenn der Versand über einen bestimmten Lieferdienst erfolgt, wird zudem dem Kunden die Buchungsnummer zur Paketverfolgung zugeschickt. Des Weiteren gilt, dass ein Kunde angerufen wird, wenn er vierzehn Tage nach Lieferung die Rechnung noch nicht beglichen hat. Diese und weitere identifizierte Prozessfragmente werden in Regeln umgesetzt, auf deren Basis dann der Prozess modelliert werden kann.*

Um das Ziel der **Konsistenthaltung bei Änderungen** (Einsatzzweck 4) zu erreichen, werden wichtige Eigenschaften bestehender Prozessmodelle, die auch bei Änderungen des Modells im Zuge der Prozessoptimierung oder bei Ad-hoc-Änderungen erhalten bleiben sollen, in Form von Regeln modelliert. Die Regeln werden dabei praktisch aus dem Prozessmodell abgeleitet und sollen hierbei die betreffenden Prozesseigenschaften möglichst exakt widerspiegeln. Die *Überprüfung* dieser Regeln wird erst bei einer Änderung des Prozessmodells oder laufender Prozessinstanzen wirklich relevant. Ggf. können die Regeln auch im Sinne des vorher genannten Einsatzzwecks als Grundlage für eine Neumodellierung des Prozesses von Grund auf dienen, wenn dieses sinnvoller ist als eine evolutionäre Optimierung des Prozessmodells. Da in den meisten Fällen somit zwischen Regelmodellierung und Regelüberprüfung möglicherweise eine große Zeitspanne liegt, kann, obwohl die Regeln üblicherweise vom Prozessbetreuer erstellt werden, nicht unbedingt davon ausgegangen werden, dass dieser zum Zeitpunkt der Auswertung noch mit allen Regeln vollständig vertraut ist.

Beispiel 3.4. *Ein Beispiel für die Verwendung von Regeln zur Konsistenthaltung ist folgender Fall: Im aktuellen Prozessmodell sind die Schritte »Mail an Kunden senden« und »Kunde anrufen« exklusiv zueinander. Auch bei zukünftigen Änderungen am Modell oder Ad-hoc-Änderungen an laufenden Prozessen soll sichergestellt werden, dass immer einer der beiden Schritte ausgeführt wird, aber niemals beide. Daher wird eine Regel »Es muss immer entweder eine Mail an den Kunden gesendet werden oder der Kunde angerufen werden, aber niemals beides« erstellt, die diese Anforderung abbildet.*

Integritätsregeln können zur **Ergänzung der Prozessspezifikation** (Einsatzzweck 5) eingesetzt werden, um Teile des Kontrollflusses in Regeln auszulagern. Diese müssen dann nicht im Kontrollflussgraphen des Prozessmodells, sondern in Form von ergänzenden Regeln modelliert werden, die mit dem Prozess verknüpft sind. Zur Laufzeit werden dann diese Regeln überprüft und, abhängig von ihrer Erfülltheit, bestimmte Prozessentscheidungen getroffen. So können bestimmte alternative Ausführungszweige in einer Prozessinstanz automatisch abgewählt werden, wenn sich im Laufe der Ausführung ergibt, dass bei Auswahl dieser Zweige eine Regel nicht mehr erfüllt werden kann. Des Weiteren kann mit solchen Regeln eine manuelle Ausnahmebehandlung implementiert werden: Tritt bei der Ausführung einer Instanz des Prozesses eine Situation auf, in der eine Regel auf keinen Fall mehr erfüllt werden kann, wird die Ausführung angehalten und der Nutzer oder Prozessbetreuer informiert. Dieser kann dann auf Grundlage der Regel versuchen, durch Ad-hoc-Änderungen die Verletzung zu beheben, damit der Prozess fortgesetzt werden kann.

Der Einsatz von Integritätsregeln zu diesem Zweck ist sinnvoll, wenn bei der Modellierung der entsprechenden Teile im Prozess dieser zu sehr aufgebläht würde, da sich z. B. viele XOR-Blöcke mit wiederholten Abfragen oder ähnlichen Zweigen ergäben.

Beispiel 3.5. *Als Beispiel sei hier ein klinischer Prozess genannt: Es gilt, dass nach Verabreichung eines Medikaments A ein Medikament B nur verabreicht werden darf, wenn ein Arzt zu Rate gezogen wurde, da die Medikamente mögliche Wechselwirkungen besitzen. Statt bei jeder Medikamentenverabreichung von B im Prozessmodell einen XOR-Block einzufügen, in dem abhängig von der vorherigen Verabreichung von A ein Schritt »Arzt konsultieren« enthalten ist, kann einfach eine Regel definiert werden »Wenn Medikament A verabreicht wurde, muss vor Verabreichung von Medikament B ein Arzt seine Zustimmung geben«. Wenn dann versucht wird, im Prozess die Verabreichung von Medikament B auszuführen, wird die Prozessinstanz angehalten, bis ein Schritt »Ärztliche Zustimmung zu Verabreichung von Medikament B« eingefügt wurde.*

Selbstverständlich sind auch weitere Einsatzmöglichkeiten denkbar, die oben genannten decken jedoch ein großes Spektrum an verschiedenartigen Nutzungsarten mit unterschiedlichen Anforderungen in allen Phasen des Business Process Lifecycle ab und dienen daher als Grundlage für die weiteren Untersuchungen und Entwicklungen in dieser Arbeit.

3.2.2 Auswertungszeitpunkte

Bei der Nutzung von Integritätsregeln zu den oben genannten Einsatzzwecken ergibt sich, dass die Prüfung der Erfülltheit der Regeln in einem Prozessmodell bzw. Instanzen dieses Modells zu verschiedenen Zeitpunkten im Business Process Lifecycle erfolgen kann. Wir betrachten dabei folgende Phasen:

1. Bei der Prozessmodellierung
2. Bei der Überarbeitung und Optimierung von Prozessen
3. Zur Laufzeit der Prozesse
4. Nach Abschluss der Ausführung
5. Bei der Veränderung von Regeln

Die letzte Phase ist im Prozess-Lebenszyklus nicht vorgesehen und kann parallel zu den anderen Phasen erfolgen. Diese verschiedenen Zeitpunkte werden im Folgenden mit ihren spezifischen Eigenschaften genauer erläutert.

In der Phase der **Prozessmodellierung**, in der vom Prozessbetreuer ausgehend von einem Prozessentwurf oder einem Realweltprozess ein neues Prozessmodell erstellt wird, können Integritätsregeln eingesetzt werden, um entsprechend dem ersten der in Abschnitt 3.2.1 vorgestellten Einsatzzwecke die Korrektheit des neu modellierten Prozesses zu gewährleisten. Während der Entwicklung des Prozesses im Prozessmodell-Editor kann dabei überprüft werden, ob das Modell die gewünschten Regeln erfüllt – d. h. ob bei einer Ausführung des Prozesses die Regeln immer erfüllt werden oder nur bei einem bestimmten Ausführungsverlauf (also in einer bestimmten Teilmenge der möglichen Ausführungsspuren) – oder ob bestimmte Regeln bei jeder möglichen Ausführung verletzt werden. Aufbauend auf diesem Überprüfungsergebnis kann der Prozess dann während der Modellierung angepasst werden.

Werden Integritätsregeln als Grundlage der Prozessentwicklung eingesetzt (Einsatzzweck 3), spielen sie eine besonders starke Rolle bei der Modellierung, da zudem darauf geach-

tet werden muss, dass jede Regel im modellierten Prozess auch Anwendung findet, d. h. dass alle zuvor in Regeln umgesetzten Prozesseigenschaften auch im entwickelten Modell umgesetzt sind. Auch möchte der Modellierer erkennen, welche Teile des Prozesses bereits fertig modelliert sind und wo noch Handlungsbedarf besteht.

In dieser Phase der Prozessentwicklung werden zudem Regeln, die als ergänzende Teile der Prozessspezifikation eingesetzt werden (Einsatzzweck 5), parallel zum Prozessmodell modelliert. Hierbei ist wichtig, dass der Modellierer, der hier als Prozess- und Regelbetreuer auftritt, erkennen kann, inwiefern die Regeln Auswirkungen auf die Prozessausführung haben – eine integrierte Betrachtung von Prozessmodell und Regeln ist notwendig. Bei diesem Einsatz von Integritätsregeln gilt, dass sie zur Modellierungszeit zwar erfüllbar sein müssen, aber nicht grundsätzlich erfüllt sein dürfen. Es gibt also mögliche Ausführungsspuren im Prozessmodell, in denen die Regeln erfüllt sind, und andere, in denen sie nicht erfüllt sind. Wäre eine Regel in allen Spuren erfüllt, würde sie die Prozessspezifikation nicht mehr ergänzen, sondern in ihr aufgehen. Wäre die Regel in keiner Spur erfüllt, widerspräche sie dem Prozess und würde seine Ausführung von vornherein verhindern.

Auch können insbesondere gegen Ende dieser Phase Regeln, die später zur Konsistenthaltung (Einsatzzweck 4) dienen sollen, erstellt werden. Hierbei muss darauf geachtet werden, dass diese wirklich Eigenschaften des modellierten Prozesses abbilden und nicht etwa im Widerspruch zum Modell stehen. Auch können bereits Debugging-Regeln zum Testen von Prozesseigenschaften (Einsatzzweck 2) erstellt und ausgewertet werden.

Bei der **Überarbeitung und Optimierung** eines modellierten Prozessmodells gilt Ähnliches wie bei der Modellierung – wiederum möchte der Prozessbetreuer überprüfen, ob das geänderte Prozessmodell Regeln, die Bedingung für seine Korrektheit sind, verletzt. Hier werden insbesondere auch die Regeln zur Konsistenthaltung (Einsatzzweck 4) wichtig, die vom veränderten Modell weiterhin erfüllt werden müssen. Regeln, die als Grundlage der Prozessentwicklung dienten (Einsatzzweck 3), sind in dieser Phase weniger von Bedeutung, können aber zur Konsistenthaltung weiterverwendet werden.

Während der **Laufzeit** von Instanzen eines Prozessmodells ist vor allem die Überprüfung von Regeln von Bedeutung, die nicht in allen möglichen Ausführungsspuren des Modells erfüllt sind. Dabei muss für eine solche Regel während der gesamten Laufzeit laufend überprüft werden, ob sie nach der derzeitigen Ausführungshistorie in der Instanz noch erfüllbar ist. Dies gilt insbesondere, wenn Regeln als Ergänzung der Prozessspezifikation (Einsatzzweck 5) dienen. Hier soll der Endbenutzer daran gehindert werden, in seiner Ar-

beitsliste einen manuellen Schritt durchzuführen, nach dem eine Situation auftreten würde, in der eine Regel nicht mehr erfüllbar ist. Wenn solch eine Situation bei einem automatischen Schritt oder beispielsweise aufgrund eines abgelaufenen Timeouts eintritt, muss dies der Prozessbetreuer erfahren, damit er dann z. B. mit einer Überwachungsanwendung die Instanz betrachten und versuchen kann, durch Ad-hoc-Änderungen die Verletzung zu beheben.

Eine besondere Bedeutung hat die Überprüfung von Regeln zur Laufzeit, wenn Ad-hoc-Änderungen durchgeführt werden oder versucht wird, durch Schemaevolution Änderungen am Prozessmodell auf laufende Instanzen zu propagieren. Durch die Änderungen können auch Regeln, die zuvor in einer Instanz als erfüllt galten, wieder verletzbar werden, was verhindert werden soll.

Auch bei Prozessinstanzen, deren **Ausführung abgeschlossen** ist, kann die Überprüfung von Regeln über ihren Ausführungs-Logs sinnvoll sein. So können Instanzen, bei deren Ausführung Fehler oder Probleme aufgetreten sind, mit Hilfe von neu erstellten Debugging-Regeln (Einsatzzweck 2) analysiert werden. Auch nachträgliche Performance-Analysen sind auf diese Weise möglich, indem Regeln, die Zeitschranken enthalten, über einer Menge von abgeschlossenen Prozessinstanzen ausgewertet werden und so ermittelt wird, in welchen Fällen diese Schranken nicht eingehalten wurden.

Ein weiterer Zeitpunkt, zu dem eine Auswertung von Integritätsregeln erfolgen kann, ist bei der **Veränderung von Regeln**, die bereits einem oder mehreren Prozessmodellen (mit ggf. bereits laufenden Instanzen) zugeordnet sind, bzw. bei der Zuweisung neuer Regeln zu existierenden Prozessmodellen. Dabei muss für die betroffenen Prozesse bzw. ihre laufenden Instanzen überprüft werden, inwiefern sich durch die neuen bzw. veränderten Regeln die Erfülltheit verändert. Wenn der Regelbretreuer die Auswirkungen seiner Änderungen bereits erkennen kann, bevor sie gespeichert und angewendet werden, kann er entscheiden, ob die sich ergebende neue Erfüllttheitssituation korrekt ist, oder ob die neue bzw. veränderte Regel fehlerhaft ist. Werden die Änderungen angewendet, müssen aufgrund der möglicherweise veränderten Erfülltheit ggf. nicht mehr den Regeln entsprechende Prozessinstanzen gestoppt werden.

3.3 Anforderungen an das System

Ausgehend von den dargestellten Einsatzszenarien werden nun Anforderungen vorgestellt, die das System erfüllen muss, um einen solchen Einsatz zu ermöglichen.

3.3.1 Grundlegendes

Die in Abschnitt 3.2 dargestellten Szenarien zeigen, dass bei vielen Einsatzmöglichkeiten die Regeln nicht von außen vorgegeben, sondern vom Prozessbetreuer selbstständig eingesetzt werden, um die Prozessmodellierung, -überarbeitung oder -analyse zu unterstützen. Der selbstständige Einsatz erfolgt dabei jedoch nur, wenn dem Prozessbetreuer durch die Integritätsregeln Arbeit abgenommen wird, die Modellierung der Regeln schnell und einfach erfolgen kann und ihre Erfülltheit nicht durch aufwändige und umständliche Untersuchungen analysiert werden muss, sondern direkt und übersichtlich erkannt und leicht verstanden werden kann. Daher ist das vordringliche Ziel, die Nutzung von Integritätsregeln so einfach und intuitiv wie möglich zu gestalten, damit die Anwender sie »freiwillig« aktiv zur Verbesserung ihrer Prozesse einsetzen.

3.3.2 Regelklassen

Um die verschiedenen in Abschnitt 3.2.1 vorgestellten Einsatzzwecke zu den verschiedenen Zeitpunkten wie beschrieben unterstützen zu können, muss das System dem Nutzer erlauben, Regeln in verschiedene Klassen einzuteilen. Wir schlagen folgende Einteilung vor:

- **Klasse 1:** Regeln, die auf keinen Fall verletzbar sein dürfen
- **Klasse 2:** Regeln, die bei der Ausführung nicht verletzt werden dürfen
- **Klasse 3:** Regeln, über deren Verletzung informiert wird
- **Klasse 4:** Regeln, die zur Laufzeit ignoriert werden

Bei den **Regeln, die auf keinen Fall verletzbar sein dürfen** (Klasse 1) handelt es sich um gesetzliche Regelungen, strikte Business Rules und sonstige Policies, bei denen bereits zur Modellierungszeit des Prozesses feststehen muss, dass sie bei jeglicher Ausführung des Prozesses nach dem Modell, also in jeder möglichen Ausführungsspur (abgesehen von möglichen Ad-hoc-Änderungen) niemals verletzt sein dürfen. Bei der Ausführung

von Prozessinstanzen muss zudem darauf geachtet werden, dass die Regeln bei Ad-hoc-Änderungen nicht verletzt werden. Diese Regeln sind insbesondere für den ersten Einsatzzweck, der Sicherstellung vorgegebener Randbedingungen, von Bedeutung, da sie es erlauben, die Verletzung der Regeln von vornherein zu verhindern. Wenn die Erfülltheit zur Entwurfszeit jedoch nicht vollständig bestimmt werden kann, etwa bei Zeitbedingungen, können Regeln dieser Klasse nicht eingesetzt werden.

Die **Regeln, die bei der Ausführung nicht verletzt werden dürfen** (Klasse 2), können hingegen in einem Prozessmodell durchaus in einzelnen (aber nicht allen) möglichen Ausführungsspuren verletzt sein. Zur Laufzeit einer Prozessinstanz muss jedoch die Ausführungssoftware sicherstellen, dass bei Entscheidungsknoten alternative Zweige, bei deren Auswahl eine Regel nicht mehr erfüllbar ist, nicht ausgewählt werden können. Auch bei Ad-hoc-Änderungen muss sichergestellt werden, dass durch sie keine Regel unerfüllbar wird. Sollte es ansonsten zu einer nicht vorher bestimmbaren Situation kommen, in der eine solche Regel nicht mehr erfüllt werden kann (beispielsweise aufgrund von Datenelementen oder auslaufenden Zeitschranken), muss das System den Prozess anhalten und den Prozessbetreuer informieren. Dieser Typ von Regeln wird insbesondere beim Einsatzzweck der Ergänzung der Prozessspezifikation (Einsatzzweck 5) benötigt sowie zur Sicherstellung von vorgegebenen Randbedingungen (Einsatzzweck 1) und wichtigen Assertions (Einsatzzweck 2), wenn die Erfülltheit zur Entwurfszeit unbestimmt ist.

Regeln, über deren Verletzung informiert wird (Klasse 3), können die Ausführung eines Prozesses nicht verhindern oder für seine Unterbrechung sorgen. Ist bei der Prozessmodellierung eine solche Regel in allen oder einigen möglichen Ausführungsspuren verletzt, muss der Betreuer eine Warnung erhalten. Tritt zur Laufzeit in einer Instanz ein Zustand auf, in der eine solche Regel nicht mehr erfüllt werden kann, wird der Prozessbetreuer informiert, aber die Prozessausführung wird fortgesetzt. Regeln dieser Klasse sind für Laufzeit-Tests (Einsatzzweck 2) wichtig, da sie die Ausführung (evtl. bereits im Produktivbetrieb) laufender Prozesse nicht beeinträchtigen und dennoch Laufzeitinformationen sammeln können.

Regeln, die zur Laufzeit ignoriert werden (Klasse 4), haben keinerlei Einfluss auf die Prozessausführung. Auf die Verletzung bzw. Verletzbarkeit dieser Regeln wird lediglich bei der Modellierung im Prozessmodell-Editor bzw. bei der Überwachung laufender oder abgeschlossener Prozessinstanzen hingewiesen. Bei einer Verletzung zur Laufzeit kann ggf. ein Log-Eintrag erstellt werden, dies ist aber nicht zwingend erforderlich, da die Regeln

auch nachträglich über den Ausführungs-Logs der abgeschlossenen Instanzen überprüft werden können. Diese Regeln werden insbesondere für einfache Debugging-Aktionen und Prozesstests (Einsatzzweck 2) benötigt: Auf diese Weise können Tests an einem Prozessmodell durchgeführt werden, die nur zur Modellierungszeit, nicht aber zur Laufzeit von Bedeutung sind.

Für bestimmte Einsatzzwecke, etwa als Grundlage für die Prozessentwicklung (Einsatzzweck 3) und zur Konsistenthaltung (Einsatzzweck 4) sowie für Tests (Einsatzzweck 2) struktureller Prozesseigenschaften, können Regeln einer beliebigen Klasse verwendet werden, je nachdem, wie stark die angegebene Regel sich bei testweisen Ausführungen des Prozesses auswirken soll bzw. ob die Regeln nach der Modellierung auch zu anderen Zwecken weiterverwendet werden sollen.

3.3.3 Ergebnisauswertung

Generell lässt sich nach Betrachtung der angegebenen Einsatzszenarien feststellen, dass eine Überprüfung von Integritätsregeln bei der Erstellung, Bearbeitung, Ausführung und Analyse von Prozessen, also durchgängig in jeder Phase des Prozesslebenszyklus erfolgen muss.

Zur Laufzeit muss die Prozess-Engine dafür sorgen, dass kein tatsächlich ausgeführter Schritt eine Regel, die zur Laufzeit erfüllt sein muss (also aus Klasse 1 oder 2 in Abschnitt 3.3.2), verletzt bzw. die Prozessinstanz bei einer solchen Verletzung sofort angehalten wird. Für diese Arbeit und somit die Betrachtung der Analyse aus Anwendersicht ist hierbei wichtig, dass der Prozessbetreuer sofort und auf geeignete Weise über Instanzen, die angehalten wurden, sowie über aufgetretene Verletzungen von Regeln aus Klasse 3 informiert wird. Diese Information muss nicht sehr ausführlich sein, eine genaue Analyse kann anschließend auf Basis des der Instanz zugeordneten (ggf. durch Ad-hoc-Änderungen veränderten) Prozessmodells in einem Analysewerkzeug erfolgen.

Auch während der Modellierung bzw. Bearbeitung von Prozessmodellen im Prozessmodell-Editor und der Betrachtung laufender oder abgeschlossener Prozessinstanzen in einem Analysewerkzeug sollten die zugeordneten Regeln laufend neu ausgewertet werden und kein manuelles Starten der Auswertung erforderlich sein. Auf diese Weise hat der Prozessbetreuer stets einen Überblick über die Regelkonformität des Prozesses und kann bei Änderungen, die dazu dienen, Regelverletzungen zu beheben, sofort erkennen, ob dies er-

folgreich war oder ggf. durch die Änderungen andere Verletzungen verursacht wurden. Von besonderer Wichtigkeit ist dies bei Regeln, die als Grundlage der Prozessentwicklung bzw. -optimierung dienen (Einsatzzweck 3) – hierbei wird der Prozess laufend angepasst, um die Regeln zu erfüllen, daher ergeben sich ständig Änderungen in der Erfülltheit, die das weitere Vorgehen bestimmen können. Aber auch bei Regeln aller anderen Typen ist eine laufende Auswertung von Vorteil, da der Prozessbetreuer komfortabler arbeiten kann, wenn er die Überprüfung nicht manuell starten muss und Fehler im Prozess sofort erkannt werden können. Insbesondere bei komplexen Regeln, deren Erfülltheit nicht in ihrer Vollständigkeit direkt erkannt werden kann, besteht so die Möglichkeit, Fehler auf experimentelle Weise durch testweise Änderungen einzugrenzen. Ebenfalls kann so verhindert werden, dass durch ein Versäumnis, die Analyse manuell zu starten, Verletzungen übersehen werden.

Hieraus lässt sich eine weitere äußerst wichtige Forderung an das System ableiten: Die Regelauswertung muss möglichst schnell und effizient erfolgen. Nur so lässt sich ein flüssiges Arbeiten gewährleisten und es ist sichergestellt, dass bei der Bearbeitung des Prozesses keine veralteten Erfülltheitsinformationen angezeigt werden, die den Bearbeiter evtl. verwirren. Im Zweifelsfall ist es sinnvoll, Geschwindigkeit vor Genauigkeit zu stellen, also ein weniger umfangreiches und genaues Ergebnis anzuzeigen statt zu lange mit der Darstellung des Ergebnisses zu warten. Der Prozessbetreuer kann dieses Ergebnis dann selbst durch eigene Maßnahmen wie etwa experimentelle Änderungen oder manuelle Einschränkungen der Auswertung, etwa die Beschränkung auf einzelne Regeln, genauer untersuchen.

Ein Ergebnis ungenau anzugeben, bedeutet nur, dass Details weggelassen werden. Die *Korrektheit* des Ergebnisses muss selbstverständlich auch bei einer ungenauen Angabe immer gewährleistet sein, da korrekte Ergebnisse die Grundlage für eine sinnvolle Nutzung von Integritätsregeln bilden und es zudem auch für die Akzeptanz des Systems wichtig ist, dass die Anwender Vertrauen in die Korrektheit haben können. Wenn das korrekte Ergebnis nicht genau ermittelt werden kann, muss dies dem Anwender deutlich gemacht werden, etwa durch eine Formulierung wie »Die Regel ist *möglicherweise* verletzt«.

Auch bei der Bearbeitung von Regeln, die bereits Prozessen zugeordnet sind, muss eine laufende Analyse der Erfülltheit über den entsprechenden Prozessmodellen durchgeführt werden. Wenn die Änderung auf laufende Instanzen propagiert werden soll, müssen auch diese überprüft werden. Somit kann der Regelbetreuer, wie in Abschnitt 3.2.2 angesprochen, bereits bei der Bearbeitung sehen, ob die veränderte Regel von den bereits existieren-

den Modellen und Instanzen erfüllt wird. Da diese Auswertung bei sehr vielen laufenden Instanzen jedoch recht zeitaufwändig sein kann, kann eine Möglichkeit angeboten werden, sie zu deaktivieren und die Überprüfung bei Bedarf manuell durchzuführen.

3.3.4 Ergebnisdarstellung

Um den Erfülltheitsgrad eines Prozessmodells bzw. einer Instanz überblicken zu können, ist eine übersichtliche Darstellung der Erfülltheit aller überprüften Regeln notwendig, aus der erkennbar ist, wie viele und welche Regeln erfüllt bzw. verletzt sind. Für Regeln, die als Grundlagen der Prozessmodellierung dienen (Einsatzzweck 3), muss für den Prozessmodellierer außerdem erkennbar sein, welche Regeln überhaupt vom Prozess abgedeckt werden, d. h. bei welchen Regeln der Bedingungsteil überhaupt vom Prozess erfüllt werden kann. Schließlich sollen alle Regeln vom Prozess abgebildet werden und nicht einzelne Regeln übersehen werden, da die Situation, in der sie gültig sind, nicht modelliert wurde.

Bei der Bearbeitung einer Integritätsregel, die bereits Prozessen zugewiesen ist, ist analog hierzu eine Übersicht notwendig, in der sichtbar ist, in welchen dieser Prozessmodelle bzw. ihrer laufenden Instanzen die geänderte Regel noch erfüllt ist und in welchen nicht. Auch die Möglichkeit, die Werte für mehrere Instanzen zu aggregieren, ist für eine bessere Übersichtlichkeit sinnvoll. Bei Auswahl eines Modells bzw. einer laufenden Instanz muss dann eine genauere Analyse möglich sein, z. B. indem zur Prozessmodellierungs- bzw. Instanzanalyse-Anwendung gewechselt wird.

Aus der Tatsache, dass Integritätsregeln als Grundlage für die Prozessmodellierung verwendet werden können, ergibt sich, dass die Ergebnisse der Erfülltheitsüberprüfung so dargestellt werden sollten, dass für den Anwender erkennbar ist, wo im Prozess bzw. der Instanz sich die Regeln auswirken, so dass die Beziehung von Regelerfülltheit und Prozessstruktur sichtbar wird. Auf diese Weise ist es dem Anwender möglich, festzustellen, in welchen Teilen des Prozesses (bestimmte Blöcke, einzelne Schritte) noch Handlungsbedarf besteht, um die Regeln zu erfüllen.

Auch sollte es möglich sein, für einzelne Regeln einfach zu analysieren, auf welche Weise sie genau verletzt bzw. erfüllt werden, indem sichtbar gemacht wird, wo im Prozessmodell bzw. einer laufenden Instanz die Aktivitätentypen aus der Regel vorkommen und welche Forderungen aus der Regel von welchen Knoten im Prozess aus welchen Gründen verletzt oder erfüllt werden. Zudem ist es bei der Betrachtung von Prozessmodellen

wichtig, erkennbar zu machen, in welchem Prozesskontext die Regel verletzt oder erfüllt ist, also bei welcher Auswahl unter alternativen Zweigen, welcher Ausführungsreihenfolge zwischen Knoten und unter welchen zeitlichen Rahmenbedingungen. Insbesondere bei Debugging-Regeln bzw. Assertions (Einsatzzweck 2), die vom Prozessmodellierer definiert werden und bei denen somit somit großes Prozess- und Regelwissen vorliegt, ist es wichtig, Verletzungen dieser Regeln möglichst genau analysierbar zu machen, damit die Ursache dieser ggf. erst zur Laufzeit auftretenden, unerwarteten Probleme schnell und exakt ermittelt werden kann.

Zur genauen Analysierbarkeit gehört auch, dass der Benutzer erkennen kann, in welcher Beziehung verschiedene Verletzungen von Regeln zueinander stehen. So kann es notwendig sein, mehrere Verletzungen zu beheben, damit eine Regel erfüllt ist (wenn die verletzten Forderungen in der Regel Und-Verknüpft sind oder eine Regel für *alle* Knoten eines bestimmten Typs erfüllt sein muss) oder nur einzelne Verletzungen behoben werden müssen (z. B. bei Oder-verknüpften Forderungen). Da die Beziehungen zwischen mehreren Verletzungen bei Regeln mit vielen umfangreichen Folgegliedern durchaus komplex sein können, ist es jedoch ebenfalls wichtig, den Benutzer nicht mit zu viel Informationen zu verwirren und zu überfordern. Es kann sinnvoller sein, die Beziehungen weniger genau anzugeben, etwa in einer linearen Liste statt einer verschachtelten Struktur. Um die genaue Struktur herauszufinden, kann der Anwender die zugrunde liegende Regel zurate ziehen oder auf experimentelle Weise mit der Behebung einer Verletzung beginnen und überprüfen, inwiefern dabei auch andere Verletzungen behoben werden.

Beispiel 3.6. *Eine Regel fordert, dass in einem Prozess entweder Aktivität A oder Aktivitäten B und C vorkommen müssen. Im betrachteten Prozessmodell kommt jedoch keine der Aktivitäten vor. Um die Regel zu erfüllen, kann nun entweder Aktivität A oder Aktivität B und C hinzugefügt werden. Es ergibt sich eine verschachtelte Beziehungsstruktur der Verletzungen der Regel:* $\big($(*es fehlt Aktivität A*) $\vee$ (*es fehlt Aktivität B*$\wedge$ *es fehlt Aktivität C*)$\big)$. *Statt diese Struktur (die im Beispiel einfach ist, aber bei realen Regeln durchaus komplexer sein kann) anzugeben, kann es sinnvoller sein, eine lineare Liste darzustellen:* (*es fehlt Aktivität A*; *es fehlt Aktivität B*; *es fehlt Aktivität C*). *Um die Beziehungen zwischen den Verletzungen zu erkennen, kann der Anwender die Regel betrachten. Alternativ kann er experimentell mit der Behebung beginnen: Fügt er eine Aktivität A in den Prozess ein, verschwinden die anderen beiden Verletzungen aus der Liste und er erkennt, dass sie in einer Alternativ-Beziehung standen. Fügt er jedoch eine Aktivität B ein, bleiben die anderen Verletzungen bestehen. Fügt er nun noch eine Aktivität A oder C ein, sind beide Verletzungen behoben – falls er sich für A entscheidet, war das Einfügen von B jedoch überflüssig. Durch die*

fehlende Genauigkeit können sich also unnötige Kompensationsschritte ergeben. (In der in Kapitel 4 vorgestellten Ergebnisstruktur wird dies durch die Einführung von Alternativmengen vermieden).

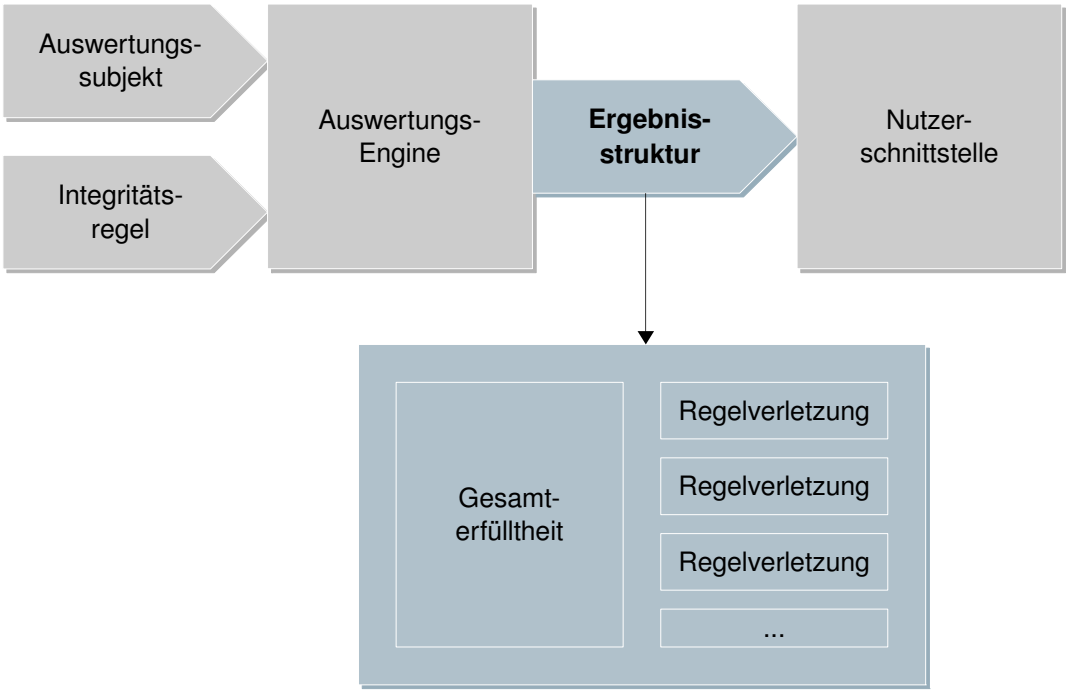
In Fällen, in denen die Regeln vorgegeben und dem Prozessbetreuer weniger geläufig sind (etwa bei Einsatzzweck 1), ist es wichtig, dass dieser möglichst genaue Hinweise bekommt, wie er durch Änderungen und Ergänzungen des Prozessmodells bzw. einer betrachteten Instanz erreichen kann, dass die Regeln erfüllt werden. Auch wenn Regeln als Grundlage der Prozessentwicklung und somit als Vorlagen für den Prozess eingesetzt werden, ist es sinnvoll, darzustellen, wie die durch eine Regel dargestellte Forderung in Prozessstrukturen umgesetzt und so in den Prozess eingebunden werden kann. Zwar wird es nicht möglich und auch nicht sinnvoll sein, darzustellen, durch welche exakten Schritte die notwendigen Änderungen durchgeführt werden können, da es hierfür meist sehr viele Möglichkeiten gibt, z. B. an welcher Stelle fehlende Aktivitäten eingefügt werden können. Sehr wohl sollte jedoch durch eine möglichst nahe an der Prozessbeschreibung orientierte Darstellung der gewünschten Eigenschaften des Prozesses erreicht werden, dass der Prozessbetreuer daraus leicht die Möglichkeiten ableiten kann, die ihm zur Verfügung stehen, um die Verletzung zu beheben.

Die den verschiedenen in Abschnitt 3.3.2 vorgestellten Regelklassen zugeordneten Regeln dienen unterschiedlichen Zwecken und werden zu unterschiedlichen Zeiten im Prozesslebenszyklus benötigt. Da sie jedoch alle auch zur Entwurfszeit ausgewertet werden, sollte das System eine Möglichkeit anbieten, die Regeln nach Klassen zu filtern und so nur die Ergebnisse der Auswertung der gerade gewünschten Regelklassen anzuzeigen. So könnten beispielsweise bei der Betrachtung der Erfülltheit einer im produktiven Einsatz laufenden Prozessinstanz mit einem Instanzanalyse-Werkzeug Regeln, die zur Laufzeit ignoriert werden, oder Debugging-Regeln ausgeblendet werden. Um eine feingliedrigere Unterscheidung zu ermöglichen und Regeln gleicher Klasse, aber unterschiedlichen Zwecks zu unterscheiden, sollte ebenfalls eine Möglichkeit angeboten werden, Regeln nach anderen Gesichtspunkten zu gruppieren (z. B. indem sie in einer gemeinsamen Datei gespeichert werden) und auch einzelne dieser Gruppen ein- und auszublenden.

In den Fällen, in denen Endanwender der Client-Anwendungen mit Regelverletzungen konfrontiert werden, z. B. wenn sie einfache Ad-hoc-Abweichungen wie das Überspringen von Schritten oder das Einfügen eines neuen Schrittes vornehmen, müssen sie eine klare, einfache Rückmeldung erhalten, dass bei der gewünschten Aktion eine Regel verletzt würde. Handelt es sich beim Anwender um einen Nutzer mit größerem Prozesswissen,

z. B. da er ebenfalls Prozess- oder Regelbetreuer ist (und den Prozess beispielsweise nur testweise ausführt), sollte die Möglichkeit bestehen, in eine erweiterte Ansicht zu wechseln, indem beispielsweise die Prozessinstanzanalyse-Anwendung geladen wird, wo die Verletzung genauer untersucht werden kann.

4 Ergebnisstruktur



Die **Ergebnisstruktur** stellt die Verbindung zwischen der verwendeten Regelauswertungs-Engine und der Nutzerschnittstelle her. Es handelt sich dabei um eine Datenstruktur, welche die Ergebnisse der Regelauswertung aufnimmt und auf deren Basis anschließend die Darstellung für den Benutzer erfolgt.

In diesem Kapitel wird eine Ergebnisstruktur beschrieben, die sowohl den Anforderungen der Auswertungs-Engines gerecht wird als auch eine anwenderangemessene Darstellung des Ergebnisses ermöglicht. Zunächst werden in Abschnitt 4.1 und 4.2 die Anforderungen an diese Struktur und ihre Voraussetzungen dargestellt, danach in den Abschnitten 4.3 – 4.5 die einzelnen Bestandteile der Struktur konzeptionell beschrieben, hergeleitet und definiert. In Abschnitt 4.6 erfolgt eine Erweiterung für ungenauere Ergebnisse, in Abschnitt 4.7 wird die Umsetzung der Ergebnisse einiger Algorithmen in die Struktur erläutert.

4.1 Anforderungen

Anforderungen an die Ergebnisstruktur werden von Seiten der Auswertungsalgorithmen gestellt, deren Ergebnisse in der Struktur repräsentiert werden sollen. Des Weiteren wurden in Abschnitt 3.3 verschiedene Anforderungen der Anwender an die Ergebnisdarstellung formuliert, die sich wiederum auf die Ergebnisstruktur auswirken, da auf der Benutzeroberfläche nur Informationen dargestellt werden können, die in der Ergebnisstruktur repräsentiert sind (vgl. Abb. 4.1).

Anforderung der Algorithmen: **Flexibilität in Ausdrucksmächtigkeit und Aussagekraft**. Da die Ergebnisstruktur darauf ausgelegt sein soll, möglichst unabhängig von den verwendeten Auswertungsalgorithmen zu sein, und bereits die in Abschnitt 2.4 vorgestellten Ansätze große Unterschiede in der Ausdrucksmächtigkeit und Struktur der möglichen Ergebnisse aufzeigen, ist es von großer Bedeutung, dass die Ergebnisstruktur flexibel genug ist, verschiedenartige Ergebnisse aufzunehmen. So sollen die Ergebnisse komplexer Algorithmen, die die Erfülltheit für jede mögliche Ausführungsspur exakt angeben, genauso aufgenommen werden können wie die Ergebnisse von Algorithmen, die lediglich eine allgemeine Aussage machen, ob eine untersuchte Regel im Auswertungssubjekt erfüllt ist und bei Nichterfülltheit ggf. ein Gegenbeispiel einer Ausführungsspur angeben, in der die Regel verletzt ist. Ebenso soll unterstützt werden, wenn Algorithmen in bestimmten Fällen keine aussagekräftigen Ergebnisse erzeugen können.

Anforderung der Anwender: **Genaue Ergebnisdarstellung**. Um den Anforderungen der Nutzer zu entsprechen, muss in der Ergebnisstruktur möglichst genau beschrieben werden,

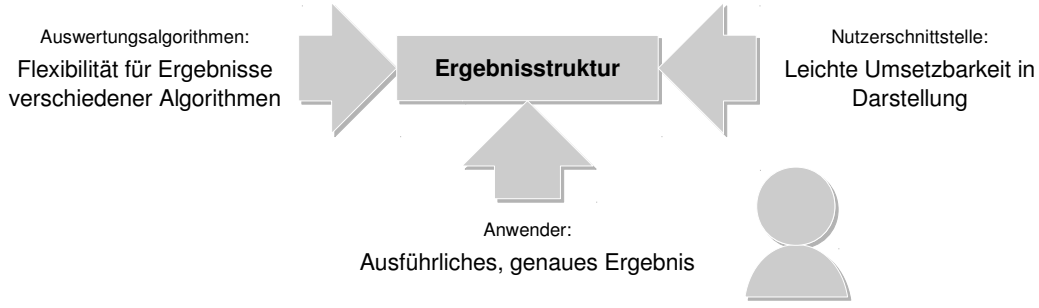


Abbildung 4.1: Anforderungen an die Ergebnisstruktur

unter welchen Umständen und auf welche Weise die Nichterfüllung einer Regel auftritt. Dies sollte zum einen in Bezug auf den jeweiligen Ausführungszustand des *Prozesses* geschehen, damit der Nutzer die Situationen in der Prozessausführung erkennt, in denen Probleme auftreten. Des Weiteren ist eine Beschreibung in Bezug auf die *Regel* notwendig, damit es möglich wird, dem Nutzer Hinweise zu geben, welche Forderung der Regel verletzt wird und wie er den Prozess ändern kann, um diese Verletzung der Regel zu beheben.

Anforderung der Nutzerschnittstelle: **Leichte Abbildung**. Die Forderung nach einer ständigen Auswertung während der Bearbeitung von Prozess und Regeln hat ebenfalls indirekt Auswirkungen auf die Ergebnisstruktur. Da Ergebnisse nach Möglichkeit in Echtzeit vorliegen sollen, sollte sie so aufgebaut sein, dass sie ohne zeitaufwändige Umrechnungs- bzw. Umkodierungsvorgänge die Ergebnisdaten des verwendeten Algorithmus aufnehmen kann und diese wiederum schnell für die Darstellung auf der Benutzeroberfläche aufbereitet werden können. Dies hat nicht nur Auswirkungen auf die logische Ergebnisstruktur, sondern auch auf ihre Umsetzung in einer Datenstruktur im Rechner, auf welche im Zuge der Beschreibung der Implementierung in Abschnitt 6.6 eingegangen wird.

4.2 Voraussetzungen

Da das Ziel ist, möglichst flexibel verschiedene Auswertungsalgorithmen zu unterstützen, muss das Ergebnis, das diese Algorithmen liefern, keinerlei Voraussetzungen erfüllen, außer, dass es korrekt sein muss. Ist es der Auswertungs-Engine nicht möglich, ein korrektes Ergebnis zu liefern, etwa da die Auswertung zu komplex wäre oder das Ergebnis nur zur Laufzeit bestimmt werden kann, darf kein möglicherweise inkorrektes Ergebnis zurückgegeben werden, sondern es muss ein korrektes unscharfes Ergebnis (siehe Abschnitt 4.6) angegeben werden.

Die hier vorgestellte Ergebnisstruktur erlaubt somit, auch recht einfache, wenig aussagekräftige Ergebnisse darzustellen, die nur aus der Information bestehen, ob die Regel im Auswertungssubjekt erfüllbar oder verletzt ist. Um den Anwender bestmöglich zu unterstützen, ist es selbstverständlich jedoch immer sinnvoll, wenn ein Algorithmus möglichst viele Ergebnisdetails liefert.

4.3 Aufbau

Um den Nutzeranforderungen entsprechend eine möglichst genaue Repräsentation des Auswertungsergebnisses zu erreichen, muss die Ergebnisstruktur nicht nur die Information umfassen, in welchen Ausführungsspuren die untersuchten Regeln erfüllt bzw. nicht erfüllt sind, sondern auch angeben, welche Teile welcher Regeln dabei jeweils in welcher Weise verletzt sind und so zur Nichterfülltheit beitragen.

Eine Möglichkeit hierzu ist, beides verbunden in einer gemeinsamen Struktur darzustellen, welche die verschiedenen Ausführungsspuren enthält, in denen die Regeln nicht erfüllt sind und dazu jeweils den verletzten Teil der entsprechenden Regel angibt. Auf diese Weise kann ein Höchstmaß an Ergebnisgenauigkeit erzielt werden, da eine exakte Zuordnung von verletzten Regelteilen zum Gesamtergebnis der Auswertung möglich ist.

Hierbei besteht jedoch das Problem, dass durch die enge Koppelung der Angabe von verletzten Regelteilen an die Beschreibung der Erfülltheit des Prozesses Flexibilität verloren geht. So kann jeweils nur dargestellt werden, welche Regelteile in einer bestimmten Ausführungsspur verletzt sind, jedoch nicht, in welchen Ausführungsspuren ein bestimmter Regelteil verletzt ist. Hierfür würde eine invers aufgebaute Struktur benötigt, in der dann wiederum die Betrachtung der Gesamterfülltheit schwierig wäre, da die Information über die Erfülltheit auf die verschiedenen verletzten Regelteile verteilt wäre.

Ein weiteres Problem der verzahnten Darstellung stellt die mangelnde Möglichkeit dar, ungenaue bzw. unvollständige (unscharfe) Ergebnisse darzustellen. Wenn beispielsweise ein Algorithmus nur eine einzelne Belegung der Variablen einer Regel liefert, mit der diese Regel verletzt ist, lässt sich aus dieser Information keine komplette Erfülltheitsstruktur ableiten, die die Erfülltheit in jedem Prozessteil sowie die entsprechenden Gründe für die Nicht-Erfülltheit beinhaltet.

Daher wird in dieser Arbeit ein Ansatz gewählt, bei dem die Ergebnisstruktur aus zwei voneinander unabhängigen Strukturen besteht, der *Gesamterfülltheit* und der *Regelverletzungsmenge*.

Die **Gesamterfülltheit** soll einen exakten Überblick darüber geben, in welchen Fällen das Auswertungssubjekt die überprüften Integritätsregeln erfüllt und in welchen Fällen nicht. In welcher Weise die Regeln in den Fällen, in denen sie nicht erfüllt sind, konkret verletzt werden, ist aus der Gesamterfülltheit nicht ersichtlich.

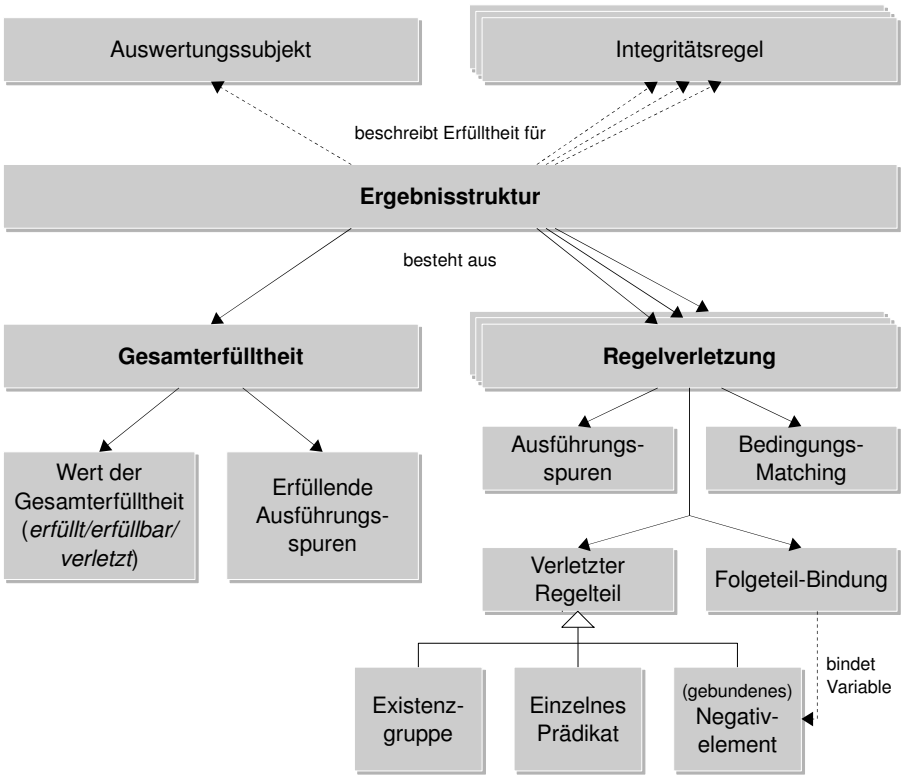


Abbildung 4.2: Überblick über den Aufbau der Ergebnisstruktur

Zur genauen Analyse der Verletzungen von Regeln dient die **Regelverletzungsmenge**. Sie stellt die Verbindung zwischen konkreten Verletzungen einer Integritätsregel bzw. eines Teils davon und den sie verletztenden Teilen des Auswertungssubjekts her. Dabei ist sie vollständig von der Gesamterfülltheitsstruktur unabhängig, erlaubt also keine direkte Untersuchung der Erfülltheit der Gesamtregel(n), sondern die ausführliche Analyse einzelner Verletzungen. Die Regelverletzungsmenge besteht aus sog. **Regelverletzungen**.

Durch die Verwendung kleinerer, unabhängiger Elemente ist die Ergebnisstruktur besser an die Darstellung für den Anwender angepasst, da sich mehrere einfache Objekte besser auf klare und verständliche Weise darstellen lassen als eine komplexe, verschachtelte Struktur. Abb. 4.2 zeigt einen Überblick über den Gesamtaufbau.

Konzeptionelle Grundlage der Regelauswertung ist, wie bereits in Abschnitt 2.3 beschrieben, eine Menge von Ausführungsspuren (*Spurmenge*). Erfolgt die Auswertung über einem Prozessmodell (Prozess-Template), so sind dies alle in diesem Modell möglichen Ausfüh-

rungsspuren. Diese Menge ist – konzeptionell betrachtet – üblicherweise unendlich, wenn man die Ausführungsdauer eines Knotens als rationale oder reelle Zahl angibt. Erfolgt die Auswertung über einer oder mehreren laufenden Prozessinstanzen, ist diese Menge beschränkt auf die in den Instanzen noch möglichen Ausführungsspuren – auch diese Menge enthält üblicherweise noch unendlich viele Elemente. Bei einer oder mehreren abgeschlossenen Prozessinstanzen ist für jede Instanz in der (dann endlichen) Menge genau eine Ausführungsspur enthalten.

Auch als Grundlage der Ergebnisstruktur werden Mengen von Ausführungsspuren verwendet. Diese sind stets (weiterhin praktisch immer unendliche) Teilmengen des Auswertungssubjekts. Für die logische Betrachtung werden diese im Folgenden zunächst als abstrakte mathematische Objekte betrachtet, bevor wir in Abschnitt 6.5 für die praktische Umsetzung eine endliche Struktur für sie entwickeln.

4.4 Gesamterfülltheit

Die Gesamterfülltheit einer Menge von Regeln über einem Auswertungssubjekt ist folgendermaßen definiert:

Definition 4.1 (Gesamterfülltheit)**.** Sei T die betrachtete Menge von Ausführungsspuren (das Auswertungssubjekt) und R eine Menge von Integritätsregeln in prädikatenlogischer Form (vgl. Abschnitt 2.3.3). Sei weiter $F \subseteq T$ die Menge aller Ausführungsspuren aus T, in denen alle Regeln aus R erfüllt sind ($F = \{t \in T | \forall\, r \in R : struct_{t,r} \vDash r\}$).

So ist die **Gesamterfülltheit** $E(T, R)$ definiert als:

$$E(T,R) = (e, F)$$

$$\text{mit } e = \begin{cases} \text{erfüllt} & \text{falls } F = T, \\ \text{erfüllbar} & \text{falls } \emptyset \subsetneqq F \subsetneqq T, \\ \text{verletzt} & \text{falls } F = \emptyset. \end{cases}$$

e wird dabei als *Wert der Gesamterfülltheit* bezeichnet, F als *Menge der erfüllenden Spuren*.

Die Gesamterfülltheit einer einzelnen Regel ergibt sich als Sonderfall dieser Definition.

Definition 4.2 (Nicht passende Regel)**.** R heißt **nicht passend** zu T, wenn in allen Ausführungsspuren aus T für alle Regeln aus R der Bedingungsteil nicht erfüllt ist, also $\forall\, r \in R, \forall\, t \in T : struct_{t,r} \not\models antecedent(r)$. In diesem Fall ist die Gesamterfülltheit stets *erfüllt*.

4.5 Regelverletzungen

Eine **Regelverletzung** beschreibt einen Teil einer Integritätsregel, eine Bindung von Variablen an Knotenausführungen sowie eine Menge von Ausführungsspuren, wenn in diesen Spuren sowohl die gesamte Regel als auch der angegebene Regelteil für die Knoten aus der Bindung verletzt ist.

Konkret besteht eine Regelverletzung aus folgenden Elementen:

- einem Verweis auf die verletzte Regel
- einer Menge von Ausführungsspuren, in denen die Verletzung auftritt
- einer Bindung der freien (positiven) Variablen des Bedingungsteils[1] an Knotenausführungen (**Bedingungs-Matching**)
- dem betroffenen Teil der Regel (**Verletzter Regelteil**)
- der Bindung der freien Variablen dieses Regelteils (**Folgeteil-Bindung**).

Für eine solche Regelverletzung muss gelten:

- in allen Ausführungsspuren der angegebenen Spurmenge ist die Regel verletzt
- bei Bindung der freien Variablen aus dem Bedingungsteil an die Knotenausführungen aus dem Bedingungs-Matching ist der Bedingungsteil in der Spurmenge erfüllt
- bei zusätzlicher Bindung der freien Variablen aus dem verletzten Regelteil an die entsprechenden Knotenausführungen ist dieser Regelteil verletzt.

Auf Grundlage der Struktur der zugrunde liegenden Regel lassen sich Regelverletzungen in Beziehung zueinander setzen: Eine Regelverletzung rv' heißt **Alternativverletzung** zu rv, falls es eine Möglichkeit gibt, die Verletztheit der Regel in den betrachteten Ausführungsspuren zu beheben, für die rv' behoben werden muss, rv aber nicht.

[1] Eine freie Variable im Kontext einer prädikatenlogischen Formel ist definiert als eine Variable, die in dieser Formel nicht durch einen Quantor gebunden wird. Die positiven Variablen des Bedingungsteils sind zwar in der Gesamtregel gebunden, treten im Bedingungsteil jedoch als freie Variablen auf, da sie außerhalb quantifiziert werden. Da sie im Bedingungsteil nicht durch einen Quantor gebunden werden, kann eine spezifische Bindung angegeben werden, was hier durch das Bedingungs-Matching geschieht.

4.5.1 Herleitung

Im Folgenden wird hergeleitet und erläutert, auf welche Weise eine Integritätsregel in einer Ausführungsspur verletzt sein kann und welche Arten von Teilformeln sinnvollerweise als *verletzte Regelteile* für Regelverletzungen verwendet werden können.

Wir betrachten hierzu jeweils eine Ausführungsspur und eine Regel, die in dieser Spur nicht erfüllt ist. Es muss mindestens eine Bindung der freien Variablen aus dem Bedingungsteil der Regel an Knotenausführungen in der Spur geben, mit der der Bedingungsteil erfüllt ist (wäre er für alle verletzt, wäre die Regel erfüllt). Wenn der Folgeteil mit dieser Bindung verletzt ist, ist der gebundene Bedingungsteil für die Verletzung der Regel mitverantwortlich. Ein erfüllter Bedingungsteil ist jedoch stets nur die Voraussetzung für eine Verletzung, daher stellt er keine eigenständige Regelverletzung dar. Stattdessen wird die entsprechende Bindung der freien Variablen aus dem Bedingungsteil als *Bedingungs-Matching* in jede Regelverletzung im Folgeteil, die bei dieser Bedingungs-Belegung auftritt, aufgenommen.

Für den Folgeteil gilt bei einer solchen Bindung der Bedingungs-Variablen: Da er verletzt ist, müssen auch alle Folgeglieder (aufgrund ihrer Oder-Verknüpfung) verletzt sein. In Abschnitt 2.3.3 wurden *Existenzgruppen* eingeführt, welche die minimalen Elemente angeben, in die sich ein Folgeglied aufteilen lässt. Da alle Folgeglieder verletzt sind, gilt: In jedem Glied muss mindestens eine der Existenzgruppen verletzt sein, da diese Und-verknüpft sind. Jede verletzte Existenzgruppe bildet den *verletzten Regelteil* einer Regelverletzung, in deren Menge der verletzenden Ausführungsspuren die betrachtete Spur liegt und deren Bedingungs-Matching sich aus der betrachteten Bindung der Bedingungsvariablen ergibt (einen Sonderfall stellen Existenzgruppen dar, die nur aus einem Negativelement bestehen, diese werden später behandelt).

Es ist nicht sinnvoll, Teilformeln einer Existenzgruppe als verletzte Regelteile zu betrachten. Wenn eine Existenzgruppe, die nicht nur aus einem einzelnen Prädikat oder Negativelement besteht, verletzt ist, bedeutet dies, dass keine Belegung für die in ihr existenzquantifizierten Variablen existiert, mit der der Rumpf der Existenzgruppe erfüllt ist. Würde eine Teilformel betrachtet, besäße diese jedoch mindestens eine (außerhalb der Teilformel existenzquantifizierte) freie Variable, für die eine spezifische Belegung angegeben werden müsste, mit der sie verletzt ist. Die Existenzgruppe fordert jedoch nur die *Existenz* von Knotenausführungen, mit denen sie erfüllt ist, und nicht, dass sie mit *bestimmten* Knotenausführungen erfüllt sein muss. So könnte eine Existenzgruppe zum Beispiel auch erfüllt

werden, indem neue Knoten zum Prozess hinzugefügt werden, mit denen die Forderungen der Gruppe erfüllt sind. Daher ist es nicht möglich, eine verletzende Belegung für die freien Variablen der Teilformel anzugeben.

Existenzgruppen, die nur aus einem Prädikat bestehen, besitzen per se keine weiteren Teilformeln, daher ist auch hier keine weitere Aufspaltung möglich.

Einen Sonderfall stellen Existenzgruppen dar, die nur ein einziges Negativelement enthalten. Ein solches Element besteht aus einer allquantifizierten Variablen und einem Rumpf, in der diese als freie Variable vorkommt. Aufgrund der Allquantifizierung muss der Rumpf für alle Belegungen der Variablen verletzt sein, somit können wir für jede Belegung, mit der der Rumpf nicht erfüllt ist, eine eigene Regelverletzung betrachten, wobei als verletzter Regelteil der Rumpf des Negativelements und als Folgeteil-Bindung die entsprechende Variable angegeben wird.

Ein weiterer Sonderfall tritt ein, wenn der Folgeteil des Regelgraphen leer ist, in der Regel also nur ein Folgeglied existiert, das $false$ ist. In diesem Fall liegt für jedes Bedingungs-Matching je eine einzelne Regelverletzung mit dem verletzten Regelteil $false$ vor.

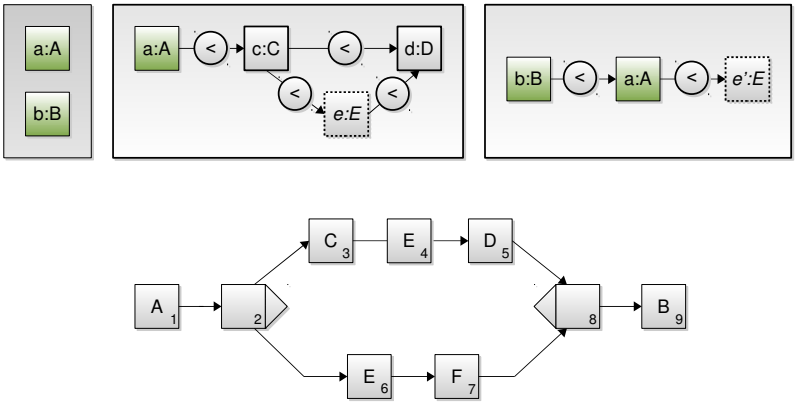


Abbildung 4.3: Ein Beispiel-Regelgraph (oben) und ein einfaches Prozessmodell (unten)

Beispiel 4.1. *Man betrachte die Regel aus Abb. 4.3 über dem Prozessmodell aus derselben Abbildung. Die zugehörige prädikatenlogische Formel lautet:*

$$\forall a \forall b \Big(\Big(C_A(a) \wedge C_B(b) \Big) \longrightarrow \Big(\Big(\big[\exists c \exists d \big(C_C(c) \wedge C_D(d) \wedge O(a,c) \wedge O(c,d) \wedge \forall e \neg (C_E(e) \wedge O(c,e) \wedge O(e,d)) \big) \big] \Big) \vee \Big(\big[\forall e' \neg (C_E(e') \wedge O(a,e')) \big] \wedge \big[O(b,a) \big] \Big) \Big) \Big)$$

Es existiert jeweils genau ein Vorkommen von A und B im Prozess. Diese sind in allen möglichen Ausführungsspuren enthalten, daher ist der Bedingungsteil mit dieser Bindung am $= (a \mapsto 1, b \mapsto 9)$ *in allen Ausführungsspuren erfüllt. Des Weiteren gilt, dass in jeder möglichen Ausführungsspur keines der Folgeglieder erfüllt wird und die Regel somit verletzt ist. Folgende Regelverletzungen existieren:*

1. *In allen Spuren eine Regelverletzung mit der Existenzgruppe aus dem ersten Folgeglied als verletztem Regelteil, da in keiner Spur Knoten vom Typ C und D mit den gewünschten Eigenschaften existieren. Wird an Knoten 2 der obere Zweig gewählt, liegt zwischen C und D ein Knoten von Typ E; wird der untere Zweig gewählt, tritt weder C noch D auf.*
2. *In allen Spuren eine Regelverletzung mit der Existenzgruppe* $\big[O(b,a)\big]$*, da die durch das Bedingungs-Matching an a und b gebundenen Knoten 1 und 9 nicht in der geforderten Reihenfolge ausgeführt werden.*
3. *In allen Spuren, die bei Knoten 2 den oberen Zweig wählen: Eine Regelverletzung des Negativelement-Rumpfs* $\neg(C_E(e') \wedge O(a,e'))$ *mit der Folgeteil-Bindung* $b_1 = (e' \mapsto 4)$*, da Knoten 4 vom Typ E ist und nach Knoten 1 auftritt, an den a gebunden ist.*
4. *In allen Spuren, die bei Knoten 2 den unteren Zweig wählen: Eine Regelverletzung des Negativelement-Rumpfs* $\neg(C_E(e') \wedge O(a,e'))$ *mit der Folgeteil-Bindung* $b_2 = (e' \mapsto 6)$*, da Knoten 4 vom Typ E ist und nach Knoten 1 auftritt, an den a gebunden ist.*

Wie zuvor erläutert, ist es nicht möglich, für Verletzung 1 eine Teilformel der Existenzgruppe als verletzten Regelteil zu betrachten. Zwar erscheint es auf den ersten Blick so, als wäre der Knoten vom Typ E in diesem Zweig (Knoten 4) der Grund für die Verletzung. Tatsächlich wäre die Regel erfüllt, wenn dieser Knoten nicht vorhanden wäre. Allerdings gibt es noch zahlreiche weitere Möglichkeiten, die Verletzung zu beheben, z. B. indem zwischen Knoten 4 und 5 ein weiterer Knoten vom Typ C eingefügt wird, indem Knoten 1 gelöscht wird oder indem Knoten 3 vor Knoten 1 ver-

schoben wird. Kommt eine Aktivität mehrfach im Prozess vor, kann die Anzahl der Möglichkeiten schnell weiter steigen. Daher kann als einzige sichere Aussage angegeben werden, dass die gesamte Existenzgruppe verletzt ist, also keine Knoten vom Typ C und D existieren, die die entsprechende Forderung erfüllen. Abschnitt 5.6.7 zeigt, dass eine solche Existenzgruppe sich auch für die Darstellung der Regelverletzung anbietet und beschreibt, wie der Anwender die einzelnen Möglichkeiten, die Verletzung zu beheben, interaktiv bestimmen kann.

Die Verletzungen 3 und 4 entsprechen dem zuvor vorgestellten Sonderfall, da die verletzte Existenzgruppe nur aus einem Negativelement besteht. Daher enthalten sie den Rumpf dieses Negativelements als verletzten Regelteil sowie je eine Bindung seiner freien Variablen.

4.5.2 Definition

Um die Regelverletzungen im Folgenden logisch definieren zu können, werden in Def. 4.3 zunächst einige Funktionen eingeführt.

Definition 4.3 (Hilfsdefinitionen)**.** Sei $r = aq\big(a \rightarrow (c_1 \vee c_2 \vee \ldots \vee c_n)\big)$ eine Integritätsregel, wie sie in Abschnitt 2.3.2 definiert wurde (inkl. der Erweiterung aus Abschnitt 2.3.3). So ist

- $antcedent(r) = a$,
- $consequences(r) = \{c_1, c_2, \ldots, c_n\}$.
- $\forall\, c_i = (eg_{i,1} \wedge eg_{i,2} \wedge \ldots \wedge eg_{i,m}) \in consequences(r)$:
  $existencegroups(c_i) = \{eg_{i,1}, eg_{i,2}, \ldots, eg_{i,m}\}$.

Sei f eine prädikatenlogische Formel. Dann ist:

- $subformulas(f)$ die Menge aller Teilformeln der Formel f.
- $preds(f)$ die Menge aller in f vorkommenden Prädikatausdrücke.
- $vars(f)$ die Menge aller in f vorkommenden (freien oder gebundenen) Variablen
- $freevars(f)$ die Menge aller freien Variablen in f.

Sei $f : A \rightarrow B$ eine Funktion, so bezeichnet $D_f = A$ die Definitionsmenge der Funktion.

Definition 4.4 ((Einfache) Regelverletzung)**.** Sei T die betrachtete Menge von Ausführungsspuren (das Auswertungssubjekt), $C \subseteq T$, r eine Integritätsregel in prädikatenlogischer Form (mit Erweiterung um Existenzgruppen, vgl. Abschnitt 2.3.3), $f \in subformulas(r)$. Dann ist

$$rv = (r, a, f, b, C)$$

mit

$$a : freevars\big(antecedent(r)\big) \to \big(\bigcap_{t \in C} nodeExs(t)\big),$$
$$b : \big(freevars(f) \setminus freevars\big(antecedent(r)\big)\big) \to \big(\bigcap_{t \in C} nodeExs(t)\big)$$

genau dann eine **(einfache) Regelverletzung** über T, wenn gilt:

$$\begin{aligned} &\forall\, t \in C : struct_{t,r} \not\models r[a_1 \leftarrow a(a_1)] \ldots [a_n \leftarrow a(a_n)] \\ &\quad \text{mit } \{a_1, \ldots, a_n\} = freevars\big(antecedent(r)\big) \end{aligned} \tag{1}$$

und

$$\begin{aligned} &\forall\, t \in C : struct_{t,r} \not\models f[a_1 \leftarrow a(a_1)] \ldots [a_n \leftarrow a(a_n)][b_1 \leftarrow b(b_1)] \ldots [b_n \leftarrow b(b_n)] \\ &\quad \text{mit } \{a_1, \ldots, a_n\} = \Big(freevars\big(antecedent(r)\big) \cap freevars(f)\Big) \\ &\qquad\quad \{b_1, \ldots, b_n\} = \Big(freevars(f) \setminus freevars\big(antecedent(r)\big)\Big) \end{aligned} \tag{2}$$

und

$$\exists\, c \in consequences(r) : \quad f \in existencegroups(c) \wedge \big(f \neq [\forall x \neg \ldots]\big) \tag{3.1}$$

oder $\exists\, c \in consequences(r), eg \in existencegroups(c) :$

$$eg = [\forall x\ f] \tag{3.2}$$

oder $f = \text{false}.$ (3.3)

In diesem Fall entspricht r der Regel, a dem Bedingungs-Matching, f dem verletzten Regelteil, C der Spurmenge und b der Folgeteil-Bindung der Regelverletzung.

Erläuterung: Das Tupel stellt genau dann eine Regelverletzung dar, wenn alles Folgende gilt (ergibt sich aus der Herleitung in Abschnitt 4.5.1):

(1) Die Regel r ist mit dem Bedingungs-Matching a über den Ausführungsspuren aus C nicht erfüllt (dies impliziert, dass der Bedingungsteil erfüllt und der Folgeteil nicht erfüllt ist).

(2) Der Regelteil f ist mit der Bindung b und dem Bedingungs-Matching a in allen Ausführungsspuren aus C nicht erfüllt.

(3) Für den Regelteil f gilt:

(3.1) f ist eine Existenzgruppe aus einem Folgeglied, die nicht nur ein Negativelement enthält oder

(3.2) f ist der Rumpf einer Existenzgruppe aus einem Folgeglied, die nur ein Negativelement enthält oder

(3.3) f ist der Folgeteil und dieser ist leer.

Beispiel 4.2. *Man betrachte erneut die Regel und das Prozessmodell aus Beispiel 4.1. Im Folgenden werden die dort nur textuell erläuterten Regelverletzungen in der in Def. 4.4 eingeführten Struktur dargestellt. Hierfür wird die dort beschriebene Regel mit r bezeichnet, die Menge aller Ausführungsspuren im dortigen Prozessmodell mit T, die Menge aller dieser Spuren, bei denen im XOR-Block der obere Zweig gewählt wird, mit T_1 und die Menge aller Spuren, bei denen der untere Zweig gewählt wird, mit T_2. Somit ergeben sich folgende Regelverletzungen:*

1. $rv_1 = (r, (a \mapsto 1, b \mapsto 9), \left[\exists c \exists d \big(C_C(c) \wedge C_D(d) \wedge O(a,c) \wedge O(c,d) \wedge \forall e \neg(\ldots)\big)\right], (), T)$
2. $rv_2 = (r, (a \mapsto 1, b \mapsto 9), [O(b,a)], (), T)$
3. $rv_3 = (r, (a \mapsto 1, b \mapsto 9), \neg(C_E(e') \wedge O(a,e')), (e' \mapsto 4), T_1)$
4. $rv_4 = (r, (a \mapsto 1, b \mapsto 9), \neg(C_E(e') \wedge O(a,e')), (e' \mapsto 6), T_2)$

Für die praktische Umsetzung kann es in vielen Fällen sinnvoll sein, mehrere Regelverletzungen, die sich nur durch das Bedingungs-Matching unterscheiden, zusammenzufassen. Wir erweitern hierzu die Definition der Regelverletzung, indem wir als Werte der Abbildung a nicht einzelne Knotenausführungen, sondern nicht-leere Mengen von Knotenausführungen verwenden.

Definition 4.5 ((Erweiterte) Regelverletzung)**.** Sei T die betrachtete Menge von Ausführungsspuren (das Auswertungssubjekt), $C \subseteq T$, r eine Integritätsregel, $f \in subformulas(r)$. Dann ist

$$rv = (r, a, f, b, C)$$

mit

$$a : freevars\big(antecedent(r)\big) \to \Big(\mathcal{P}\big(\bigcap_{t \in C} nodeExs(t)\big) \setminus \{\emptyset\}\Big),$$
$$b : \big(freevars(f) \setminus freevars\big(antecedent(r)\big)\big) \to \big(\bigcap_{t \in C} nodeExs(t)\big)$$

genau dann eine **(erweiterte) Regelverletzung** über T, wenn für alle

$$a' : freevars\big(antecedent(r)\big) \to \big(\bigcap_{t \in C} nodeExs(t)\big)$$

mit

$$\forall\, v \in freevars\big(antecedent(r)\big) : a'(v) \in a(v)$$

gilt:

$rv' = (r, a', f, b, C)$ ist eine (einfache) Regelverletzung.

($\mathcal{P}$ gibt die Potenzmenge, also die Menge aller Teilmengen einer Menge, an.)

Beispiel 4.3. *Bei den Regelverletzungen aus Beispiel 4.2 besteht keine Möglichkeit einer Zusammenfassung, da jede positive Variable aus dem Bedingungsteil der Beispielregel im Prozessmodell nur einmal vorkommt. Als erweiterte Regelverletzungen ergeben sich somit dieselben Verletzungen, nur wird das Bedingungs-Matching nun als* $(a \mapsto \{1\}, b \mapsto \{9\})$ *angegeben.*

Befände sich aber nach Knoten 9 ein weiterer Knoten 10 vom Typ B, verdoppelte sich die Anzahl der einfachen Regelverletzungen, da für jede bisherige Verletzung eine weitere mit Bedingungs-Matching $(a \mapsto 1, b \mapsto 10)$ *angegeben werden müsste. Diese acht einfachen Regelverletzungen lassen sich durch vier erweiterte Regelverletzungen ersetzen:*

1. $rv'_1 = (r, (a \mapsto \{1\}, b \mapsto \{9,10\}), \big[\exists c \exists d \big(C_C(c) \wedge C_D(d) \wedge O(a,c) \wedge O(c,d) \wedge \ldots\big)\big], (), T)$
2. $rv'_2 = (r, (a \mapsto \{1\}, b \mapsto \{9,10\}), [O(b,a)], (), T)$
3. $rv'_3 = (r, (a \mapsto \{1\}, b \mapsto \{9,10\}), \neg(C_E(e') \wedge O(a,e')), (e' \mapsto 4), T_1)$
4. $rv'_4 = (r, (a \mapsto \{1\}, b \mapsto \{9,10\}), \neg(C_E(e') \wedge O(a,e')), (e' \mapsto 6), T_2)$

Auch die zu Beginn von Abschnitt 4.5 angesprochenen Alternativverletzungen lassen sich nun formal definieren.

Definition 4.6 (Alternativverletzung)**.** Seien $rv = (r, a, f, b, C)$ und $rv' = (r', a', f', b', C')$ (erweiterte) Regelverletzungen. Dann heißt rv' eine **Alternativverletzung** zu rv, wenn gilt:

$$(r' = r) \wedge \big(\forall\, v \in freevars\big(antecedent(r)\big) : a'(v) \cap a(v) \neq \emptyset\big) \wedge (C' \cap C) \neq \emptyset \qquad (1)$$

und (mit $c \in consequences(r) : f \in subformulas(c)$):

$$\exists\, c' \in consequences(r') : f' \in subformulas(c') \wedge c' \neq c \qquad (2)$$

Es ist leicht erkennbar, dass diese Definition symmetrisch ist: Ist rv' Alternativverletzung zu rv, so ist auch rv Alternativverletzung zu rv'.

Erläuterung: Eine Regelverletzung rv' ist, wie zu Beginn des Abschnitts 4.5 beschrieben, eine Alternativverletzung zu einer anderen Regelverletzung rv, wenn es eine Möglichkeit gibt, die Verletzung der Regel zumindest in einer Teilmenge der zugrunde liegenden Ausführungsspuren zu beheben, indem rv' behoben wird, rv aber nicht. Dies ist *nicht* der Fall, wenn beiden Verletzungen verschiedene Regeln oder Bedingungsmatchings zugrunde liegen (da alle Regeln für alle Bedingungsmatchings erfüllt sein müssen) oder die zugrunde liegenden Ausführungsspuren disjunkt sind. Diese Fälle werden durch Bedingung (1) ausgeschlossen. Die verletzten Regelteile müssen in verschiedenen Folgegliedern liegen, was Bedingung (2) sicherstellt.

Beispiel 4.4. *Für die erweiterten Regelverletzungen aus Beispiel 4.3 ergeben sich folgende Alternativverletzungen:*

- *Für rv'_1: rv'_2, rv'_3 und rv'_4, da die Regel jeweils identisch ist, die Bedingungs-Matchings übereinstimmen, sich die Spurmengen überschneiden und der verletzte Regelteil von rv'_1 aus dem ersten, der der anderen Verletzungen aus dem zweiten Folgeglied stammt.*
- *Für rv'_2, rv'_3 und rv'_4: rv'_1 aus denselben Gründen.*

4.6 Ergebnisunschärfe

Um die Ergebnisse möglichst vieler verschiedener Auswertungsalgorithmen aufnehmen zu können, muss die Ergebnisstruktur möglichst flexibel sein und auch weniger aussagekräftige, »unscharfe« Ergebnisse aufnehmen können. Hierfür notwendige Anpassungen werden im Folgenden vorgestellt.

Eine wichtige Möglichkeit, unscharfe Ergebnisse darzustellen, ist durch den Aufbau der Ergebnisstruktur bereits gegeben: Da es sich um keine große, verschachtelte Struktur handelt, sondern die einzelnen Regelverletzungen unabhängig voneinander sind, muss nicht die vollständige Regelverletzungsmenge angegeben werden, sondern es kann auch nur eine **Teilmenge der Regelverletzungen** verwendet werden. Dies ist z. B. bei Auswertungsalgorithmen von Bedeutung, die nur eine einzelne Belegung der Regelvariablen zurückgeben, mit der eine Regel verletzt ist. Hieraus lassen sich leicht eine oder mehrere Regelverletzungen ableiten, jedoch üblicherweise nicht die gesamte Menge.

Sind im Auswertungssubjekt Schleifen enthalten, *können* ggf. auch gar nicht alle Regelverletzungen praktisch zurückgegeben werden, da ihre Menge unendlich sein kann. In diesem Fall ist es sinnvoll, nur Regelverletzungen für Iterationen anzugeben, die mit Hilfe der dem Benutzer zur Verfügung gestellten Methoden zur Schleifenanalyse (Schleifenexpansion, siehe Abschnitt 5.6.4) manuell ausgewählt wurden.

Eine weitere Möglichkeit der Repräsentation von Ergebnisunschärfe stellen **unscharfe Spurmengen** dar. Wenn sich aus dem Ergebnis eines Algorithmus keine exakten Informationen über die genaue Spurmenge, in der die Regel erfüllt ist oder eine Regelverletzung zutrifft, ermitteln lassen, kann stattdessen eine solche unscharfe Spurmenge angegeben werden. Hierzu werden zwei Mengen angegeben: Eine *obere Schranke* und eine *untere Schranke*, die Teilmenge der oberen Schranke ist. Es gilt dann: Die tatsächliche Spurmenge, die durch die

unscharfe Spurmenge beschrieben wird, liegt zwischen unterer und oberer Schranke – die untere Schranke ist also eine Teilmenge der tatsächlichen Spurmenge und diese eine Teilmenge der oberen Schranke. Bei der Angabe der Gesamterfülltheit durch eine unscharfe Spurmenge beispielsweise gibt die untere Schranke eine Menge von Spuren an, in denen der Algorithmus eindeutig aussagen kann, dass die Regeln erfüllt sind. Die obere Schranke gibt eine Menge von Spurmengen an, von denen der Auswertungsalgorithmus nicht weiß, ob die Regeln in ihnen erfüllt sind oder nicht. Für alle Spuren, die nicht in der oberen Schranke liegen, ist sich der Algorithmus sicher, dass die Regeln verletzt sind.

Definition 4.7 (Unscharfe Spurmenge)**.** Seien T_O, T_U Mengen von Ausführungsspuren mit $T_U \subseteq T_O$. Dann heißt

$$T_\sim = (T_U, T_O)$$

eine **unscharfe Spurmenge** mit oberer Schranke T_O und unterer Schranke T_U.

Eine Ausführungsspur t heißt **möglicherweise enthalten** in $T_\sim$, wenn $t \in T_O$.

Eine Ausführungsspur t heißt **sicher enthalten** in $T_\sim$, wenn $t \in T_U$.

Mithilfe der unscharfen Spurmenge lassen sich nun unscharfe Definitionen für Gesamterfülltheit und Regelverletzungen angeben.

Definition 4.8 (Unscharfe Gesamterfülltheit)**.** Sei T die betrachtete Menge von Ausführungsspuren (das Auswertungssubjekt) und R eine Menge von Integritätsregeln. Sei $F \subseteq T$ die Menge aller Ausführungsspuren aus T, in denen alle Regeln aus R erfüllt sind ($F = \{t \in T | \forall\, r \in R : struct_{t,r} \vDash r\}$).

Sei weiter $F_\sim = (F_U, F_O)$ eine unscharfe Spurmenge nach Def. 4.7 mit $F_U \subseteq F \subseteq F_O$.

Dann heißt

$$E_\sim(T, R) = (e_\sim, F_\sim)$$

mit

$$e_\sim \in \{\text{erfüllbar}, \text{evtl. verletzt}, \text{evtl. erfüllt}, \text{unbestimmt}\}$$

unscharfe Gesamterfülltheit von R in T, wenn gilt:

- Falls $e_\sim$ = erfüllbar: $F \neq \emptyset \wedge F \neq T$
- Falls $e_\sim$ = evtl. verletzt: $F \neq T \wedge F_U = \emptyset$
- Falls $e_\sim$ = evtl. erfüllt: $F \neq \emptyset \wedge F_O = T$
- Falls $e_\sim$ = unbestimmt: $F_U = \emptyset \wedge F_O = T$

In Definition 4.8 wird zwar die Menge F der Spuren, in denen die Regeln erfüllt sind, verwendet – diese Menge ist in der Praxis jedoch unbekannt (ansonsten müsste kein unscharfes Ergebnis verwendet werden). Die Definition verlangt lediglich, dass diese unbekannte Menge von der angegeben unscharfen Spurmenge abgedeckt wird sowie dass einzelne Eigenschaften dieser Menge bekannt sein müssen, damit bestimmte Werte für $e_\sim$ verwendet werden dürfen – sind keine solchen Eigenschaften bekannt, muss der Wert *unbestimmt* verwendet werden.

In Fällen, in denen auf Grundlage der Erfülltheit einer Regel im Prozessmodell automatisierte Aktionen durchgeführt werden, etwa indem die Instantiierung eines Prozessmodells unterbunden wird, wenn eine Regel im Prozessmodell nicht mindestens *erfüllbar* ist, darf ein Algorithmus kein Ergebnis wie *evtl. verletzt* oder *unbestimmt* zurückgeben, da ansonsten die Prozessausführung möglicherweise verhindert wird, obwohl alle Regeln erfüllbar sind.

Definition 4.9 (Unscharfe Regelverletzung)**.** Sei T die betrachtete Menge von Ausführungsspuren (das Auswertungssubjekt), $C \subseteq T$, r eine Integritätsregel und $rv = (r, a, f, b, C)$ eine Regelverletzung.

Sei weiter $C_\sim = (C_U, C_O)$ eine unscharfe Spurmenge mit $C_U \subseteq C \subseteq C_O$.

Dann heißt $rv_\sim = (r, a, f, b, C_\sim)$ **Unscharfe Regelverletzung** über T mit Regel r, Bedingungs-Matching a, verletztem Regelteil f, Spurmenge $C_\sim$ und Bindung b.

Häufig sind unscharfe Regelverletzungen jedoch nicht notwendig: da die Spurmenge einer Regelverletzung nur *eine* Menge von Spuren angeben muss, in denen die Verletzung auftritt, und dies nicht zwingend *alle* betroffenen Spuren sein müssen, kann es genügen, eine normale Regelverletzung anzugeben mit der Menge aller Spuren, von denen feststeht,

dass die Regelverletzung in ihnen auftritt. Da der Anwender bei Regelverletzungen erwarten kann, dass diese nicht vollständig sind, ist dieses Vorgehen legitim und kann die Übersichtlichkeit erhöhen. Bei der Gesamterfülltheit, bei der eine vollständige Angabe erwartet wird, ist hingegen die Verwendung unscharfer Spurmengen in unklaren Fällen zwingend notwendig.

4.7 Ergebnisbeispiele

Wie aussagekräftig ein in der vorgestellten Ergebnisstruktur dargestelltes Auswertungsergebnis ist, hängt stark vom verwendeten Auswertungsalgorithmus ab. Es können spezifische Algorithmen entwickelt werden, die möglichst umfangreiche und optimal in der Ergebnisstruktur darstellbare Ergebnisse erzeugen, oder bereits bestehende Algorithmen zur Regelauswertung, von denen einige Typen in Abschnitt 2.4 vorgestellt wurden, eingesetzt werden. In diesem Abschnitt soll beispielhaft beschrieben werden, wie die bei Verwendung solcher Algorithmen entstehenden Ergebnisse auf die in diesem Kapitel vorgestellte Ergebnisstruktur abgebildet werden könnten.

Bei **graphbasierten Algorithmen** hängt das Ergebnis stark vom verwendeten Verfahren ab. Wird als Ergebnis beispielsweise einer oder mehrere Teilgraphen des Prozessmodellgraphen zurückgegeben, in denen die untersuchte Regel *erfüllt* ist, lässt sich die Gesamterfülltheit leicht angeben: Da jeder Teilgraph eine Menge von Ausführungsspuren repräsentiert, entspricht, falls feststeht, dass das zurückgegebene Ergebnis vollständig ist (also *alle* Ausführungsspuren angibt, in denen die Regel erfüllt ist), die Menge der erfüllbaren Spuren der durch die Teilgraphen repräsentierten Menge, der Wert der Gesamterfülltheit ergibt sich hieraus direkt. Ist nicht sicher, dass das Ergebnis vollständig ist, muss eine unscharfe Gesamterfülltheitsangabe erfolgen, deren unscharfe Spurmenge als untere Schranke die Ausführungsspuren der Teilgraphen enthält und als obere Schranke das gesamte Auswertungssubjekt, da die Regel theoretisch in allen Spuren erfüllt sein könnte.

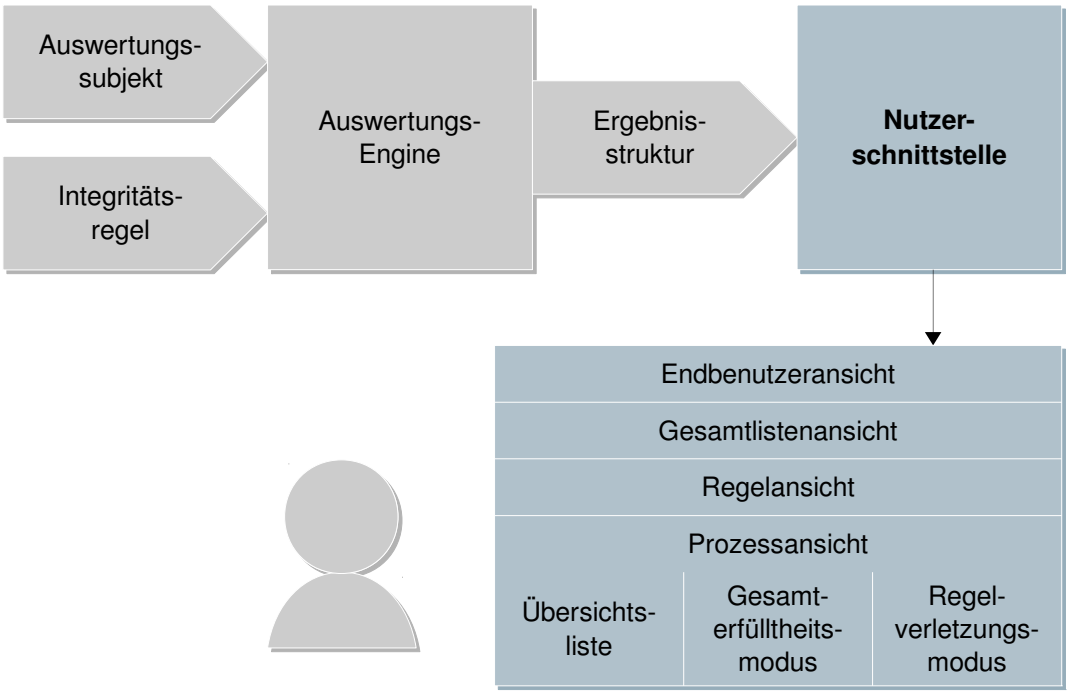
Gibt der Algorithmus stattdessen Teilgraphen zurück, in deren Ausführungsspuren die Regel *verletzt* ist, ergibt sich die Gesamterfülltheit aus dem Komplement der Menge der durch sie repräsentierten Spuren. Bei einer unvollständigen Angabe ist wieder eine unscharfe Spurmenge erforderlich, deren obere Schranke dieser invertierten Menge entspricht und deren untere Schranke die leere Menge ist, da die Regel theoretisch in allen Spuren verletzt sein könnte.

Beschränkt sich die Ausgabe des Algorithmus auf die Angabe der erfüllenden oder verletzenden Spuren, können ohne weiteren Aufwand keine Regelverletzungen angegeben werden, was zwar in der Ergebnisstruktur erlaubt ist, da die Menge der Regelverletzungen nicht vollständig sein muss, jedoch kein sehr aussagekräftiges Ergebnis darstellt. Wenn der Algorithmus einzelne Regelteile getrennt auswertet, kann er jedoch ggf. so angepasst werden, dass er Zwischenergebnisse für die einzelnen Existenzgruppen zurückgibt, wenn diese verletzt sind, welche dann in Regelverletzungen umgewandelt werden können (wenn das Bedingungs-Matching aus dem Ergebnis abgeleitet werden kann). Ansonsten kann ggf. der Algorithmus einzeln für alle Existenzgruppen (und ggf. Bedingungs-Matchings) aufgerufen und die jeweils erhaltenen verletzenden Teilgraphen in Regelverletzungen umgewandelt werden, wenn sie nicht leer sind und nicht innerhalb der Teilgraphen liegen, in denen die Gesamtregel erfüllt ist.

Bei **modellbasierten Verfahren** wird als Ergebnis häufig ein *Gegenbeispiel* zurückgegeben, etwa eine Belegung der Variablen der Regel mit Knotenausführungen, mit der die Regel verletzt ist. Daraus lassen sich leicht eine oder mehrere Regelverletzungen ableiten – Bedingungs-Matching und ggf. Folge-Bindung lassen sich aus der Belegung ableiten, verletzte Regelteile lassen sich ermitteln, indem in die einzelnen Existenzgruppen die Knotenausführungen eingesetzt werden und jede Gruppe dann analysiert wird – wird sie zu *falsch* ausgewertet, ist die Existenzgruppe verletzter Regelteil einer Regelverletzung. Enthält die Regel weder Reihenfolge- noch Zeitprädikate, lässt sich die verletzende Spurmenge aus dem Gegenbeispiel ableiten, indem die Menge aller Spuren bestimmt wird, in denen alle in der zurückgegebenen Belegung verwendeten Knotenausführungen vorkommen. Andernfalls kann direkt nur eine unscharfe Spurmenge angegeben werden, deren obere Schranke diesen Spuren entspricht und deren untere Schranke die leere Menge darstellt, da die Regel ggf. nur verletzt ist, wenn die Knoten in den Spuren bestimmte Bedingungen erfüllen.

Die Gesamterfülltheit lässt sich meist nur unscharf angeben: Wird ein Gegenbeispiel zurückgegeben, ist sicher, dass die Regel in mindestens einer Ausführungsspur verletzt ist, aber es sind nicht alle verletzenden Spuren bekannt. Als Wert der Gesamterfülltheit kann daher nur *evtl. verletzt* angegeben werden, als erfüllende Spurmenge eine unscharfe Spurmenge, deren untere Schranke leer ist und deren obere Schranke dem Komplement der aus dem Gegenbeispiel ableitbaren Spurmenge entspricht, oder, wenn diese unscharf ist, dem Komplement ihrer unteren Schranke. Wird kein Gegenbeispiel zurückgegeben, ist die Regel in allen Spuren erfüllt, bei den erfüllenden Spuren handelt es sich also um das gesamte Auswertungssubjekt, der Wert der Gesamterfülltheit ist *erfüllt*.

5 Nutzerschnittstelle



In diesem Kapitel wird die in dieser Arbeit entwickelte Nutzerschnittstelle vorgestellt, die es dem Anwender erlaubt, die Erfülltheit von Integritätsregeln in einem Geschäftsprozess – auf Modellebene sowie für laufende bzw. abgeschlossene Instanzen – interaktiv und auf eine an seine Ziele und Aufgaben angepasste Weise zu untersuchen. Hierzu werden die Überprüfungsergebnisse, die in der in Kapitel 4 definierten Ergebnisstruktur vorliegen, in eine anwenderfreundliche Darstellung umgesetzt.

In Abschnitt 5.1 werden zunächst kurz die aus den Anforderungen abgeleiteten Ziele der Nutzerschnittstelle beschrieben, bevor in Abschnitt 5.2 ein Überblick über die Grundlagen der entwickelten Schnittstelle gegeben wird. Die verschiedenen dort eingeführten Ansichten werden anschließend in den Abschnitten 5.3 – 5.6 detailliert vorgestellt.

5.1 Ziele

Grundlegende Aufgabe der in diesem Kapitel entwickelten Nutzerschnittstelle ist es, das Ergebnis der Regelauswertung dem Anwender in einer Form darzustellen, die seinem gedanklichen Modell entspricht und den Zielen, die er durch die Nutzung des Systems erreichen möchte, entgegenkommt.

Die Nutzerschnittstelle basiert dabei auf den Anforderungen, die im Zuge der Anforderungsanalyse in Abschnitt 3.3.4 ermittelt wurden. Wichtige Forderungen waren hierbei:

- die übersichtliche Darstellung der Erfülltheit verschiedener Regeln in einem Prozess,
- die Darstellung der Erfülltheit einer Regel in verschiedenen Prozessen und Prozessinstanzen,
- das Anbieten einfacher Rückmeldungen für Endbenutzer,
- die Erkennbarkeit der Ausführungssituationen und Prozessteile, in denen Regelverletzungen auftreten,
- die genaue Analysierbarkeit einzelner Regelverletzungen,
- die Darstellung der Beziehungen zwischen Regelverletzungen.

Besonderes Augenmerk liegt in dieser Arbeit auf einer zielführenden Darstellung der Regelerfülltheit. Die Nutzerschnittstelle soll dem Anwender Hinweise geben, aufgrund derer er Möglichkeiten findet, die Regelverletzungen zu beheben. Bei Betrachtung komplexerer Integritätsregeln ist es in den meisten Fällen nicht möglich, dem Anwender die konkreten Schritte mitzuteilen, die er unternehmen muss, um eine Regelverletzung zu beheben. Das Problem ist hierbei, dass es für die Behebung häufig zahlreiche Möglichkeiten gibt, zwischen denen nicht das System, sondern nur der Anwender entscheiden kann, da für die Entscheidung ggf. Ziele und Semantik des Prozesses von Bedeutung sind, die sich aus der Prozessbeschreibung im System nicht ableiten lassen. Unser Konzept basiert daher darauf, dem Anwender stattdessen darzustellen, wie das gewünschte Ergebnis aussehen muss bzw. welche verschiedenen Möglichkeiten für korrekte Ergebnisse es gibt. Dies soll in einer Form geschehen, die es dem Anwender ermöglicht, selbst die im jeweiligen Prozess günstigste Möglichkeit zur Erreichung des gewünschten Ergebnisses herauszufinden. Dies wird dadurch unterstützt, dass wir dem Anwender Position und Art einer zu behebenden Regelverletzung möglichst genau beschreiben, damit er seine Handlungsmöglichkeiten besser einschätzen kann.

5.2 Grundlagen

Im Zuge der Anforderungsanalyse wurden in Abschnitt 3.1 die Anwender in die Nutzergruppen *Prozessbetreuer*, *Regelbetreuer* und *Endbenutzer* eingeteilt, die dementsprechend verschiedene Nutzungsschwerpunkte haben: Während die Grundlage der Arbeit von Prozessbetreuern stets ein Prozessmodell bzw. eine daraus abgeleitete Prozessinstanz darstellt, arbeiten Regelbetreuer hauptsächlich an der Erstellung und Bearbeitung von Integritätsregeln. Endbenutzer hingegen kommen bei ihrer Arbeit mit Client-Anwendungen ausschließlich bei zur Laufzeit auftretenden Fehlern mit dem Compliance-System in Kontakt.

Daher betrachten wir für die Nutzerschnittstelle unterschiedliche Ansichten, die in verschiedenen der in Abschnitt 2.5 vorgestellten Systemkomponenten verwendet werden. Die einfachste Ansicht stellt dabei die **Endbenutzeransicht** dar, die in Abschnitt 5.3 vorgestellt wird. Einen kompletten Überblick über die Erfülltheit aller Prozesse, Instanzen und Regeln liefert die für Prozess- und Regelbetreuer gedachte **Gesamtlistenansicht**, die wir in Abschnitt 5.4 einführen. Die **Regelansicht**, die hauptsächlich von Regelbetreuern verwendet wird, wird in Abschnitt 5.5 erläutert. Die umfangreichsten Analysemöglichkeiten bietet die in Abschnitt 5.6 eingeführte **Prozessansicht**, da tiefergehende Untersuchungen der Regelerfülltheit stets nur im Kontext von Prozessmodellen oder -instanzen erfolgen können. Die Prozessansicht kann auch von anderen Ansichten aus aufgerufen werden, um die Details verletzter Regeln genauer zu analysieren. Die Betrachtung der Nutzerschnittstelle zur Regelanalyse in dieser Ansicht stellt daher den Schwerpunkt dieses Kapitels dar.

5.3 Endbenutzeransicht

Wie bereits in Kapitel 3 beschrieben, sollen Endanwender möglichst selten mit Integritätsregeln und ihrer Verletzung in Kontakt kommen. Daher werden grundsätzlich alle Regelverletzungen, die nicht durch manuelle Aktionen des Endanwenders, sondern z. B. durch Überschreitung einer Zeitschranke im Prozess oder Fehler bei einem automatischen Schritt entstehen, diesem auch nicht angezeigt. Stattdessen wird, je nach Regelklasse der verletzten Regel, der Prozessbetreuer informiert und ggf. die Prozessinstanz angehalten. Der Prozessbetreuer kann dann die Prozessinstanz mit einer geeigneten Anwendung in einer ausführlicheren Ansicht (siehe Abschnitt 5.6) betrachten und die Verletzung analysieren. Prinzipiell ist auch die Durchführung automatischer, vordefinierter Kompensationsschritte im

Sinne eines *Exception Handling* durch die Prozess-Engine möglich, solche ohne Nutzerinteraktion ablaufenden Schritte liegen jedoch außerhalb des Themenbereichs dieser Arbeit.

Auch in Fällen, in denen Regelverletzungen durch Aktionen eines Endanwenders verursacht werden, soll er in seinem Arbeitsfluss möglichst wenig gestört werden. Da es, wie in Abschnitt 5.1 beschrieben wurde, meist nicht möglich ist, eine Regelverletzung automatisiert zu beheben, einem Endanwender jedoch häufig das notwendige Prozesswissen sowie die Nutzerprivilegien fehlen, um eine selbstständige Behebung auf Grundlage der Prozessbeschreibung durchzuführen, ist es sinnvoll, Nutzeraktionen wie das Durchführen manueller Schritte oder einfacher Ad-hoc-Änderungen, die zu Regelverletzungen führen, bereits im Vorfeld zu verhindern. Dies kann geschehen, indem z. B. entsprechende Steuerelemente deaktiviert werden. Damit der Anwender für die dabei ggf. entstehenden Einschränkungen seines Handlungsspielraums Verständnis hat, ist es sinnvoll, bei Mausbewegung über ein deaktiviertes Steuerelement eine kurze, einfache Beschreibung der verletzten Regel darzustellen, die grob den Sinn der Regel angibt. Diese Beschreibung kann beispielsweise im Regel-Editor vom Regelbetreuer als zusätzliche Eigenschaft bei der Erstellung der Regel angegeben werden.

Um eine Regelverletzung auf solche Weise zu verhindern, muss eine Client-Anwendung entsprechend angepasst werden, um die Auswirkungen der Durchführung verschiedener Aktionen auf die Prozessinstanz im Vorfeld zu berechnen und für die berechneten Folgezustände die Regelerfülltheit überprüfen zu lassen. In Fällen, in denen dies zu aufwändig ist oder wenn die Client-Anwendung es nicht unterstützt, muss die Server-Software, die in jedem Fall vor der tatsächlichen Durchführung einer Aktion diese auf die Verletzung von Integritätsregeln prüft, eine Fehlermeldung erzeugen, welche dem Anwender dann von der Client-Anwendung dargestellt wird und ebenfalls einen Hinweis auf den Sinn der verletzten Regel enthält.

5.4 Gesamtlistenansicht

Die Gesamtlistenansicht dient dem Prozess- und Regelbetreuer dazu, eine vollständige Übersicht über die Erfülltheit aller Regeln, Prozessmodelle und Prozessinstanzen zu erhalten. Sie kann beispielsweise in einer Prozess-Überwachungsanwendung verwendet werden. Variationen dieser Ansicht werden auch in der Regel- und Prozessansicht verwendet, die in den nachfolgenden Abschnitten 5.5 und 5.6 vorgestellt werden.

Die Gesamtlistenansicht besteht aus einer Liste, die für die Erfülltheit jeder Integritätsregel in jedem Prozessmodell sowie jeder Prozessinstanz je einen Eintrag enthält.

Jeder Eintrag in der Liste besteht aus mehreren Spalten: einer Angabe über das Prozessmodell, der ID der Prozessinstanz (diese Spalte enthält den Wert »Modell«, wenn die Erfülltheitsangabe sich nicht auf eine einzelne Instanz, sondern auf das Prozessmodell bezieht), einer (im Vorfeld vom Regelbetreuer festgelegten oder automatisch erzeugten) Kurzbeschreibung der Integritätsregel sowie einer Angabe der Gesamterfülltheit der Regel im entsprechenden Auswertungssubjekt.

Zur Angabe der Regel ist jeweils in einer zusätzlichen Spalte die entsprechende Regelklasse (vgl. Abschnitt 3.3.2) angegeben, welche durch ein Symbol gekennzeichnet wird. Die folgende Liste zeigt die unterstützten Regelklassen und ihre entsprechenden Symbole:

- – Regeln, die auf keinen Fall verletzbar sein dürfen (Klasse 1)
- – Regeln, die bei der Ausführung nicht verletzt werden dürfen (Klasse 2)
- – Regeln, über deren Verletzung informiert wird (Klasse 3)
- – Regeln, die zur Laufzeit ignoriert werden (Klasse 4)

Die Angabe der Gesamterfülltheit erfolgt durch eine textuelle Beschreibung sowie durch ein spezifisches Symbol, das in der Farbe der jeweiligen Regelklasse gehalten ist. Dabei werden folgende Gesamterfülltheitsstufen unterstützt (die Symbole sind hier beispielhaft in der Klasse 1 zugeordneten Farbe dargestellt):

- – Erfüllt, da nicht passend (vgl. Def. 4.2)
- – Erfüllt (vgl. Def. 4.1)
- – Evtl. erfüllt (vgl. Def. 4.8)
- – Erfüllbar (vgl. Def. 4.1)
- – Erfüllbar, mit unscharfer Spurmenge (vgl. Def. 4.8)
- – Evtl. verletzt (vgl. Def. 4.8)
- – Verletzt (vgl. Def. 4.1)
- – Erfülltheit unbestimmt (vgl. Def. 4.8)

Würden die Ergebnisse als einfache flache Liste dargestellt, wäre dies nicht nur für den Anwender unübersichtlich, sondern würde auch eine sehr aufwändige Analyse erforderlich

machen, da alle Ergebnisse auf einmal berechnet werden müssten. Daher werden die Einträge der Liste stets nach einer der drei Spalten »Integritätsregel«, »Prozessmodell« und »Prozessinstanz« gruppiert. Hierbei werden alle Einträge, die in dieser Spalte denselben Wert enthalten, zu einer Gruppe zusammengefasst. Es kann auch nach weiteren Spalten gruppiert werden, wodurch dann innerhalb einer Gruppe wiederum Einträge zu Untergruppen zusammengefasst werden. In der Liste wird dann für jede Gruppe ein Eintrag angezeigt, der sich vom Anwender expandieren lässt, wodurch die enthaltenen Einträge (oder Untergruppen) angezeigt werden. Zu jedem Gruppeneintrag wird die jeweilige aggregierte Gesamterfülltheit angezeigt, außerdem können in zusätzlichen Spalten weitere Informationen zu den jeweiligen Prozessmodellen, Instanzen bzw. Regeln, deren Gruppierung der jeweilige Eintrag repräsentiert, enthalten sein.

Die Gesamtlistenansicht bietet dem Anwender mehrere vorgegebene sinnvolle Gruppierungsmöglichkeiten, aus denen er wählen kann:

- Gruppierung nach der Spalte »Integritätsregel«
- Gruppierung nach der Spalte »Integritätsregel«, dann nach »Prozessmodell«
- Gruppierung nach der Spalte »Prozessmodell«, dann nach »Prozessinstanz«
- Gruppierung nach der Spalte »Prozessmodell«, dann nach »Integritätsregel«
- Gruppierung nach der Spalte »Prozessinstanz«

Eine Gruppierung nur nach der Spalte »Prozessmodell« ist nicht sinnvoll, da dadurch die Einträge für verschiedene Integritätsregeln und verschiedene Prozessinstanzen gemischt dargestellt würden. Eine Gruppierung nur nach Spalte »Prozessinstanz« ist hingegen sinnvoll, da jede Instanz genau einem Prozessmodell zugeordnet ist. Ebenfalls sinnvoll ist die Gruppierung nur nach der Spalte »Integritätsregel«, da es sinnvoll sein kann, die Erfülltheit einer Regel in allen Instanzen aller Prozessmodelle gemischt zu betrachten.

Wie die Anforderungsanalyse in Kapitel 3 zeigte, sind nicht in jeder Situation alle Ergebnisse von Bedeutung. Daher erlaubt es die Listenansicht, Filteroperationen durchzuführen, um die Übersichtlichkeit zu erhöhen und spezifischere Analysen zu ermöglichen. Es lässt sich auswählen, ob auch Prozessmodelle bzw. Instanzen angezeigt werden, in denen alle Regeln erfüllt sind, oder nur solche, in denen mindestens eine (potentiell) verletzt ist. Auch lässt sich einstellen, ob nur laufende, nur abgeschlossene oder alle Instanzen angezeigt werden sollen. Ebenfalls filtern lässt sich nach Regelklassen, so dass nur Einträge für

Integritätsregeln bestimmter Klassen angezeigt werden. Eine weitere Filtermöglichkeit ergibt sich, indem eingestellt werden kann, ob nur die Erfülltheit in Prozessmodellen, nur die Erfülltheit in Prozessinstanzen oder alle Erfülltheitseinträge angezeigt werden (dies beeinflusst auch die entsprechenden Aggregationsmöglichkeiten).

Eine weitere Möglichkeit der Anpassung besteht darin, eine Sortierung nach verschiedenen Kriterien durchzuführen. Mögliche Sortierkriterien sind der Erfülltheitsstatus der Modelle/Instanzen, die Anzahl der laufenden/abgeschlossenen Instanzen der Prozessmodelle, die bisherigen Laufzeiten der Instanzen oder die Regelklassen. Generell ist eine Sortierung nach den Inhalten jeder der dargestellten Spalten möglich. In der obersten Gruppierungsstufe erfolgt dabei zunächst eine Sortierung der Gruppen nach dem ausgewählten Kriterium, soweit es auf die jeweilige Gruppierung zutrifft (ist beispielsweise auf der ersten Ebene nach Prozessmodellen gruppiert, als Sortierkriterium jedoch die Regelklasse gewählt, kann auf dieser Stufe keine Sortierung erfolgen). Die Einträge bzw. Untergruppen der jeweiligen Gruppe werden ebenfalls nach dem Kriterium sortiert, sofern es zutrifft. Es können auch sekundäre und weitere Sortierkriterien angegeben werden, nach denen Gruppen oder Einträge sortiert werden, die durch ein vorhergehendes Kriterium nicht sortiert wurden.

Durch die vielseitigen Möglichkeiten der Anpassung der Listenansicht ist es dem Anwender möglich, die Darstellung seiner jeweiligen Situation anzupassen und so stets mit wenigen Schritten die Informationen zu erhalten, die er benötigt.

Bei Doppelklick auf die Spalte »Integritätsregel« eines Eintrags – oder auf den Eintrag einer Gruppe, wenn nach dieser Spalte gruppiert wurde – wird die entsprechende Integritätsregel im Regel-Editor geöffnet. Analog lässt sich durch Doppelklick auf die Spalte »Prozessmodell« der Prozessmodell-Editor für das jeweilige Modell und mit der Spalte »Prozessinstanz« die Instanzanalyse-Anwendung für die gewählte Instanz starten. Auf diese Weise lässt sich die Gesamtlistenansicht als Ausgangspunkt für weitere Analysen unter Zuhilfenahme der im Folgenden vorgestellten Ansichten nutzen.

5.5 Regelansicht

Die Regelansicht wird in der Regeleditierungskomponente des Systems vom Regelbetreuer eingesetzt. In dieser Komponente erfolgt die Bearbeitung einer neu erstellten oder bestehenden Integritätsregel auf Grundlage ihres Regelgraphen. Entsprechend den Anforderun-

gen aus Kapitel 3.3 soll dabei möglichst zu jeder Zeit erkennbar sein, inwiefern die Regel, falls sie bereits Prozessen zugeordnet ist, in diesen verletzt oder erfüllt ist.

Hierzu wird in der von uns entworfenen Nutzerschnittstelle der Regeleditor um eine Listenansicht erweitert, die der Liste aus der Gesamtlistenansicht entspricht, allerdings reduziert auf die einzelne betrachtete Regel. Dadurch entfällt die Spalte »Integritätsregel« und die angebotenen Gruppierungsmöglichkeiten reduzieren sich auf folgende:

- Gruppierung nach der Spalte »Prozessmodell«
- Keine Gruppierung

Eine Gruppierung nur nach der Spalte »Prozessinstanz« ist nicht sinnvoll, da für jede Prozessinstanz nur ein Eintrag enthalten ist. Stattdessen ist im Gegensatz zur Gesamtlistenansicht auch die flache Darstellung ohne Gruppierung sinnvoll, da durch die Beschränkung auf eine Integritätsregel die Ergebnismenge bereits ausreichend eingeschränkt ist.

Um die Erfülltheit der Regel in einem Prozessmodell oder einer Instanz genauer zu analysieren, lässt sich wiederum durch Doppelklick auf das entsprechende Element in der Liste die im Folgenden beschriebene Prozessansicht öffnen, wodurch die Gesamterfülltheit sowie die Regelverletzungen der betrachteten Regel in der Prozessdarstellung weitergehend untersucht werden können.

5.6 Prozessansicht

Eine Prozessansicht wird in allen Komponenten des Systems eingesetzt, in denen Prozessmodelle, laufende oder abgeschlossene Prozessinstanzen visualisiert werden. Zu nennen sind hier insbesondere Komponenten zur Betrachtung einzelner laufender Prozessinstanzen und zur Durchführung von Ad-hoc-Änderungen, der Prozessvorlagen-Editor sowie Komponenten zur Log-Analyse.

Da Prozesse die grundlegenden Elemente eines BPM-Systems darstellen, ist in der Prozessansicht auch die genaueste Analyse der Erfülltheit von Integritätsregeln möglich. Neben dem Prozessbetreuer, für dessen Arbeit diese Ansicht die Grundlage bildet, kann sie auch vom Regelbetreuer genutzt werden, um detailliertere Informationen zu der Erfülltheit einer Regel im Prozesskontext zu erhalten.

Wir reichern hierzu die Prozessdarstellung um Compliance-Informationen an, welche eine genaue Analyse der Regelerfülltheit ermöglichen. Ebenfalls wird die Prozessansicht um eine Übersichtsliste erweitert, die einen Überblick über die Erfülltheit der Regeln und alle Regelverletzungen bietet. Sie bietet ebenfalls Filter- und Auswahlmöglichkeiten, die sich auch auf die Prozessdarstellung auswirken, und kann somit als Kontrollkomponente eingesetzt werden.

Abschnitt 5.6.1 beschreibt zunächst die Herausforderungen für die Darstellung der Ergebnisse in der Prozessansicht. Die Übersichtsliste wird in Abschnitt 5.6.2 genauer vorgestellt, Abschnitt 5.6.3 beschreibt die hierfür notwendigen textuellen Regelverletzungsangaben. Die restlichen Abschnitte Abschnitt 5.6.4 – Abschnitt 5.6.7 widmen sich der Anpassung und Erweiterung der Prozessdarstellung zur Integration der Auswertungsergebnisse.

5.6.1 Herausforderungen

Eine der größten technischen Herausforderungen der Integration der Erfülltheitsangabe in die Prozessansicht stellt der begrenzte Bildschirmplatz dar. Für die Darstellung der Erfülltheit können Knoten von Bedeutung sein, die sich in weit voneinander entfernten Teilen des Prozessmodells befinden und so nie gemeinsam auf dem Bildschirm zu sehen sind. Insbesondere erschwert dies auch eine übersichtliche Darstellung aller »neuralgischen Punkte« in einem Prozessmodell, an denen Regelverletzungen vorliegen. Dieser Herausforderung wird in Abschnitt 5.6.4 begegnet. Auch die in Abschnitt 5.6.2 vorgestellte Übersichtsliste hilft hier, da in ihr alle Regelverletzungen, auch nicht sichtbare, aufgelistet werden.

Eine weitere Herausforderung ergibt sich durch die notwendige Darstellung von Schleifen. In verschiedenen Iterationen von Schleifen können verschiedene Erfülltheitssituationen vorliegen. Insbesondere kann die Erfülltheit davon abhängen, ob beispielsweise in der ersten Iteration in der Schleife ein bestimmter Ausführungsweg gewählt wurde und in der zweiten Iteration ein anderer. Des Weiteren können Schleifen potentiell unendlich oft ausgeführt werden, wodurch es nicht möglich ist, alle möglichen Iterationen auszuwerten. Das in dieser Arbeit beschriebene Lösungskonzept für die Analyse der Erfülltheit in Schleifen durch den Anwender wird ebenfalls in Abschnitt 5.6.4 besprochen.

Eine wichtiges Element bei der Umsetzung der zuvor vorgestellten Ergebnisstruktur in der Nutzerschnittstelle stellt die Darstellung der Spurmengen dar, die in der Ergebnisstruktur als grundlegendes Element verwendet werden. Die Aufgabe, diese oft unendlichen Men-

gen verschiedener Ausführungsspuren verständlich darzustellen, wird in Abschnitt 5.6.5 behandelt.

Um dem Anwender einen Überblick über die Erfülltheit im Prozess zu geben sowie die Prozessteile, in denen Regelverletzungen vorliegen, hervorzuheben, ist eine übersichtliche Darstellung der Erfülltheit notwendig. Der hierzu entwickelte Gesamterfülltheitsmodus wird in Abschnitt 5.6.6 vorgestellt. Der komplexeste Bestandteil der Erfülltheitsanzeige ist die detaillierte Darstellung einzelner Regelverletzungen zur genauen Analyse durch den Benutzer. Diese wird in Abschnitt 5.6.7 beschrieben.

5.6.2 Übersichtsliste

Die Übersichtsliste in der Prozessansicht dient dazu, einen Überblick über die Erfülltheit der Integritätsregeln im derzeit geöffneten Prozessmodell bzw. der untersuchten Prozessinstanz zu geben. Sie ist ähnlich aufgebaut wie die in der Gesamtlistenansicht verwendete Liste, enthält jedoch nicht nur Informationen über die Gesamterfülltheit, sondern auch über die einzelnen Regelverletzungen.

So ist für jede Regelverletzung im aktuellen Auswertungssubjekt ein Eintrag enthalten, welcher eine kurze textuelle Beschreibung der Verletzung enthält, die im Folgenden in Abschnitt 5.6.3 eingeführt wird. Ebenfalls ist ein farblich gekennzeichnetes Symbol dargestellt, das die Regelklasse der verletzten Regel angibt und dem in der Gesamtlistenansicht verwendeten Symbol entspricht. In weiteren Spalten ist die verletzte Integritätsregel angegeben sowie die Regeldatei, die diese Regel enthält. Diese beiden Spalten werden jedoch nicht dargestellt, nach ihnen kann nur gruppiert werden.

Bei Klick auf eine Regelverletzung in der Liste wird diese auf die in Abschnitt 5.6.7 beschriebe Weise in der Prozessdarstellung angezeigt.

Ähnlich wie bei der Darstellung in der Gesamtlistenansicht bestehen auch in dieser Liste Möglichkeiten zur Gruppierung und Aggregation. So sind in der Standardansicht alle Regelverletzungen einer Regel gruppiert, wobei bei jeder Regel die entsprechende Gesamterfülltheit analog zur Darstellung in der Gesamtlistenansicht angezeigt wird. Diese Gruppierung kann vom Nutzer jedoch auch deaktiviert werden. Bei Auswahl einer Regel besteht auch die Möglichkeit, diese im Regel-Editor zu öffnen. Es ist zudem möglich, die Regeln weitergehend zu gruppieren, indem Regeln, die in derselben Datei gespeichert sind, zu einer Gruppe zusammengefasst werden, für die eine aggregierte Erfülltheitsangabe erfolgt.

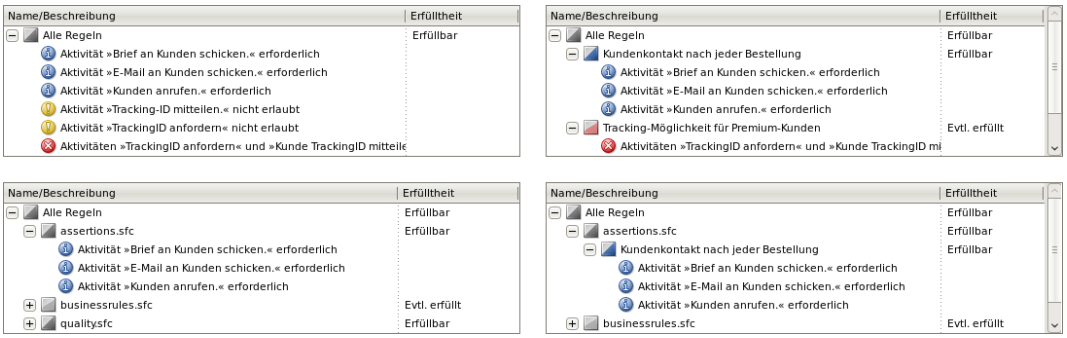


Abbildung 5.1: Übersichtsliste. Oben links: ungruppiert; Oben rechts: nach Regeln gruppiert; Unten links: nach Dateien gruppiert; Unten rechts: nach beidem gruppiert

Zusätzlich wird die Gesamterfülltheit aller Regeln als oberstes Element in der Liste angezeigt. Verschiedene Gruppierungsmöglichkeiten sind in Abb. 5.1 dargestellt.

Auch für diese Liste kann eingestellt werden, dass auch Regeln angezeigt werden sollen, die in allen Spuren erfüllt sind (und somit keine Regelverletzungen enthalten). Auf diese Weise kann sich der Anwender einen Überblick über die Erfülltheit aller Regeln verschaffen. Auch besteht die Möglichkeit, nach verschiedenen Regelklassen zu filtern, so dass nur die Regelverletzungen der ausgewählten Klassen angezeigt werden. Dies bietet sich an, da in verschiedenen Phasen des Prozesslebenszylus verschiedene Regelklassen von Bedeutung sein können, wie in Kapitel 3 dargestellt wurde. Die hier durchgeführte Filterung beeinflusst auch die aggregierten Erfülltheitsangaben sowie die Darstellung des Prozesses im in Abschnitt 5.6.6 beschriebenen Gesamterfülltheitsmodus. Auch können, wenn nach Regeln gruppiert wurde, in der Liste eine oder mehrere Regeln ausgewählt werden, in diesem Fall wird nur die Erfülltheit dieser Regeln im Gesamterfülltheitsmodus der Prozessdarstellung angezeigt.

Wiederum ist eine Sortierung nach verschiedenen Kriterien möglich, etwa nach der Erfülltheit oder der Verletzungsbeschreibung. Sortiert werden die Regelverletzungen innerhalb der Aggregationsgruppen sowie auch die Gruppen selbst.

5.6.3 Textuelle Regelverletzungen

Damit Regelverletzungen in der Listenansicht in Textform dargestellt werden können, müssen Beschreibungen für sie definiert werden. Da der in der Liste verfügbare Platz begrenzt

ist, kann keine vollständige Erläuterung der Verletzung erfolgen, sondern es können nur die wichtigsten Elemente erwähnt werden. Für eine genauere Analyse kann die Regelverletzung in der Prozessdarstellung betrachtet werden, wie in Abschnitt 5.6.7 beschrieben wird.

In der textuellen Beschreibung wird nur der verletzte Regelteil der Regelverletzung angegeben. Handelt es sich bei diesem um den Ausdruck *false*, bedeutet dies, dass der Folgeteil der Regel leer ist, die Regel also verletzt ist, da der Bedingungsteil erfüllt wurde. Als textuelle Beschreibung wird dann »*Verletzt, da Bedingung erfüllt*« verwendet.

Ein weiterer einfacher Fall liegt vor, wenn es sich beim verletzten Regelteil um ein einzelnes Negativelement (ohne Quantisierung) handelt. In diesem Fall enthält die Regelverletzung stets auch eine Belegung für die im Negativelement definierte Variable. In diesem Fall wird als textuelle Beschreibung »*Aktivität ›A‹ nicht erlaubt*« verwendet, wobei A die Aktivität der an die Variable gebundenen Knotenausführung angibt. Besitzt der entsprechende Knoten keine Aktivität, wird stattdessen die ID des Knotens verwendet. Im Negativelement ggf. enthaltene Prädikate werden aus Platzgründen nicht angegeben.

Ebenfalls gesondert betrachtet werden kann der Fall, dass der verletzte Regelteil aus einem einzigen Prädikat mit zwei Variablen aus dem Bedingungsteil besteht. In diesem Fall wird als textuelle Beschreibung angegeben: »*Aktivität ›A‹ muss vor ›B‹ auftreten*«, falls es sich um ein Reihenfolgeprädikat handelt, »*Zeitliche Einschränkung zwischen Aktivitäten ›A‹ und ›B‹ verletzt*«, falls es sich um ein Zeitprädikat handelt oder »*Knoten müssen verschieden sein*«, falls es sich um ein Ungleichheitsprädikat handelt. Die Aktivitäten entstammen den Knotenausführungen, die den Variablen durch das Bedingungs-Matching zugeordnet sind.

Wenn keiner der zuvor betrachteten Sonderfälle eintritt, handelt es sich beim verletzten Regelteil um eine Existenzgruppe, in der mindestens eine existenzquantifizierte Variable enthalten ist. In diesem Fall wird nicht die komplette Existenzgruppe textuell beschrieben, sondern nur die Aktivitätentypen der enthaltenen existenzquantifizierten Variablen: »*Aktivität ›A‹ erforderlich*«, falls nur eine solche Variable existiert, oder »*Aktivitäten* A_1, A_2, ... *und* A_n *erforderlich*« andernfalls. Auf diese Weise sind zwar ebenfalls in der Existenzgruppe enthaltene Prädikate und Negativelemente nicht aus der textuellen Beschreibung ersichtlich, dies kann jedoch aus Gründen der Prägnanz nicht vermieden werden. Um die weiteren Elemente der Existenzgruppe zu analysieren, muss die Regelverletzung im Prozess betrachtet werden.

5.6.4 Anpassung der Prozessdarstellung

Wie in Abschnitt 5.6.1 beschrieben wurde, passen komplexe Prozesse oft nicht komplett in den auf dem Bildschirm zur Verfügung stehenden Platz. Dem kann zwar durch Scrolling der Darstellungsfläche begegnet werden, allerdings ist es schwierig, einen Überblick über die Erfülltheit der Regeln im Prozess zu erlangen, wenn er nicht in seiner Vollständigkeit betrachtet werden kann. Einen der einfachsten Mechanismen, dem zu begegnen, stellt die Bereitstellung von Zoomfunktionalität dar. Daher wird die Möglichkeit angeboten, nach Belieben im Prozessmodell ein- und wieder auszuzoomen. Allerdings ergibt sich durch Zooming das Problem, dass bei kleineren Zoomstufen der dargestellte Text nur noch schwer lesbar ist und auch die Darstellung weiterer komplexerer Erfülltheitsinformationen schwer fällt, da für einzelne Elemente nur noch sehr wenig Platz zur Verfügung steht.

Daher sieht die von uns beschriebene Nutzerschnittstelle vor, dass die Prozessdarstellung angepasst werden kann, um zum jeweiligen Zeitpunkt nicht benötigte Prozessteile auszublenden und so den vorhandenen Platz besser für die jeweils wichtigen Elemente zu nutzen. Hierzu hat der Anwender die Möglichkeit, Blöcke des Prozesses, die derzeit nicht relevant sind, temporär zu einem einzigen Knoten zusammenzufassen. Hierzu wird ein in der Prozessansicht entweder bereits bereitgestelltes oder ansonsten neu hinzuzufügendes Rechtecks-Auswahlwerkzeug erweitert. Statt nur einzelne Knoten auszuwählen, wenn der Anwender mit diesem Werkzeug einen rechteckigen Bereich im Prozess markiert, werden automatisch die größtmöglichen komplett im Rechteck enthaltenen Blöcke bzw. Folgen von Blöcken mit jeweils einer Gruppierungs-Markierung umgeben, die einen Button enthält. Mit Hilfe dieses Buttons lässt sich der ausgewählte Bereich dann zu einem einzigen Knoten zusammenfassen. Dieser Knoten enthält wiederum einen Button, mit dem er wieder »aufgeklappt« und der in ihm enthaltene Prozessteil angezeigt werden kann. Die Gruppierung bleibt dabei jedoch erhalten, damit der Nutzer den Block später mit einem Klick auf den entsprechenden, in der Gruppierung enthaltenen Button wieder einklappen kann. Ein weiterer Button in der Gruppierung ermöglicht, diese wieder vollständig aufzulösen.

Dieser Mechanismus stellt eine Erweiterung und Adaption des in vielen Software-Entwicklungsumgebungen wie z. B. Eclipse [11] vorhandenen *Code Folding*, mit dem mehrere Zeilen eines Software-Quellcodes zu einer zusammengefasst werden können, für Prozessmodelle dar und wird daher in dieser Arbeit als **Process Block Folding** bezeichnet. Abb. 5.2 zeigt die Funktionsweise des Mechanismus.

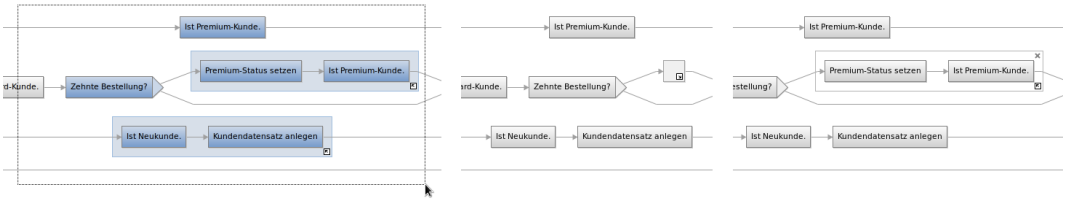


Abbildung 5.2: Process Block Folding. Links: Auswahl von Blöcken; Mitte: zusammengefasster Block; Rechts: wieder aufgeklappter Block

Da sich die Struktur der Prozessbeschreibung beim Zusammenfassen und Aufklappen von größeren Blöcken erheblich ändern kann, kommt es, wenn dies abrupt geschieht, zu einem sprunghaften Wechsel in der Darstellung, wodurch der Anwender sich ggf. im veränderten Prozess neu orientieren muss. Daher ist es nicht nur aus ästhetischen, sondern insbesondere ergonomischen Gesichtspunkten sinnvoll, den Ein- und Ausklappvorgang zu animieren, damit sich die Prozessänderungen in einer flüssigen Bewegung ergeben und der Anwender die Veränderungen mitverfolgen kann, wodurch seine Orientierung im Prozess erhalten bleibt.

Da sich die Tatsache, welche Teile eines Prozesses wichtig sind und welche nicht, bei der interaktiven Analyse der Erfülltheit rasch ändern kann, kann es für den Anwender mühsam sein, ständig nicht relevante Blöcke ein- und relevante auszuklappen. Außerdem sind Änderungen der Relevanz von Prozessbestandteilen möglicherweise nicht erkennbar, wenn sie sich außerhalb des auf dem Bildschirm sichtbaren Bereichs oder innerhalb eines eingeklappten Blocks abspielen. Daher ist es hilfreich, wenn das System einen Mechanismus anbietet, mit dem automatisch Prozessteile so ein- und ausgeklappt werden, dass im jeweiligen Analysekontext relevante Prozessteile sichtbar und weniger relevante Prozessteile ausgeblendet sind, um den vorhandenen Bildschirmplatz optimal auszunutzen.

Daher stellen wir das Verfahren des **AutoFolding** vor, das ein solches automatisches Process Block Folding durchführt und vom Anwender bei Bedarf ein- und ausgeschaltet werden kann. Ist es aktiviert, wird bei Änderungen der Erfülltheit des betrachteten Ausführungssubjekts oder einem Wechsel des Darstellungsmodus durch den Benutzer stets vom System eine Menge von im jeweiligen Kontext wichtigen Knoten ermittelt, deren Sichtbarkeit durch Aufklappen von zuvor automatisch eingeklappten Blöcken sichergestellt wird, während andere Prozessteile automatisch gruppiert und eingeklappt werden, wenn es der vorhandene Bildschirmplatz erfordert. Auf diese Weise automatisch erzeugte Blockgruppierungen haben eine gestrichelte Außenlinie und können so von manuell erzeugten Grup-

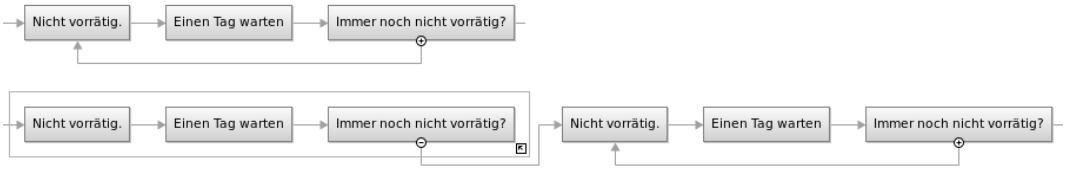


Abbildung 5.3: Schleifenexpansion. Oben: Nicht-expandierte Schleife, eine Iteration dargestellt; Unten: Schleife nach einer Expansion auf zwei Iterationen

pen unterschieden werden. Werden sie später im Zuge des AutoFoldings wieder aufgeklappt, werden solche Gruppierungen automatisch wieder aufgelöst.

Vom Anwender erzeugte und manuell eingeklappte Gruppierungen werden niemals durch das AutoFolding aufgeklappt. Auf diese Weise haben vom Anwender vorgegebene Zusammenfassungen stets Vorrang vor dem AutoFolding. Wenn eine vom Anwender erzeugte Gruppierung jedoch manuell aufgeklappt wurde und keine wichtigen Knoten enthält, kann sie automatisch eingeklappt und (da dies nicht manuell geschah) später auch wieder aufgeklappt werden.

Welche Knoten im Prozess im jeweiligen Darstellungsmodus als wichtig gelten und durch das AutoFolding sichtbar gemacht werden, wird im Zuge der Beschreibungen der einzelnen Modi in den Abschnitten 5.6.5 – 5.6.7 erläutert.

Eine weitere Anpassung der Prozessdarstellung, die nicht der Erhöhung der Übersichtlichkeit, sondern der erweiterten Analyse dient, stellt die **Schleifenexpansion** dar. Diese behandelt das in Abschnitt 5.6.1 beschriebene Problem der Analyse von Schleifen. Hierfür werden die Endknoten von Schleifen im Prozess mit einem Button versehen, der zum Expandieren der Schleife dient. Bei einem Klick auf diesen Button wird in der Prozessbeschreibung eine Kopie der Schleife erstellt und nach dem Endknoten eingefügt, während im ursprünglichen Schleifenblock die Rücksprungkante entfernt wird. Auf diese Weise wird eine Iteration der Schleife ausgelagert, so dass Regelverletzungen bzw. Erfülltheitseigenschaften dargestellt werden können, die in dieser Iteration, nicht jedoch in anderen Iterationen von Bedeutung sind. Der Vorgang kann wiederholt werden, um mehrere Iterationen zu untersuchen. Außerdem enthält die jeweils letzte Iteration einen Button, mit der sie entfernt und wieder in die Schleife integriert werden kann. In Abb. 5.3 ist die Wirkung der Schleifenexpansion dargestellt.

Durch die Schleifenexpansion ergibt sich, dass für die Erfülltheitsangabe in der Prozessdarstellung jeder dargestellte Knoten genau einer Knotenausführung entspricht – dies ist die Grundlage aller folgenden Betrachtungen. Ein in einer Schleife enthaltener Knoten steht daher also nicht mehr für alle möglichen Knotenausführungen in allen Iterationen, sondern nur für die tatsächlich dargestellte Iteration, in der er enthalten ist. Daher werden die Begriffe *Knoten* und *Knotenausführung* in Bezug auf die Prozessdarstellung im Folgenden weitgehend synonym verwendet.

5.6.5 Spurlinien

Bei der Integration der Erfülltheitsangabe in die Prozessdarstellung ist es sowohl für die Gesamterfülltheit als auch die Regelverletzungen erforderlich, Mengen von Ausführungsspuren darzustellen, in denen Regeln erfüllt bzw. nicht erfüllt sind. Hierzu wird das Konzept der **Spurlinien** verwendet, das im Folgenden eingeführt wird.

Eine Spurlinie ist eine in die Prozessdarstellung integrierte zusammenhängende Form aus Liniensegmenten, die eine oder mehrere alternative Ausführungsspuren angibt. Eine Spurlinie verläuft parallel zu den Kontrollflusskanten des Prozessgraphen und beginnt stets als einzelne Linie am Startknoten des Prozesses und endet wiederum als einzelne Linie am Endknoten, kann sich dazwischen jedoch aufspalten und wieder verschmelzen.

Definition 5.1 (Spurlinie)**.** Eine Spurlinie ist definiert als Teilgraph des durch Schleifenexpansion ggf. erweiterten Kontrollflussgraphen des Prozesses. Der Graph der Spurlinie enthält alle Knoten (die je einer Knotenausführung entsprechen) und Kanten des erweiterten Kontrollflussgraphen (ohne Schleifenrücksprung- und Sync-Kanten), die nicht auf sog. *abgewählten Zweigen* eines XOR-Blocks oder in *übersprungenen Iterationen* einer Schleife liegen. Bei jedem XOR-Block können beliebig viele Zweige bis auf einen abgewählt werden – außer, der XOR-Block liegt selbst in einem abgewählten Zweig eines ihn umgebenden Blocks und ist somit komplett nicht im Teilgraphen enthalten. Bei einem Schleifenendknoten einer Iteration einer Schleife, nach dem noch eine oder mehrere weitere Iterationen derselben Schleife in der Darstellung vorhanden sind, ist es möglich, alle diese Iterationen zu überspringen, also ihre Knoten und Kanten nicht in den Graphen der Spurlinie aufzunehmen, und stattdessen eine direkte Iterationsübersprungskante vom betrachteten Endknoten zum auf die Schleife folgenden Knoten hinzuzufügen.

Auf diese Weise ergibt sich der Graph einer Spurlinie stets als zusammenhängender Teilgraph, dessen Startknoten dem Startknoten des Prozessgraphen und dessen Endknoten dem Endknoten des Prozessgraphen entspricht und dessen weitere Knoten jeweils mindestens eine ausgehende und eine eingehende Kante besitzen.

Eine Spurlinie kann an zusätzliche *Bedingungen* gebunden werden, welche Reihenfolge- und Zeitbeziehungen zwischen den Knotenausführungen der durch die Linie beschriebenen Ausführungsspuren angeben. Die Bedingungen werden durch einen logischen Ausdruck angegeben, der die in Abschnitt 2.3.2 verwendeten Prädikate O und T_I verwendet, wobei außer einzelnen Prädikaten keine anderen Teilausdrücke negiert sein dürfen. Die Spurlinie repräsentiert dann nur diejenigen Ausführungsspuren, für die der Ausdruck erfüllt ist. Kommt eine in einem Prädikat verwendete Knotenausführung in einer der Ausführungsspuren *nicht* vor, gilt das Prädikat (bzw. wenn es negiert ist, das negierte Prädikat) in dieser Spur ebenfalls als *erfüllt*.

Jede Spurlinie repräsentiert eine Menge von Ausführungsspuren. Dabei gilt: Eine Ausführungsspur wird genau dann von der Spurlinie repräsentiert, wenn es einen Teilgraphen des Graphen der Spurlinie gibt, der alle Knotenausführungen der Ausführungsspur und keine nicht in der Ausführungsspur enthaltenen Knotenausführungen enthält. Dieser Teilgraph darf außer dem Start- und Endknoten des Prozesses keine Knoten ohne Nachfolger (also keine »Sackgassen«) oder ohne Vorgänger besitzen. Außerdem müssen für die Ausführungsspur alle Bedingungen der Spurlinie erfüllt sein.

Soll eine Menge von Ausführungsspuren in Form von Spurlinien dargestellt werden, müssen diese nur die Spuren aus der Menge abbilden, bei denen die Anzahl der durchgeführten Iterationen der im Prozess enthaltenen Schleifen der Anzahl der derzeit in der Prozessansicht dargestellten Iterationen der jeweiligen Schleifenausführungen entspricht. In diesem Fall werden in den Spurlinien keine Iterationsübersprungkanten verwendet. Es können jedoch auch Spurlinien angezeigt werden, die Spuren repräsentieren, in denen weniger Iterationen durchgeführt werden. In diesem Fall werden Iterationsübersprungkanten benötigt. Dies ist insbesondere dann sinnvoll, wenn in der darzustellenden Spurmenge keine Spuren enthalten sind, bei denen die enthaltenen Iterationen der derzeitigen Darstellung entsprechen.

Um eine allgemeine Menge von Ausführungsspuren darzustellen, benötigt man üblicherweise mehrere Spurlinien, da eine einzelne Spurlinie nicht alle möglichen Kombinationen

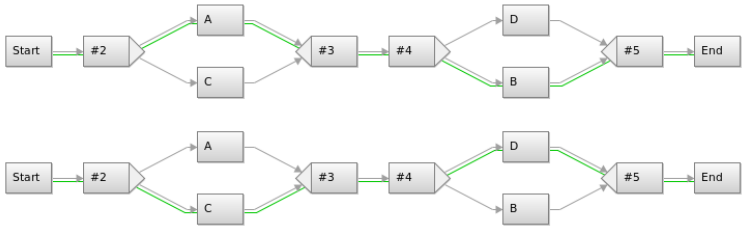


Abbildung 5.4: Zwei Spurlinien, die nicht vereinigt werden können

von Ausführungsspuren darstellen kann. Liegen beispielsweise zwei XOR-Blöcke in einer Sequenz hintereinander und wird in einer Ausführungsspur im ersten Block Zweig 1 und im zweiten Block Zweig 2 ausgewählt, in einer anderen Ausführungsspur hingegen im ersten Block Zweig 2 und im zweiten Block Zweig 1, gibt es keine Spurlinie, die genau diese beiden Spuren repräsentiert, da sich bei Vereinigung der den beiden Ausführungsspuren entsprechenden Teilgraphen ein Teilgraph ergibt, durch den auch andere Ausführungsspuren beschrieben werden, etwa die Auswahl von Zweig 1 in beiden XOR-Blöcken (vgl. Abb. 5.4). Allerdings gilt:

Lemma. *Jede Spurmenge, die Teilmenge der möglichen Ausführungsspuren eines Prozessmodells ist, lässt sich durch eine endliche Menge von Spurlinien darstellen.*

Erläuterung: Die darzustellende Menge von Ausführungsspuren ist, obwohl nur eine beschränkte Anzahl von Iterationen berücksichtigt wird, häufig unendlich groß. Allerdings lassen sich mehrere dieser Spuren zu jeweils einer Äquivalenzklasse strukturell identischer Ausführungsspuren zusammenfassen, die sich nur in der Reihenfolge und den Zeitpunkten der Knotenausführungen unterscheiden. Die Anzahl dieser Äquivalenzklassen ist endlich, da sich die verschiedenen Klassen nur durch die jeweils gewählte Ausgangskante an den XOR-Splitknoten- und Schleifenendknotenausführungen unterscheiden und es aufgrund der Beschränkung der Iterationen nur eine endliche Anzahl von Knotenausführungen und somit auch nur endlich viele Auswahlmöglichkeiten gibt. Jede dieser Äquivalenzklassen lässt sich durch eine triviale Spurlinie darstellen, bei der an XOR-Splits alle Zweige bis auf einen abgewählt und bei Schleifenendknoten, an denen die Schleife verlassen wird, die restlichen Iterationen übersprungen werden und indem die Unterschiede der in der Äquivalenzklasse enthaltenen Spuren durch eine Bedingung zu der Spurlinie angegeben wird. Diese Bedingung kann zwar prinzipiell ein unendlicher Ausdruck sein, wodurch er

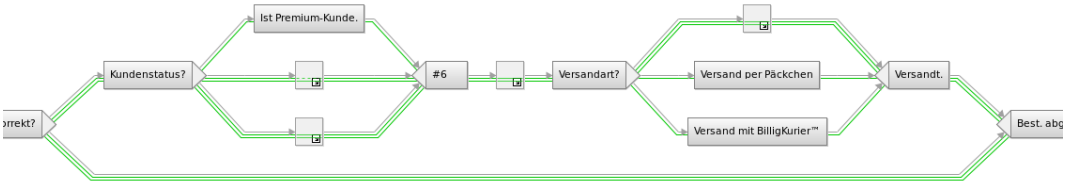


Abbildung 5.5: Prozessmodell mit zwei Spurlinien und teilweise eingeklappten Blöcken

nur noch theoretisch beschrieben werden kann, jedoch sind solche Ausdrücke für die Darstellung der Spurlinie irrelevant.

Die **graphische Darstellung** einer Spurlinie ergibt sich, indem die Kanten ihres Graphen parallel zu den jeweiligen Kontrollflusskanten in den Prozessgraphen eingezeichnet werden, gekennzeichnet durch eine an die jeweilige Rolle der Spurlinie angepasste Farbe. Iterationsübersprungskanten werden dabei nicht gezeichnet.

Für Teilgraphen, die zu einzelnen Knoten eingeklappt wurden und in denen die Spurlinie somit nicht gezeichnet werden kann, wird sie als einzelne gerade Linie dargestellt, die auf den Knoten gezeichnet wird. Ist der komplette eingeklappte Teilgraph im Spurliniengraph enthalten, erfolgt die Darstellung entsprechend der restlichen Spurlinie, ist nur ein Teil des enthaltenen Teilgraphen durch die Spurlinie repräsentiert, als gestrichelte Linie (vgl. Abb. 5.5).

Wenn mehrere Spurlinien vorhanden sind, die sich an Split- und Joinknoten auch gegenseitig überschneiden können, kann es für den Anwender schwierig werden, einzelne Spurlinien zu verfolgen. Daher besteht die Möglichkeit, die Maus über eine Spurlinie zu bewegen, wodurch diese hervorgehoben und alle anderen Spurlinien ausgeblendet werden, wie Abb. 5.6 zeigt. Durch einen Klick auf die Spurlinie kann dieser Zustand fixiert werden, so dass die Hervorhebung nicht bei weiteren Mausbewegungen wieder verschwindet, sondern erst, wenn der Anwender außerhalb der Spurlinie klickt.

Zur Darstellung, dass eine Spurlinie an eine Bedingung gebunden ist ist, werden alle Kanten im zugeordneten Graphen, die ausschließlich auf Wegen liegen, welche eine in einem Prädikat der Bedingung vorkommende Knotenausführung enthalten, gestrichelt gezeichnet.

Wenn die zugrunde liegende Spurmenge unscharf ist (vgl. Abschnitt 4.6), wird ihre obere Schranke durch Spurlinien dargestellt, wobei Wege, die Ausführungsspuren entsprechen,

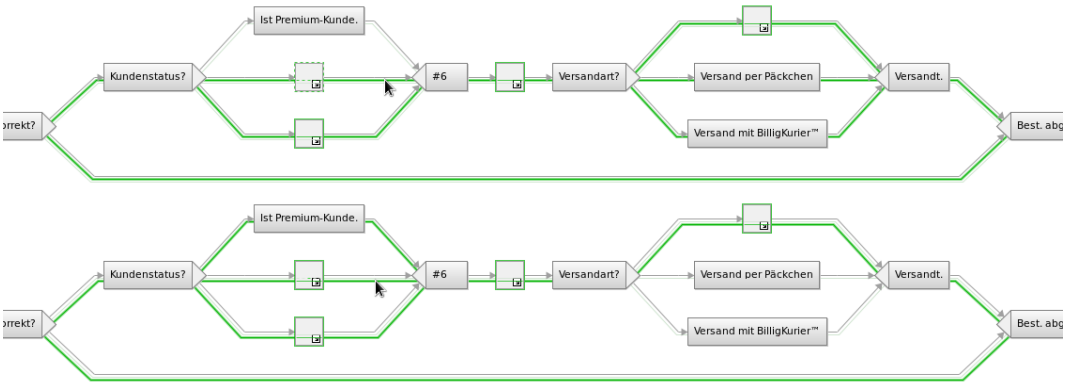


Abbildung 5.6: Zwei Spurlinien (aus Abb. 5.5), davon je eine hervorgehoben

welche nicht sicher in der Menge enthalten sind (also nicht auch in der unteren Schranke liegen), ebenfalls als gestrichelte Linien gezeichnet werden, jedoch mit einem größeren Abstand zwischen den Strichen, um sie von bedingten Spuren zu unterscheiden. Ist die obere bzw. untere Schranke auf einem solchen Weg von einer Bedingung betroffen, werden beide Strichlängen gemischt.

Ist AutoFolding aktiviert, gelten die Split-Knoten aller XOR-Blöcke, bei denen in einer Spurlinie mindestens ein Zweig abgewählt ist (die aber selbst nicht auf einem abgewählten Zweig liegen), als wichtig. Auf diese Weise ist sichergestellt, dass der jeweilige XOR-Block nicht eingeklappt wird und somit die abgewählten Zweige sichtbar sind (da immer nur ganze Blöcke eingeklappt werden können, ist sichergestellt, dass bei Sichtbarkeit des Split-Knotens auch alle Zweige sichtbar sind). Ebenfalls ist bei allen Schleifen, bei denen in mindestens einer Spurlinie mindestens eine dargestellte Iteration übersprungen wird, der Schleifenstartknoten der letzten Iteration als wichtig markiert. Auf diese Weise ist sichergestellt, dass der Schleifenblock nicht in einem eingeklappten Bereich liegt und somit auch die übersprungenen Iterationen sichtbar sind.

Um eine übersichtliche Darstellung zu erhalten, werden nie mehr als fünf Spurlinien dargestellt. Sollte es das jeweilige Ergebnis, insbesondere bei der Darstellung der Gesamterfülltheit, erforderlich machen, mehr Spurlinien anzuzeigen, so werden die ersten vier Spurlinien wie oben beschrieben dargestellt. Die restlichen Spurlinien werden zu einer einzigen, grau dargestellten Kombinations-Spurlinie zusammengefasst. Der ihr zugeordnete Teilgraph entspricht der Vereinigung der Teilgraphen aller enthaltenen Spurlinien. Durch Klick auf diese Kombinations-Spurlinie kann durch die Spurlinien »gescrollt« werden, es

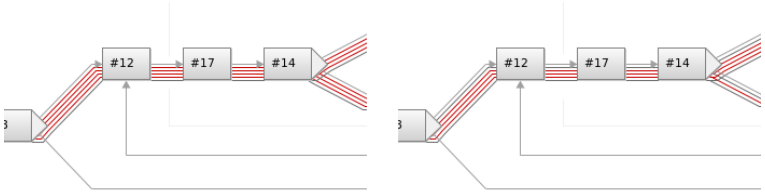


Abbildung 5.7: Darstellung bei zu vielen Spurlinien. Links vor dem Scrollvorgang, rechts nach Klick auf die unterste Spurlinie

wird also eine weitere Spurlinie dargestellt, dafür werden die ersten beiden Spurlinien zu einer Kombinations-Linie zusammengefasst, mit der in die andere Richtung zurück gescrollt werden kann, wie in Abb. 5.7 dargestellt ist.

5.6.6 Gesamterfülltheitsmodus

Der Gesamterfülltheitsmodus ist die Standardansicht der in die Prozessbeschreibung eines Prozessmodells oder einer Prozessinstanz integrierten Ergebnisdarstellung. Er beinhaltet zwei Elemente zur Darstellung der Erfülltheit: Die Darstellung der Gesamterfülltheit durch Spurlinien und eine Übersicht über Regelverletzungen durch Knotenmarkierungen.

Für die Darstellung der **Gesamterfülltheit** kann der Anwender zwischen zwei Varianten wählen: Es kann entweder die Erfülltheit oder die Verletztheit der Regeln im Auswertungssubjekt dargestellt werden. Bei einer Darstellung der *Erfülltheit* wird durch eine oder mehrere grüne Spurlinien die Menge der Ausführungsspuren dargestellt, in denen die Regeln erfüllt sind (Menge F aus Def. 4.1). Ist die Darstellung der *Verletztheit* ausgewählt, wird stattdessen die Menge der Ausführungsspuren, in denen die Regel *nicht* erfüllt ist, durch Spurlinien dargestellt, die in diesem Fall in Rot gehalten sind. Diese Menge V ergibt sich als Komplementärmenge zu F (in der Menge T aller Ausführungsspuren im Auswertungssubjekt – $V = T \setminus F$). Diese Auswahlmöglichkeit ist sinnvoll, da je nach Ergebnis die eine oder andere Darstellungsform besser geeignet sein kann: Ist die Regel in vielen Spuren erfüllt und nur in einigen verletzt, liefert die Darstellung der Verletztheit ein übersichtlicheres Ergebnis; ist die Regel nur in wenigen Spuren erfüllt, kann die Darstellung der Erfülltheit einen besseren Überblick liefern. In Abb. 5.8 sind beide Varianten für dasselbe Ergebnis dargestellt.

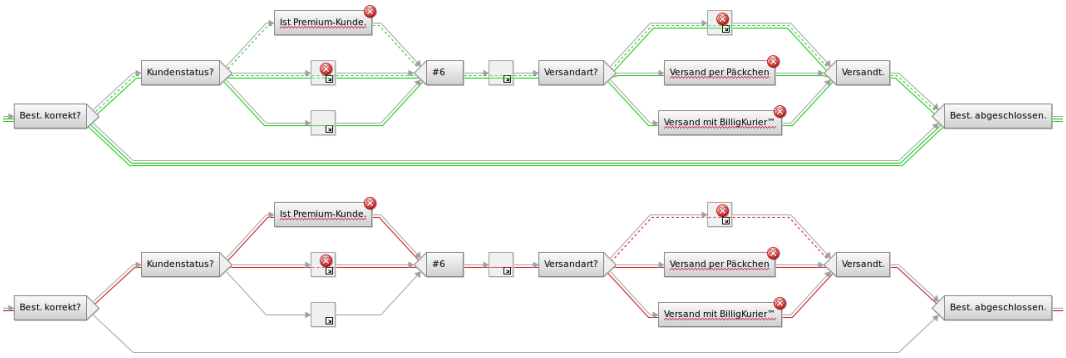


Abbildung 5.8: Darstellung der Gesamterfülltheit eines Prozessmodells. Oben: Darstellung der erfüllenden Spuren; Unten: Darstellung der verletzenden Spuren

Bei einer unscharfen Spurmenge ergibt sich die unscharfe Komplement-Spurmenge, indem als untere Schranke das Komplement der oberen Schranke und als obere Schranke das Komplement der unteren Schranke der ursprünglichen Menge verwendet wird.

Wird eine Spurlinie, die an eine Bedingung gebunden ist, durch Mausbewegung oder Klick hervorgehoben, wird diese Bedingung an der Spurlinie dargestellt, sofern sie feststeht und es sich um keinen zu komplexen Ausdruck handelt. Bedingungen, die nur aus einem einzelnen nicht-negierten Prädikat oder einer Und-Verknüpfung von nicht-negierten Prädikaten bestehen, können dargestellt werden, indem die Knotenausführungen aus den jeweiligen Prädikaten durch Pfeile verbunden werden, die bei Zeitprädikaten zusätzlich mit einem Uhrensymbol versehen sind (vgl. Abb. 5.9). Die Darstellung der Pfeile erfolgt in der Farbe der Spurlinien, bei Hover über ein Uhrensymbol wird das zugeordnete Zeitintervall angezeigt. Bei einem komplexeren Ausdruck ist die Darstellung auf diese Weise nicht möglich. In diesem Fall kann eine Bestimmung der genauen Erfülltheit nur über die manuelle Betrachtung der Regelverletzungen erfolgen oder indem versucht wird, die Komplexität des Ergebnisses durch Regelfilterung (vgl. Abschnitt 5.6.2) zu reduzieren. Eine Darstellung komplexerer Bedingungen an den Spurlinien ist nicht sinnvoll, da die Spurlinien nur einen Überblick über die Erfülltheit geben sollen und für genauere Untersuchungen die Regelverletzungen vorgesehen sind.

Die Übersicht über die Regelverletzungen erfolgt, indem alle Knotenausführungen, die in einer Regelverletzung vorkommen (indem sie in den Bindungen a bzw. b aus Def. 4.5 an Regelvariablen gebunden werden), besonders hervorgehoben werden. Hierzu wird für jede Knotenausführung bestimmt, ob sie in Regelverletzungen vorkommt und wenn ja, in

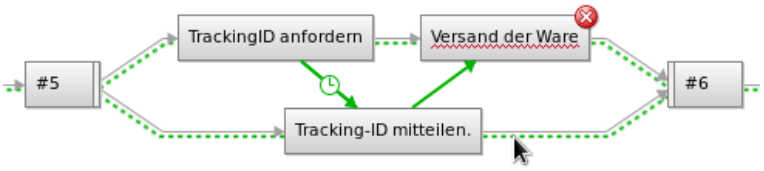


Abbildung 5.9: Darstellung einer bedingten Spurlinie bei Hervorhebung

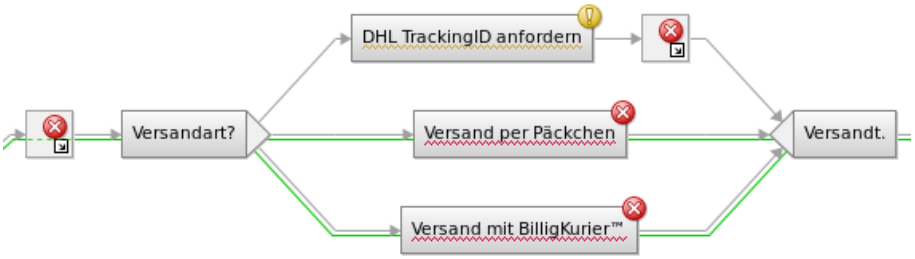


Abbildung 5.10: Markierte Knoten in einem Prozessmodell

welchen. Existiert mindestens eine solche Regelverletzung, wird die Knotenausführung in der Prozessdarstellung hervorgehoben, indem ihr Name unterschlängelt und mit einem Markierungssymbol versehen wird. Das Symbol richtet sich nach der stärksten Regelklasse der jeweils verletzten Regeln und entspricht dem in der Übersichtsliste (Abschnitt 5.4) verwendeten Symbol für diese Klasse, die Farbe der Unterschlängelung entspricht der Farbe des Symbols, wie Abb. 5.10 zeigt.

Diese Darstellung kann nur für sichtbare Knotenausführungen erfolgen. Liegt eine Knotenausführung in einem zu einem einzigen Knoten zusammengefassten Prozessteil, wird dieser Knoten durch ein entsprechendes Symbol hervorgehoben. Liegt eine markierte Knotenausführung außerhalb des sichtbaren Bereichs, wird an den Rand der Darstellungsfläche ein Pfeil in der Hervorhebungsfarbe gezeichnet, der in die Richtung des Knotens zeigt.

Jeder markierte Knoten gilt für das AutoFolding als wichtiger Knoten. Dadurch ist sichergestellt, dass (wenn der Anwender nicht manuell Prozessteile zusammenfasst) kein markierter Knoten in einem eingeklappten Block liegt.

Die in der Übersichtsliste durchgeführten Filteraktionen (siehe Abschnitt 5.6.2) wirken sich auch auf die Prozessdarstellung aus. So werden nur Regelverletzungen für Regeln darge-

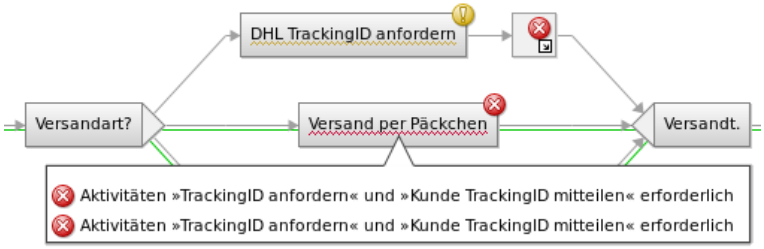


Abbildung 5.11: Liste mit Regelverletzung nach Klick auf Knotenmarkierung

stellt, die der Filterung entsprechen. Auch die Spurlinien stellen nur die Gesamterfülltheit in diesen Regeln dar.

Bei Klick auf eine Regelverletzungsmarkierung an einer Knotenausführung in der Prozessdarstellung wird eine Liste mit allen Regelverletzungen, in denen die entsprechende Knotenausführung vorkommt, angezeigt (vgl. Abb. 5.11). Darin wird jede Regelverletzung durch ihre textuelle Beschreibung und das der Regelklasse entsprechende Symbol repräsentiert. Bei Mausbewegung über einen Eintrag der Liste wird die entsprechende Regelverletzung temporär im Prozess dargestellt, wie in Abschnitt 5.6.7 beschrieben wird. Bei Klick auf den Eintrag wird diese Darstellung fixiert und die Liste geschlossen.

Für Regelverletzungen, die keine Knotenbindungen enthalten, kann keine Knotenmarkierung erfolgen. Für die Ermittlung dieser Regelverletzungen muss daher die Übersichtsliste (vgl. Abschnitt 5.6.2) konsultiert werden, die alle Regelverletzungen enthält und bei Klick ebenfalls im Prozess darstellt.

Ist kein Prozessmodell, sondern eine laufende oder abgeschlossene Prozessinstanz geöffnet, werden in der Prozessdarstellung zusätzlich, wie in [9] vorgesehen, Markierungen angezeigt, die angeben, welche Knoten bereits ausgeführt wurden, welche abgewählt wurden und welche sich gerade in Ausführung befinden. Damit das Vorhandensein von Markierungen zur Darstellung der Erfülltheit und des Ausführungszustandes beim Anwender nicht zu Verwirrung führt, werden die Ausführungs-Markierungen in Grau dargestellt, während die Erfülltheits-Markierungen farbig gehalten sind. Um dennoch auf den ersten Blick bereits einen Überblick zu erhalten, welche Knoten ausgeführt wurden und welche nicht, werden abgewählte Knotenausführungen mit geringerem, laufende und abgeschlossene Knotenausführungen mit etwas höherem Kontrast als normale Knoten dargestellt.

5.6.7 Regelverletzungsmodus

Im **Regelverletzungsmodus** wird eine einzelne Regelverletzung in der Prozessdarstellung angezeigt. Das Ziel ist dabei, dem Anwender zu ermöglichen, diese Verletzung möglichst genau zu analysieren und ihm Hinweise zu geben, wie er sie beheben kann. Um vom Gesamterfülltheitsmodus in diesen Modus zu wechseln, kann in der Übersichtsliste oder in der Liste, die bei Klick auf eine Verletzungsmarkierung im Gesamterfülltheitsmodus erscheint, eine Regelverletzung ausgewählt werden. Um zum Gesamterfülltheitsmodus zurückzukehren, kann auf eine beliebige freie Stelle in der Prozessdarstellung geklickt werden.

Die Darstellung der Regelverletzung in diesem Modus erfolgt, indem der Bedingungsteil der verletzten Regel sowie der verletzte Regelteil aus der Regelverletzung als **Regelgraph-Fragment** in der Prozessdarstellung angezeigt wird. Dabei werden alle Regelknoten, deren entsprechende Variablen durch das Bedingungs-Matching a (vgl. Def. 4.5) sowie die Bindung b an eine oder (bei erweiterten Regelverletzungen) mehrere Knotenausführungen im Prozess gebunden sind, an der Position dieser Knotenausführung(en) in die Prozessdarstellung integriert dargestellt. Dabei handelt es sich grundsätzlich stets um Bedingungs-Occurrence- und Folge-Absence-Knoten. Nicht gebundene (»freie«) Regelknoten werden in einem speziellen Bereich angezeigt, der sich frei auf dem Bildschirm positionieren lässt. Die in diesem Bereich dargestellten Knoten werden dabei nach Möglichkeit automatisch so angeordnet, dass Reihenfolge-Kanten zwischen ihnen stets von links nach rechts zeigen und die Anordnung somit einer gewünschten Anordnung im Prozess entspricht. Alle Knotenausführungen des Prozesses, an die ein Regelknoten gebunden ist, werden für das AutoFolding als wichtige Knoten betrachtet.

Die Regelknoten (sowohl »freie« als auch an Prozess-Knotenausführungen gebundene) werden, analog zur Darstellung in Regelgraphen, durch die den Prädikaten entsprechenden Kanten verbunden, die jeweils analog zur Graphdarstellung durch Symbole gekennzeichnet werden, welche die Art der Kante sowie ihre Zuordnung zu Bedingungs- oder Folgeteil angeben.

Durch diese Darstellung der Regelverletzung, die in Abb. 5.12 gezeigt wird, erkennt der Anwender schnell, welche im Prozess vorhandenen Knoten laut Regel nicht erlaubt sind, welche Knoten fehlen, welche Forderungen fehlende oder bereits vorhandene Knoten erfüllen müssen sowie – durch die Darstellung des Bedingungsteils – unter welchen Bedingungen diese Forderungen gelten.

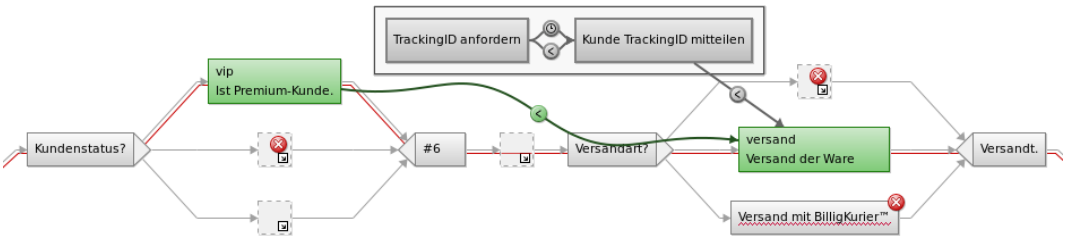


Abbildung 5.12: Darstellung einer Regelverletzung durch Regelgraphfragment und Spurlinie

Zusätzlich werden im Regelverletzungsmodus (anstelle der Spurlinien aus dem Gesamterfülltheitsmodus) **Spurlinien** angezeigt, die die Spurmenge C der Regelverletzung repräsentieren, welche die Ausführungsspuren angibt, in denen die Regelverletzung gilt.

Um die dargestellte Forderung des verletzten Regelteils zu erfüllen und somit die Regelverletzung zu beheben, bestehen (wenn Bedingungs- und Folgteil nicht leer sind) für den Anwender zwei grundsätzliche Möglichkeiten: Er kann entweder dafür sorgen, dass die Bedingung nicht mehr erfüllt ist oder er kann die Forderung erfüllen. Beides kann geschehen, indem Knoten aus dem Prozess entfernt werden, die von Occurrence-Knoten im Bedingungsteil oder Absence-Knoten im Folgeteil der Regel gematcht werden, indem Knoten hinzugefügt werden, die auf Absence-Knoten im Bedingungsteil oder Occurrence-Knoten im Folgeteil der Regel passen, oder indem vorhandene Knoten verändert bzw. verschoben werden. Diese Aktionen können vom Anwender direkt in dieser Ansicht durchgeführt werden, um sofort die Auswirkungen der Aktionen auf die Erfülltheit zu beobachten.

Durch die Darstellung der freien Knoten (im hierfür vorgesehenen Bereich) ergibt sich für den Anwender insbesondere ein Anhaltspunkt, Knoten welcher Aktivitätentypen mit welchen Zusatzanforderungen im Prozess fehlen. Um diese Knoten hinzuzufügen, bietet das System eine direkte Möglichkeit: So kann ein Regelknoten aus dem Regelfragment per Drag & Drop in die Prozessdarstellung gezogen werden. Wird er auf einem vorhandenen Prozessknoten abgelegt, erscheint ein Fenster, aus dem eine zum Aktivitätentyp des Regelknotens passende Aktivitätenvorlage ausgewählt werden kann, die anschließend dem Knoten zugewiesen wird (eine ggf. bereits zugewiesene Aktivität wird dabei überschrieben). Wird er auf einer Kontrollflusskante abgelegt, wird zwischen den beiden durch die Kante verbundenen Prozessknoten ein neuer Knoten eingefügt, dem wiederum eine aus

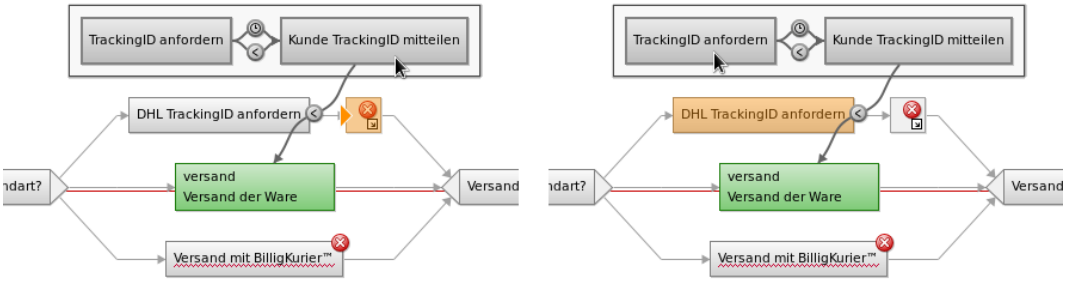


Abbildung 5.13: Hervorhebung von Prozessknoten bei Mausbewegung über Regelgraphknoten. Rechts: Sichtbarer Knoten; Links: Knoten in eingeklapptem Bereich

einer Liste auswählbare, passende Aktivität zugewiesen wird. Auf diese Weise lässt sich das Prozessmodell schnell und einfach auf Grundlage der Regel anpassen.

Wird die Maus über einen der freien Occurrence-Knoten bewegt, werden in der Prozessdarstellung alle Knotenausführungen, die dem Aktivitätentyp des Occurrence-Knotens entsprechen, hervorgehoben. Hierdurch ergeben sich für den Anwender Anhaltspunkte, welche bereits vorhandenen Knotenausführungen die Forderung nach dem Vorhandensein eines Knotens des angegebenen Typs erfüllen würden, wenn nicht zusätzliche Anforderungen (Reihenfolge-, Zeit- oder Ungleichheitsbedingungen) verletzt wären. Es werden auch Knotenausführungen hervorgehoben, die nicht auf der (durch die Spurlinien dargestellten) Spurmenge der Regelverletzung liegen. Auf diese Weise besteht für den Anwender die Möglichkeit, zu erkennen, ob die Regelverletzung behoben werden könnte, indem z. B. ein Knoten auf einen anderen XOR-Zweig verschoben würde. Liegt eine auf solche Weise markierte Knotenausführung außerhalb des sichtbaren Bildschirmbereichs, wird am entsprechenden Bildschirmrand ein Pfeilsymbol angezeigt, das in Richtung der Knotenausführung zeigt, damit sie vom Anwender nicht übersehen wird. Liegt sie innerhalb eines eingeklappten Bereichs, wird ein Pfeil gezeichnet, der in diesen Bereich zeigt (vgl. Abb. 5.13).

Außerdem wird, während eine Regelverletzung in diesem Modus betrachtet wird, eine (durch den Anwender frei positionierbare) Liste mit den **Alternativverletzungen** der betrachteten Verletzung angezeigt. Mit dieser Liste kann der Anwender schnell erkennen, welche Regelverletzungen statt der derzeit betrachteten behoben werden können. Die Liste verhält sich analog zur Liste der Regelverletzungen, die bei Klick auf eine Verletzungsmar-

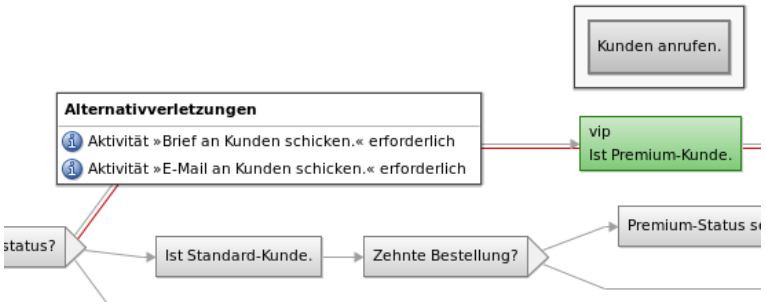
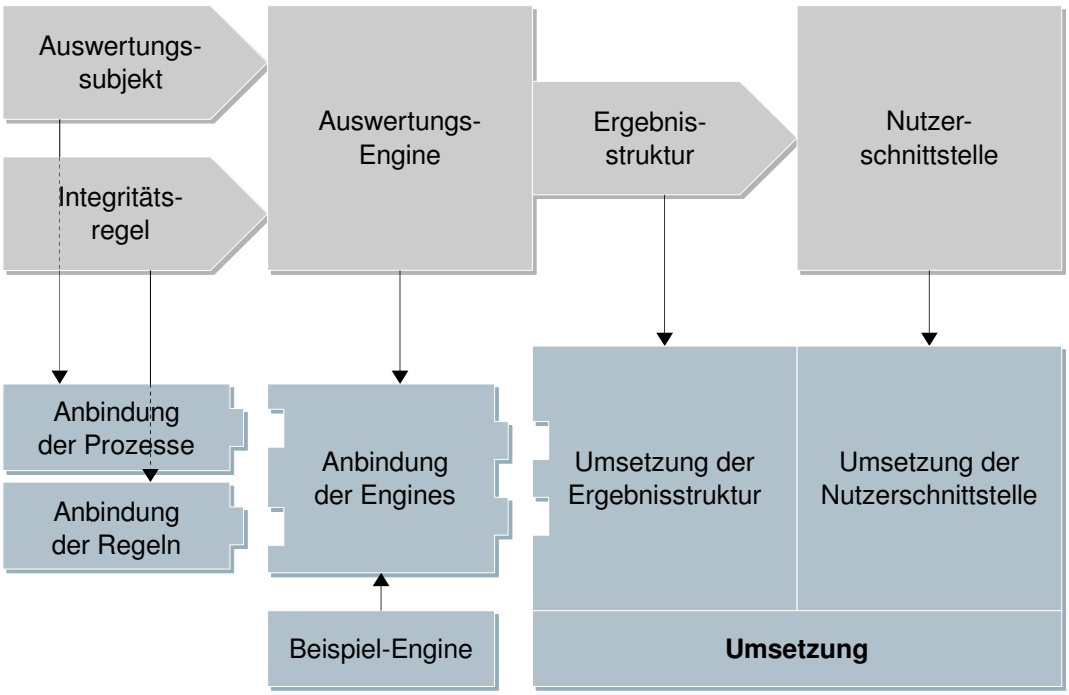


Abbildung 5.14: Liste mit Alternativverletzungen

kierung an einem Prozessknoten erscheint: Es werden textuelle Beschreibungen der entsprechenden Verletzungen angezeigt, bei Mausbewegung über eine Verletzung wird diese temporär in der Prozessdarstellung angezeigt und bei einem Klick wechselt die Ansicht vollständig zur ausgewählten Regelverletzung. Eine Beispielliste ist in Abb. 5.14 dargestellt.

Für die integrierte Darstellung des Regelfragments in der Prozessdarstellung sind einige Sonderfälle zu beachten: Liegt eine Knotenausführung, an die ein Knoten aus dem Regelfragment gebunden ist, in einem zu einem einzelnen Knoten zusammengefassten Prozessteil, wird dieser Knoten hervorgehoben. Ein weiterer Sonderfall ergibt sich, wenn mehrere Regelknoten an dieselbe Knotenausführung im Prozess gebunden sind. Dies ist möglich, wenn zwischen ihnen keine Ungleichheits- oder Reihenfolgebeziehung angegeben ist. In diesem Fall werden die entsprechenden Regelknoten überlappend an der Position der jeweiligen Knotenausführung dargestellt, einzelne Regelknoten können durch Mausklick in den Vordergrund geholt werden.

6 Praktische Umsetzung



Wir haben die Ergebnisse dieser Arbeit – die in Kapitel 4 vorgestellte Ergebnisstruktur und die in Kapitel 5 eingeführte Nutzerschnittstelle – beispielhaft in einer prototypischen Demonstrator-Anwendung umgesetzt, um die Realisierbarkeit zu demonstrieren und weitere aufbauende Arbeiten zu ermöglichen. Dieses Kapitel beschreibt Aufbau und Funktionsweise dieses Demonstrators.

6.1 Grundlegendes

Als Grundlage der Umsetzung der Ergebnisse dieser Arbeit mussten Repräsentationen für die zugrundeliegenden Elemente – Aktivitätentypen, Integritätsregeln, Prozessmodelle und Ausführungsspuren – gewählt werden. Diese werden in Abschnitt 6.2 bis 6.5 vorgestellt. Den nächsten Schritt stellte die Entwicklung einer Datenstruktur zur Abbildung der Ergebnisstruktur dar, die sich gut für die Aufnahme der Ergebnisse der Auswertungsalgorithmen und für die Darstellung der Daten an der Nutzerschnittstelle eignet. Diese wird in Abschnitt 6.6 beschrieben.

Bei der Umsetzung der Nutzerschnittstelle beschränkten wir uns auf die in Abschnitt 5.6 vorgestellte Prozessansicht, da diese die mit Abstand umfangreichste Ansicht darstellt und neben der detaillierten Darstellung der Erfülltheit im Prozess auch eine Übersichtsliste enthält, der die in Regel- und Gesamtlistenansicht verwendeten Listen weitgehend entsprechen. Die Prozessansicht kann in verschiedenen Anwendungen eines Prozess-Management-Systems verwendet werden, die wichtigste Komponente stellt hierbei der Prozessvorlagen-Editor dar. Daher wurde für den Demonstrator die Einbindung der Prozessansicht in diese Komponente umgesetzt. Die Umsetzung der Nutzerschnittstelle wird in Abschnitt 6.7 vorgestellt.

Um die Funktionalität von Ergebnisstruktur und Nutzerschnittstelle und ihre Eignung für reale Auswertungsergebnisse zu demonstrieren, wurde ebenfalls eine Auswertungs-Engine benötigt, die Integritätsregeln über einem Auswertungssubjekt auswerten und die Ergebnisse in der Ergebnisstruktur repräsentieren kann. Hierfür wurde ein einfacher Auswertungsalgorithmus in eine Beispiel-Engine umgesetzt, mit dem Ziel, möglichst detaillierte Ergebnisse zu erhalten. Die Anbindung von Algorithmen an das System wird in Abschnitt 6.8, der verwendete Beispiel-Algorithmus in Abschnitt 6.8.2 beschrieben.

Die Implementierung des Demonstrators erfolgte auf Grundlage der vom Institut für Datenbanken und Informationssysteme der Universität Ulm und AristaFlow [1] entwickelten *AristaFlow BPM Suite* [9]. Hierbei wurde der Demonstrator als Plugin für den *AristaFlow Process Template Editor* entwickelt, welcher auf der *Eclipse*-Plattform [11] basiert. Daher erfolgte die Entwicklung in der Programmiersprache *Java* [14], unter Verwendung der Entwicklungsbibliotheken von AristaFlow und Eclipse. Die Benutzeroberfläche verwendet das in Eclipse eingesetzte *Standard Widget Toolkit* (SWT) [30]. Der erstellte Code wurde mittels *JavaDoc* dokumentiert.

Mit dem entwickelten Demonstrator ist es somit möglich, im Prozessvorlagen-Editor Integritätsregeln über AristaFlow-Prozessvorlagen auswerten zu lassen und die Auswertungsergebnisse mit der in dieser Arbeit entwickelten Nutzerschnittstelle zu analysieren.

6.2 Aktivitätentypen

Aktivitätentypen werden grundsätzlich durch die abstrakte Klasse `ActivityType` repräsentiert. Diese bietet Methoden, um den Namen und die Obertypen des jeweiligen Aktivitätentyps abzufragen. Für die tatsächlichen Aktivitätentypen stehen zwei Subklassen zur Verfügung: `BaseActivityType` für Aktivitätentypen, die *AristaFlow Activity Templates* entsprechen, und `AbstractActivityType` für abstrakte Aktivitätentypen. Letztere werden im Demonstrator jedoch nicht verwendet. Ein `BaseActivityType`-Objekt ist jeweils fest einem *AristaFlow Activity Template* zugeordnet. Die Klasse `ActivityTypeRegistry` bietet die Möglichkeit, für ein *AristaFlow Activity Template* die zugehörige `BaseActivityType`-Klasse zu erhalten.

6.3 Integritätsregeln

Integritätsregeln werden durch eine verschachtelte Datenstruktur aus Objekten verschiedener Klassen repräsentiert, welche eine einfache Abbildung in/aus Regelgraphen und prädikatenlogischen Regeln ermöglicht. Eine Integritätsregel wird durch ein Objekt der Klasse `IntegrityRule` umgesetzt. Diese Klasse enthält drei Felder: ein Feld vom Typ `Antecedent`, das den Bedingungsteil angibt, ein Array vom Typ `ConsequenceElement`, welches die Folgeglieder beinhaltet, und das Feld `ruleClass`, das die Regelklasse angibt, wobei die möglichen Werte durch Konstanten repräsentiert werden, die den Regelklassen aus Abschnitt 3.3.2 entsprechen.

Die Klasse `Antecedent` enthält drei Felder: ein Array vom Typ `OccurrenceVariable`, das die positiven Variablen aus dem Bedingungsteil und ihre Typisierungen enthält, ein Array vom Typ `RelationPredicate`, welches die diese Variablen verbindenden zweielementigen Prädikate repräsentiert, sowie ein Array vom Typ `NegativeElement`, das die Negativelemente des Bedingungsteils enthält. Die Klasse `ConsequenceElement` beinhaltet ein Array vom Typ `ExistenceGroup`, das für jede Existenzgruppe des Folgeglieds je

ein Objekt enthält. Die Klasse `ExistenceGroup` ist aufgebaut wie die Klasse `Antecedent` und enthält wiederum Arrays mit positiven Variablen, Prädikaten und Negativelementen (tatsächlich besitzen beide Klassen eine gemeinsame abstrakte Oberklasse, `RuleMainExpression`, die diese Felder enthält).

Die Klasse `NegativeElement` besitzt zwei Felder: ein Objekt der Klasse `AbsenceVariable`, das die im Negativelement definierte Variable repräsentiert, sowie ein Array von `RelationPredicate`-Objekten, welche die Prädikate des Negativelements darstellen. Die Klassen `OccurrenceVariable` und `AbsenceVariable` sind von der Klasse `Variable` abgeleitet, welche ein `String`-Feld mit dem Variablennamen sowie ein Feld vom Typ `ActivityType`, das die Typisierung der Variablen angibt, enthält. Die abstrakte Klasse `RelationPredicate` besitzt zwei Felder vom Typ `Variable`, die auf die erste und zweite im Prädikat verwendete Variable verweisen. Als Subklassen für die eigentlichen Prädikate existieren `OrderPredicate` für Reihenfolgebeziehungen, `InequalityPredicate` für Ungleichheitsbeziehungen und `TemporalPredicate` für Zeitbeziehungen, wobei letztere Klasse die zusätzlichen Felder `intervalStart` und `intervalEnd` vom Typ `double` enthält, die das mit der Zeitbeziehung verknüpfte Interval angeben. Bei einem nach oben offenem Intervall wird der Wert `Double.POSITIVE_INFINITY` als `intervalEnd` verwendet.

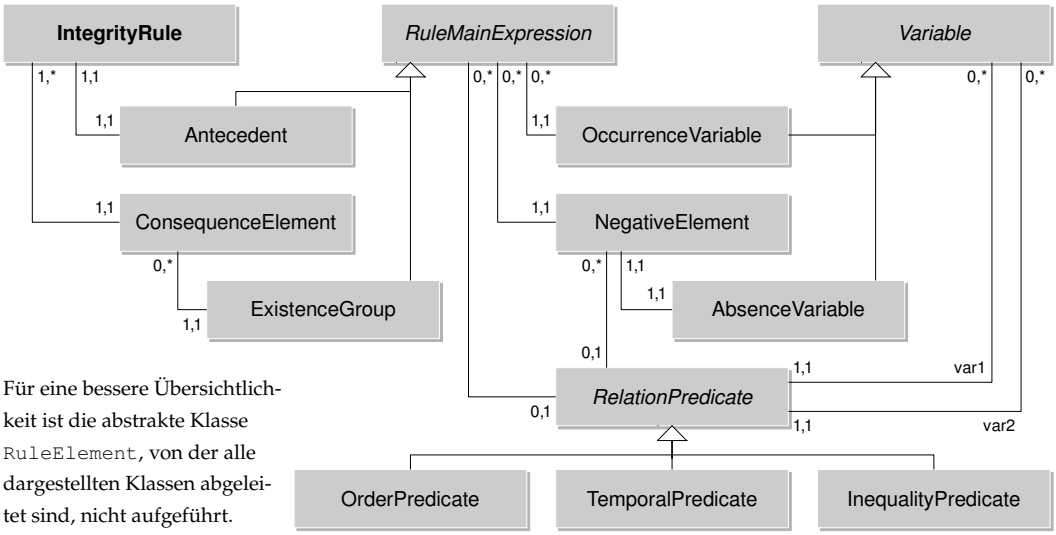
Alle beschriebenen Klassen zur Umsetzung von Regelteilen erweitern die abstrakte, leere Klasse `RuleElement`. Die gesamte Klassenstruktur ist in Abb. 6.1 dargestellt.

6.4 Prozessmodelle

Für die Umsetzung der Prozessmodelle und -knoten greift der Demonstrator direkt auf die durch die AristaFlow-API bereitgestellte Funktionalität zurück. Prozessmodelle werde dementsprechend durch AristaFlow-`Template`-Objekte, Knoten durch ihre `nodeID` repräsentiert.

6.5 Ausführungsspuren

Wie in Kapitel 4 dargestellt wurde, stellt eine Grundlage für die Angabe der Gesamterfülltheit und für Regelverletzungen jeweils eine Menge von Ausführungsspuren dar, die



Für eine bessere Übersichtlichkeit ist die abstrakte Klasse `RuleElement`, von der alle dargestellten Klassen abgeleitet sind, nicht aufgeführt.

Abbildung 6.1: Klassenstruktur zur Darstellung von Integritätsregeln

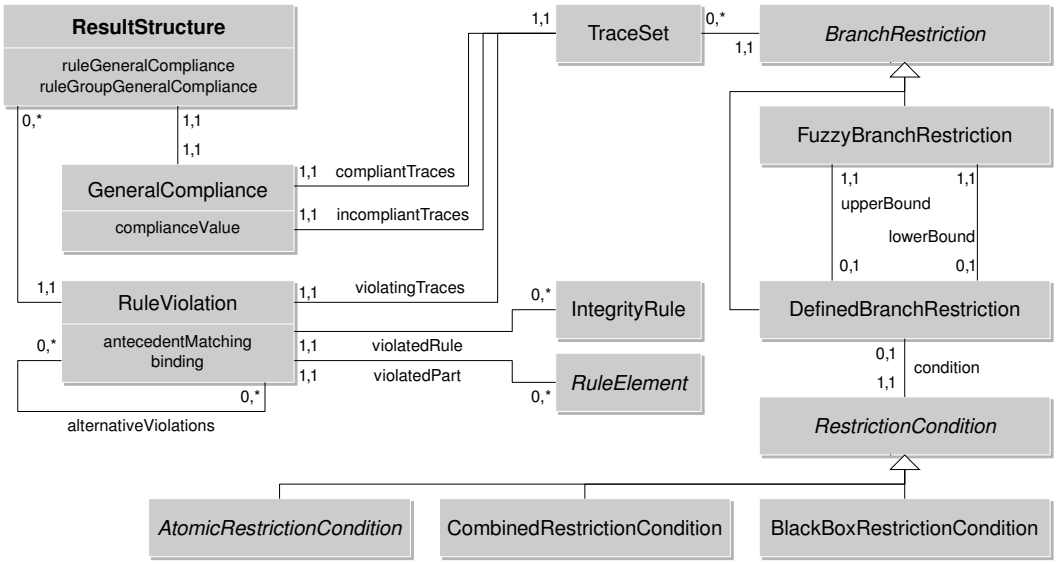


Abbildung 6.2: Klassenstruktur für die Ergebnisstruktur und Spurmengen

eine Teilmenge der durch das Prozessmodell (bzw. einer durch Ad-Hoc-Änderungen modifizierten Variante) beschriebenen Ausführungsspuren darstellt. Diese ergeben sich üblicherweise durch Einschränkungen wie die Auswahl bestimmter Zweige in XOR-Verzweigungen, Festlegung der Ausführungsreihenfolge von Knoten auf parallelen Spuren oder Einschränkungen der zeitlichen Beziehungen zwischen Knotenausführungen aus dem Prozessmodell.

Wie bereits in Abschnitt 4.3 und bei der Einführung der Spurlinien in Abschnitt 5.6.5 beschrieben wurde, enthalten diese Mengen aufgrund der beliebig langen Ausführungszeiten von Knoten und der theoretisch beliebig oft ausführbaren Schleifen häufig unendlich viele Ausführungsspuren, wenn die Auswertung nicht auf eine einzige Spur beschränkt ist. Für die praktische Umsetzung kann eine solche Menge von Ausführungsspuren daher nicht durch Aufzählung aller möglicher Spuren angegeben werden, sondern es ist eine endliche Datenstruktur notwendig, welche die in der Menge enthaltenen Spuren beschreibt. Als Sonderfall kann solch eine Datenstruktur auch zur Beschreibung einer einzelnen Ausführungsspur verwendet werden.

Das Konzept hinter der in dieser Umsetzung verwendeten Datenstruktur zur Beschreibung von Spurmengen entspricht weitgehend dem der in Abschnitt 5.6.5 eingeführten Spurlinien. Dies ist sinnvoll, da beide Konzepte der Darstellung von Spurmengen dienen – Spurlinien auf dem Bildschirm, die Datenstruktur im Speicher. Spurlinien stellen somit eine Visualisierung der in der hier vorgestellten Datenstruktur gespeicherten Spurmengen dar.

Eine Spurmenge wird durch ein Objekt der Klasse `TraceSet` repräsentiert. Dieses beinhaltet eine Menge von Objekten der abstrakten Klasse `BranchRestriction`, die in einem Array abgelegt sind. Für nicht unscharfe Spurmengen sind dies Objekte der Subklasse `DefinedBranchRestriction`. Die Elemente des `TraceSet` besitzen eine Oder-Semantik, d.h. eine Ausführungsspur ist in der betrachteten Spurmenge enthalten, wenn sie durch mindestens eine der im `TraceSet` enthaltenen `BranchRestriction`s repräsentiert wird.

6.5.1 Definierte Zweigeinschränkungen

Eine `DefinedBranchRestriction` (definierte Zweigeinschränkung) beschreibt eine Teilmenge der in einem Prozessmodell möglichen Ausführungsspuren. Ein Objekt dieser Klasse repräsentiert alle Ausführungsspuren, bei denen bestimmte Entscheidungen während

der Prozessausführung in bestimmter Weise getroffen werden und deren Knotenausführungen in bestimmten Beziehungen zueinander stehen.

Die Entscheidungen an XOR-Verzweigungen werden durch das Feld `deSelectedBranchCodes` repräsentiert. Dies ist eine durch eine `HashMap` realisierte Abbildung von Knotenausführungen von XOR-Split-Knoten auf Mengen von XOR-Zweig-IDs. Eine Ausführungsspur ist nur dann in der durch das Objekt beschriebenen Menge enthalten, wenn bei Auftreten einer der in der Schlüsselmenge der `HashMap` enthaltenen Knotenausführungen keiner der durch diese Abbildung angegebenen Zweige ausgewählt wird.

Die Entscheidungen über das Fortsetzen oder Verlassen von Schleifen werden durch das Feld `loopIterations` angegeben. Dies ist eine `HashMap`, die Knotenausführungen von Schleifenstartknoten jeweils eine Folge von abgeschlossenen Integer-Intervallen zuordnet. Das letzte Intervall der Folge kann auch nach oben offen sein. Die Knotenausführungen stellen jeweils die erste Knotenausführung einer Ausführung der Schleife dar. Eine Ausführungsspur ist nur dann in der durch das `DefinedBranchRestriction`-Objekt beschriebenen Teilmenge enthalten, wenn die Anzahl der in der Spur durchgeführten Iterationen der jeweiligen Schleifenausführung in einem der durch dieses Feld angegebenen Intervalle liegt.

Die zusätzlichen Bedingungen für Beziehungen zwischen bestimmten Knotenausführungen werden durch das Feld `condition` bestimmt, welches ein Objekt vom Typ `RestrictionCondition` enthält. Eine Ausführungsspur ist nur dann in der Zweigeinschränkung enthalten, wenn alle in ihr enthaltenen Knotenausführungen die durch dieses Objekt repräsentierten Bedingungen erfüllen. Unterklassen von `RestrictionCondition` beschreiben einzelne Reihenfolge- oder Zeitrelationen zwischen zwei Knotenausführungen, die auch negiert sein können, sowie verschachtelte Und- oder Oder-Verknüpfungen zwischen mehreren Beziehungsangaben. Analog zu Spurlinien gilt, dass eine Reihenfolge- oder Zeitrelation, die eine nicht in der Spur enthaltene Knotenausführung betrifft, in dieser Spur als erfüllt gilt (ist sie negiert, gilt die Relation selbst als nicht erfüllt und die negierte Relation somit als erfüllt).

Eine besondere Unterklasse stellt `BlackBoxRestrictionCondition` dar. Da in Spurlinien nur einfache Bedingungen dargestellt werden können, ist es nicht notwendig, Bedingungsausdrücke mit hoher Komplexität in der Ergebnisstruktur anzugeben. Stattdessen kann als `condition` ein Objekt dieser Klasse verwendet werden, das nur eine Liste der

Knotenausführungen enthält, die von einer Bedingung betroffen sind, aber nicht den genauen Aufbau der Bedingung angibt.

Analog zu Spurlinien gilt, dass sich jede beliebige Teilmenge der Ausführungsspuren eines Prozessmodells durch eine endliche Menge von `DefinedBranchRestriction`-Objekten darstellen ließe, wenn man unendlich große Bedingungsausdrücke zuließe und die Anzahl der Iterationen in Schleifen beschränkt. Da jedoch nur endliche Bedingungsausdrücke möglich sind und die Anzahl der Iterationen im Allgemeinen unbeschränkt ist, lassen sich nicht alle Teilmengen vollständig repräsentieren. Da an der Nutzerschnittstelle jedoch grundsätzlich nur endliche Objekte dargestellt werden können, ist diese Einschränkung in der praktischen Anwendung nicht relevant: Nur für Iterationen, die auf der Nutzeroberfläche in der Prozessdarstellung angezeigt werden, sind ausführliche Angaben in den Spurmengen der Ergebnisstruktur notwendig, da nur diese auch durch Spurlinien dargestellt werden können. Statt theoretisch unendliche Bedingungsausdrücke anzugeben, kann, wie oben beschrieben, ein Objekt der Klasse `BlackBoxRestrictionCondition` verwendet werden, welches die sichtbaren, in der Bedingung enthaltenen Knotenvorkommen enthält.

6.5.2 Unscharfe Zweigeinschränkungen

Eine unscharfe Spurmenge wird durch ein `TraceSet` repräsentiert, das neben `DefinedBranchRestriction`-Objekten auch `FuzzyBranchRestriction`-Objekte enthält. Diese sind Datenstrukturen mit zwei Feldern, `lowerBound` und `upperBound`, welche `DefinedBranchRestriction`-Objekte enthalten, die die obere bzw. untere Schranke der durch das `FuzzyBranchRestriction`-Objekt repräsentierten unscharfen Teil-Spurmenge darstellen. `lowerBound` muss dabei eine Teilmenge der von `upperBound` repräsentierten Spuren darstellen. `lowerBound` kann den Wert `null` enthalten, wenn die untere Schranke einer leeren Spurmenge entspricht.

Dies entspricht keiner 1:1-Abbildung der Definition unscharfer Spurmengen (Def. 4.7): Während dort je eine komplette Spurmenge als obere und untere Schranke angegeben wird, ist bei der Umsetzung je eine obere und untere Schranke pro Zweigeinschränkung angegeben. Eine Abbildung einer logischen unscharfen Spurmenge auf eine Menge von unscharfen Zweigeinschränkungen ist jedoch problemlos möglich, da obere und untere Schranke durch je eine endliche Menge von `DefinedBranchRestriction`s darstellbar sind, welche sich wiederum (ggf. unter weiterer Aufspaltung einzelner Objekte) zu `FuzzyBranchRestriction`s zusammenfassen lassen.

6.5.3 Knotenausführungen

Eine Knotenausführung wird durch ein Objekt der Klasse `NodeEx` repräsentiert. Als direkte Umsetzung von Def. 2.1 enthält ein Objekt dieser Klasse im Feld `node` die dem AristaFlow-Prozessmodell entstammende ID des Knotens sowie im Feld `iterations` eine Liste der Iterationsnummern für die den Knoten enthaltenden Schleifen. Diese Liste wird in Form eines `int`-Arrays gespeichert.

6.6 Ergebnisstruktur

Grundlage für die Umsetzung der Ergebnisstruktur stellt die Klasse `ResultStructure` dar. Ein Objekt dieser Klasse repräsentiert das Ergebnis der Auswertung einer oder mehrerer Integritätsregeln über dem Auswertungssubjekt, also im Demonstrator stets über dem Prozessmodell.

Analog zur konzeptionellen Ergebnisstruktur besitzt diese Klasse zwei Felder: Einen Verweis auf ein Objekt der Klasse `GeneralCompliance`, welches die Gesamterfülltheit enthält, und ein `Array` von `RuleViolation`-Objekten, die die Regelverletzungen angeben.

Zusätzlich ist das Feld `ruleGeneralCompliance` enthalten, welches eine Abbildung enthält, die für jede überprüfte Integritätsregel den Grad der Gesamterfülltheit angibt. Dieser Grad wird durch eine Konstante angegeben, analog zu dem Wert, der in der im Folgenden vorgestellten Klasse `GeneralCompliance` im Feld `complianceValue` verwendet wird. Ebenfalls ist ein Feld `ruleGroupGeneralCompliance` enthalten, welches für Gruppen von Regeln einen gemeinsamen Gesamterfülltheitswert angibt. Diese Funktionalität wird genutzt, um an der Nutzeroberfläche gemeinsame Ergebnisse für alle in einer Datei enthaltenen Regeln anzugeben.

Die Struktur der für die Ergebnisstruktur verwendeten Klassen ist in Abbildung 6.2 dargestellt.

6.6.1 Gesamterfülltheit

Die Klasse `GeneralCompliance` enthält die Felder `complianceValue`, `compliantTraces` und `incompliantTraces`. `complianceValue` ist ein Wert, der den Grad der

Gesamterfülltheit aller betrachteten Regeln angibt. Folgende Konstanten können als Wert verwendet werden:

- `SATISFIED` entspricht dem Wert *erfüllt* aus Def. 4.1.
- `SATISFIED_BECAUSE_ANTECEDENT_VIOLATED` entspricht dem Wert *nicht passend* aus Def. 4.2.
- `SATISFIABLE` entspricht dem Wert *erfüllbar* aus Def. 4.1.
- `VIOLATED` entspricht dem Wert *verletzt* aus Def. 4.1.
- `SATISFIABLE_FUZZY` entspricht dem Wert *erfüllbar* aus Def. 4.8.
- `VIOLATED_OR_SATISFIABLE` entspricht dem Wert *evtl. verletzt* aus Def. 4.8.
- `SATISFIABLE_OR_SATISFIED` entspricht dem Wert *evtl. erfüllt* aus Def. 4.8.
- `UNCERTAIN` entspricht dem Wert *unbestimmt* aus Def. 4.8.

`compliantTraces` ist ein Feld vom Typ `TraceSet`, das die Menge der Ausführungsspuren angibt, in denen die betrachteten Regeln erfüllt sind.

`incompliantTraces` ist ebenfalls ein Feld vom Typ `TraceSet`, das die Menge aller Ausführungsspuren angibt, in denen die betrachteten Regeln *nicht* erfüllt sind. Dies wird für die Darstellung der Spurlinien verletzender Spuren benötigt. In der logischen Definition der Gesamterfülltheit ist dieser Wert nicht enthalten, da er sich logisch direkt als Komplement der Menge F aus Def. 4.1 ergibt ($V = T \setminus F$).

Die angegebenen Spurmengen werden zur Darstellung der Erfülltheit im Prozessmodell verwendet und müssen daher auch nur die derzeit dargestellten Iterationen enthalten. Zwar müsste in Fällen, in denen nicht dargestellte Spuren für die Erfülltheit relevant sind, theoretisch eine unscharfe Spurmenge angegeben werden, in deren oberer Schranke die weiteren Iterationen enthalten sind, da diese aber nicht dargestellt werden können, ist diese Angabe überflüssig. Der `complianceValue`-Wert hingegen muss auch nicht dargestellte Iterationen berücksichtigen.

6.6.2 Regelverletzungen

Die Klasse `RuleViolation`, deren Instanzen jeweils eine Regelverletzung repräsentieren, besitzt folgende Felder, die bis auf `alternativeViolations` jeweils genau einem Feld aus dem Tupel *rv* in Def. 4.5 entsprechen:

- `violatedRule` – Ein Verweis auf das `IntegrityRule`-Objekt der verletzten Regel.
- `AntecedentMatching` – Ein `Map`-Objekt, das den `OccurrenceVariable`-Objekten aus dem Bedingungsteil der Regel je ein Array von `NodeEx`-Objekten zuordnet und dem Bedingungs-Matching *a* aus *rv* entspricht.
- `violatedPart` – Ein `RuleElement`-Objekt aus `violatedRule`, das den verletzten Regelteil angibt.
- `binding` – Ein `Map`-Objekt, das der `AbsenceVariable` aus dem `violatedPart` ein `NodeEx`-Objekt zuordnet, falls es sich beim verletzten Regelteil um ein einzelnes Negativelement handelt. Andernfalls hat dieses Feld den Wert `null` (vgl. *b* aus *rv*).
- `violatingTraces` – Ein `TraceSet`-Objekt, das die Ausführungsspuren angibt, in denen die Regelverletzung auftritt.
- `alternativeViolations` – Ein Array von `RuleViolation`-Objekten, das die Alternativverletzungen dieser Verletzung angibt (vgl. Def. 4.6). Dieses Feld kann erst gesetzt werden, wenn alle `RuleViolation`-Objekte erzeugt wurden.

6.7 Nutzerschnittstelle

Wie bereits zu Anfang dieses Kapitels beschrieben, wurde der Demonstrator als Plugin in den *AristaFlow Process Template Editor* integriert. Durch die Erweiterbarkeit der AristaFlow-Plattform kann so direkt auf im System vorhandene Prozessmodelle zugegriffen werden, wodurch sich der Demonstrator mit realen Prozessen einsetzen lässt. Die Erweiterungen der Prozessansicht, die in Abschnitt 5.6 vorgestellt wurden, sind jedoch so umfangreich, dass es nicht umsetzbar war, mittels eines externen Plugins die vorhandene Prozessbearbeitungs-Komponente des *Process Template Editor* um alle in dieser Arbeit eingeführten Elemente zu erweitern. Daher wurde für den Demonstrator eine neue Bearbeitungskomponente erstellt, der sog. **Rule Based Process Editor** (RBPE). Die umgesetzte Komponente bietet keine Möglichkeiten zur Bearbeitung, da der Fokus auf der Visualisierung der Constrainterfüllung lag. Somit ist für die tatsächliche Editierung von Prozessen weiterhin die Originalkomponente des *AristaFlow Process Template Editor* erforderlich. Es wäre jedoch möglich, die entwickelte Komponente um Editierfähigkeiten zu erweitern.

Der *Rule Based Process Editor* wird durch eine eigene Eclipse-Editor-Komponente realisiert, die in der Klasse `RBPEditor` vorliegt. Dieser Editor ist den AristaFlow-Prozess-

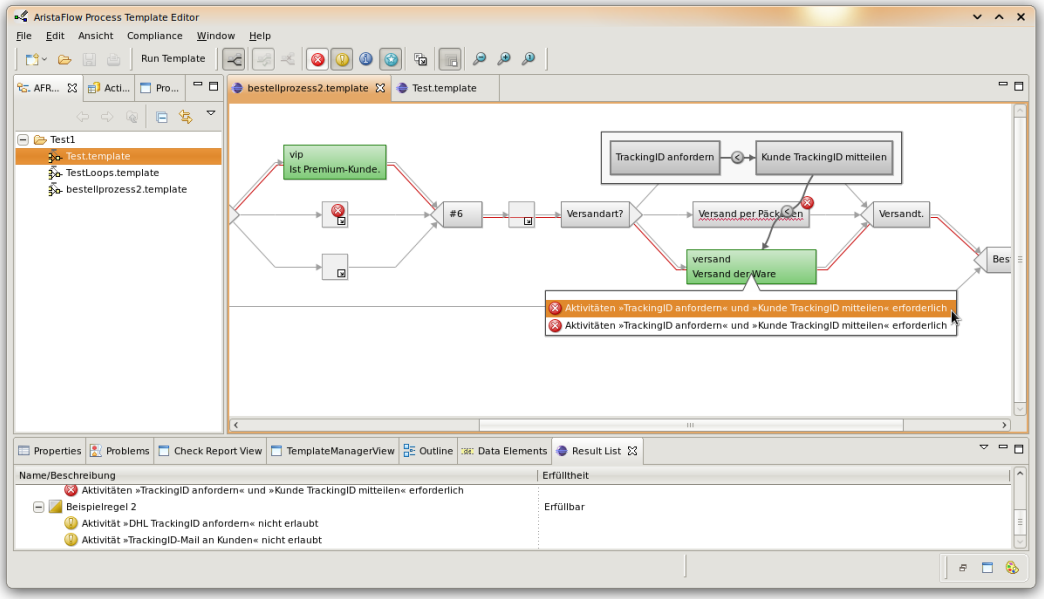


Abbildung 6.3: Der im Demonstrator umgesetzte Rule Based Process Editor

Templatedateien zugeordnet, so dass der Anwender bei Klick mit der rechten Maustaste auf eine Templatedatei wählen kann, ob er sie in der ursprünglichen AristaFlow-Editierkomponente oder mit dem *Rule Based Process Editor* öffnen möchte. Abb. 6.3 zeigt den im Demonstrator umgesetzten Editor.

6.7.1 Prozessdarstellung

Zur Darstellung des Prozessmodells sowie der verschiedenen Inhalte des Auswertungsergebnisses wird im Demonstrator ein Ebenen-Konzept verwendet. Verschiedene Ebenen stellen verschiedene Elemente unabhängig voneinander auf der Anzeigefläche dar. Für die Darstellung einer Ebene ist jeweils ein Objekt der Klasse `Painter` zuständig, das eine Breite und Höhe sowie einen Referenzpunkt, der innerhalb oder außerhalb der Ebene liegen kann, besitzt. Jeder `Painter` ist dafür zuständig, bei Aufruf seiner Methode `paint` den Inhalt der entsprechenden Ebene auf den Bildschirm zu zeichnen, so dass der Referenzpunkt auf einem vorgegebenen Bildschirmpunkt zu liegen kommt. Dieser sog. Ausrichtungspunkt ist für alle Ebenen gleich. Auf diese Weise werden die verschiedenen Ebenen zueinander ausgerichtet, außerdem lässt sich über ihre Größenangaben unter Berücksichtigung dieser Anordnung der für die Darstellung benötigte Platz berechnen. Ist dieser Platz

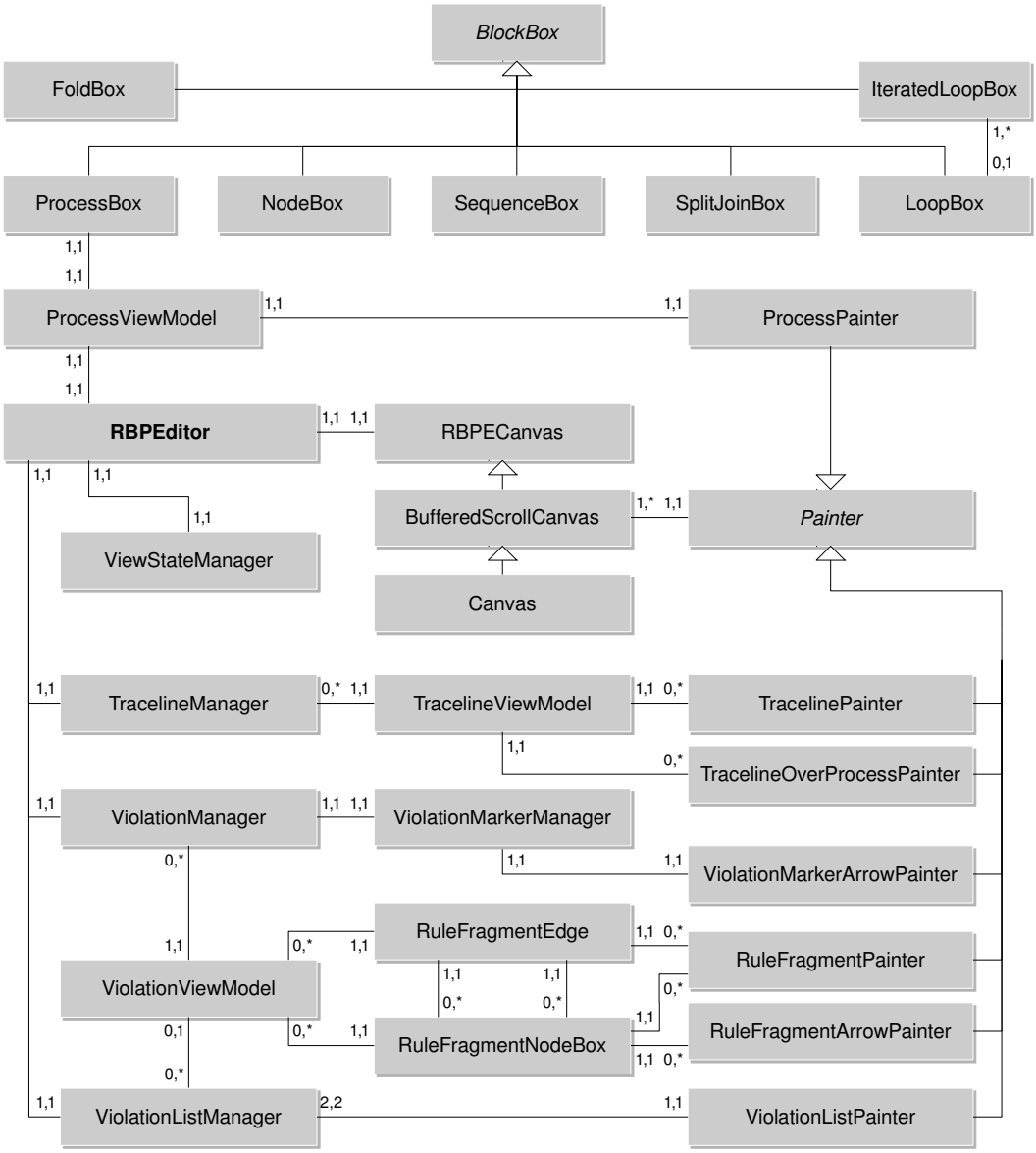


Abbildung 6.4: Klassenstruktur zur Prozess- und Ergebnisdarstellung

größer als die verfügbare Bildschirmfläche, werden Bildlaufleisten (Scrollbars) angezeigt, mit deren Hilfe sich der Ausrichtungspunkt und somit die gesamte Darstellung verschieben lässt.

Die Daten, die eine Darstellungsebene anzeigt, werden in verschiedenen spezifischen `ViewModel`-Objekten gehalten. Diese für jede Ebene spezifischen Objekte stellen eine an die Darstellung angepasste Sicht auf das jeweils zugrundeliegende Datenmodell dar. Einen Überblick über die für die Darstellung des Prozesses und der Auswertungsergebnisse verwendeten Klassen bietet Abb. 6.4.

Als Kontrollkomponenten innerhalb der Nutzerschnittstelle dienen `Manager`-Objekte. Diese erzeugen auf Grundlage des jeweiligen Prozessmodells die notwendigen `ViewModel`-Komponenten, blenden ggf. Ebenen ein oder aus und bieten Methoden an, mit denen sich die Darstellung, z. B. als Reaktion auf Nutzeraktionen, beeinflussen lässt.

Eine grundlegende `Manager`-Komponente ist der `ViewStateManager`. Dieser verwaltet den aktuellen Zustand der Nutzeroberfläche, welcher über Methodenaufrufe geändert werden kann. Über solche Zustandsänderungen werden `Manager` durch einen Event-Mechanismus informiert, so dass diese die Darstellung entsprechend anpassen können. Der `ViewStateManager` unterstützt folgende Zustände:

- `STATE_NODATA` – Es liegen keine Erfülltheitsinformationen vor.
- `STATE_OVERVIEW` – Es ist der Gesamterfülltheitsmodus aktiv, in dem Regelverletzungen durch Knotenmarkierungen gekennzeichnet sind und die Gesamterfülltheit durch Spurlinien angezeigt wird.
- `STATE_VIOLATION` – Es wurde eine Regelverletzung fest ausgewählt, diese wird nun im Regelverletzungsmodus durch Regelfragmente und Spurlinien dargestellt.
- `STATE_VIOLATIONLIST_HOVER` – Es wird temporär eine Regelverletzung dargestellt, da sich die Maus über dem entsprechenden Eintrag in einer Liste (z. B. der Liste der Alternativverletzungen) befindet. Die Darstellung entspricht `STATE_VIOLATION`, allerdings unterscheiden sich ggf. die dargestellten Regelverletzungen. Wird die Maus vom gewählten Eintrag wegbewegt, wechselt die Darstellung zurück zum vorherigen Zustand, `STATE_VIOLATION` oder `STATE_OVERVIEW`.

Die Grundlage der gesamten Prozessdarstellung bildet die Klasse `RBPECanvas`, die die Darstellungsfläche (Canvas) des *Rule Based Process Editors* bereitstellt. Im Konstruktor dieser Klasse werden alle `Painter` instantiiert, außerdem werden in dieser Klasse Nutzerak-

tionen in Aufrufe an den entsprechenden Managern umgewandelt. Diese Klasse erweitert die Klasse `BufferedScrollCanvas`, welche für das Zeichnen auf der Darstellungsfläche zuständig ist und hierzu nacheinander die Zeichenmethoden der verfügbaren `Painter` aufruft, um die Darstellung Ebene für Ebene aufzubauen. Hierzu verfügt `BufferedScrollCanvas` über eine Liste mit `Painter`-Objekten für alle Ebenen, deren Reihenfolge bestimmt, in welcher Reihenfolge sie gezeichnet werden. Der `BufferedScrollCanvas` verwaltet zudem die Scrollfunktionalität der Zeichenebene.

6.7.2 Prozessmodell

Die grundlegende Datenstruktur für die Darstellung des Prozessmodells ist in der Klasse `ProcessViewModel` enthalten. Wichtigster Bestandteil ist dabei eine Struktur von verschachtelten `BlockBox`es, die durch die in der ADEPT-Prozessbeschreibungssprache [27] verwendete Blockstrukturierung ermöglicht wird (vgl. Abschnitt 2.1.1). Ein `BlockBox`-Objekt repräsentiert dabei einen Block (oder eine Folge von Blöcken) im Prozessmodell. Verschiedene Subklassen stehen für verschiedene Typen von Blöcken:

- `NodeBox` für einen einzelnen Knoten im Prozess.
- `SequenceBox` für eine Sequenz aus zwei oder mehr anderen Boxen.
- `SplitJoinBox` für einen XOR- oder AND-Block; Enthält Boxen für Split- und Joinknoten sowie die einzelnen Zweige.
- `LoopBox` für eine Schleife; Enthält Boxen für Schleifenstart- und -endknoten sowie den Schleifenrumpf.
- `ProcessBox` für den äußersten Block eines Prozessmodells; Enthält Boxen für Prozessstart- und -endknoten sowie den gesamten Prozessinhalt.

Bei der Instantiierung eines Objekts vom Typ `ProcessViewModel` wird das übergebene AristaFlow-Prozesstemplate in diese Struktur umgewandelt. Zwei weitere Subklassen für eingeklappte Prozessteile und Schleifenexpansion werden später vorgestellt.

Jedes `BlockBox`-Objekt ist für die Ausrichtung der in ihm enthaltenen `BlockBox`es zuständig, die verschiedenen Subklassen ordnen die enthaltenen Blöcke somit je nach Art des Blocks horizontal bzw. vertikal an. Hierfür ist die Methode `calculate` zuständig, die rekursiv für jeden Block aufgerufen wird und dabei die Positionen der enthaltenen Blöcke sowie die Größe des Blocks festlegt.

Jede `BlockBox` besitzt eine Methode `draw`, bei deren Aufruf sie sich und (durch rekursiven Aufruf) ihre untergeordneten Boxen an eine angegebene absolute Position auf der Darstellungsfläche zeichnen muss. Eine `NodeBox` zeichnet dabei ihren Rahmen und Inhalt, andere Boxen rufen die `draw`-Methoden ihrer Kind-Boxen auf, um diese darzustellen, und zeichnen ebenfalls die Verbindungskanten sowie eventuelle Zusatzelemente.

Die Klasse `ProcessPainter` stellt die Ebene bereit, welche für die Darstellung des Prozessmodells zuständig ist. Fordert der Canvas den `ProcessPainter` zum Zeichnen der Ebene auf, ruft dieser die Methode `draw` in der Wurzel-`ProcessBox` auf, wodurch rekursiv das gesamte Prozessmodell gezeichnet wird.

Jede `NodeBox` enthält einen Verweis auf das `NodeEx`-Objekt der von ihr dargestellten Knotenausführung. Damit auch ein schnelles Mapping von der Modell- in die Darstellungsebene möglich ist, enthält das `ProcessViewModel` eine Umsetzungstabelle, die jedem `NodeEx`-Objekt, soweit es dargestellt wird, die für seine Darstellung verantwortliche `NodeBox` zuordnet. Beim Erzeugen und Zerstören einer `NodeBox` wird dieser Tabelle automatisch ein Eintrag hinzugefügt bzw. dieser wieder entfernt.

Da Synchronisationskanten im Prozessmodell die Blockstrukturierung durchbrechen, können diese nicht in der `BlockBox`-Struktur repräsentiert werden. Stattdessen enthält das `ProcessViewModel` eine Liste mit je einem `SyncEdgeView`-Objekt für jede dieser Kanten im Prozess. Jedes dieser Objekte enthält einen Verweis auf die Start- und End-`NodeBox` der Kante. Nach dem Zeichnen des Prozessmodells im `ProcessPainter` wird dann für jedes dieser Objekte eine Kante zwischen die entsprechenden `NodeBox`es gezeichnet.

Process Block Folding

Um das in Abschnitt 5.6.4 vorgestellte **Process Block Folding** – also das Zusammenfassen von Teilen eines Prozesses zu einem Knoten – zu ermöglichen, wird eine weitere Subklasse von `BlockBox` verwendet, `FoldBox`. Eine `FoldBox` dient als Container für eine weitere Box. Sie besitzt zwei mögliche Zustände, zwischen denen mittels Methodenaufrufen gewechselt werden kann, `open` und `closed`. Im Zustand `open` entspricht die Größe der Box der der enthaltenen Box, diese wird normal gezeichnet und von der `FoldBox` mit einem Rahmen und zwei Buttons zum Einklappen und Entfernen der Zusammenfassung versehen. Im Zustand `closed` hat die Box nur noch die Größe eines einzelnen Knotens, außer-

dem wird die enthaltene Box nicht gezeichnet. Stattdessen enthält sie als einzigen Inhalt einen Button zum Aufklappen.

Wenn der Anwender auf der Oberfläche mit dem Auswahlwerkzeug einen rechteckigen Bereich auswählt, werden die enthaltenen Sequenzen von Boxen beim Zeichnen der Oberfläche durch den `ProcessPainter` mit einer Hervorhebung und mit einem Einklapp-Button versehen. Bei Klick auf diesen Button wird eine neue `FoldBox` erstellt, die die ausgewählten Boxen enthält (handelt es sich um Teile einer Sequenz, wird hierfür eine neue `SequenceBox` erstellt). In der Eltern-`BlockBox` werden dann die entsprechenden Boxen durch die neue `FoldBox` ersetzt. Wird in einer `FoldBox` der Button zum Entfernen gewählt, wird die `FoldBox` wieder durch ihren Inhalt ersetzt (eine ggf. zuvor neu erzeugte `SequenceBox` wird dabei wieder entfernt).

Das `ProcessViewModel` enthält eine Umsetzungstabelle, die jedem Knotenvorkommen, dessen `NodeBox` innerhalb einer geschlossenen `FoldBox` liegt, die äußerste der sie enthaltenden geschlossenen `FoldBox`en zuweist. Beim Öffnen und Schließen von `FoldBox`-Objekten wird diese Tabelle automatisch angepasst. Dadurch kann z. B. beim Zeichnen einer Sync-Kante erkannt werden, ob eine oder beide Knoten, zwischen denen die Kante verläuft, in einer geschlossenen `FoldBox` liegen, und das Zeichnen entsprechend angepasst werden.

Schleifenexpansion

Für die **Schleifenexpansion** wird ebenfalls eine weitere Subklasse von `BlockBox` eingesetzt, `IteratedLoopBox`. Eine `IteratedLoopBox` repräsentiert eine expandierte Schleife und enthält eine Sequenz von Boxen, die jeweils eine Iteration darstellen. Damit nicht benötigte Iterationen leicht eingeklappt werden können, sind alle bis auf die letzte Iteration in jeweils einer `FoldBox` enthalten. Die einzelnen Iterationen selbst sind durch `Loop-Box`es realisiert, wobei bei allen außer der letzten keine Schleifenrücksprungkante gezeichnet wird.

Eine `LoopBox` besitzt eine Methode, mit der eine weitere Iteration hinzugefügt werden kann. Diese wird aufgerufen, wenn der Schleifenexpansionsbutton betätigt wird. Befindet sich die `LoopBox` nicht in einer `IteratedLoopBox`, wird diese zunächst erzeugt. Nun wird der enthaltenden `IteratedLoopBox` eine neue Iteration hinzugefügt, indem eine Kopie der `LoopBox`, aller enthaltenen Boxen sowie aller betroffenen `SyncEdgeView`s er-

zeugt wird. Dabei wird für jede enthaltene `NodeBox` bei der zugeordneten `NodeEx` in der jeweiligen Stufe der Iterationszähler erhöht.

Analog besitzt die Klasse `IteratedLoopBox` eine Methode zum Entfernen der letzten Iteration, bei deren Aufruf die letzte enthaltene `LoopBox` gelöscht wird. Ist nur noch eine Iteration vorhanden, wird die `IteratedLoopBox` automatisch entfernt und wieder durch die verbleibende `LoopBox` ersetzt.

AutoFolding

Die Automatisierung des *Process Block Folding*, das **AutoFolding**, wird durch eine eigene Komponente durchgeführt, die durch die Klasse `AutoFolder` realisiert ist. Eine Instanz dieser Klasse ist für das AutoFolding in einem Prozessmodel zuständig. Um einen AutoFolding-Durchgang durchzuführen, wird das Feld `importantNodeExs` der Instanz auf die Menge der für das AutoFolding als wichtig geltenden Knoten gesetzt und die Methode `autoFold` aufgerufen. Dieser wird auch die verfügbare Breite der Darstellungsfläche übergeben.

In der Methode `autoFold` wird zunächst mittels der Methode `calculateImportance` unter Verwendung der Menge der wichtigen Knotenausführungen für jede `BlockBox` ihre Wichtigkeit berechnet, welche im Feld `isImportant` abgelegt wird. Dieses Feld kann folgende Werte annehmen: `UNIMPORTANT`, wenn die Box keinerlei wichtige Knoten enthält, `CONTAINS_IMPORTANT`, wenn einige, aber nicht alle untergeordneten Boxen wichtig sind, und `IMPORTANT`, wenn alle untergeordneten Boxen wichtig sind oder die Box selbst wichtig ist. Eine `NodeBox` ist wichtig, wenn die der Box entsprechende Knotenausführung in der Menge der wichtigen Knoten enthalten ist. Andere `BlockBox`es berechnen diesen Wert anhand der Wichtigkeit ihrer Kind-Boxen.

Nun wird der `BlockBox`-Baum erneut rekursiv durchlaufen. Dabei wird für jede Box die Methode `autoFoldBlock` aufgerufen, welche je nach Wert des jeweiligen `isImportant`-Felds unterschiedliche Aktionen durchführt. Ist die Box als `IMPORTANT` markiert, werden alle in der Box enthaltenen `FoldBox`es (bzw. die Box selbst, wenn sie eine `FoldBox` ist), die vom AutoFolder angelegt wurden, geöffnet und entfernt. Boxen, die vom Anwender erstellt, aber vom AutoFolder geschlossen wurden, werden geöffnet, aber nicht entfernt. Ist die Box als `UNIMPORTANT` markiert, werden enthaltene Boxen (ggf. auch die Box selbst) zu `FoldBox`es zusammengefasst bzw. vorhandene `FoldBox`es eingeklappt, wenn der zur

Verfügung stehende Platz ansonsten nicht ausreicht. Boxen, die als `CONTAINS_IMPORTANT` markiert sind, werden aufgeklappt, wenn es sich um `FoldBox`es handelt. Enthaltene Boxen werden entsprechend ihrer Wichtigkeit behandelt.

Wird nach diesem Vorgang der zur Verfügung stehende Platz nicht vollständig ausgenutzt, erfolgt ein weiterer Durchlauf durch den Baum, in dem als unwichtig markierte Boxen, die bisher eingeklappt waren, geöffnet werden, soweit dafür Platz zur Verfügung steht.

Zur Steuerung des AutoFolding dient die Klasse `AutoFoldingManager`, von der jeder `RBPEditor` eine Instanz hält. Für jeden der im `ViewStateManager` möglichen Zustände verwaltet der `AutoFoldingManager` parallel eine Liste der in diesem Zustand wichtigen Knoten. Bei einem Wechsel des Zustands wird im `AutoFolder` die Liste der wichtigen Knoten entsprechend ausgetauscht. Außerdem bietet die Klasse Methoden, um das AutoFolding zu aktivieren bzw. deaktivieren.

Das AutoFolding wird ausgelöst, wenn sich der Zustand des `ViewStateManager`s, die Prozessstruktur, das Auswertungsergebnis oder der zur Verfügung stehende Platz ändert.

Animationen

Wie in Abschnitt 5.6.4 beschrieben wurde, sollen Änderungen an der Prozessdarstellung nach Möglichkeit animiert werden, um abrupte Sprünge in der Darstellung, die den Anwender verwirren könnten, zu vermeiden. Für die Koordination der Animationen sorgt die Klasse `AnimationManager`. Sie ist dafür zuständig, Animationen durchzuführen und zu vermeiden, dass in einem Objekt mehrere Animationen zeitgleich ablaufen, die sich gegenseitig stören können. Ebenfalls sorgt der `AnimationManager` dafür, dass bei der gleichzeitigen Animation mehrerer Objekte in jedem Animationsschritt das Prozessmodell nur einmal neu gezeichnet wird.

Ein `AnimationManager` verwaltet eine Liste aller derzeit animierten Objekte. Objekte, die animiert werden können, implementieren das Interface `Animatable`. Alle 50 Millisekunden ruft der `AnimationManager` in allen enthaltenen Objekten die Methode `animate` aus, in der das jeweilige Objekt einen Animationsschritt durchführen muss. Anschließend veranlasst der `AnimationManager` eine Neuberechnung der Prozessgeometrie durch Aufruf der Methode `calculate` am `ProcessPainter` sowie ein Neuzeichnen aller Ebenen. Objekte, deren Animationen abgeschlossen sind, werden aus der Liste

entfernt. Ist die Liste leer, stellt der `AnimationManager` seine Arbeit ein, bis wieder ein Element hinzugefügt wird.

Animationen werden in den Klassen `FoldBox` und `IteratedLoopBox` eingesetzt, die hierfür das Interface `Animatable` implementieren. Methoden, die Änderungen an der Prozessdarstellung verursachen, wie das Ein- und Ausklappen von `FoldBox`es oder das Hinzufügen bzw. Entfernen von Iterationen in einer `IteratedLoopBox`, besitzen einen Parameter, mit dem der `AnimationManager` übergeben werden kann. Ist dieser nicht `null`, wird die entsprechende Aktion nicht sofort ausgeführt, sondern das entsprechende Objekt wird beim `AnimationManager` registriert. Bei jedem Aufruf von `animate` wird ein Schritt in der Änderung der Darstellung durchgeführt, etwa die Größe der `FoldBox` dem aktuellen Animationsschritt entsprechend geändert.

6.7.3 Auswertungsergebnis

Die Darstellung des Auswertungsergebnisses im Prozess erfolgt größtenteils durch zusätzliche Ebenen, welche durch weitere `Painter`-Klassen sowie weitere `Manager` und `ViewModel`s realisiert werden.

Spurlinien

Eine Spurlinie wird durch die abstrakte Klasse `TracelineViewModel` realisiert. Diese bietet Methoden an, die für XOR- und Schleifenblöcke zurückgeben, welche Zweige bzw. Iterationen in der Spurlinie enthalten sind und in welchem Linienstil sie gezeichnet werden sollen. Ebenfalls wird der Anfangs-Linienstil für den Prozess angegeben. Als Linienstile stehen durchgezogene Linien, verschiedene gestrichelte Linien (bei Bedingungen oder unscharfen Ergebnissen) sowie der Wert `TRACELINE_SKIPPED` für nicht enthaltene Zweige/Iterationen zur Verfügung.

Die Implementierung der Methoden erfolgt in verschiedenen Unterklassen für unscharfe und nicht-unscharfe Spurlinien. Diese stellen jeweils nur eine dünne Zwischenschicht über den ihnen zugrunde liegenden `DefinedBranchRestriction`- und `FuzzyBranchRestriction`-Objekten dar. Dies ist möglich, da diese Datenstrukturen so konstruiert wurden, dass eine Zweigeinschränkung genau einer Spurlinie entspricht.

Die Darstellung der Spurlinien erfolgt durch eine spezielle Painter-Klasse, den TracelinePainter. Eine Instanz dieses Painters ist für das Zeichnen aller Spurlinien in einem Prozessmodell zuständig und liegt in der Liste der Painter vor dem ProcessPainter, so dass die Spurlinien-Ebene »unter« dem Prozessmodell liegt. Größe und Referenzpunkt entsprechen denen des ProcessPainters. Ein TracelinePainter hält eine Liste mit TracelineViewModels für alle darzustellenden Spurlinien. Wird die paint-Methode aufgerufen, wird nacheinander für jede Spurlinie ein rekursiver Durchlauf durch die BlockBox-Struktur durchgeführt und für jeden Block die Spurlinie im aktuellen Linienstil gezeichnet. Für die ProcessBox sowie die einzelnen Zweige der XOR-SplitJoinBoxes und die Iterationen von (Iterated)LoopBoxes wird zuvor beim TracelineViewModel der zu verwendende Linienstil angefragt. Hat dieser den Wert LINESTYLE_SKIPPED, so wird für den entsprechenden Zweig/die entsprechende Iteration keine Linie gezeichnet.

In geschlossenen FoldBoxes müssen die Spurlinien *über* die jeweilige Box im Prozessmodell gezeichnet werden. Hierfür zuständig ist ein zweiter Painter, der TracelineOverProcessPainter, welcher in der Ebenenliste des Canvas nach dem ProcessPainter auftritt und somit über dessen Ausgabe zeichnet.

Enthält die durch eine Spurlinie dargestellte Zweigeinschränkung als Bedingung eine einzelne Reihenfolge- und Zeitbeziehung oder eine Konjunktion von solchen Beziehungen, werden für diese, wenn die jeweilige Spurlinie ausgewählt ist, Verbindungskanten zwischen den entsprechenden Knoten gezeichnet. Da diese über dem Prozessmodell liegen sollen, wird dazu ebenfalls der TracelineOverProcessPainter verwendet, der hierfür vom TracelineViewModel eine Liste der RestrictionCondition-Objekte für die einzelnen darzustellenden Beziehungen erhält.

Für die Verwaltung der darzustellenden Spurlinien ist die Klasse TracelineManager zuständig. Sie kann parallel mehrere Sätze von Spurlinien verwalten: Die Spurlinien für die erfüllenden Spuren der Gesamterfülltheit, die verletzenden Spuren der Gesamterfülltheit (beide im Zustand STATE_OVERVIEW), die Spuren der derzeit ausgewählten Regelverletzung (im Zustand STATE_VIOLATION) und die Spuren der derzeit dargestellten temporären Regelverletzung (im Zustand STATE_VIOLATIONLIST_HOVER). Der TracelineManager setzt die darzustellenden Spurlinien des TracelinePainters dabei stets auf die dem jeweiligen Zustand – sowie im Zustand STATE_OVERVIEW dem aktuellen Darstellungsmodus (erfüllend oder verletzend) – entsprechenden TracelineViewModels.

Regelverletzungen

Eine Regelverletzung wird für die Darstellung durch ein Objekt der Klasse `ViolationViewModel` repräsentiert. Es wird aus einem `RuleViolation`-Objekt erzeugt. Dabei wird der verletzte Regelteil der Regelverletzung in eine Liste aus `RuleFragmentNodeBox`- und `RuleFragmentEdge`-Objekten umgewandelt, welche den Knoten und Kanten in der Regelgraph-Repräsentierung des Regelteils entsprechen. Bei Regelknoten, die in der `RuleViolation` durch das Bedingungs-Matching oder die Bindung im Folgeteil an Prozessknotenausführungen gebunden werden, wird im jeweiligen `RuleFragmentNodeBox`-Objekt die entsprechende `NodeBox` vermerkt. Wird der Regelknoten an mehrere Knotenausführungen gebunden, werden für ihn mehrere `RuleFragmentNodeBox`-Objekte erzeugt, auch die angeschlossenen Kanten treten dann jeweils mehrfach auf.

Ein `ViolationViewModel`-Objekt enthält auch eine Liste mit `ViolationViewModel`s der Alternativverletzungen der jeweiligen Regelverletzung. Diese wird aus der Liste der Alternativverletzungen im `RuleViolation`-Objekt erzeugt. Ebenfalls enthalten ist eine Methode, welche eine textuelle Beschreibung der Regelverletzung nach Abschnitt 5.6.3 zurückgibt.

Für die Erzeugung und Verwaltung von `ViolationViewModel`-Objekten ist die Klasse `ViolationManager` zuständig, von der pro `RBPEditor` eine Instanz existiert. Dieser Manager erzeugt aus der Liste der Regelverletzungen in der Ergebnisstruktur, die von der Auswertungs-Engine erzeugt wird, eine Liste mit allen entsprechenden `ViolationViewModel`-Objekten. Außerdem beinhaltet er Verweise auf die derzeit im Prozess als Regelfragmente dargestellten Regelverletzungen. Hierbei können zwei Regelverletzungen parallel verwaltet werden, eine für den Zustand `STATE_VIOLATION` und eine für den Zustand `STATE_VIOLATIONLIST_HOVER`. Er besitzt ebenfalls Methoden, um diese Regelfragmente ein- bzw. auszublenden. Die Darstellung der Fragmente wird später beschrieben.

Für die Verwaltung der Regelverletzungsmarkierungen an Prozessknoten im Gesamterfülltheitsmodus wird eine Instanz der Klasse `ViolationMarkerManager` eingesetzt. Diese erzeugt aus allen `RuleFragmentNodeBox`-Objekten eine Abbildung, die den `NodeBox`-Objekten die gebundenen `ViolationViewModel`s zuordnet. Anschließend werden den `NodeBox`-Objekten entsprechend der Regelklassen der verletzten Regeln Verletzungsmarkierungen zugewiesen. Die Klasse `NodeBox` ist selbst für das Zeichnen der jeweiligen Markierung in Form eines Symbols und der gewellten Unterstreichung des Knotennamens zuständig. Geschlossenen `FoldBox`es wird ebenfalls jeweils eine Markierung zugewiesen,

wenn sie markierte NodeBoxes enthalten, auch sie sind für die Darstellung der Markierung selbst zuständig.

Für die Darstellung der Pfeile, die in Richtung markierter Knoten außerhalb des Darstellungsbereichs zeigen, existiert eine eigene Painter-Klasse namens ViolationMarkerArrowPainter.

Regelverletzungslisten

Für die Anzeige der Alternativverletzungen der derzeit dargestellten Regelverletzung sowie für die Auswahl einer Regelverletzung nach Klick auf eine Verletzungsmarkierung werden in die Prozessdarstellung eingebettete Listen benötigt. Hierfür stehen die Klassen ViolationListManager und ViolationListPainter zur Verfügung. Ein ViolationListManager steuert beide Arten von Listen für einen RBPEditor. Für die Darstellung ist pro Liste ein eigener ViolationListPainter notwendig.

Die Klasse ViolationListManager bietet Methoden, um die Liste der Regelverletzungen an einer Verletzungsmarkierung anzuzeigen bzw. auszublenden. Die Informationen über die anzuzeigenden Regelverletzungen stammen dabei aus dem ViolationMarkerManager, die textuelle Beschreibung aus dem ViolationViewModel. Die Liste der Alternativverletzungen wird automatisch angezeigt bzw. ausgeblendet, wenn eine Regelverletzung im Prozess angezeigt wird. Die Daten für ihren Inhalt stammen vom ViolationViewModel. Der ViolationListManager übergibt die anzuzeigenden Elemente an den jeweiligen ViolationListPainter.

Regelgraph-Fragmente

Die Darstellung des verletzten Regelteils einer Regelverletzung als Regelgraph-Fragment übernimmt eine weitere Ebene auf dem Canvas, der RuleFragmentPainter. Grundlage bilden hierbei die RuleFragmentNodeBoxes und RuleFragmentEdges aus dem zugehörigen ViolationViewModel-Objekt.

Bevor das Regelfragment gezeichnet werden kann, müssen die RuleFragmentNodeBoxes positioniert werden. Ziel ist es, eine Positionierung zu erreichen, die möglichst der erwünschten Anordnung der Knoten im Prozessmodell entspricht und bei der möglichst selten Knoten durch Kanten überdeckt werden. Nur bei Knoten, die an Prozessknoten ge-

bunden sind, steht die Positionierung durch die Positionen der entsprechenden `NodeBox`es bereits fest.

Um die ungebundenen Regelknoten zu positionieren, wird der Graph, der sich aus den Knoten und Kanten des `ViolationViewModel` ergibt, zunächst in seine maximalen zusammenhängenden Komponenten zerlegt, wobei Knoten, die an Prozessknoten gebunden sind, nicht berücksichtigt werden. Anschließend werden die Knoten innerhalb jeder Komponente unter Verwendung eines auf dem bekannten Verfahren zur topologischen Sortierung von Kahn [17] basierenden Algorithmus so angeordnet, dass Reihenfolge- und Zeitkanten immer von links nach rechts zeigen (dies ist möglich, da gültige Regeln in diesen Kanten keine Zyklen besitzen), wobei nicht in Reihenfolge- und Zeitbeziehung zueinander stehende Knoten auch parallel dargestellt werden können. Die Zusammenhangskomponenten selbst werden horizontal nebeneinander gesetzt, wobei sich die Reihenfolge nach der Anordnung der mit den Komponenten verbundenen Prozessknoten richtet. Aus der Breite und Höhe dieser Folge von Zusammenhangskomponenten ergibt sich auch die Größe der durch den `RuleFragmentPainter` repräsentierten Ebene. Der Referenzpunkt der Ebene wird so gesetzt, dass sich ihr Inhalt möglichst nahe an den mit den Zusammenhangskomponenten verbundenen Prozessknoten befindet, ohne sie zu überdecken.

Beim Zeichnen des Regelfragments wird der Bereich, in dem sich die ungebundenen Knoten befinden, mit einem Rahmen umgeben. Anschließend zeichnet der `RuleFragmentPainter` die gebundenen Knoten an die Positionen der mit ihnen verbundenen `NodeBox`es. Die nicht gebundenen Regelknoten werden an den zuvor berechneten Positionen dargestellt. Anschließend werden die Kanten zwischen den Knoten gezeichnet. Werden zwei Knoten durch mehrere Kanten verbunden, werden diese gewölbt dargestellt, so dass die an den Kanten angebrachten Symbole, die die Kantentypen angeben, nicht überlappen. Kanten, die weit voneinander entfernte Knoten verbinden, werden als geschwungene Linien gezeichnet, um die Wahrscheinlichkeit, dass sie andere Knoten oder Prozesskanten überdecken, zu verringern.

Analog zum `ViolationMarkerArrowPainter` existiert auch für Regelfragmente eine zusätzliche `Painter`-Klasse, die Pfeile in Richtung außerhalb des sichtbaren Bereichs liegender `RuleFragmentNodeBox`es zeichnet. Diese Aufgabe übernimmt die Klasse `RuleFragmentArrowPainter`.

6.7.4 Übersichtsliste

Für die Listenansicht, die die Gesamterfülltheit der überprüften Regeln sowie alle Regelverletzungen übersichtlich darstellt, wird ein eigenes Unterfenster verwendet, das in der Klasse `ResultListView` umgesetzt wurde. Es erhält nach der Auswertung der Integritätsregeln in einem Prozessmodell das `ResultStructure`-Objekt und stellt es mit Hilfe eines speziellen `ContentProvider`s in einem SWT-`Tree`-Objekt dar. Je nach Auswahl des Anwenders werden die einzelnen Regelverletzungen dabei vom `ContentProvider` nach Regeln und/oder Regeldateien (die in Form von `RuleGroup`-Objekten umgesetzt sind) gruppiert. Wenn ausgewählt wird, dass erfüllte Regeln nicht angezeigt werden sollen, werden diese mittels einer speziellen `ViewerFilter`-Klasse ausgefiltert.

Bei Auswahl von einer oder mehreren Regeln bzw. Regelgruppen in der Liste werden die für die Prozessansicht auszuwertenden Regeln auf diese Auswahl eingeschränkt und eine Neuauswertung durchgeführt, deren Ergebnis im Prozessmodell dargestellt wird (vgl. Abschnitt 6.8.1). Wird eine einzelne Regelverletzung ausgewählt, wird diese durch den `RuleFragmentPainter` im Prozessmodell dargestellt.

6.8 Auswertungsalgorithmus

Damit verschiedene mögliche Auswertungsalgorithmen an das System angebunden werden können, muss jeder Algorithmus eine Klasse bereitstellen, welche das Interface `ValidationEngine` implementiert. Dieses Interface enthält eine einzige Methode, `validate`. Diese Methode besitzt folgende Parameter:

- `validationSubject` – Das Auswertungssubjekt, repräsentiert durch ein Objekt der Klasse `DefinedBranchRestriction`. Jedes gültige Auswertungssubjekt lässt sich durch ein Objekt dieser Klasse ausdrücken. Für diesen Demonstrator handelt es sich jedoch immer um ein Objekt, bei dem kein Zweig abgewählt, alle Iterationen ausgewählt sowie keine Bedingungen vorhanden sind, das also das vollständige Prozessmodell repräsentiert.
- `ruleGroups` – Eine Liste von `RuleGroup`-Objekten, welche jeweils die zu überprüfenden Regeln aus einer Regeldatei enthalten.
- `displayedIterations` – Ein Objekt vom Typ `HashMap<NodeEx, Integer>`, welches Knotenausführungen auf natürliche Zahlen abbildet. Bei den Knotenausführun-

gen handelt es sich dabei um Ausführungen des Startknotens der jeweils ersten Iteration einer Schleife. Eine Abbildung $(n \mapsto i)$ bedeutet dabei, dass bei der Schleifenausführung, welche mit der Knotenausführung n beginnt, im derzeit an der Benutzeroberfläche dargestellten Prozessmodell i Iterationen sichtbar sind, dieses Vorkommen der Schleife in der Prozessansicht also $(i-1)$ mal expandiert wurde. Diese Informationen können vom Algorithmus genutzt werden, um nur für Iterationen, die auch tatsächlich dargestellt werden, ausführliche Ergebnisse anzugeben.

Die implementierende Klasse muss als Ergebnis dieser Methode ein Objekt vom Typ `ResultStructure` zurückgeben, das die Erfülltheit der angegebenen Regeln im Auswertungssubjekt angibt. Dabei muss für jede Regel aus jeder Datei ein Eintrag im Feld `ruleGeneralCompliance` existieren, der den Wert der Gesamterfülltheit für diese Regel angibt, und für jede Datei ein Eintrag im Feld `ruleGroupGeneralCompliance`, der den Wert für die Menge der in der Datei enthaltenen Regeln angibt. Das Feld `generalCompliance` gibt die Gesamterfülltheit für die Menge aller Regeln aus allen Dateien an, das Feld `ruleViolations` enthält alle bei der Auswertung ermittelten Regelverletzungen.

6.8.1 Anbindung

Den Aufruf des Auswertungsalgorithmus übernimmt die Klasse `ValidationManager`, von der jeder `RBPEditor` eine Instanz besitzt. Diese Klasse hält Verweise auf das `ValidationEngine`-Objekt der verwendeten Auswertungs-Engine, alle zu überprüfenden Integritätsregeln in `RuleGroup`-Objekten (eine `RuleGroup` für jede Regeldatei), sowie ggf. eine Teilmenge der Regeln, auf die die Darstellung im Prozessmodell eingeschränkt sein soll (wenn in der Übersichtsliste Regeln ausgewählt wurden). Die Klasse bietet eine Methode an, die von anderen Komponenten aufgerufen wird, wenn eine Neuauswertung erfolgen muss, z. B. wenn das Prozessmodell geändert wurde, sich durch Schleifenexpansion neue `displayedIteration`s ergeben oder eine neue Auswahl der zu überprüfenden Regeln erfolgte. Daraufhin ruft der `ValidationManager` die `validate`-Methode der Auswertungs-Engine auf und übergibt die Inhalte der erhaltenen Ergebnisstruktur an die verschiedenen `Manager`-Objekte für die einzelnen Darstellungsebenen sowie die Übersichtsliste. Anschließend wird die Darstellung aktualisiert. Wenn in der Übersichtsliste die für die Prozessdarstellung auszuwertenden Regeln eingeschränkt wurden, wird die `validate`-Methode zweimal aufgerufen: einmal mit allen Regeln für die Darstellung in der

Liste und einmal mit den Regeln aus der Einschränkung für die Darstellung im Prozessmodell.

6.8.2 Beispielalgorithmus

Für den Demonstrator wurde als Beispielimplementierung eine einfache Auswertungs-Engine umgesetzt. Diese nutzt intern die Klasse `ExtendedDefinedBranchRestriction`, eine Unterklasse von `DefinedBranchRestriction`, welche diese um verschiedene Funktionen erweitert. Hierzu verwendet sie eine zweigbasierte Sicht auf das Prozessmodell. Auf diese Weise kann sie Methoden anbieten, die es ermöglichen, aus der Zweigeinschränkung alle Ausführungsspuren, die eine bestimmte Knotenausführung enthalten bzw. nicht enthalten, zu entfernen. Dazu werden XOR-Zweige bzw. Schleifeniterationen, welche die Knotenausführung (rekursiv) enthalten bzw. für ihre Nicht-Ausführung sorgen, im `DefinedBranchRestriction`-Objekt abgewählt, indem beispielsweise die entsprechenden Zweige dem Feld `deselectedBranchCodes` hinzugefügt werden. Des Weiteren bietet die Klasse auf dieser Grundlage Methoden an, um zwei Zweigeinschränkungen zu *schneiden*, also eine Zweigeinschränkung zu erhalten, die die Schnittmenge der von beiden Einschränkungen repräsentierten Spurmengen darstellt, sowie Zweigeinschränkungen zu *vereinigen*, also eine ihrer Vereinigungsmenge entsprechende Zweigeinschränkung zu erhalten, wenn sie kompatibel sind.[1]

Der eigentliche Auswertungsalgorithmus, der diese Funktionen verwendet, ist in der Klasse `SimpleValidationEngine` umgesetzt. Er nutzt das Entfernen von Zweigen und die Mengenoperationen, um Spurmengen zu ermitteln, in denen die Integritätsregel erfüllt bzw. verletzt ist, wobei bei Schleifen nur die derzeit dargestellten Iterationen überprüft werden. Der Algorithmus wird im Folgenden kurz skizziert, eine detaillierte Beschreibung würde jedoch den Rahmen dieser Arbeit sprengen.

Zunächst werden für alle positiven Variablen des Bedingungsteils einer zu überprüfenden Regel alle derzeit dargestellten Knotenausführungen ermittelt, die dem Typ der jeweiligen Variablen entsprechen. Diese werden zu allen möglichen Bedingungs-Matchings kombiniert, es werden also alle möglichen Bindungen der Bedingungsvariablen an diese

[1] Die Vereinigung ist nicht immer möglich: wenn z. B. in zwei aufeinanderfolgenden XOR-Blöcken in beiden Einschränkungen verschiedene Zweige abgewählt wurden, kann es sein, dass keine einzelne Zweigeinschränkung existiert, die die Vereinigungsmenge der von beiden Einschränkungen dargestellten Spurmengen repräsentiert. In so einem Fall gibt der Algorithmus zwei Zweigeinschränkungen zurück. Da mehrere Einschränkungen in einem `TraceSet` Oder-Semantik besitzen, entspricht dieses Ergebnis ebenfalls einer Vereinigung, erhöht jedoch die Ergebniskomplexität, weshalb der Algorithmus versucht, dies zu vermeiden.

Knotenausführungen ermittelt (enthält der Bedingungsteil keine positiven Variablen, wird ein leeres Bedingungs-Matching erzeugt). Für jedes dieser Bedingungs-Matchings wird die Regel nun weiter ausgewertet. Dazu wird zum einen die Menge der Spuren bestimmt, in denen der Bedingungsteil erfüllt ist, und zum anderen die Menge der Spuren, die den Bedingungsteil verletzen. Wie diese Mengen bestimmt werden, wird später beschrieben. Anschließend wird für das aktuelle Bedingungs-Matching der Folgeteil ausgewertet. Dazu werden für jede Existenzgruppe in jedem Folgeglied wiederum alle möglichen Belegungen der positiven Variablen der Existenzgruppe mit Knotenausführungen aus dem Prozess ermittelt und für jede Belegung die Menge der Spuren bestimmt, in denen die Existenzgruppe erfüllt ist sowie die Menge der Spuren, in denen sie verletzt ist. Ist letztgenannte Menge nicht leer, wird eine Regelverletzung mit der Menge der verletzenden Spuren und der derzeitigen Bindung für die jeweilige Existenzgruppe erzeugt. Anschließend werden die Ergebnismengen der verschiedenen Belegungen, der Existenzgruppen sowie der Folgeglieder vereinigt bzw. geschnitten nach dem Schema, das in Abb. 6.5 dargestellt ist. Daraufhin wird die Menge der Ausführungsspuren, in der der Folgeteil verletzt ist, mit der Menge der Spuren, in denen der Bedingungsteil erfüllt ist, geschnitten – dies ergibt die Menge der Spuren, in denen die Regel für das Bedingungs-Matching verletzt ist. Die Spuren, in denen der Bedingungsteil verletzt ist, werden mit den Spuren, in denen der Folgeteil erfüllt ist, vereinigt und ergeben die Menge der Spuren, in denen die Regel erfüllt ist. Anschließend werden die Ergebnismengen der verschiedenen Bedingungs-Matchings wiederum vereinigt bzw. geschnitten, wie in der Abbildung dargestellt ist. Werden mehrere Regeln ausgewertet, werden die Ergebnisse nochmals vereinigt bzw. geschnitten.

Die Spurmengen der erzeugten Regelverletzungen werden zum Schluss noch mit der Menge der verletzenden Spuren der jeweiligen Regel geschnitten; Ist die Schnittmenge leer, stellt die jeweilige Verletzung keine tatsächliche Regelverletzung dar (da beispielsweise ein anderes Folgeglied erfüllt ist und somit die Verletzung der Existenzgruppe aufhebt) und wird entfernt.

Zur Auswertung des Bedingungsteils bzw. einer Existenzgruppe für eine bestimmte Belegung ihrer Variablen werden zunächst alle (den Bedingungsteil/die Existenzgruppe potentiell erfüllenden) Spuren berechnet, in denen alle den Variablen zugeordneten Knotenausführungen vorkommen. Dazu werden die Methoden der Klasse `ExtendedDefinedBranchRestriction` verwendet. Ebenso wird auf diese Weise die Menge der Spuren berechnet, in der nicht alle Knotenausführungen vorkommen – in diesen Spuren ist der Bedingungsteil/die Existenzgruppe sicher verletzt. Anschließend wird überprüft, ob die

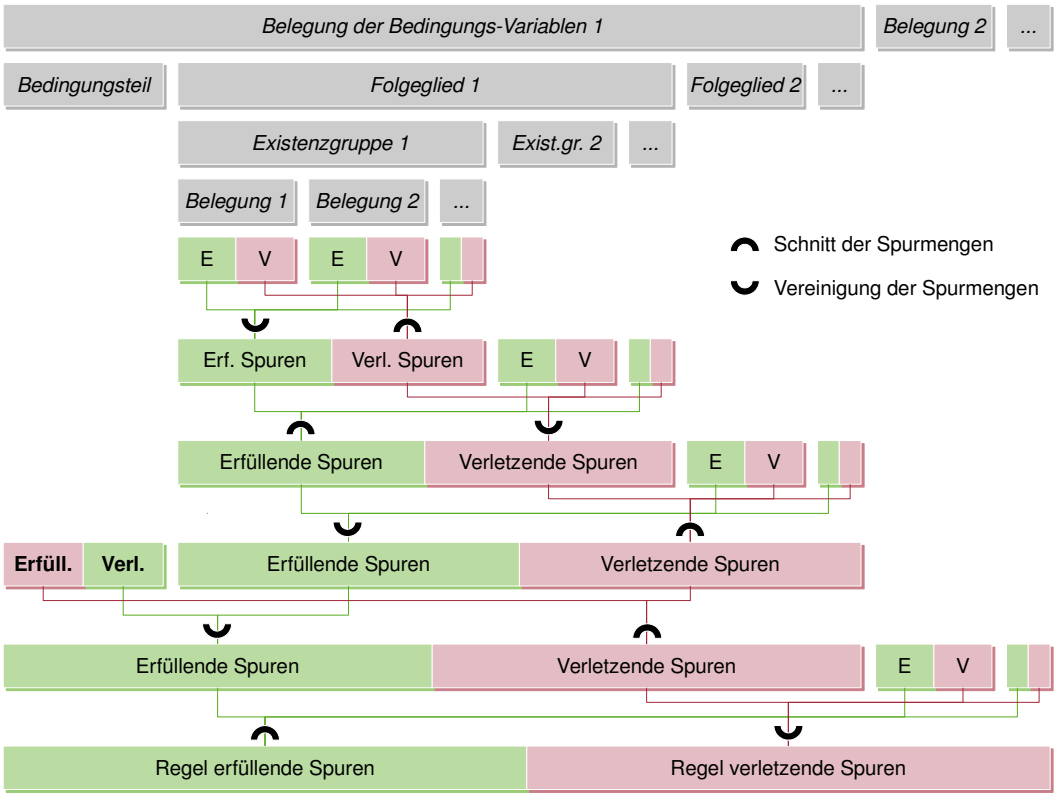


Abbildung 6.5: Funktionsweise des Beispielalgorithmus

Prädikate des Bedingungsteils zwischen den Knotenausführungen erfüllt sind – ist dies nicht der Fall, ist der Bedingungsteil in allen Spuren verletzt, ansonsten werden zu den erfüllenden Spuren ggf. Bedingungen hinzugefügt sowie diese mit invertierten Bedingungen zu den verletzenden Spuren hinzugefügt. Ebenfalls werden die Negativelemente überprüft und die erfüllenden bzw. verletzenden Spuren entsprechend angepasst. Als Ergebnis dieser Auswertung ergibt sich je eine Menge von `ExtendedDefinedBranchRestriction`-Objekten für die Spuren, in denen der Bedingungsteil bzw. die Existenzgruppe für das Matching erfüllt ist und jene, in denen er/sie verletzt ist.

Das Gesamtergebnis der Auswertung ergibt sich aus den ermittelten Spur- und Verletzungsmengen. Der *Wert der Gesamterfülltheit* berechnet sich folgendermaßen: Ist die Menge der erfüllenden Spuren leer, ist der Wert *verletzt*, außer, im Prozessmodell ist mindestens eine Schleife enthalten – in diesem Fall ist der Wert *evtl. verletzt*, da es möglich ist, dass bei der Ausführung nicht dargestellter (und somit nicht überprüfter) Iterationen die Re-

geln in einigen Ausführungsspuren erfüllt sind. Ist die Menge der verletzenden Spuren leer, ist der Wert *erfüllt* – bei Schleifen analog *evtl. erfüllt*. Ansonsten gilt: Enthält die Menge der erfüllenden Spuren ein `ExtendedDefinedBranchRestriction`-Objekt, bei dem keine Zweige abgewählt und alle Iterationen ausgewählt wurden, das jedoch eine Bedingung besitzt, ist das Ergebnis *evtl. erfüllt*, da die Bedingung möglicherweise immer erfüllt ist, was der Algorithmus nicht ermitteln kann.[2] Enthält die Menge der verletzenden Spuren alle Zweige des Prozessmodells mit einer Bedingung, ist das Ergebnis analog hierzu *evtl. verletzt*. Enthalten beide Mengen alle Zweige des Prozessmodells und eine Bedingung, könnten die Regeln also sowohl *erfüllt* als auch *verletzt* sein und das Ergebnis ist somit *unbestimmt*. In jedem anderen Fall hat die Gesamterfülltheit den Wert *erfüllbar*.

Der Algorithmus verwendet keine `FuzzyBranchRestriction`-Objekte, da mittels der `ExtendedDefinedBranchRestriction`s die Zweigentscheidungen für die dargestellten Iterationen stets exakt bestimmt werden können. Lediglich die enthaltenen `RestrictionCondition`-Objekte können nicht ausgewertet werden (ggf. ergeben sich bei den Mengenoperationen auch `BlackBoxRestrictionConditions`) und führen zu den ggf. unscharfen Angaben des Werts der Gesamterfülltheit, genauso wie die Tatsache, dass nur sichtbare Iterationen ausgewertet werden.

Ein Vorteil dieses Algorithmus ist, dass er relativ leicht verständlich ist und ohne externe Komponenten auskommt sowie umfangreiche Ergebnisdaten liefert, die sich gut eignen, die Möglichkeiten der in dieser Arbeit beschriebenen Nutzerschnittstelle zu demonstrieren. Allerdings stellt er eine relativ naive Herangehensweise dar, die einen häufigen Durchlauf über das Prozessmodell erfordert und deshalb bei großen Prozessen schnell ineffizient werden kann. Wenn einzelne Aktivitätentypen, die in der Regel im Bedingungsteil auftreten, mehrfach im Prozess vorkommen, muss für sehr viele Kombinationsmöglichkeiten der Zuordnung von Knotenausführungen zu Bedingungsvariablen jeweils der komplette Folgeteil ausgewertet werden, was ebenfalls ineffizient ist. Außerdem ist das Ergebnis in einigen Fällen unscharf, da bei Schleifen nur die dargestellten Iterationen ausgewertet werden sowie Zeitbeziehungen und Verhältnisse zwischen Reihenfolgebeziehungen nicht aufgelöst werden, wie oben beschrieben wurde.

[2] Eine erzeugte Spurmenge könnte beispielsweise die Bedingung $(a < b) \vee \neg(a < b)$ beinhalten, die immer erfüllt ist, was der Algorithmus jedoch nicht erkennt – somit kann er in einem solchen Fall nicht aussagen, ob die Regel nur *erfüllbar* oder *erfüllt* ist und muss ein unscharfes Ergebnis zurückgeben. Im angegebenen Beispiel wäre dies leicht zu erkennen, allerdings kann es auch zu wesentlich komplexeren Fällen kommen.

7 Zusammenfassung und Ausblick

In dieser Diplomarbeit wurden Konzepte vorgestellt und prototypisch umgesetzt, die es erlauben, die Einhaltung von Integritätsregeln in Prozessmodellen und -instanzen anschaulich darzustellen und es Anwendern von BPM-Systemen ermöglichen, Verletzungen der Regeln zu analysieren, Möglichkeiten für ihre Behebung zu erkennen sowie auf Grundlage der Auswertungsergebnisse Entscheidungen zu treffen und Anpassungen durchzuführen.

Nachdem in Kapitel 2 die zugrunde liegenden Konzepte besprochen wurden, erfolgte in Kapitel 3 eine Analyse der aus Anwendersicht bestehenden Anforderungen an die Auswertung von Integritätsregeln und insbesondere die Darstellung und Vermittlung der Auswertungsergebnisse. Dabei wurden die Anwender zunächst in verschiedene Nutzergruppen eingeteilt. Daraufhin wurden Anwendungsszenarien für verschiedene Nutzungsmöglichkeiten von Integritätsregeln aufgestellt, woraus dann die Anforderungen abgeleitet wurden. Dabei wurden auch zusätzliche Anwendungsgebiete für Integritätsregeln vorgestellt, die weit über die einfache Bestimmung, ob ein Prozess vorgeschriebene Regeln einhält oder nicht, hinausgehen: die Modellierung von Prozessen auf Grundlage von durch Regeln repräsentierten bedingten Prozessfragmenten, das Testen von Prozessen durch Assertion-Regeln oder die Ableitung von spezifischen Regeln aus einem Prozessmodell zur Erhaltung wichtiger Eigenschaften bei der Optimierung der Prozesse im Verlauf ihres Lebenszyklus. Die Anforderungsanalyse zeigte, wie die verschiedenen Anwendungen eine anwenderorientierte Ergebnisdarstellung notwendig machen.

Aufbauend auf den ermittelten Anforderungen wurde in Kapitel 4 eine Ergebnisstruktur beschrieben, die in der Lage ist, die Ergebnisse der Regelauswertung abzubilden, dabei Flexibilität für die verschiedenartigen Ergebnisse unterschiedlicher Algorithmen bietet und zugleich die Basis für eine anwenderorientierte Darstellung bildet. Dabei wurde eine zweigeteilte Struktur verwendet, die aus einer Gesamterfülltheitsangabe und einer Menge von Regelverletzungen besteht und trotz ihres einfachen Aufbaus detaillierte Ergebnisangaben aufnehmen kann. Um auch weniger aussagekräftige Ergebnisse abbilden zu können, wurde zudem die Unterstützung von Ergebnisunschärfe eingeführt.

Anschließend wurde in Kapitel 5 das Konzept einer Nutzerschnittstelle vorgestellt, die den zuvor untersuchten Anforderungen entsprechend die Daten aus der Ergebnisstruktur aufbereitet und dem Anwender präsentiert, um ihn zielführend bei der Erfüllung seiner Aufgaben zu unterstützen. Neben verschiedenen flexibel anpassbaren Listenansichten, die es – basierend auf den Ergebnissen der Anforderungsanalyse – verschiedenen Anwendern ermöglichen, in unterschiedlichen Situationen für ihre jeweiligen Aufgaben eine optimale Übersicht über die Regelerfülltheit zu erhalten, wurde ausführlich eine in die Prozessdarstellung integrierte, interaktiv analysierbare Visualisierung der Auswertungsergebnisse vorgestellt. Diese hat das Ziel, die in der zuvor beschriebenen Ergebnisstruktur vorliegenden Compliance-Informationen, die je nach Algorithmus in Umfang und Aussagekraft variieren können, möglichst detailliert und an die Aufgaben der Anwender angepasst darzustellen. Zur Erhöhung der Übersichtlichkeit steht dabei Zoomfunktionalität und (auf Wunsch automatisches) *Process Block Folding* zur Verfügung, durch Schleifenexpansion kann das Auswertungssubjekt angepasst werden. Die Auswertungsergebnisse werden in Form von Spurlinien, Knotenmarkierungen und vom Anwender einblendbaren Regelgraph-Fragmenten dargestellt.

Schließlich wurde die Umsetzbarkeit dieses Konzepts durch eine prototypische Implementierung demonstriert, die in Kapitel 6 beschrieben wurde. Dabei wurde der Fokus auf die Umsetzung der zuvor beschriebenen interaktiven Konzepte zur Darstellung der Auswertungsergebnisse in der Prozessansicht gelegt.

7.1 Erweiterungsmöglichkeiten

Die Ergebnisse dieser Arbeit bieten zahlreiche Ansatzpunkte für weitergehende Untersuchungen und Ergänzungen. Eine Möglichkeit stellt dabei die Anbindung anderer Auswertungsalgorithmen an die Ergebnisstruktur dar. Bei Anbindung mehrerer Algorithmen könnten auch vergleichende Untersuchungen angestellt werden, um das Verhältnis zwischen Auswertungsaufwand und Aussagekraft der Ergebnisse verschiedener Algorithmen zu untersuchen.

Weiter besteht die Möglichkeit, den entwickelten Demonstrator zu einer voll funktionsfähigen Editierkomponente zu erweitern, um ihn als vollständigen Ersatz für die Komponente zur Prozesseditierung des *AristaFlow Process Template Editor* einzusetzen, oder alternativ die bestehende Komponente um die Ergebnisse dieser Arbeit zu ergänzen. Mit einem

solchen vollständigen regelbasierten Prozessmodell-Editor könnten Anwendertests durchgeführt werden, um zu evaluieren, inwiefern die in dieser Arbeit entwickelten Konzepte bei der tatsächlichen Anwendung die Prozessentwicklung sinnvoll unterstützen können sowie ob die Bedienung sich den Anwendern intuitiv erschließt. Des Weiteren ließe sich der Demonstrator erweitern, um auch die Verwendung in anderen Softwarekomponenten als dem Prozessmodell-Editor zu unterstützen, etwa bei der Durchführung von Ad-hoc-Änderungen oder der Analyse von Ausführungs-Logs.

Das dieser Arbeit zugrunde liegende Regelmodell beschränkt sich auf strukturelle Prozesseigenschaften – das Vorkommen/Nichtvorkommen von Aktivitäten im Prozess sowie Reihenfolge- und Zeitbeziehungen zwischen Knoten. Hier bestände die Möglichkeit für weitergehende Arbeiten, dies um andere Aspekte zu erweitern, etwa die mit Aktivitäten verknüpften Ein- und Ausgabedaten oder personelle und technische Ressourcen, die für die Bearbeitung von Aufgaben benötigt werden. So könnten beispielsweise Regeln beschrieben werden, die fordern, dass mehrere Aktivitäten von derselben Person durchgeführt werden müssen oder eine Aktivität nicht ausgeführt werden darf, wenn bei einer vorhergehenden Aktivität ein Ausgabedatum einen bestimmten Wert angenommen hat. Dies ließe sich beispielsweise durch Hinzufügen weiterer Prädikattypen in den Regeln erreichen, die sich auf zusätzliche Kanten- bzw. Knotentypen in Regelgraphen abbilden ließen. Bei einer solchen Erweiterung könnten die in dieser Arbeit vorgestellten Konzepte der Ergebnisdarstellung mit nur geringem Aufwand angepasst werden, indem für die Darstellung der Regelverletzungen im Prozessmodell die in den Regelgraphen eingeführten zusätzlichen Kanten- und Knotentypen übernommen werden. Bei laufenden bzw. abgeschlossenen Prozessinstanzen wäre ebenfalls eine Erweiterung der Prozessdarstellung um die jeweiligen tatsächlichen Daten- bzw. Ressourceninformationen sinnvoll, die sich bei der Ausführung des Prozesses ergeben haben.

Zur detaillierten Untersuchung von Regelverletzungen wird in dieser Arbeit die Prozessansicht verwendet. Es ist jedoch auch denkbar, die Erfülltheit detailliert in der Regelgraphansicht des Regel-Editors darzustellen, wenn Regeln bearbeitet werden, die bereits Prozessmodellen zugeordnet wurden. Hierzu könnte beispielsweise der Regelgraph analog zum Prozessgraphen um Knotenmarkierungen erweitert werden. Die in dieser Arbeit vorgestellte Ergebnisstruktur könnte hierzu weitgehend übernommen werden.

Um temporale Eigenschaften laufender oder abgeschlossener Prozessinstanzen besser visualisieren zu können, ist ebenfalls denkbar, eine zeitbasierte Prozessansicht einzuführen,

in der die bereits ausgeführten Knoten auf einer Zeitachse angeordnet werden. So könnten Regeln, die Zeitbeziehungen enthalten, noch besser analysiert werden.

Weitere Ergänzungen wären auch in Bezug auf das Prozessmodell selbst möglich. In dieser Arbeit wird davon ausgegangen, dass alle Aktivitäten korrekt ausgeführt werden können bzw. im Fehlerfall der Prozess angehalten wird und eine manuelle Fehlerbehandlung durchgeführt werden muss. Wird eine automatische Fehlerbehandlung, etwa durch im Prozessmodell definierte Kompensationsschritte, ermöglicht, sollte der Fehlerfall bei der Analyse der Erfülltheit von Integritätsregeln berücksichtigt werden, was sich auch in der Ergebnisdarstellung widerspiegeln müsste. Durch die Flexibilität der Ergebnisstruktur und -darstellung lassen sich die Ergebnisse dieser Arbeit jedoch leicht an solche Erweiterungen anpassen: Durch die Ermöglichung der Fehlerbehandlung wird das Auswertungssubjekt lediglich um weitere mögliche Ausführungsspuren erweitert, die Ergebnisstruktur müsste somit nicht angepasst werden, es würde lediglich notwendig, eine Darstellung für diese Ausführungsspuren in der Prozessansicht zu finden. Dies könnte beispielsweise durch eine manuelle Anpassungsmöglichkeit durch den Benutzer, ähnlich der Schleifenexpansion, gelöst werden, indem sich nach beliebigen Prozessschritten in der Darstellung die entsprechenden Fehlerbehandlungsschritte einblenden ließen.

Ebenfalls nicht betrachtet wurden in dieser Arbeit hierarchische Workflows – Prozesse, in denen einzelne Knoten weitere Prozesse repräsentieren können. Die Darstellung der Erfülltheit könnte hierbei z. B. erfolgen, indem Knoten, die verschachtelte Prozesse enthalten, ähnlich dem Process Block Folding »aufgeklappt« werden können, um die Erfülltheit der Regeln in den enthaltenen Knoten zu analysieren. Analog zu der Darstellung »eingeklappter« Knoten im Process Block Folding könnte bei nicht expandierten Knoten ein aggregierter Erfülltheitswert dargestellt werden.

Das in dieser Arbeit vorgestellte Darstellungskonzept nutzt wesentlich den blockstrukturierten Aufbau von ADEPT-Prozessen als Grundlage für eine übersichtliche Ergebnisdarstellung durch Spurlinien sowie für die Konzepte des Process Block Folding und der Schleifenexpansion. Die Ergebnisstruktur hingegen ist durch die Verwendung von allgemeinen Ausführungsspuren (bis auf die in den Spuren verwendete schleifenorientierte Definition von Knotenausführungen) weitgehend unabhängig von der Prozessstruktur. Weiterführende Arbeiten könnten untersuchen, inwiefern sich auf Grundlage einer angepassten Ergebnisstruktur Darstellungskonzepte für andere Prozessstrukturen entwickeln lassen.

7.2 Schlusswort

Wie die Anforderungsanalyse gezeigt hat, bietet die Überprüfung von Compliance zahlreiche neue Möglichkeiten, die Entwicklung, Analyse und Ausführung von Prozessen zu unterstützen. Durch an die spezifischen Aufgaben und Anforderungen der Anwender angepasste Analysemöglichkeiten der Integrität von Geschäftsprozessen wird die Verwendung von Regeln in BPM-Systemen angenehmer und effizienter, da durch eine detaillierte und verständliche Ergebnisdarstellung Fehler im Prozess schneller und einfacher lokalisiert und behoben werden können. In der Folge führt die Verwendung von Integritätsregeln zu einer Verbesserung der Qualität der Prozesse. Wenn für den Anwender ersichtlich wird, dass Integritätsregeln nicht nur Fehler in Prozessen reduzieren, sondern auch die Prozessentwicklung selbst vereinfachen können, steigt zudem die Motivation der Prozessbetreuer, Regeln selbstständig einzusetzen. Die in dieser Arbeit vorgestellten Konzepte stellen einen wichtigen Schritt dar, um diese Ziele zu erreichen.

Literaturverzeichnis

[1] ARISTAFLOW GMBH: *AristaFlow BPM Suite*. `http://www.aristaflow.com/`

[2] AWAD, A. ; WESKE, M. : Visualization of compliance violation in business process models. In: *Proc. BPI'09*, 2009

[3] AWAD, A. ; SMIRNOV, S. ; WESKE, M. : Towards Resolving Compliance Violations in Business Process Models. In: *Proceedings of GRCIS*, 2009

[4] BOTTONI, P. ; KOCH, M. ; PARISI-PRESICCE, F. ; TAENTZER, G. : Consistency Checking and Visualization of OCL Constraints. In: *«UML» 2000 – The Unified Modeling Language*, Springer Berlin/Heidelberg, 2000, S. 294–308

[5] CARRO, M. ; HERMENEGILDO, M. : Some Design Issues in the Visualization of Constraint Logic Program Execution. In: *Proc. Joint Conference on Declarative Programming (AGP'98)*, 1998

[6] CLARKE, E. M. ; GRUMBERG, O. ; PELED, D. A.: *Model Checking*. MIT Press, 2000

[7] DADAM, P. ; REICHERT, M. ; RINDERLE, S. ; JURISCH, M. ; ACKER, H. ; GÖSER, K. ; KREHER, U. ; LAUER, M. : ADEPT2 – Next Generation Process Management Technology. In: *Proceedings Fourth Heidelberg Innovation Forum*, 2007

[8] DADAM, P. ; KUHN, K. ; REICHERT, M. ; BEUTHER, T. ; NATHE, M. : ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen. In: *Proc. 25 GI-Jahrestagung und 13. Schweizer Informatikertag GISI 95, Zürich*, 1995

[9] DADAM, P. ; REICHERT, M. ; RINDERLE-MA, S. ; GOESER, K. ; KREHER, U. ; JURISCH, M. : Von ADEPT zur AristaFlow BPM Suite – Eine Vision wird Realität: „Correctness by Construction“ und flexible, robuste Ausführung von Unternehmensprozessen / Ulmer Informatik-Berichte. 2009 (2009-02). – Forschungsbericht

[10] DE PAUW, W. ; JENSEN, E. ; MITCHELL, N. ; SEVITSKY, G. ; VLISSIDES, J. ; YANG, J. : Visualizing the Execution of Java Programs. In: *Software Visualization* LNCS 2269 (2002), S. 151–162

[11] ECLIPSE FOUNDATION: *Eclipse project*. `http://www.eclipse.org/`

[12] EL KHARBILI, M. ; STEIN, S. ; MARKOVIC, I. ; PULVERMÜLLER, E. : Towards a Framework for Semantic Business Process Compliance Management. In: *Proceedings of GRCIS 2008*, 2008

[13] GHOSE, A. ; KOLIADIS, G. : Auditing Business Process Compliance. In: *Service-Oriented Computing – ICSOC 2007* (2007), S. 169 – 180

[14] GOSLING, J. ; JOY, B. ; STEELE, G. ; BRACHA, G. : *The Java™ Language Specification*. Third Edition. Addison-Wesley `http://java.sun.com/docs/books/jls/`

[15] GOVERNATORI, G. ; MILOSEVIC, Z. : Dealing with contract violations: formalism and domain specific language. In: *EDOC 2005*, IEEE Press, 2005, S. 46–57

[16] GOVERNATORI, G. ; MILOSEVIC, Z. ; SADIQ, S. : Compliance checking between business processes and business contracts. In: *EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*. Washington, DC, USA : IEEE Computer Society, 2006, S. 221–232

[17] KAHN, A. B.: Topological sorting of large networks. In: *Commun. ACM* 5 (1962), Nr. 11, S. 558–562

[18] LIU, Y. ; MÜLLER, S. ; XU, K. : A static compliance-checking framework for business process models. In: *IBM Systems Journal* 46 (2007), Nr. 2, S. 335–362

[19] LY, L. T. ; RINDERLE-MA, S. ; DADAM, P. : Integration and verification of semantic constraints in adaptive process management systems. In: *Data and Knowledge Engineering* 64 (1) (2008), S. 3–23

[20] LY, L. T. ; RINDERLE-MA, S. ; DADAM, P. : Design and Verification of Instantiable Compliance Rule Graphs in Process-Aware Information Systems. In: *22nd International Conference on Advanced Information Systems Engineering (CAiSE'10)*, 2010

[21] LY, L. T. ; RINDERLE-MA, S. ; GÖSER, K. ; DADAM, P. : On enabling integrated process compliance with semantic constraints in process management systems – Requirements, challenges, solutions. In: *Information Systems Frontiers* (2009)

[22] MARJANOVIC, O. : Dynamic Verification of Temporal Constraints in Production Workflows. In: *Agile Development Conference/Australasian Database Conference* (2000)

[23] MUTHUKUMARASAMY, J. ; STASKO, J. T.: Visualizing Program Executions on Large Data Sets Using Semantic Zooming / Graphics, Visualization, and Usability Center, College of Computing, Georgia Institute of Technology. 1995 (GIT-GVU-95-02). – Forschungsbericht

[24] OBJECT MANAGEMENT GROUP: *MOF 2.0/XMI Mapping, Version 2.1.1*, Dezember 2007. `http://www.omg.org/spec/XMI/2.1/`

[25] OBJECT MANAGEMENT GROUP: *Business Process Model and Notation (BPMN), Version 1.2*, Januar 2009. `http://www.omg.org/spec/BPMN/1.2/`

[26] REICHERT, M. : *Dynamische Ablaufänderungen in Workflow-Management-Systemen*, Universität Ulm, Dissertation, Mai 2000

[27] REICHERT, M. ; DADAM, P. : ADEPTflex – Supporting Dynamic Changes of Workflows Without Loosing Control / Ulmer Informatik-Berichte. 1997 (97-07). – Forschungsbericht

[28] RINDERLE, S. : *Schema Evolution in Process Management Systems*, Universität Ulm, Dissertation, Oktober 2004

[29] SCHÖNING, U. : *Logik für Informatiker*. Spektrum Akademischer Verlag, Heidelberg, 2000

[30] SWT-ENTWICKLER, ECLIPSE FOUNDATION: *SWT: The Standard Widget Toolkit*. `http://www.eclipse.org/swt/`

[31] VAN DER AALST, W. : Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: *Proc. BPM '00*, 2000

[32] VAN DER AALST, W. ; HOFSTEDE, A. ; WESKE, M. : Business process management: A survey. In: *Lecture Notes in Computer Science* (2003), S. 1–12

Abbildungsverzeichnis

1.1 Business Process Lifecycle nach [32] und modifizierte Variante 3
1.2 Schematische Darstellung der Aufgabenstellung 5

2.1 Knoten- und Kantentypen in ADEPT 11
2.2 Blockstrukturierung in einem ADEPT-Prozessmodell 12
2.3 Das Zustandsmodell eines Knotens in ADEPT (aus [28], S. 61) 13
2.4 Prozessmodell mit verschachtelten Schleifen 15
2.5 Beispiel-Prozessmodell zu Ausführungsspuren 17
2.6 Graphische Darstellung der Elemente eines Regelgraphen 21
2.7 Erlaubte Kanten in Regelgraphen 22
2.8 Kanten aus dem Folgeteil mit Knoten aus dem Bedingungsteil. 23
2.9 Beispielregel 24
2.10 Beispielregel mit Hervorhebung der Regelglieder und Negativelemente . . . 27
2.11 Beispielregel mit farbiger Hervorhebung der Existenzgruppen 28

3.1 Betrachtete Nutzergruppen und ihre Interaktion mit dem BPM-System 38

4.1 Anforderungen an die Ergebnisstruktur 58
4.2 Überblick über den Aufbau der Ergebnisstruktur 61
4.3 Beispiel-Regelgraph und einfaches Prozessmodell 65

5.1 Übersichtsliste 87
5.2 Process Block Folding 90
5.3 Schleifenexpansion 91
5.4 Zwei Spurlinien, die nicht vereinigt werden können 94
5.5 Prozessmodell mit zwei Spurlinien und teilweise eingeklappten Blöcken . . . 95
5.6 Zwei Spurlinien, davon je eine hervorgehoben 96
5.7 Darstellung bei zu vielen Spurlinien 97
5.8 Darstellung der Gesamterfülltheit eines Prozessmodells 98

5.9 Darstellung einer bedingten Spurlinie bei Hervorhebung 99
5.10 Markierte Knoten in einem Prozessmodell 99
5.11 Liste mit Regelverletzung nach Klick auf Knotenmarkierung 100
5.12 Darstellung einer Regelverletzung durch Regelgraphfragment und Spurlinie 102
5.13 Hervorhebung von Prozessknoten bei Mausbewegung über Regelgraphknoten 103
5.14 Liste mit Alternativverletzungen 104

6.1 Klassenstruktur zur Darstellung von Integritätsregeln 109
6.2 Klassenstruktur für die Ergebnisstruktur und Spurmengen 109
6.3 Der im Demonstrator umgesetzte Rule Based Process Editor 116
6.4 Klassenstruktur zur Prozess- und Ergebnisdarstellung 117
6.5 Funktionsweise des Beispielalgorithmus 133

Definitionsverzeichnis

Definition 2.1 Knotenausführung 15
Definition 2.2 Potentielle Ausführungsspur 16
Definition 2.3 Ausführungsspur 16
Definition 2.4 Interpretation der Prädikatensymbole 29
Definition 2.5 Interpretation einer Integritätsregel 30
Definition 4.1 Gesamterfülltheit 62
Definition 4.2 Nicht passende Regel 62
Definition 4.3 Hilfsdefinitionen 67
Definition 4.4 (Einfache) Regelverletzung 68
Definition 4.5 (Erweiterte) Regelverletzung 70
Definition 4.6 Alternativverletzung 71
Definition 4.7 Unscharfe Spurmenge 73
Definition 4.8 Unscharfe Gesamterfülltheit 73
Definition 4.9 Unscharfe Regelverletzung 74
Definition 5.1 Spurlinie 92

Beispielverzeichnis

2.1 Blockstrukturierung 12
2.2 Knotenausführungen 15
2.3 Ausführungsspuren 17
2.4 Aktivitätentypen 18
2.5 Regelgraph 24
2.6 Regel in prädikatenlogischer Form (ohne Erweiterung) 26
2.7 Regel in prädikatenlogischer Form (mit Erweiterung) 28
2.8 Interpretation einer Integritätsregel 30
3.1 Regeln zur Sicherstellung vorgegebener Randbedingungen 42
3.2 Regeln zum Testen von Prozesseigenschaften 43
3.3 Regeln, die als Grundlage der Prozessentwicklung dienen 44
3.4 Regel zur Konsistenthaltung bei Änderungen 45
3.5 Regel zur Ergänzung der Prozessspezifikation 45
3.6 Beziehungen zwischen Verletzungen einer Regel 54
4.1 Regelverletzungen 65
4.2 Formale Regelverletzungen 69
4.3 Erweiterte Regelverletzungen 70
4.4 Alternativverletzungen 71

Index

Absence-Knoten, 20
Ad-hoc-Änderungen, 2, 33
ADEPT, 2, 6, 10
Aktivität, 11
Aktivitätentyp, 18, 107
    Editor, 33
Algorithmus, *siehe* Auswertungsalgorithmus
Alternativverletzung, 63
    Darstellung, 103
    Definition, 71
AND-Block, 12
Animation, 123
Arbeitsliste, 33
AristaFlow, 34, 106
Ausführungs-Logs, 2
Ausführungsspur, 15
    Darstellung, 93
    Definition, 16
    Einsatz, 62
    Umsetzung, 108
Ausführungsspuren, 10
Auswertungs-Engine, 31, 129
Auswertungsalgorithmus, 31, 129
    Beispielalgorithmus, 131
Auswertungssubjekt, 10
AutoFolding, 90, 122

Bedingungs-Matching, 63, 64, 68
Bedingungsteil, 20, 107
Business Process Lifecycle, 3, 46, 51
Business Process Management, 2
Business Rules, 3, 42

Compliance, *siehe* Erfülltheit

Demonstrator, 105

Endanwender, 37, 79
    Anforderungen, 41
    Aufgaben, 40
Endbenutzeransicht, 79
Erfülltheit, 3, 24, 53, 86
Ergebnisstruktur, 57, 113
    Aufbau, 60
Ergebnisunschärfe, 72
Erweiterte Regelverletzung, 70
Existenzgruppe, 27, 108
    Anwendung, 64

Folding, *siehe* Process Block Folding
Folgeglied, 20, 107
Folgeteil, 20
Folgeteil-Bindung, 63, 68

Gesamterfülltheit, 60
    Darstellung, 81, 86, 97

Definition, 62
Umsetzung, 114
unscharfe, 73
Gesamterfülltheitsmodus, 97
Gesamtlistenansicht, 80

Integritätsregel, 3, 19
Interpretation, 29
Prädikatenlogische Formalisierung, 24
Regelgraph, *siehe* Regelgraph
Umsetzung, 107
Iteration, 14
Iterationsnummer, 15, 113

Joinknoten, 11

Knotenausführung, 15, 92, 113
Knotenmarkierung, 99
Kontrollflusskante, 11

Laufzeit, 47, 51
Lebenszyklus, *siehe* Business Process Lifecycle
Listenansicht, 81, 84, 86
Filterung, 82, 87, 99
Gruppierung, 82, 84, 86
Umsetzung, 129

Model Checking, 6, 32

Negativelement, 25, 65, 108
Nutzergruppen, 36
Nutzerschnittstelle, 77

Occurrence-Knoten, 20

Prädikat, 24
Prädikatenlogik, 6
Prädikatenlogische Formel, 24
Process Block Folding, 89, 120
Prozessansicht, 84, 106, 115
Prozessbeschreibungssprache, 10
Prozessbetreuer, 36
Anforderungen, 40
Aufgaben, 38
Prozessinstanz, 2
Prozesslebenszyklus, *siehe* Business Process Lifecycle
Prozessmanagement, 2
Prozessmodell, 2, 11, 108
Prozessmodell-Editor, *siehe* Prozessvorlagen-Editor
Prozessmodellierung, 46, 51
Prozessvorlage, *siehe* Prozessmodell
Prozessvorlagen-Editor, 33, 38, 84, 106

Quantifizierung, 25

Rücksprungkante, 12
Regel-Editor, 39, 83
Regelansicht, 83
Regelbetreuer, 36, 83
Anforderungen, 41
Aufgaben, 39
Regeleditor, 33
Regelfragment, *siehe* Regelgraph-Fragment
Regelgraph, 20
Darstellung, 21
Semantik, 22
Regelgraph-Fragment, 101, 127
Regelkante, 20
Regelkanten, 101
Regelklasse

Darstellung, 81
Regelklassen, 49
Regelknoten, 20, 101
Regelverletzung, 61, 63
Darstellung, 86, 98, 101
Definition, 68
Erweiterte Definition, 70
Textuelle Beschreibung, 87
Umsetzung, 114, 126
unscharfe, 74
Regelverletzungsmodus, 101
Reihenfolgekante, 21

Schemaevolution, 2
Schleife, 12
Schleifenblock, 12
Schleifenexpansion, 91, 121
SeaFlows, 6, 19
Splitknoten, 11
Spurlinie, 92
Bedingung, 93, 95, 98
Gesamterfülltheit, 97
Regelverletzung, 102
Umsetzung, 124
Unschärfe, 95
Spurmenge, 61, 110
unscharfe, 72, 112
Subtyp, 18
Sync-Kante, *siehe* Synchronisationskante
Synchronisationskante, 13

Typisierungsprädikat, 24, 26

Übersichtsliste (Prozessansicht), 86, 106, 129
Ungleichheitskante, 21
Unschärfe, *siehe* Ergebnisunschärfe

Variable, 24
Verletzter Regelteil, 63, 68

WSM-Netze, 10

XOR-Block, 12

Zeitkante, 21
Zoom, 89
Zweigeinschränkung, 110
Bedingung, 111

Name: Philipp Merkel Matrikelnummer: 594451

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den _____ _____

Philipp Merkel